

CLOUD COMPUTING AND DISTRIBUTED SYSTEMS



Prof. Rajiv Misra
Computer Science and Engineering
IIT Patna



INDEX

S. No	Topic	Page No.
	<i>Week 1</i>	
1	Introduction to Cloud Computing	1
2	Virtualization	27
3	Hotspot Mitigation for Virtual Machine Migration	56
	<i>Week 2</i>	
4	Server Virtualization	83
5	Software Defined Network	111
6	Geo-distributed Cloud Data Centers	149
	<i>Week 3</i>	
7	Leader Election in Rings (Classical Distributed Algorithms)	179
8	Leader Election (Ring LE & Bully LE Algorithm)	200
9	Design of Zookeeper	226
	<i>Week 4</i>	
10	Time and Clock Synchronization in Cloud Data Centers	261
11	Global State and Snapshot Recording Algorithms	298
12	Distributed Mutual Exclusion	318
	<i>Week 5</i>	
13	Consensus in Cloud Computing and Paxos	370
14	Byzantine Agreement	394
15	Failures & Recovery Approaches in Distributed Systems	439
	<i>Week 6</i>	
16	Design of Key-Value Stores	475
17	Design of HBase	516
	<i>Week 7</i>	
18	Peer to Peer Systems in Cloud Computing	538

Week 8

19	MapReduce	574
20	Introduction to Spark	618
21	Introduction to Kafka	651

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute Of Technology, Patna

Lecture - 01
Introduction to Cloud Computing

Introduction to Cloud Computing. Preface, content of this lecture you we will discuss a brief introduction to cloud computing.

(Refer Slide Time: 00:20)

Preface

Content of this Lecture:

- In this lecture; we will discuss a brief introduction to Cloud Computing and also focus on the aspects i.e. Why Clouds, What is a Cloud, Whats new in todays Clouds and also distinguish Cloud Computing from the previous generation of distributed systems.



The diagram illustrates the three-tier architecture of cloud computing. It features a central cloud icon containing three distinct layers: 'Application' at the top, 'Platform' in the middle, and 'Infrastructure' at the bottom. Various icons representing different services like databases, servers, and storage are scattered around the cloud, symbolizing the interconnected nature of cloud resources.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

And also focus on the aspects such as; why clouds, what is a cloud, what is new in today's cloud and also distinguish cloud computing from the previous generations of computing system that is, the distributed system.

(Refer Slide Time: 00:40)

Scalable Computing Over the Internet

- Evolutionary changes that have occurred in **distributed and cloud computing** over the past 30 years, **driven by applications with variable workloads and large data sets**.
- Evolutionary changes in machine architecture, operating system platform, network connectivity, and application workload.
- Distributed computing** system uses multiple computers to solve large-scale problems over the Internet. Thus, **distributed computing becomes data-intensive and network-centric**.
- The **emergence of computing clouds** instead demands high-throughput computing (HTC) systems built with distributed computing technologies.
- High-throughput computing (HTC)** appearing as computer clusters, service-oriented architecture, computational grids, peer-to-peer networks, Internet clouds, and the future Internet of Things.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Scalable computing over the internet, so the evolutionary changes that have occurred in distributed and cloud computing over past 30 years are driven by the applications, with variable workload and large data sets. These evolutionary changes are happening in the machine architectures, operating system, platform, network connectivity and application workloads. The distributed computing systems uses multiple computer to solve large scale problems over the internet. Thus, distributed computing becomes the data intensive and network centric.

The emergence of computing clouds, so instead the demands of high throughput computing systems build with distributed computing systems has led to the emergence of the need of the cloud. So, in high throughput computing we will see that the systems which appears they are basically the computing clusters, service oriented architecture, computational grids, peer to peer networks, internet cloud and the future of internet of things.

(Refer Slide Time: 02:12)

The Hype of Cloud: Forecasting

- Gartner in 2009 – Cloud computing revenue will soar faster than expected and will **exceed \$150 billion** by 2013. It will represent 19% of IT spending by 2015.
- IDC in 2009: "Spending on IT cloud services will triple in the next 5 years, reaching **\$42 billion**."
- Forrester in 2010 – Cloud computing will go from **\$40.7 billion** in 2010 to **\$241 billion** in 2020.
- Companies and even federal/state governments using cloud computing now: **fbo.gov**

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Let us see the hype of cloud which was forecasted around 10 years back, and using that particular predictions we will see how far these predictions are towards moving the cloud further. Gartner in 2009 has predicted that the cloud computing will soar its revenue much faster; than the expected at that point of time and we will exceed cross a huge revenue mark. And it will represent the 19 or 20 % growth of IT's spending in terms of the cloud computing by 2015.

Similarly, IDC in 2009 also predicted the same thing that; spending in IT cloud services will triple in the next 5 years. And Forrester in 2010 has predicted also the same thing; that is the cloud computing will go as far as spending is concerned in 2010 to 2020; crossing a several that is 5 folds' increase. So, companies and even new federal governments are also using the cloud computing.

(Refer Slide Time: 03:48)

Many Cloud Providers

- AWS: Amazon Web Services
 - EC2: Elastic Compute Cloud
 - S3: Simple Storage Service
 - EBS: Elastic Block Storage
- Microsoft Azure
- Google Compute Engine/AppEngine
- Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera
- And 100s more...



Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, with this particular forecast let us see; what is the current status, how many key players in the cloud is available as the providers. The first one is called as Amazon Web Service, which is a most prominent cloud provider. So, it provides the cloud services in 3 different types; the first one is called Elastic Compute Cloud, the second one is S3 that is Simple Storage Service, the third one is called EBS Elastic Block Storage. The second cloud provider which is also well known is called Microsoft Azure, and Amazon and Microsoft Azures normally they provide similar kind of cloud services.

The third one is called the Google Compute Engine that is also known as App Engine, that is also a cloud provider. Besides these 3 prominent cloud providers there are several other cloud providers such as; Rightscale, Salesforce, EMC, Gigaspaces, 10gen, Datastax, Oracle, VMWare, Yahoo, Cloudera and there are 100 many more in this particular arena.

(Refer Slide Time: 05:09)

Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- **Private clouds** are accessible only to company employees
- **Public clouds** provide service to any paying customer:
 - **Amazon S3 (Simple Storage Service)**: store arbitrary datasets, pay per GB-month stored
 - **Amazon EC2 (Elastic Compute Cloud)**: upload and run arbitrary OS images, pay per CPU hour used
 - **Google App Engine/Compute Engine**: develop applications within their App Engine framework, upload data that will be imported into their format, and run

Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, there are categories of clouds which are available, they are called as public cloud and the private cloud. Public clouds are accessible internal to the companies and they are also managed internally by the company, and all it is basically like energy usage. It is maintenance all are owned by the company itself; hence it is called a private cloud. It is not available it is not accessible by outside people.

The other type of cloud is called a public cloud, which provide the services to any paying customer; that means, it is open for anyone who wants to use the cloud by paying the cost, that is why it is called a public cloud. The cloud provider has to basically maintained, then bare the cost of energy and so on and so forth. Only the customers who want to use it they have to just pay as they use it. The example of public clouds are the Amazon S3, Amazon S3 is a Simple Storage Service will store arbitrary datasets, and the users has to pay the money as per the amount of space which are basically rented are used for the storage, that is; in terms of GB per month.

The second kind of public cloud is given by the Amazon as EC2 that is; Elastic Compute cloud, and this particular elastic compute cloud will provide the compute services to the client. So, as per as any user can upload and run an arbitrary operating system images. And based on this several operating system, which are basically given and different applications can basically be used be run on this kind of system.

These particular different operating systems, images and applications which runs will require now the CPU. So, therefore, the user has to pay, how many CPUs they require; that means, instead of numbers, but they have to pay as per the CPU hour, which is being used by the applications.

Similarly, the third kind of service which basically is categorized under the public cloud is, example is called Google app engines; here the users can develop their applications within this app engine framework and they upload their data that will be imported into their format, and it can run. So, it will give you that Google app engine gives more flexibility in terms of the directly the programming customer can do. And they can use the entire framework to solve their applications, and they can pay the money accordingly the use.

(Refer Slide Time: 08:20)

Customers Save: Time and Money

- With AWS, a new server can be up and running in **three minutes** compared to **seven and a half weeks** to deploy a server internally and a **64-node Linux cluster** can be online in five minutes (compared with three months internally).
- With Online Services, reduce the IT **operational costs** by roughly **30%** of spending.
- A private cloud of virtual servers inside its datacenter has saved nearly **crores of rupees annually**, because the company can share computing power and storage resources across servers.
- 100s of startups can harness large computing resources without buying their own machines.

Cloud Computing and Distributed Systems | Introduction to Cloud Computing

As far as the customers are concerned, in this cloud scenario the customers will save the time and money how that we are going to explain over here. So, if AWS is being used, then basically a new server can be up and running and within a fraction of minutes that is; 3 minutes compared to several weeks and months to purchase the server, and then basically put it into the service. All these particular cost of invoicing purchasing and installation will be now reduced and only 2 or 3 minutes are required to install a new server and run. So, time is saved if the cloud is opted as the method of computing instead of owning the own servers.

Another example is regarding with the online services will reduce this operational cost by around 30 % of the spending of the internal company, why because operational cost is not at all required. The only amount of money it has to be paid whatever as per the use. Therefore, it is a saving of money also if the cloud is used for computing purposes. A private cloud of the virtual servers inside the data centre has saved nearly crores of rupees annually because company can share; computing power and storage resources across the servers.

So, again this is also going to be very cost effective usage, if even if the private cloud is maintained inside the data centre. We will see the economics when it is required to be go for the private cloud versus the public cloud. Also there are various startup companies they can harness large computing resources without buying their own machines. This also is based on the economics calculations, whether to go for a your own private cloud or a own or basically use the public cloud, but as far as we will see all these things why, so most many options are available; and these options open up new arena of a cloud computing. So, what is a cloud?

(Refer Slide Time: 10:59)

What is a Cloud?

- Advances in virtualization make it possible to see the growth of Internet clouds **as a new computing paradigm**.
- i.e. dramatic differences between developing software for millions to use **as a service** versus distributing software to run on their PCs."

History:

- In 1984, John Gage Sun Microsystems gave the slogan, "**The network is the computer.**"
- In 2008, David Patterson UC Berkeley said, "**The data center is the computer.**"
- Recently, Rajkumar Buyya of Melbourne University simply said: "**The cloud is the computer.**"
- Some people view **clouds as grids** or **clusters** with changes through virtualization, since clouds are anticipated to process huge data sets generated by the traditional Internet, social networks, and the future IoT.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Now, here we will see that the advances in the virtualization makes it possible, the growth of internet clouds as a new computing paradigm that is; there is a dramatic difference between developing a software for millions to use as a service versus developing a software and distributing it to run on their own PCs. So, the architecture in

a cloud computing is now slightly changed, where the software will be given as a service to the millions rather than the software is to be distributed to run on their PCs.

So, the cloud has changed the new paradigm and let us trace back through the history. In 1984 John Gage of Sun Microsystems gave the slogan that the network is the computer. Similarly, later on in 2008 David Patterson of UC Berkeley has said that; the data center is the computer recently Rajkumar Buyya of Melbourne university simply said the cloud is the computer. So, just see the way the paradigm is shifting the definition of computer is not changing from network to the data center, now it is the cloud.

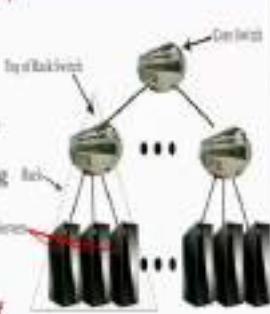
So, some people view the clouds as a grid or a clusters, with changes through the virtualization these clouds are anticipated to process huge data sets which are generated by the traditional internet, social networks and future IOTs.

(Refer Slide Time: 12:46)

What is a Cloud?

- A single-site cloud (as known as "Datacenter") consists of
 - Compute nodes (grouped into racks)
 - Switches, connecting the racks
 - A network topology, e.g., hierarchical
 - Storage (backend) nodes connected to the network
 - Front-end for submitting jobs and receiving client requests
 - (Often called "three-tier architecture")
 - Software Services
- A geographically distributed cloud consists of
 - Multiple such sites
 - Each site perhaps with a different structure and services

Cloud Computing and Distributed Systems Introduction to Cloud Computing



So, we will see the inside what is there in the cloud. So, if you go inside we will see; that there are two kinds of setup you will see inside the cloud. The first one is called single side cloud that is within one premises. That cloud is called the datacenter, which comprises of the compute nodes which are grouped into the racks, which are shown here these are the servers, which are grouped into these particular racks, this compute nodes are there sometimes they are also called as server.

Then comes the switches which are connecting these racks. So, every rack will have a top of the rack switch, which is basically mentioned over here and they are connecting all the racks. Then comes this particular network topology which will be of 2 level, and within this particular rack will also be the storage backend nodes connected to the network so; that means, there are nodes also within that particular rack which are primarily meant for the storage purposes. They are having the SSDs within it, so basically used primary for the storage.

Now, frontend will be there for submitting the jobs and receiving the client requests. So, often this can be treated as 3 tier architecture. Here there is a core switch which we will connect all the different top of the rack switches. So, this particular hierarchy and there is a software services, which will basically will be used to run the applications on this kind of structure, which is basically nothing but a datacenter with the clusters within it.

Now, there may be there are different cloud providers, which has deployed this kind of set up at more than one sites they are called geographical distributed clouds. And which comprises of multiple search sites that is; multiple search datacenters which are connected together and each site perhaps with a different structure and the services running within it they can communicate over the fast network.

(Refer Slide Time: 15:09)

Computing Paradigm Distinctions

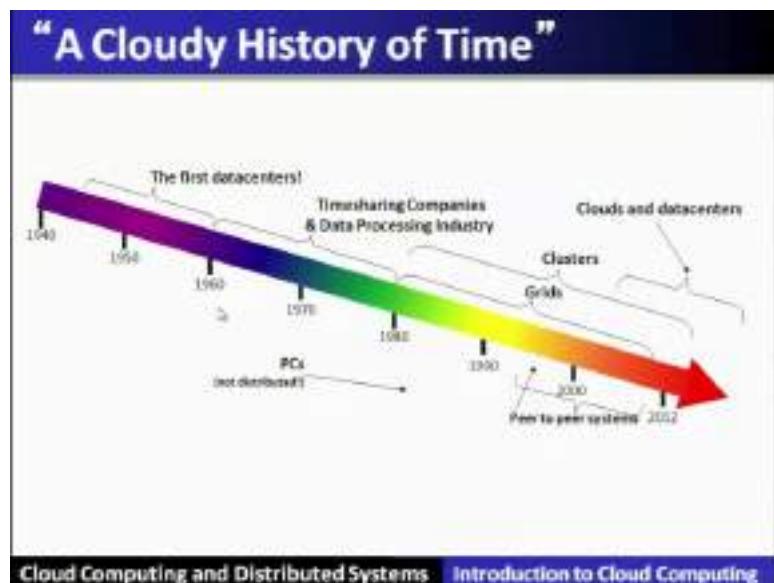
- Cloud computing overlaps with distributed computing.
- Distributed computing:** A *distributed system* consists of multiple autonomous computers, having its own memory, communicating through *message passing*.
- Cloud computing:** Clouds can be built with physical or virtualized resources over large data centers that are distributed systems. Cloud computing is also considered to be a form of *utility computing or service computing*.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, that was some of the interval the description of the cloud, what comprises of the cloud. Now, this cloud what is basically the computing paradigm which makes the distinction as a cloud computing that we will see. So, there is a wide overlap between the cloud and the distributed computing. Distributed computing comprises of multiple autonomous computers having their own memory and they communicate through the messages called message passing. As far as the cloud is concerned cloud can be built with the physical or virtualized resources over the large data centers that are distributed systems.

So, basically on these particular distributed systems, the virtualization of the resource will create pool of virtualized resources. And this pool of virtualized resources can be allocated to the applications. And therefore, the cloud computing is having an overlap with a distributed system and also a flexibility or elasticity, which is called in terms of computing resources; so, the cloud computing is also considered to be a form of utility computing or the service computing.

(Refer Slide Time: 16:27)



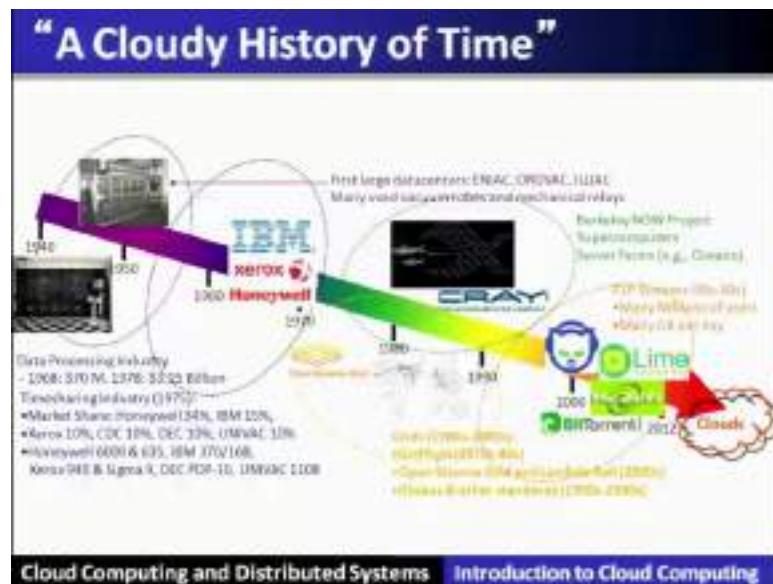
Let us trace back the history of the development of a cloud system. So, 1940s we will see we have seen that; here we have seen that the ENIAC the bigger the big data centers like ENIAC and ENIAC system was installed. And these particular systems in 1940s were housed in a big room. And that was the data centers we call those big rooms, which are full of CPUs called data centers, but primarily with a slower, slower computing facilities

which we have right now. Then afterwards then came the timesharing companies and the data processing industries were transformed into a timesharing system where terminals and the PCs terminals were used to express those systems.

And also the data if it is quite large then it is given in the form of a punch cards. And that industry was called data processing industry. Then there is a slight change after 1980s and the systems were now become the PCs personal computers and personal computers were given to the people directly to use it. And at the same point of time the grids were also evolved clusters were also evolved. Then using these particular systems, the peer to peer systems were formed they are the precursor or precursor to the current cloud computing systems.

Now, the current cloud computing systems are basically having the same setup, that is called datacenters which were there in 1940s so and 1960s, so the same cycle is being repeated, but with the different notions of the computing, so that more data or a big data can be computed into this kind of datacenters.

(Refer Slide Time: 18:37)



So, this is what is being summarized over here that; the precursor to the clouds are basically the peer to peer why because many PCs were available and they were, they were connected together to form a computing, and a cloud is basically further advancement of these kind of systems.

(Refer Slide Time: 19:01)

Scalable Computing Trends: Technology

- **Doubling Periods** – storage: 12 months, bandwidth: 9 months, and CPU compute capacity: 18 months (what law is this?)
- **Moore's law** indicates that processor speed doubles every 18 months.
- **Gilder's law** indicates that network bandwidth has doubled each year in the past.
- Then and Now
 - Bandwidth
 - 1985: mostly 56Kbps links nationwide
 - 2015: Tbps links widespread
 - Disk capacity
 - Today's PCs have TBs, far more than a 1990 supercomputer

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Now, as far as the amount of data and the flexibility in the applications for these resources, let us to the scalable computing trends and the technology is also evolving around that. So, we will see how the trend is in the technology perspective has taken up the shape and has given, now the birth for the cloud computing. So, as far as the hardware is concerned in the hardware we have seen the scalability or the growth that; it was and we such that the storage after every 12 months it was envisaged that it will be doubled.

Similarly, the bandwidth after every 9 month will be doubled similarly the compute CPU compute capacity also every 18 month will be doubled. Doubled in the sense with the same cost you will get the doubled the capacities, double the speed and double the bandwidth; that means, the doubling phenomena. So, what is the law behind this doubling of the periods? So, Moore's law indicates that the processor speeds doubles every 18 months. Although there is no basically doubling in the terms of the speed, but horizontally the development is taking place. So, number of course, are being packed more in a corresponding chip that is the trend now.

Similarly, the gilders law indicates that the network bandwidth has doubled each year in the past. Similarly, we see that earlier the bandwidth was in kbps kilobits per second. In 2015 we will see a terabytes per second, was a link speed of the same amount or a same cost. So, there is a tremendous increase and it is being followed the principles of a

doubling period. Similarly, the disk capacity today's PCs have terabytes for more than 1990s super computers.

(Refer Slide Time: 21:13)

The Trend toward Utility Computing

- Aiming towards autonomic operations that can be self-organized to support dynamic discovery. Major computing paradigms are composable with *QoS and SLAs (service-level agreements)*.
- In 1965, MIT's Fernando Corbató of the Multics operating system envisioned a computer facility operating "like a power company or water company".
- **Plug** your thin client into the computing Utility **and Play** Intensive Compute & Communicate Application
- **Utility computing** focuses on a business model in which customers receive computing resources from a paid service provider,
- All **grid/cloud platforms are regarded as utility service providers**.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, this all is moving towards the reality of utility computing. Let us see what do we mean by duality computing. So, aiming towards autonomic operations that can be self-organized to support dynamic discovery major computing paradigms are composable with quality of service and service level agreement SLAs. In 1965 MIT's Fernando Corbato of Multics operating system envisaged that; computing facility like operating systems like a power company are basically work like or like a water company.

So, power company or a water company works like the plug and play; that means, the power is available in the homes in the form of a socket, whenever required you can use the plug and continue to use it without having knowing the problems or the production at the power stations. Similarly, the water companies problems without knowing it, you can open the tap water will come out, using the same concept of utility computing.

So, computing also should be provided in the similar manner if it is then it is called the utility computing. That is the thin client can plug-in into the computing utility and play; that means, one can use the compute and also can run the applications. So, cloud in some form is realizing the hope of the computing utility computing. So, utility computing focuses on business model in which customers receive computing services from a paid

service provider. So, here all the grid oblique the cloud platforms are regarded as the utility service providers.

(Refer Slide Time: 23:14)

Features of Today's Clouds

- I. **Massive scale:** Very large data centers, contain tens of thousands sometimes hundreds of thousands of servers and you can run your computation across as many servers as you want and as many servers as your application will scale.
- II. **On-demand access:** Pay-as-you-go, no upfront commitment.
 - And anyone can access it
- III. **Data-intensive Nature:** What was MBs has now become TBs, PBs and XBs.
 - Daily logs, forensics, Web data, etc.
- IV. **New Cloud Programming Paradigms:** MapReduce/Hadoop, NoSQL/Cassandra/MongoDB and many others.
 - Combination of one or more of these gives rise to novel and unsolved distributed computing problems in cloud computing.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Features of today's clouds, so there are 4 different features which will categorize the cloud or the applications which are basically the cloud problems. The first one is massive scale, very large data centers contains tens of 100s of 1000s of servers. And you can run your applications across as many servers as you want and as many servers as your application will scale, that is called massive scale. We will see in more details of what do you mean by this massive scale and in terms of the cloud problem.

The second aspect or the feature which will classify as a cloud problem or a cloud computing is called on demand access. So, on demand access means pay as you go pay as you use; that means, it is different from upfront cost, upfront cost means you have to pay in advance and then whether you use that up to that level or not, that is not that is in contrast to that. So, on demand access the pay as you go, so these particular model also classify the problem into the cloud problem.

Third feature which classifies the problem into a cloud problem is called Data intensive nature. So, what was megabytes earlier now has become a bigger size that is in a terabyte Petabytes and Zettabytes. So, this size of data is growing and if the data size is quite large, then those problems falls into the cloud problems. Examples are daily logs forensics reports then weblogs, which will continuously generate the data that becomes

of that size data and it has to be solved in a data intensive nature, that is the computing of that category of that system is required that is called a cloud computing.

4th feature is called a new cloud programming paradigm. So, the problems of the big data or a large scale data or a data intensive nature of applications, require a new cloud programming paradigms for example, map reduce and it is open source version is called a new programming paradigm, which is used to solve this particular problems. So, this new programming paradigms also are classified as one of the features of the cloud problem.

Another thing is called the key value store, if it is then the systems like Cassandra is being used. Similarly, if the database is in the form of NoSQL then MongoDB is basically the programming paradigm which is being used. So, newer programming paradigms are available and if they are required then basically the problem is categorized in this particular today's cloud. Now, if we will see that if one or more of these above features are available then only we can classify the problem into a cloud computing problem.

(Refer Slide Time: 26:52)

I. Massive Scale

- Facebook [GigaOm, 2012]
 - 30K in 2009 -> 60K in 2010 -> 180K in 2012
- Microsoft [NYTimes, 2008]
 - 150K machines
 - Growth rate of 10K per month
 - 80K total running Bing
 - In 2011, Microsoft Search had 110K machines (4 sites)
- Yahoo! [2009]:
 - 100K
 - Split into clusters of 4000
- AWS EC2 [Randy Bias, 2009]
 - 40K machines
 - 8 cores/machine
- eBay [2012]: 50K machines
- HP [2012]: 380K in 180 DCs
- Google: A lot



Cloud Computing and Distributed Systems Introduction to Cloud Computing

Let us see in more detail what do you mean by the first feature of a cloud computing that is called a massive scale. Take for example, the Facebook application, Facebook application as of 2012 we have seen this particular data that there were 30000 servers were deployed in 2009, which has grown up to 60000 servers in 2010. That is in one year

it was doubled number of servers and in 2012 it is 118000 servers are deployed. So, the scale is basically keeps on changing and it has become 180000 servers are used to run one application that is Facebook is a massive scale.

Similarly, the Microsoft, Microsoft in 2008 were using 150000 machines and that growth rate was 10000 machines per month. And we can see that 80000 different servers were running one application which is called Bing. Similarly, in 2013 Microsoft cosmos application required 110 thousand machines. And those machines that many number of machines were deployed in 4 different regions. So, that is basically called as the massive scale and this massive scale is required to serve these applications, which is termed as the cloud. Similarly, Yahoo in 2009 has 100000 servers and that splits up into the cluster of 4000.

So, that is there are different sites of at most 4000 servers and they were together if we see that they becomes 100000 servers which runs this Yahoo service. So, Yahoo is basically providing the Google or using the cloud service. The next one is Amazon EC 2, we see that in 2009 40000 machines were required to run this particular system or application EC 2. And each machine was basically of 8 core systems. Similarly, eBay required a 50000 machines to run the applications, HP 380000.

So, as far as the Google is concerned it requires lot many number of servers the total numbers are not disclosed by the Google, but it is basically known that it is quite large than any of the above companies which we have. So, it is basically a massive scale. So, this is the first requirement of the cloud problem.

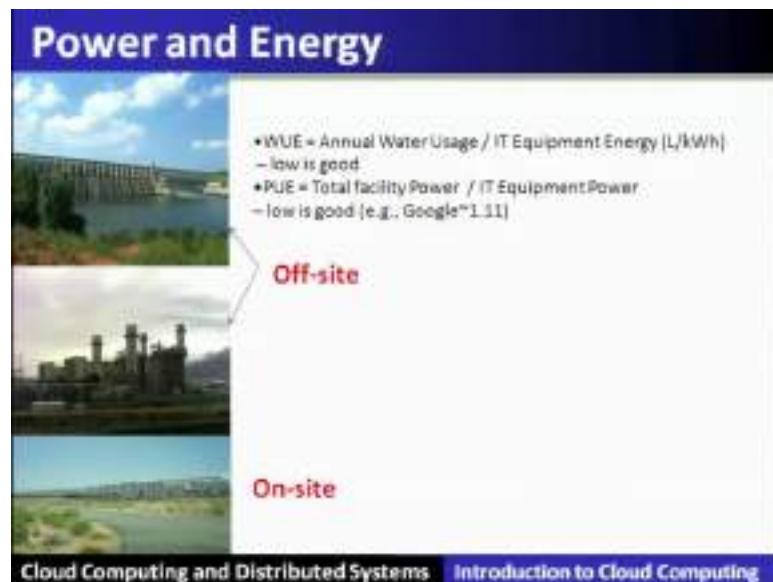
(Refer Slide Time: 29:49)



Now, what is there inside the massive scale that we can see what is there inside the datacenter. So, at one side and at several sides these datacenters will house lot of servers racks and they are all connected together. So, that you can see in this particular room this is called a data centre it has lot of racks full of racks the entire room is filled, and all these racks are connected on the right side you can see this back side of the rack, they are all servers which are interconnected with each other.

And, within inside every server you can see the boards, they are servers are nothing but they are basically in the form of a blades or in the boards, all are fitted within this particular rack and they have been powered there have been communicate they are been connected through the network

(Refer Slide Time: 30:46)



Such a datacenter requires huge power to basically run and, and also this power is being generated through the power stations and also when so much of power is required within one room, so lot of heat is being generated. So, how to cool it so all these are basically the requirement for the maintenance and it requires the cost, how to reduce this particular energy is one of the challenge here in the cloud computing datacenters. So, the water which is used to cool down, and so the annual water usage that is water usage is also measured as annual water usage divided by the IT equipment energy which is being used if this particular parameter is low then it is good.

Similarly, the power utility is also measured by the total facility power divided by IT equipment power; if it is low then it is good. So, Google has shown that his particular data center is achieving 1.11 less than 1 is not possible, but it is very close to 1 so; obviously, it is trying to use as much as it is power drawn for a computing purposes without any much energy wastage.

(Refer Slide Time: 32:11)

Cooling



- Air sucked in
- Combined with purified water
- Moves cool air through system

Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, the cooling there are different types of methods are used to cool this datacenter and some of them are like air sucked in and water is also combined with purified water. And water moves the cool air through the system. There are various methods of cooling which are used in the datacenter of a cloud.

(Refer Slide Time: 32:32)

II. On-demand access: *AAS Classification

- **On-demand:** renting vs. buying one. E.g.:
 - AWS Elastic Compute Cloud (EC2): a few cents to a few \$ per CPU hour
 - AWS Simple Storage Service (S3): a few cents per GB-month
- **HaaS: Hardware as a Service**
 - Get access to barebones hardware machines, do whatever you want with them, Ex: Your own cluster
 - Not always a good idea because of security risks
- **IaaS: Infrastructure as a Service**
 - Get access to flexible computing and storage infrastructure. **Virtualization** is one way of achieving this; subsume HaaS.
 - Ex: Amazon Web Services (AWS: EC2 and S3), OpenStack, Eucalyptus, Rightscale, Microsoft Azure, Google Cloud.

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Now, the second feature of a cloud is on demand access. So, in the industry terms it is being classified as AAS classification so; that means, hardware as a service IaaS means infrastructure as a service, then PaaS means platform as a service, SaaS means software

as a service. So, on demand axis is one of the important features of a cloud problem, on demand means; that you are not buying you are renting or you are paying as you use. So, for example, AWS elastic compute cloud EC 2; that means, you can pay as you use the CPUs for it. How many CPUs are required, how many CPU per hour is required for the application only that is being paid off by the customer.

Similarly, AWS also provides another cloud service that is called storage service S 3, AWS simple storage service. So, it has to be paid as per the use; that means, GB per month. How much space or a storage space per month you use and that amount is to be paid that is called on demand access to the resources like computing storage and therefore, it is been classified. The first one is called hardware as a service; that means, you can get access to the bare bone hardware machines and do whatever you want with them; that means, your own cluster can be owned by someone else as a service.

But it is not a good idea because of the security risks. Therefore, the another one which is called infrastructure as a service is more popular than hardware as a service, and is being provided for the public to be and many companies are use are family giving infrastructure as a service to the customers. So, that access to the flexible computing and storage infrastructure in infrastructure as a service and this is done through the virtualization. So, virtualization has achieved this infrastructural service, and infrastructure as a service has subsumed the hardware as a service. So, hardware as a service is not being used up in the industry, but infrastructure as a service is used and that also covers hardware as a service within it.

For example, Amazon web services AWS, EC2 and S3 is example of a infrastructure as a service, OpenStack is also an example of a infrastructure service. Eucalyptus is also an example RightScale, Microsoft Azure and Google cloud they are example of infrastructure as a service.

(Refer Slide Time: 35:33)

II. On-demand access: *AAS Classification

- **PaaS: Platform as a Service**
 - Get access to flexible computing and storage infrastructure, coupled with a software platform (often tightly coupled)
 - Ex: Google's AppEngine (Python, Java, Go)
- **SaaS: Software as a Service**
 - Get access to software services, when you need them. subsume SOA (Service Oriented Architectures).
 - Ex: Google docs, MS Office on demand

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Now, another kind of AAS classification for on demand axis is called platform as a service. Now, get access to the flexible computing and restorage infrastructure coupled with the programming platform. So, often they are tightly coupled so; that means, the programming paradigm is given and the people the users can use the programming to basically run their applications examples are Google app engine.

This is not as flexible as infrastructure as a service, but it is easy to use platform as a service. Another way of AAS classification is called software as a service. So, get access to the software services when you when you need them. And this will subsume the service oriented architectures. So, given that software as a service is available. So, the service oriented architecture is being subsumed the example of a software as a service are the Google's doc and MS office on demand these are the software as a service.

(Refer Slide Time: 36:44)

III. Data-intensive Computing

- Computation-Intensive Computing
 - Example areas: MPI-based, High-performance computing, Grids
 - Typically run on supercomputers (e.g., NCSA Blue Waters)
- Data-Intensive
 - Typically store data at datacenters
 - Use compute nodes nearby
 - Compute nodes run computation services
- In data-intensive computing, the **focus shifts from computation to the data:**
- CPU utilization no longer the most important resource metric, instead I/O is (disk and/or network)

Cloud Computing and Distributed Systems Introduction to Cloud Computing

The third important feature for a cloud problem or a cloud computing classification is called data intensive computing. This is in contrast to a computation intensive computing earlier there was a computation intensive computing. So, the data was small and which has to be computed very fast.

So, therefore, MPI based high performance computing cluster or grids were formed and also the development of supercomputer was to compute the data very fast. So, that is called computation intensive computing, but the trend is now changing, here the applications have a large data and data cannot be moved, where the compute nodes where the computations are, but the computation has to move where ever the data is required for it is computation, that is why the size of the data is too big. And that is why it is called a data intensive computing.

So, with the typical data intensive computing that is one of the key features of a cloud computing systems requires to store the huge amount of data in first hand at the datacenter. The second is use the compute node nearby; that means, the data cannot move because it is the big size, the compute nodes nearby can compute. So, compute nodes runs the computation service that is called data intensive computing.

So, in data intensive computing the focus shifts from the computation intensive computing to the data intensive computing. So, that is the focus shifted from computation to the data. Here the CPU utilization is no longer the important resource

metric, but instead the IO input output that is the disk and also the network is important why because; in a data intensive computing IO is very important similarly the network is important.

(Refer Slide Time: 38:54)

IV. New Cloud Programming Paradigms

- Easy to write and run highly parallel programs in new cloud programming paradigms:
 - **Google:** MapReduce and Sawzall
 - **Amazon:** Elastic MapReduce service (pay-as-you-go)
 - **Google (MapReduce)**
 - Indexing: a chain of 24 MapReduce jobs
 - ~200K jobs processing 50PB/month (in 2006)
 - **Yahoo!** (Hadoop + Pig)
 - WebMap: a chain of several MapReduce jobs
 - 300 TB of data, 10K cores, many tens of hours (~2008)
 - **Facebook** (Hadoop + Hive)
 - ~300TB total, adding 2TB/day (in 2008)
 - 3K jobs processing 5STB/day
 - **NoSQL:** MySQL is an industry standard, but Cassandra is 2400 times faster

Cloud Computing and Distributed Systems Introduction to Cloud Computing

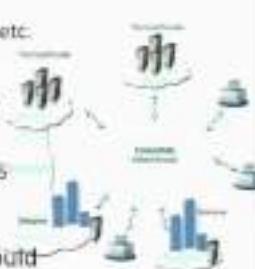
Now, the next important features of the cloud problem is called new cloud programming paradigm. It has to provide the easy to write and run highly parallel programs in the new cloud programming paradigms. For example, the Google provides a new programming paradigm which is called MapReduce and Sawzall. So, Amazon also provides elastic MapReduce service, where you have to pay as you go. Google also provides the MapReduce in the same manner. So, map Google also uses the MapReduce in the form of indexing, for indexing it requires a chain of MapReduce of 24 MapReduce jobs.

So, just see that the MapReduce solves a big problem or the data intensive computing. Similarly, the Yahoo also has used the Hadoop that is a open source version of the MapReduce and it is own version that is called pig. Facebook also uses the hadoop and the hive where in 300 terabytes of total data size is being basically computed or being processed. Another thing is called a new programming paradigm is NoSQL, which is in contrast to the MySQL which is of industry standard. Similarly, the key value store that is called Cassandra is also 2400 times faster than MySQL.

(Refer Slide Time: 40:35)

Two Categories of Clouds

- Can be either a (i) public cloud, or (ii) private cloud
- Private clouds** are accessible only to company employees
- Example of popular vendors for creating private clouds are VMware, Microsoft Azure, Eucalyptus etc.
- Public clouds** provide service to any paying customer
- Examples of large public cloud services include Amazon EC2, Google AppEngine, Gmail, Office365 and Dropbox etc.
- You're starting a new service/company: should you use a public cloud or purchase your own private cloud?



Cloud Computing and Distributed Systems Introduction to Cloud Computing

So, we can see that as far as the cloud is concerned there are two types or two categories of cloud public and private. So, as you know that the private clouds are accessible only within the company and public clouds basically are being provided as a service to the customer.

(Refer Slide Time: 40:54)

Single site Cloud: to Outsource or Own?

- Medium-sized organization: wishes to run a service for M months
 - Service requires 128 servers (1024 cores) and 524 TB
 - Outsource** (e.g., via AWS): monthly cost
 - \$3 costs: \$0.12 per GB month, EC2 costs: \$0.10 per CPU hour (costs from 2009); Storage = \$0.12 X 524 X 1000 ≈ \$62 K
 - Total = Storage + CPUs = \$62 K + \$0.10 X 1024 X 24 X 30 ≈ \$136 K
 - Own**: monthly cost
 - Storage = \$349 K / M; Total = \$ 1555 K / M + 7.5 K (includes 1 sysadmin / 100 nodes)
 - using 0.45:0.4:0.15 split for hardware:power:network and 3 year lifetime of hardware
 - Business makes more profit if:
 - \$349 K / M + 362 K (rental)
 - ≈ 1000 K / M + 25 K = 11.25 K overall
 - Startups use clouds a lot
 - Cloud providers benefit monetarily most from storage

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Now, the question about whether to use the private cloud or to use a public cloud is basically a matter of economics. So, for example, if let us say we have taken example of a medium size organization, which runs it is computing services for let us say M months.

So, the services in requires 128 different servers and 524 terabyte of a space. Now if it is outsources using Amazon AWS services on a monthly basis. So, let us compute the cost. So, for a storage S 3 will cost let us say about \$ 62 K. And CPU will be costing around how much that is comes out to be 1024 times 24 times 30 that comes total comes out to be \$ 136 K for the outsourcing of the entire computing infrastructure.

Now, in contrast to that if you want to own a private cloud and let us understand the cost of that storage. If you purchase \$ 349 K divided by the total number of months you are going to use it. Similarly, you have to add some more cost for it is maintenance in the terms of the man power system administration and so on. If you compute what you will see here is that this particular outsource is if it is equated to this particular purchasing it over a particular period of months you will obtain a breakeven analysis.

And in the breakeven analysis if you see in the slides that if the number of months is more than 6 months, then it is better to own the infrastructure that is go for a private cloud. And if the number of months is more than 12 months for the overall then only you can go for the private cloud. If it is less than 6 months and less than 12 months for the overall, then the breakeven says that you should not go for the private one you have to go for the public one. Therefore, the startup companies which uses such a infrastructure, but for a little period of time maybe less than one year for the experimentation all it is better for them to go for the cloud, that is why clouds are becoming more popular for these companies.

(Refer Slide Time: 43:49)

Conclusion

- Clouds build on many previous generations of distributed systems
- Characteristics of cloud computing problem
 - **Scale, On-demand access, data-intensive, new programming**

Cloud Computing and Distributed Systems Introduction to Cloud Computing

Conclusion, so clouds build on many previous generations of distributed systems. So obviously, you have to see that the development of distributed systems with the virtualization has built the new generation of the cloud computing systems. These cloud computing systems have the features or the characteristics in the cloud problems, they are called massive scale, on demand access, data intensive computation at a programming paradigm.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 02
Virtualization

Virtualization; preface content of this lecture.

(Refer Slide Time: 00:17)

Preface

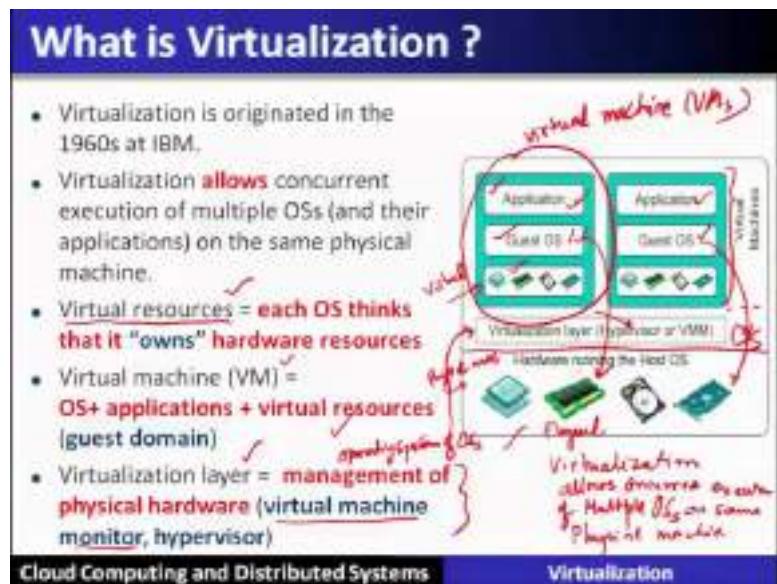
Content of this Lecture:

- In this lecture, we will discuss virtualization technology its importance, benefits, different models and key approaches to virtualization in CPU, Memory and Device Virtualization.

Cloud Computing and Distributed Systems Virtualization

We will discuss virtualization technology, its importance, benefits, different models used for virtualization. And the key approaches to the virtualization with respect to the CPU, memory and device virtualization is a part of this lecture. What is virtualization?

(Refer Slide Time: 00:45)



Virtualization is originated in the year late that is 1960's at IBM. So, it is not a new concept. So, virtualization allows concurrent execution of multiple operating systems, and their applications on the same physical machine.

So, in this example you can see here, that there is one piece of hardware that is CPU memory disk and devices. This particular piece of hardware, if it is being used by more than one application and it is corresponding operating systems, then this particular way of sharing the common hardware across multiple such machines which are the virtual machines. So, this will allow the concurrent execution of multiple operating systems and also their applications simultaneously over a same piece of hardware. So, this is possible using the virtualization.

So, virtualization allows the concurrent execution of multiple operating systems on the same physical machine. So, this is a physical machine and now it has multiple virtual machines; that means, operating system it is application and it is virtual resources together is called a virtual machine. So, what becomes the virtual resource? So, here each operating each operating system thinks that it owns the entire hardware resources. So, these hardware resources, this operating system which is called a guest operating system, thinks that it owns these hardware resources.

Similarly, the other guest operating system, and there may be many more such virtual machines residing and concurrently executing simultaneously. They think that this

hardware they own, and if they think this way then it is called the virtual resources. So, the physical resources are being shared that is why they are called virtual resources.

Now, the concept of a virtual machine is that set of operating system which is called a guest operating system. And the application which runs on that corresponding operating system and the virtual resources together is called the virtual machine. So, these particular several operating systems will now be accessing the same piece of hardware. So, that means, this guest operating system will be either they have own this entire resources or some part of the resources which are available as a part of the physical machine.

So, there is a requirement which is called the virtualization layer or you can also think of an operating system of several operating systems. So, operating system is required which will basically provide the physical resources to be shared across multiple operating systems. And this particular operating system of operating systems is called basically the virtualization layer. So, virtualization layer is also called as hypervisor or virtual machine monitor. So, the virtualization layer or hypervisor or virtual machine monitor is nothing but an operating system of operating systems. So, operating systems of the virtual machine requires to be managed by or requires to be supported by this virtualization layer that is the hypervisor.

Similarly this operating system or which is called a virtualization layer or a hypervisor will use these hardware's or will manage the physical hardware resources and will provide the virtualized view to the several operating systems which are called a guest operating systems. Defining the virtualization, a virtual machine is an efficient isolation of the duplicate of the real machine.

(Refer Slide Time: 06:11)

Defining Virtualization

- A virtual machine is an **efficient, isolated duplicate of the real machine.**
- Supported by a virtual machine monitor (VMM): **if hypervisor**
 - Provides environment **essentially identical with the original machine**
 - Programs show **at worst only minor decrease in speed**.
 - VMM is in **complete control of system resources**.
(This means that the virtual machine monitor has full control to make decisions, who accesses which resources and when)



Cloud Computing and Distributed Systems Virtualization

So, here you can see that this particular virtual machine will duplicate the real physical machine. So, this is to be done in an isolated in an efficient manner. So, virtual machine is an efficient isolated duplicate of the real machine, which is supported by the virtual machine monitor or the hypervisor.

This particular virtual machine monitor or the hypervisor will provide environment that is essentially identical with the original machine. So, all the resources which are available as part of the physical machine will; that means, this guest operating system will have an illusion as if they basically are owning these physical resources. And that particular illusion is being supported by the virtual machine monitor or and hypervisor.

In other sense, this particular illusion is to be done or is to be achieved in a real sense. In the sense it is a copy of all the physical resources will be available as a part of the virtual resources to the guest operating systems. Hence the one of the important goals of virtual machine monitor is fidelity. The second goal of virtual machine monitor is that the programs show at most only the minor decrease in the speed.

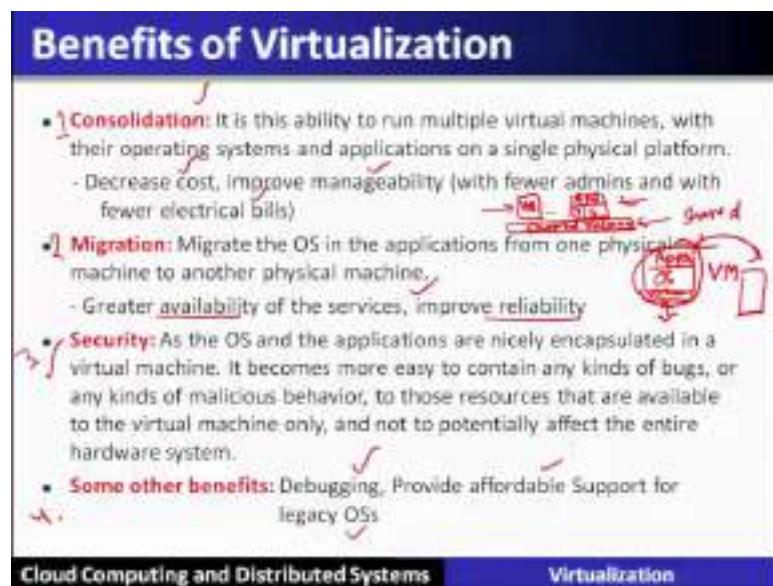
So, for example, these resources are being shared across several guest operating systems. So, not all resources are available to these guest operating systems which are available as far as the physical machine is concerned physical resources, but some part of the physical resources is available.

Hence, this particular way the programs will have only a small decrease in the speed. Because the full resources are not available, but other full resources are shared across the resources. Hence the performance also is one of the requirements that there will be an efficient or a good performance in doing so; that is, in sharing the resources across several operating systems.

Now, the third one which is supported is basically that virtual machine monitor is in the complete control of the physical resources; that means, if it is not done in that manner then, these operating guest operating system may interfere. The use of these physical resources with the other guest operating systems, hence a complete isolation and safety is also the third important requirement of the virtual machine monitor or the hypervisor.

So, basically the virtual machine monitor will take the full control and then makes the decision of a controlled access of these resources to or across this guest operating systems, to ensure the safety and also ensures the isolation across different virtual machines which are running over the same physical machine.

(Refer Slide Time: 10:05)



Benefits of the virtualization: the first benefit is the consolidation. We will understand what you mean by the consolidation. So, consolidation is the ability to run multiple virtual machines with their operating systems and applications on single physical platforms. So, again I am giving you a illustration that this is a VM virtual machine 1 2

and so on several virtual machine which comprises of an OS and application. They will run on shared resources. This particular way is called a consolidation.

In the sense, that this physical resource will be shared across multiple virtual machines or a many virtual machines. This is called a consolidation, because it will decrease the cost why because the same set of hardware resources cannot be fully utilized by one physical set of operating system and applications. If there are several such virtual machines running; obviously, the overall cost will decrease, and also will improve the manageability of all these resources and a fewer other electricity bills and other maintenance cost. Hence the consolidation is one of the key factors to utilize the hardware's or the physical resources. And this is done by the virtualization and this process is called consolidation.

Now second important benefit of virtualization is the migration. So, that means, migrate the operating system in the application from one physical machine to another physical machine. Now given that the application and it is corresponding operating system they are bundled together with the virtual resources, and this is called virtual machine. Now this will be detached from the physical machine this complete package can be migrated or may be copied to some other place whenever there is a scope of improvement or efficiency or availability or improving the reliability.

There are various regions which requires the mobility to migrate the entire virtual machine from one physical machine to another physical machine; thereby, increasing the availability and also will improve the reliability and the performance of the system. So, the maintenance here becomes quite automatic and manageable with the migration in place. So, this is another benefit of the virtualization.

Third benefit is called security. So, as the operating system and applications are nicely encapsulated in a virtual machine that we have seen in the migration also. Therefore, it becomes an easy to contain any kind of bugs or any kind of malicious behavior to those resources that are available to the virtual machine only, and not potentially affect the entire hardware system.

Therefore, it ensures the security if multiple virtual machines are basically accessing the same or sharing the same set of hardware in this particular scenario. So, whatever

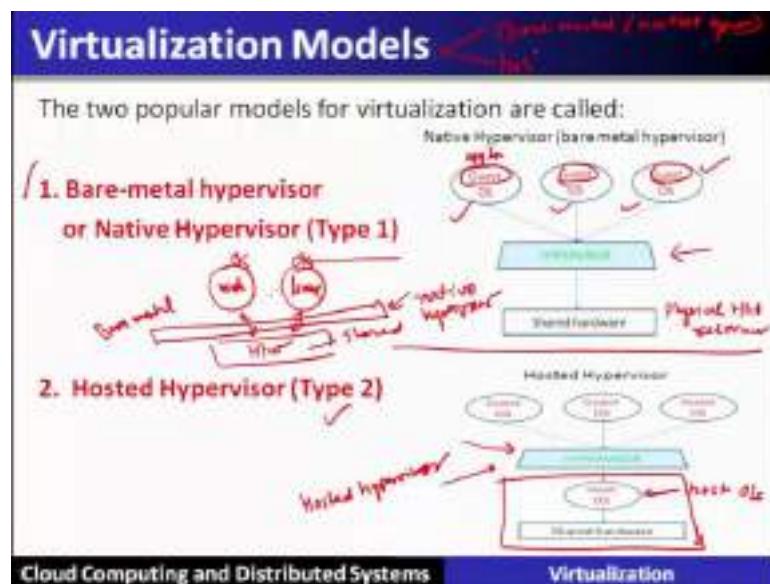
problem is happening at one virtual machine will not suffer to the other virtual machine that is called security and isolation.

Now both are some other benefits for example, whenever a debugging is required to be done into a one virtual machine. It will not affect the other virtual machines. Hence this particular debugging or malfunctioning at one virtual machine will be isolated or will not disturb the other virtual machines. And the process and the efficiency of the machine or usage will go on.

Similarly, it will also provide the affordable support for legacy operating systems. What do you mean by this is that, the legacy operating systems to support in a new hardware a new hardware some more provisions are being provided by the architects of this particular machines.

Now, there is no longer required of this kind of things. Why because in the virtualization whatever is the legacy operating system and applications they are detached from the physical machine. Therefore, it will provide an affordable support for the legacy operating system in this manner this is another benefit of virtualization.

(Refer Slide Time: 15:23)



Now, let us see different models which are basically most prominent for virtualization. There may be there are various other models available, but we are only going to discuss most important or a prominent model for virtualization. The first one is called the bare

metal hypervisor, or it is also called the native hypervisor and it is called as a type one hypervisor.

See in this particular example, over the hardware in the hypervisor sits just above the hardware or a shared physical machine or a physical hardware resources to support various different operating system and their corresponding applications. Then it is called the bare metal hypervisor.

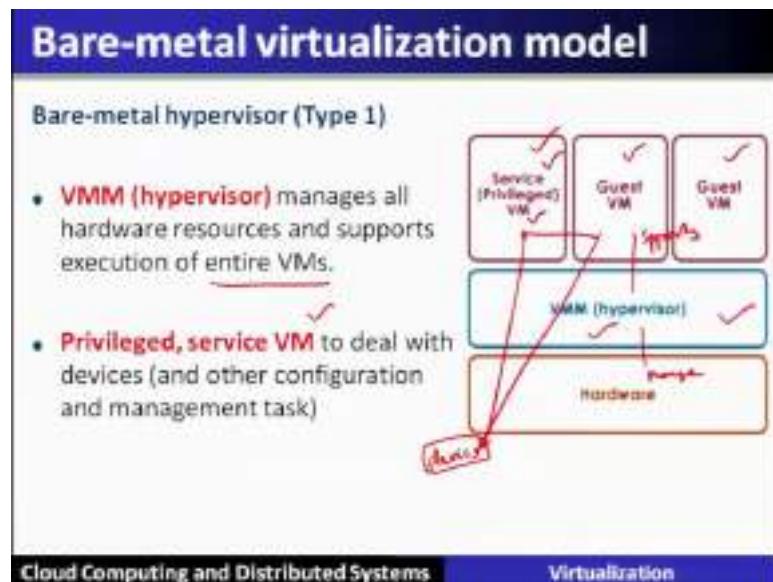
So, in another way that you can think of for example, there is a Microsoft windows, this is Linux. There are different operating systems now running over the same piece of hardware, which was earlier not possible, either the windows was running or the Linux was running in this particular moved now all different operating systems OS can run simultaneously over a same piece of hardware; that means, the hardware is shared across different operating system running. That is being possible with the help of the virtualization layer, which is called bare metal hypervisor or it is in need the hypervisor.

Why because, it will basically sit just above the physical hardware and that is why it is called in native hypervisor. The different operating systems which will be using the hardware using with the support of bare metal hypervisor they are called the guest operating systems.

Now, second type of virtualization model is called a hosted hypervisor or a type 2 hypervisor in hosted hypervisor shared hardware continues to use it is own operating system, and that is called a host operating system. Now to support several other operating systems by this particular operating system to share the hardware it requires another layer which is called virtualization layer which is called a hosted hypervisor.

So now, the physical machine comprises it is hardware and the operating system which will be shared across multiple operating system with the support of the hypervisor. So, hypervisor sits just above the host operating system and gives and supports the guest operating system. If that is the model, then it is called type 2 hypervisor. It is also called as hosted hypervisor. So, these 2 models we have seen; that is, the bare metal or a native hypervisor, the other one is called a hosted hypervisor.

(Refer Slide Time: 19:25)

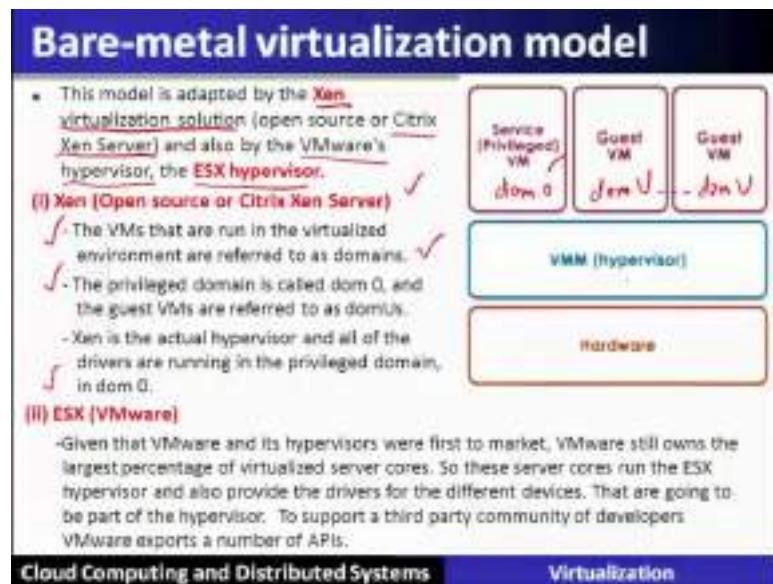


Let us see some details inside this particular virtualization model which is called as a bare metal virtualization model. So, in a bare metal hypervisor, this particular hypervisor that is virtual machine monitor sits just above the hardware. So, this is called a bare metal hardware why because, this hypervisor directly sits over the hardware and manages the hardware resources. This will also support for a several virtual machines with the help of a privileged virtual machine.

So, virtual machine monitor the hypervisor manages the hardware resources, this will manage hardware resources. And also supports the execution of several or entire virtual machines. Now there is a concept of privileged service virtual machine to deal with the devices and other configuration and the management task.

For example, the devices to share the devices across the guest virtual machines; so, virtual machines would like to access the device is directly. This is done through the service or the privileged virtual machine. This will support the axes of devices to the other guest virtual machines. Hence, the virtual machine monitor that is the bare metal hypervisor with the help of service privileged virtual machine continues to support the use of the configurations and management task related to the devices.

(Refer Slide Time: 21:24)



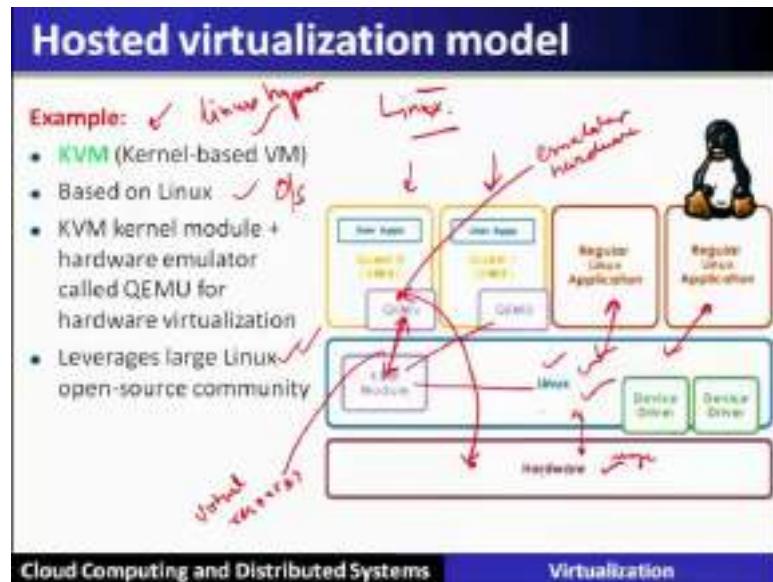
Now we can see that the bare metal virtualization model was adopted here in Xen virtualization solution, which is an open source or it is also available with Citrix Xen Server. This bare metal virtualization model is also used by the VM wares hypervisor which is called ESX hypervisor.

So, let us see these two industry standard or these two hypervisors which are used in the industry which basically are adopting the bare metal virtualization model. The first one is we are going to discuss is Xen.

So, the virtual machines that are run in virtualized environment, they are termed as the domains. The privileged domain is called domain 0, and the guest VMs are referred to as domain u. This is the terminologies which are used in Xen hypervisor. So, then is the actual hypervisor now all the drivers are running in the privileged domain. That is in the domain 0 that we have earlier explained in the previous slide of bare metal virtualization model.

Now another way of using the bare metal virtualization is seen here in VMware ESX hypervisor. Now given that VMware and it is hypervisors were first to market, we are very still owns the largest percentage of virtualized server course. So, the server course run ESX hypervisor run also provide the drivers to the different devices. They are going to be part of hypervisor to support the third party community of the developers. So, VMware exports a number of APIs.

(Refer Slide Time: 23:39)



Now, let us see hosted hypervisor is rated or type 2 hypervisor. So, in this model at the lowest level there is a full-fledged operator host operating system that manages all the hardware resources. So, this operating system which is called a hosted operating systems sits just above the hardware, and its task is to manage the hardware resources.

Now, the host operating systems will integrate a virtual machine monitor. That is responsible for providing the virtual machine with their virtual platform interface. And for managing all of the context switching schedule. So, let us see the example over here.

Now, in the hosted hypervisor examples, hypervisor example the native applications; native applications; that means, who are using the same operating system can directly be supported by the hosted hypervisor using the host operating system in the hosted hypervisor. Now in the hosted hypervisor, we will use the host operating system or the host operating system will integrate virtual machine monitor to support the different virtual machines.

So, the different virtual machine will have their own guest operating systems, different guest operating systems are being supported by the virtual machine monitor which in turn will basically use the host operating system. And this host operating system is having the responsibility of managing the entire hardware. So, the hardware is shared through the host operating system via this hosted hypervisor across all the VMs or across all the guest operating systems. So, this particular way is supported and if it is then it is called the hosted hypervisor.

The good part of hosted hypervisor is that this host operating system. It is internal functioning will be leveraged to basically support this virtual machine monitor or the hosted hypervisor; so that it will continue to support several virtual machines. And at the same point of time it will also continue to use the legacy applications which will try to use the physical machine and several applications will be sharing the hosted operating system.

Similarly, the host operating system will have the support for the devices which are again being available being shared across all the guest operating system and the native operating system through hosted hypervisor. So, this model is called hosted virtualization model which continues to use its host operating system. And so, that it will also support the virtual machine monitor. And this is called hosted virtualization or hosted hypervisor, which intern supports several virtual machines.

The example of a hosted virtualization model is seen in a kernel based, in a Linux based hypervisor, which is called KVM that is called kernel based virtual machine. It is based on the Linux operating system. So, Linux operating system you know that it sits over the hardware, and continues to support its applications. And we will manage the hardware resources, also the device drivers.

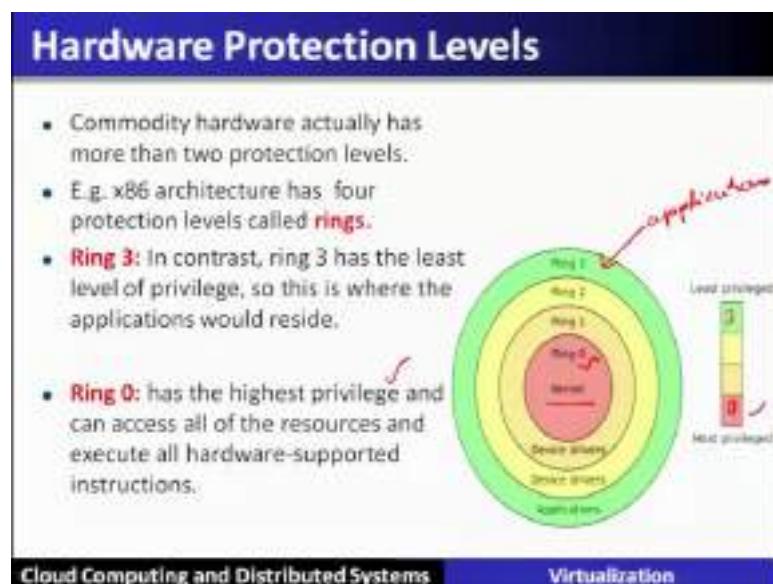
Now, whenever there is a virtual machines on the Linux platform. Then this Linux supports or integrates the KVM module that is the kernel virtual machine the kernel based VM. So, this particular module which is being supported by the Linux operating system intern which will support with the help of QEMU. Similarly, it is supported by QEMU. So, this QEMU is emulator, this is the hardware emulator.

So, QEMU is a hardware emulator; that means, this hardware which is available this will be reflected or will be made available to the guest operating system or to the virtual machines. And this particular QEMU with the help of KVM module will provide the virtual resources.

So, QEMU is other hardware emulator which will give a complete picture of the hardware resources which are available and this particular KVM module with the QEMU are going to support several VMs. This is also going to be important, why because this way that is hosted virtualization using the Linux operating system is going to be very useful why because Linux is an open source community.

So, whatever it is development taking place in the Linux will continue to be basically used up, here in this way of supporting this virtual machines in the Linux operating system.

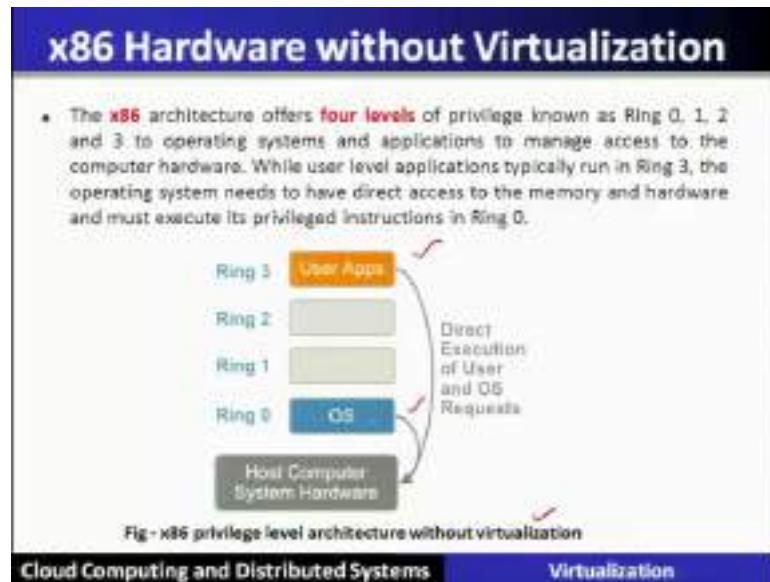
(Refer Slide Time: 29:56)



Now, let us see the hardware, production levels which are being used here in the hypervisors or in the virtual machine monitors. Now you know there are different commodity hardware's and these commodity hardware's has more than 2 production levels. For example, x86 architecture has 4 different production levels which are called as a ring, where as you see that kernel resides in the ring 0.

So, that means, kernel 0 has the highest privilege. So, level 0 has the highest privilege whereas the level 3 has the least privilege. So obviously, the least privilege; that means, the applications will basically be dealt at the level 3 or a ring 3 privilege privileges whereas, the highest privileges are where the kernel or a code of the operating systems.

(Refer Slide Time: 31:03)



Now we can see here the diagram or the illustration of the x86 privilege level architecture. Without virtualization, here you can see that the user application are at ring 3 privilege level, where as the highest privilege level that is ring 0 will be assigned to the operating system kernel.

So, operating system is basically directly managing the hardware resources and supports all the applications. So, any user if non privileged instructions directly can be accessed through the hardware, but the privileged instructions are done through the operating systems, here in the normal scenario without any virtualization.

(Refer Slide Time: 31:52)

Processor Virtualization (Trap-and-Emulate)

- Guest instructions
 - executed **directly by hardware**
 - for non-privileged operations:
hardware speeds=> efficiency
 - for privileged operations: trap to hypervisor
- Hypervisor determines what needs to be done:
 - **If illegal operation:** terminate VM ✓
 - **If legal operation:** emulate the behavior the guest OS was expecting from the hardware ✓

Cloud Computing and Distributed Systems Virtualization

Now as far as the processor also supports the virtualization in the form of a trap and emulate instructions or constructs. So, the guest instruction that is the guest operating systems executed directly by the hardware; now for the non-privileged operations to ensure the efficiency and it can run at the hardware speeds. Now for the privileged operations, this will give a trap to the hypervisor; that means, the guest operating system will generate a trap to the hypervisor, and this is like a system call and hypervisor inter will call on behalf of the guest operating system to execute the privileged instructions.

So, hypervisor will determine what is to be done. If whenever there is a trap the hypervisor will ensure whether it is illegal operation, then it will terminate the virtual machine. If it is a legal operation, then it will emulate the behavior of the guest operating system which is expecting to be executed at the hardware speed from the hardware. So, without any much loss of efficiency with the help of with the support of hypervisor all the privilege instructions are also executed at the guest operating system level.

(Refer Slide Time: 33:16)

Problems with Trap-and-Emulate

- X86 Pre 2005
- ~4 rings, no root/non-root modes yet
- -hypervisor in ring 0, guest OS in ring 1
- BUT: 17 privileged instructions do not trap! Fail silently!
 - E.g. interrupt enable/disable bit in privileged register; POPF/PUSHF instructions that access it from ring 1 fail silently
- Hypervisor doesn't know, so it doesn't try to change settings
- OS doesn't know, so assumes change was successful!

Cloud Computing and Distributed Systems Virtualization

Let us see the scenario to support this particular ring is basically to support this trap at emulate instructions. So, all this particular ring 0 1 2; they are basically categorized as non-root mode privilege levels. So, user can directly execute the hardware in the normal situation; so, that the application can get the hardware speed or efficiency. Whenever the guest of whenever there is a privileged instructions generated by the guest operating system, this will trap to the virtual machine monitor which works at the root level or a root mode privileged level.

Now this particular trap intern will generate a hypervisor called to the hardware and this will be executed. So, here we have seen that there are some 17 different privileged instructions with this particular guest operating system requires to be executed with the help of the hypervisors. So, there are different mechanisms to support these 17 different privileged instructions different models of the hypervisor that we will see in the next slide.

(Refer Slide Time: 34:47)

Binary Translation

- **Main idea:** rewrite the VM binary to never issue those 17 instructions
- Pioneered by Mendel Rosenblum's group at Stanford, commercialized as VMware
- Rosenblum awarded ACM Fellow for "reinventing virtualization"

Cloud Computing and Distributed Systems Virtualization

So, this particular concept of binary translation is to rewrite this virtual machine binary to never issue those 17 different trap or instructions. This particular idea of binary translation was pioneered by Mendel Rosenblum of his Stanford which was commercialized in VMware software. And for this work which is which is renamed as reinventing virtualization has conferred an award to Rosenblum as ACM fellow.

(Refer Slide Time: 35:28)

Full-Virtualization: Binary Translation

Binary translation:

- **Goal:** full virtualization=guest OS not modified
- **Approach:** dynamic binary translation

1. Inspect code blocks to be executed
2. If needed, translate to alternate instruction sequence
 - e.g., to emulate desired behavior, possibly even avoiding trap
3. Otherwise, run at hardware speeds
 - Cache translated blocks to amortize translation costs

Fig.: Binary translation approach to x86 virtualization

Cloud Computing and Distributed Systems Virtualization

Now, let us see the binary translation how it works which has revolutionized the VMware software and which is having a good a market in case of in this virtualization or

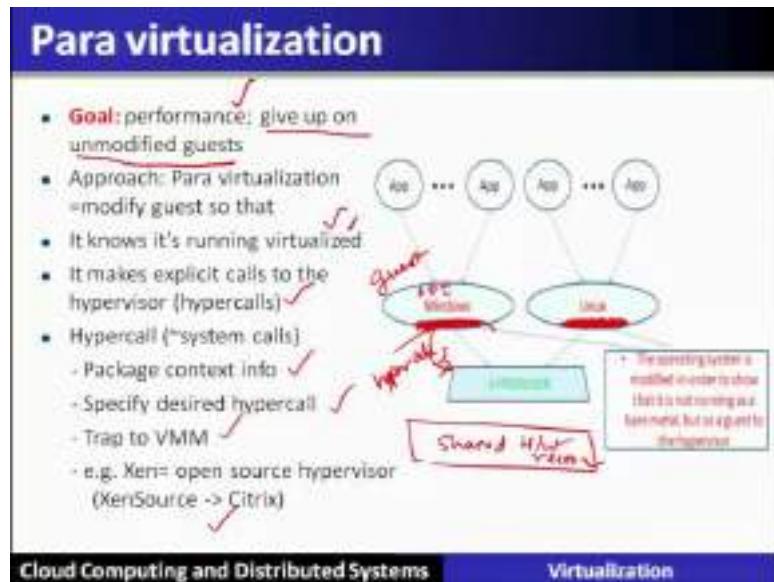
in a cloud computing scenario. So, binary translation here in the goal is that the guest operating system is not modified. If it is, then it is called a full virtualization, and this full virtualization is done through the binary translation.

So, let us see the approach that when such privileged instructions are generated it cannot be known beforehand. It will be done or it will be detected at dynamically during the runtime. So, dynamically at the dynamic time these binary translation has to be done; that means, those privileged instructions has to be identified by the hypervisor in full virtualization and it has to execute it or it has to process it. If it is not able to execute then it will feel silently. Now for that this binary translation will do, let us see the steps or the approach of battery translation for such privilege instructions in case of full virtualization where the guest operating system is not modified.

So, first it will inspect the blocks to be executed. If needed, then it will translate to an alternate instruction sequence. For example, to emulate the desired behavior possibly even avoiding the trap. So, it has to know what instruction what privilege instruction will come in the future just by inspecting the block code. Now otherwise it will run at the hardware speed.

So, the cash translated blocks are there; that means, instead of trapping one instruction, all the instruction all such instructions are inspected in the blocks and they will be given at one go that is the batches are being given to run at the hardware speed. That is all done in the binary translation to make the efficient use of it and also. The cache will translate the blocks to amortized the translation cost for several instructions if they are batch. So, isn't so that particular translation cost will be reduced in this manner.

(Refer Slide Time: 38:12)



So, in contrast to the full virtualization, there is another method of virtualization which is called para virtualization. Let us see the goal of Para virtualization. Goal is basically the performance; it will give up on unmodified guest. So, the guest is no longer modified. So, guest operating systems are to be modified to basically get the performance of this particular privileged instruction translation and execution at a higher efficiency.

So, the approach of a Para virtualization is to modify the guest operating system for example, this is the guest operating system. Only a little portion of the guest operating system is modified, but as far as the APIs are concerned which are supporting different application they are not modified, then some internal part is modified which is interfacing with the hypervisor. So, that means, only the privileged instructions which are required to be directly access to the hardware is being modified.

Now in this particular approach which is called a Para virtualization, where the guest operating system is a little modified so that all the applications which are running in this particular model knows that it is running the virtualized in a virtualized scenario. So, this means that it makes an explicit call to the hypervisor, and this is called a hypervisor call.

So, it will use this modified it will make a hyper call. So, hyper calls are like system calls, and they are having the packaged context information is specify the desired hyper calls, and then it will trap to the virtual machine monitor that is the hypervisor. Example of this particular Para virtualization is done in the Xen supported by the Citrix.

(Refer Slide Time: 40:29)

Para virtualization: Review !

- What percentage of Guest OS code may need modification with para Virtualization?
 - 10 %
 - 50%
 - 30%
 - 2%
- **Answer:** less than 2%.
 - This can be shown by a proof-of-construction by XEN
 - Xen is a para virtualized hypervisor.

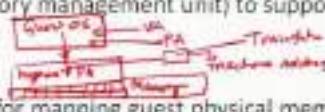
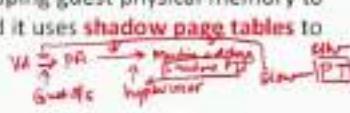
Cloud Computing and Distributed Systems

Virtualization

Now, let us see how much in para virtualization. What is the percentage of guest operating system code that needs to be modified in the para virtualization. It is more than it is 50 %, means some of the code is of the guest operating system is to modified or it is less than 50 % that is 30 % code is to modified, or even it is lesser than 10 %, or it is less than 2 %. So, only less than 2 % code is modified in para virtualization to it. So, it is doable, it is and this particular proof is given in Xen para virtualized hypervisor.

(Refer Slide Time: 41:12)

Memory Virtualization

- To run multiple virtual machines on a single system, one has to virtualize the **MMU**(memory management unit) to support the guest OS.
- The **VMM** is responsible for mapping guest physical memory to the actual machine memory, and it uses **shadow page tables** to accelerate the mappings.
- The VMM uses **TLB** (translation lookaside buffer) hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access.

Cloud Computing and Distributed Systems

Virtualization

Memory virtualization to run multiple virtual machines on a single machine one has to virtualize, the memory management unit to support the guest operating systems. The virtual machine monitor is responsible for mapping the guest physical memory to the actual machine memory. And it uses the shadow page tables to accelerate the mapping. Virtual machine monitor uses translation look aside buffer hardware to map the virtual memory directly to the physical to the machine memory to avoid the 2 level of translation on every access.

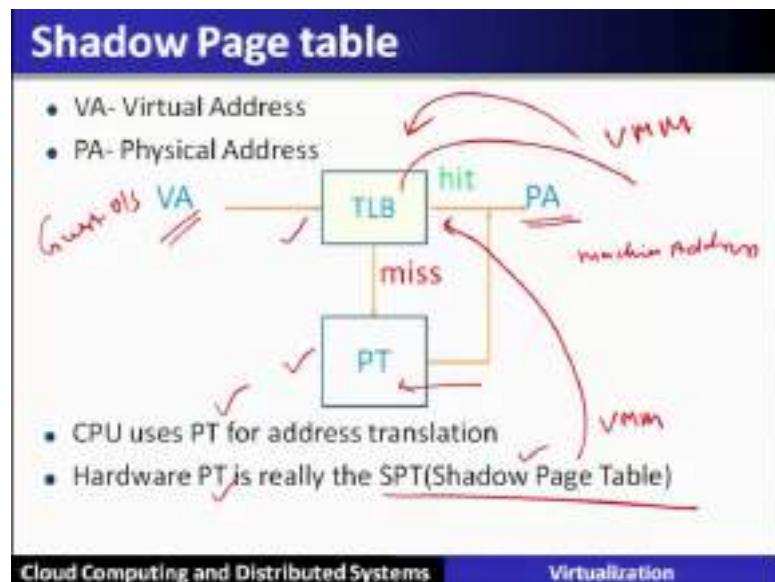
So, let us see through examples that whenever the guest operating system, q a memory address and this particular memory address will be taken by the hypervisor. So, the guest operating system will have the virtual address, and he has it is view of the physical address. This physical address when it is given to the hypervisor, because hypervisor is actually managing the hardware or the memory let us say.

So, this particular physical address is to be translated into the memory into the machine address. So, this particular there are 2 level so, virtual address is to be translated into the physical address this is done by the guest operating system. This particular physical address will be translated to the machine address, and this is to be done with by the hypervisor. So, just see that there are 2 in direction. Now here will be added over, here hence this memory operation becomes slow.

Now, let us see how this efficiency aspect is being covered here in the memory virtualization. So, the virtual machine monitor will use a concept which is called the shadow page table. So now, there are 2 page tables; one is called shadow page table which will be maintained by the hypervisor and there will be a physical page table which will be maintained by the hardware. If both are same then basically the efficiency will be achieved at the hypervisor level itself. So, virtual machine monitor uses the translation look aside buffer hardware to map the virtual memory directly to the machine memory to avoid 2 levels of translation on every access.

So, using TLB the virtual address is directly translated into the to the machine address with the help of TLB by the hypervisor. Now it may be possible that translation look aside buffer which is a cache will not have the information about this particular virtual address then it will have a miss. Then it will go into a physical or a page table, and then it will be resolved by the page table and in that process it will be updated the translation.

(Refer Slide Time: 45:46)

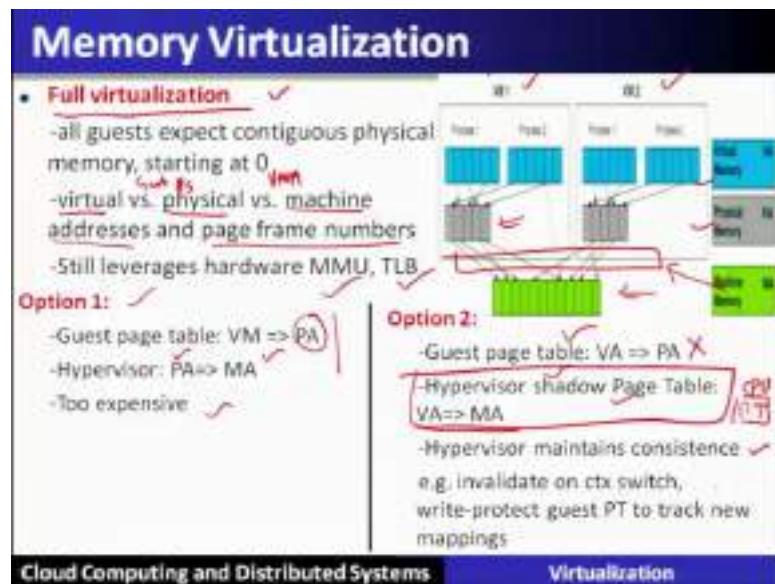


Let us see this particular method of using the shadow page table. So, this particular shadow page table will be maintained by this one, will be maintained by the virtual machine monitor. So, here whenever there is a virtual address given by the guest operating system; so, this particular hardware is a part of virtual machine monitor.

So, it will directly give the machine address. So, it will have a look into the translation look aside buffer that is a cache for that corresponding virtual address. And if it is there then it will directly generate the machine address. So, this particular TLB or the; so, basically otherwise it will access to the page table.

CPU uses the page table for the address translation and the hardware page table which is there with the CPU is really the shadow page table which will be maintained by the virtual machine monitor. If both are same, then basically there will be a direct hit. Otherwise, if there is a miss then from the physical translation that information is basically stored in the shadow page table so that the future access to back virtual addresses can be fasten up. So, in this way the memory virtualizations achieves and efficiency as far as.

(Refer Slide Time: 47:33)



Now let us see the memory virtualization in the case of a full virtualization. Full virtualization means that the guest operating system is not modified; that means, the virtual addresses are directly given virtual addresses which are converted by the guest operating systems physical address and that is given to the hypervisor. So, there are 3 different levels as I told you the virtual address, physical address this will be translated in the guest operating system versus this physical address versus machine address. This will be translated by the hypervisor.

So, basically in full virtualization the hypervisor has to leverage the hardware knowledge of machine or a memory management unit and translation look aside buffer as I told you earlier. So, there are 2 options option one says that the guest operating system will have it is festival which will translate into a physical address. But this physical address is different than the machine address.

So, but that information is having at the hypervisor level in the full virtualization. So, this physical address will be translated into the machine address, but as we have seen that this particular 2 level translation is too expensive and the memory accesses become slower.

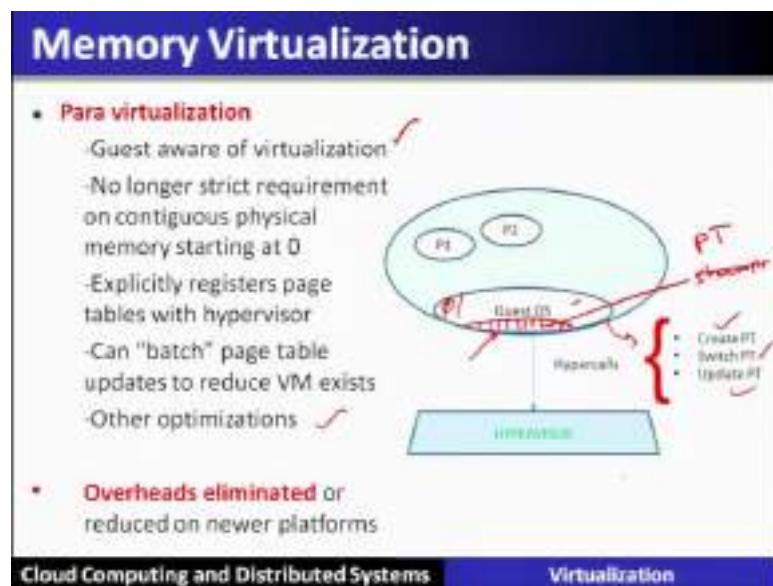
So, this particular hardware example, we can see over here this is the virtual machine one virtual machine 2. They have their virtual addresses which are generated by the process. And they can translate into the physical memory virtual memory physical memory.

When this physical memory or physical addresses are given, then basically the hypervisor comes in between and it will translate into the machine address. The hardware which in the previous slide we have seen that it will used to accelerate this.

The other option is that the guest page table that is when the guest is doing from virtual address to the physical address translation. That is done through the guest page table is it really required. So, hypervisor will now use its shadow page table which will directly translate the virtual address into the machine address. Hence, this particular guest page table can be avoided this particular translation can be avoided directly with this particular hypervisor.

So, hyper for this the hypervisor maintains a consistency, consistency between the shadow page table and the page table which is there with the CPU at the hardware. So, both are we have seen the previous example, how they are going to be updated so that the consistency between shadow page table and the page table is maintained; so that the virtual address can directly be translated into the machine address in most of the cases.

(Refer Slide Time: 50:52)



Now, in case of Para virtualization, let us see the memory virtualization; how it will takes place? Now in Para virtualization the guest is aware of the that it is running in the virtualized scenario; that means, the guest operating system will be modified a little bit that these in tables which hypervisor was earlier doing in full virtualization will be available at the guest operating system. So, guest operating system will have something

called shadow page table. So, directly it can be translated with the help of hypervisor calls.

So now the thing is here explicitly it will register the page table with the hypervisor. So, the page table will be registered with the hypervisor. Now the hypervisor can batch the page table updates to reduce virtual memory exist. And this will also have the other optimizations by way of batching it together.

Now as far as to maintain the page table at the guest operating system level, it will issue the hyper calls to create the page table and to switch the page table and to update the page table. So, all these hyper calls will continue to maintain the consistent page table at all the guest operating system levels. Now here this particular overhead will be reduced and this will make a more efficient way to access the memory using para virtualization.

(Refer Slide Time: 52:36)

Device Virtualization

- **For CPUs and memory**
- Less diversity, ISA (instruction set architectures) level standardization of interface
- **For devices**
- High diversity
- Lack of standard specification of device interface and behavior
- Three Key models for device virtualization:
 - (i) Passthrough Model
 - (ii) Hypervisor Direct Model
 - (iii) Split Device Driver Model

Cloud Computing and Distributed Systems Virtualization

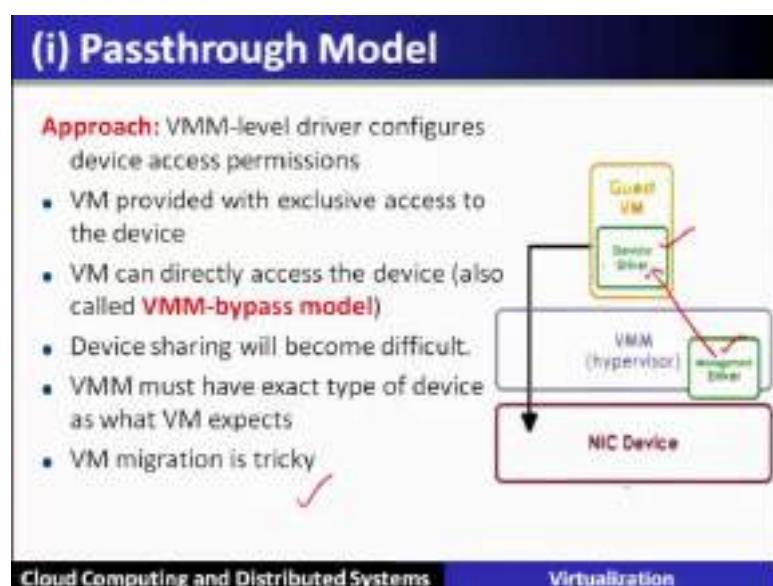
Now, we are going to see the device virtualization. After looking up the CPU and memory virtualization, which is mostly standardized as far as the interfaces are concerned and instruction set architecture given the instruction set architecture these are all standardized and they are less diverse also. So, CPU and memory virtualization has proper algorithm and proper shape and different models basically supports that.

Now, for the devices are having high diversity, and also all the devices lacks in the standard specification interfaces and the behavior. So, there is no common standard

which basically will follow for all the devices. So, different devices different types of devices will have its own standards. And also there are different diversity as I told you different type of devices we will have different behaviors. So, as far as device virtualization is concerned is not so state forward; like, we have seen using the techniques for the CPU and the memory.

So, we are going to see the device the 3 different key models for the virtualization of devices. They are the pass through model hypervisor direct model split device driver model. Let us see one by one all 3 different type of models used in the device virtualization.

(Refer Slide Time: 54:11)

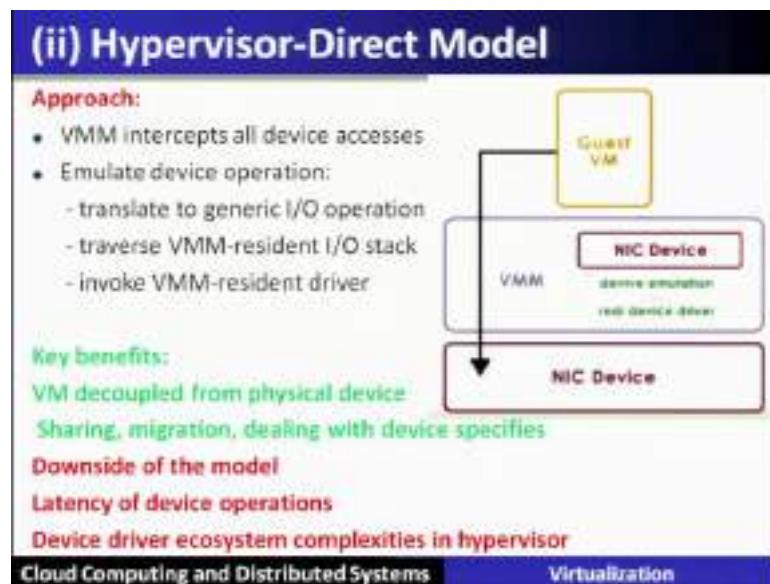


Now, first model is called pass through model. Here the approaches that virtual machine monitor level configures the devices and device access permissions to be given to the guest virtual machines to access the devices. So, virtual machines are provided with the exclusive access to the devices with the support of virtual machine monitor, which basically will have its own management routine which will basically supports this particular direct access which will bypass the virtual machine hypervisor. So that is the guest operating systems, the guest virtual machines can directly access the devices and in an efficient manner.

The device sharing is very difficult in this particular scenario. So, virtual machine monitor must have exact type of device as what the virtual machine expects. Now here

the virtual machine migration is tricky because, the drivers of guest virtual machines or the guest operating system directly are accessing or tied up with the devices. So, the migration is more difficult or a tricky. So, in the next model we will see how this problem is going to be solved.

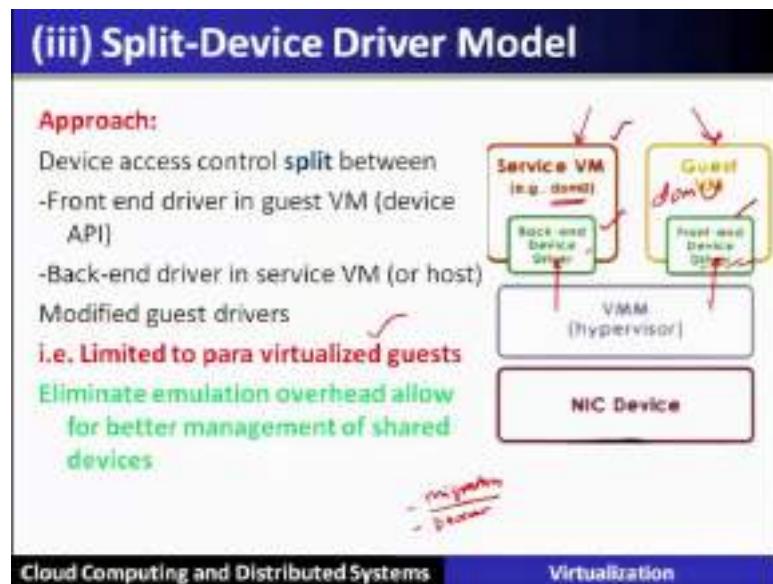
(Refer Slide Time: 55:55)



Hypervisor direct model approach virtual machine monitor intercepts all device accesses. Emulate the device operations in the sense it will translate to the generic I/O operations, and traverse the virtual machine monitor resident I/O stack. Invoke virtual machine monitor resident driver, and it will allow the access; that means, the devices are sometimes or it is virtualized. So, the guest operating system is supporting or using supported by the proper way of generic device operations through the virtual machine monitor.

The key benefits here are that virtual machine is decoupled from the physical device, unlike in the previous method which we have seen. And this will support the sharing migration are dealing with the device is specifies now downside of the model is that the devices are accessed through the virtual machine monitor. Hence it will give it will add to the latency for the devices for the device operations in the virtual machine accesses to the device. So, device driver ecosystem is the adding the complexity in the hypervisor level.

(Refer Slide Time: 57:30)



Now, the next method is called is split device driver method. Here the device access control is a split between the front end driver in the guest VM and back end driver in the service VM or the host let us see through this particular example. This device access control is split into 2 parts. The first one is called front end device driver in the guest VM that is through the device API. The other one is called back end device driver in the service VM. So, service VM as you know that it is called domain 0 and the guest VM or the guest operating system or the VMs is called domain U.

Now, here this particular split device driver model is limited to the para virtualized guest operating system, why because this is going to be para virtualized. Now here it will eliminate the emulation overhead and it will allow for better management of the shared devices, because hypervisor will support the guest operating system through the front end device drivers. And also the service VM that is the privileged VM which will provide the back end device driver together they will basically allow to access the device driver directly.

So, with this is split device driver model, all the important aspects that is the migration is possible here in this case, and also it will support using para virtualized method. The device accesses or being given to the virtual machines, so, that; they may access it efficiently.

(Refer Slide Time: 59:38)

Conclusion

- In this lecture, we have **defined virtualization** and discussed the main virtualization approaches.
- We have also described **processor virtualization, memory virtualization and device virtualization** used in virtualization solutions, such as Xen, KVM and the VMware .

Cloud Computing and Distributed Systems Virtualization

So, conclusion in this lecture we have defined virtualization and discuss the main virtualized virtualization approaches. We have also described the processor virtualization, memory virtualization and device virtualization, used for virtualized solutions. And also we have shown some of the solutions such as Xen KVM and VMware virtualization methods or hypervisors.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 03
Hotspot Mitigation for Virtual Machine Migration

Hotspot Mitigation for Virtual Machine Migration; preface content of this lecture.

(Refer Slide Time: 00:24)

The slide has a blue header bar with the word "Preface" in white. Below it is a white content area with a red title "Content of this Lecture:" followed by two bullet points. At the bottom, there is a footer bar with two tabs: "Cloud Computing and Distributed Systems" and "Hotspot Mitigation for VM Migration".

Preface

Content of this Lecture:

- In this lecture, we will discuss **hot spot mitigation techniques** to automate the task of monitoring and detecting hotspots by determining a new mapping of physical to virtual resources and initiating the necessary migrations.
- We will discuss a **black-box approach** that is fully OS- and application-agnostic and a **gray-box approach** that exploits OS- and application-level statistics.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

We will discuss hotspot mitigation techniques and algorithms to automate the task of monitoring and detecting hotspots in the cloud system by determining new mapping of physical to the virtual resources and initiating the virtual machine migration. This all we will discuss in the topic hotspot mitigation techniques for virtualized data centers; here we will cover 2 different schemes one is called black box approach that is fully operating system and application agnostic. The other approach is called as a grey box approach that exploits operating system and application level statistics. We will also see the paper which has given these 2 approaches to solve the hotspot mitigation problem using virtual machine migration for virtualized data center.

(Refer Slide Time: 01:38)

Enterprise Data Centers

- **Data Centers are composed of:**
 - Large clusters of servers
 - Network attached storage devices
- **Multiple applications per server**
 - Shared hosting environment
 - Multi-tier, may span multiple servers
- **Allocates resources to meet Service Level Agreements (SLAs)**
- **Virtualization increasingly common**



Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Enterprise data centers are composed of large number of clusters of servers. So in a typical example you can see here 2 pictures, which shows the racks full of the servers and it comprises a large cluster and this all is packed in a room and this is called data center. This data center has along with the servers the storage devices are also there along with them and sufficient network devices are there to connect with the outside world these particular data centers are used to run many applications and many applications are running on a particular server or a particular application is running on a spanning multiple servers also.

(Refer Slide Time: 02:48)

Benefits of Virtualization

- Run multiple applications on one server
 - Each application runs in its own virtual machine
- Maintains isolation
 - Provides security
- Rapidly adjust resource allocations
 - CPU priority, memory allocation
- VM migration
 - "Transparent" to application
 - No downtime, but incurs overhead

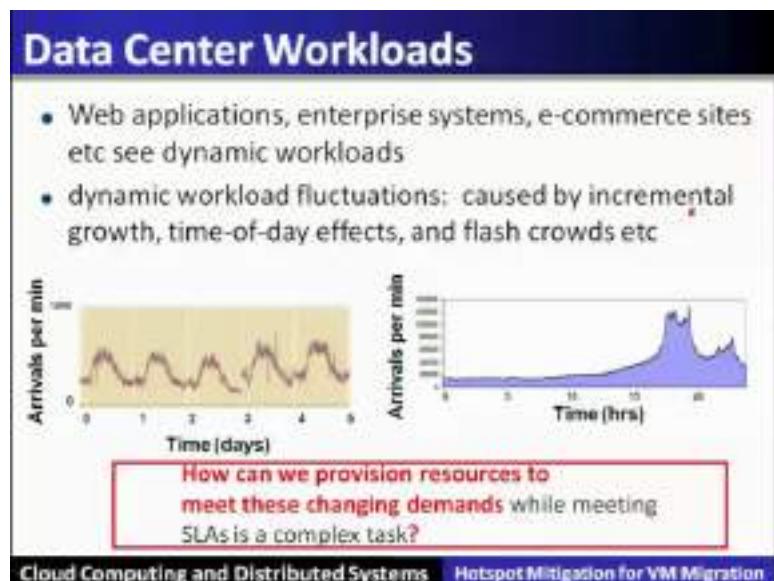
How can we use virtualization to more efficiently utilize data center resources?

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, this will provide a shared hosting environment for the applications to run in this environment, this becomes a multi tier and may span over multiple servers those applications. Now another important thing is the allocation of resources to these applications has to meet to the service level agreements, which is a contract between the customer and application provider and that application in turn will require these resources from the data center to be allocated to the application.

Hence the allocation of the resources has to meet the service level agreements and in this environment of sharing the data center of resources across many applications is done only with the help of the technique which is called virtualization. Let us see the benefits of virtualization, it will enable to run multiple applications on one server; that means, the server will be multiplexed across different applications. Second benefit of virtualization is that it will maintain the isolation across different application, which together in a particular machine. So, the Co-located applications are provided with insufficient isolation and the security provisions.

(Refer Slide Time: 05:13)



So, that they may not interfere with each other and yet they will be multiplexing and sharing the resources of the work of the physical machines or the servers. Third benefit here is that it will rapidly adjust the resource allocation to the application for example, the application may require additional CPU on a priority basis or it will require more memory to be allocated to the application all these are possible in the scenario of

virtualized data center. Fourth benefit of virtualization is the virtual machine migration that will be transparent to the application.

So, the application will not know about where this particular server is being used for running that particular application and it is all transparent. Virtual machine migration will support the cause of adjusting the resource allocation, if let us say more resources are required and that server which is currently running the application does not have that many number of resources. So, it will be migrated this is called virtual machine migration and in this migration there will not be any downtime for the applications and also it will incur the less overhead this is all possible with the help of virtualization techniques.

So, how can we use virtualization to more efficiently utilize the data center resource becomes a very important task for the researchers to ponder over. Let us see the data center workloads these data centers are very popular to host wide variety of applications, such as web applications enterprise system applications e-commerce websites which are running for the customers. So, these applications require the data center to be used and they will generate the dynamic workloads of different workloads. This dynamic workload will fluctuate due to the incremental growth of the workloads and also during the time of the day affects.

For example, there are some peak hours when many customers are accessing these e-commerce website or doing the navigation during the office hours may be the peak time. So, the time of the day also affects this dynamic workload fluctuations and also due to the flash crowds, sometimes we will generate lot of workload and this will guarantee or this will generate the dynamic workload fluctuations.

This all fluctuation has to be absorbed by these data center who are running those applications. So, here how can we provision resources to meet these challenging demands of the workload or a dynamic workloads fluctuations, which are basically due to the incremental growth or during the day time of the day effects or during or due to the flash crowds. So, how can we provision these resources? To meet these challenging demands while meeting the service level agreements is not a trivial task.

(Refer Slide Time: 09:12)

Provisioning Methods

- **Hotspots** form if resource demand exceeds provisioned capacity
- **Over-provisioning**
 - Allocate for peak load
 - Wastes resources
 - Not suitable for dynamic workloads
 - Difficult to predict peak resource requirements
- **Dynamic provisioning**
 - Adjust based on workload
 - Often done manually — *agility*
 - Becoming easier with virtualization

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Now, we will see the provisioning methods which will allocate the resources as per the workload or a dynamic workload demand and what are the issues involved we will call it as the provisioning methods. Now if the resource demands exceed the provisioned capacity, then we say that the hotspots will form. Over provisioning of the resources means that they are allocated as per their peak load calculations and they have to be ensured as per the service level agreement slash.

So, over provisioning though there is lot of resources and also is not very suitable for dynamic workloads and also very difficult to predict the peak resource requirements. So, here we will see this over provisioning methods, which will basically how we will ensure it according to the service level agreements. Another thing is called dynamic provisions; that means, due to the workload fluctuations a dynamic provisioning also is required to be in place.

So, this will adjust the resource allocation based on the dynamic workload fluctuation, often it is done manually but it lacks with the agility. If it is done through the in a manual way we will understand this way that why manually will lack the agility; therefore, dynamic provisioning has to be done in a automatic manner.

(Refer Slide Time: 10:15)

Problem Statement

How can we automatically (i) monitoring for resource usage,(ii) hotspot detection, and (iii) mitigation i.e. determining a new mapping and initiating the necessary migrations (i.e. **detect and mitigate Hotspots**) in virtualized data centers?

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Now let us understand the problem for this particular discussion, so given this particular scenario how can we automatically monitor for the resource usage and how can we detect the hotspots and then meeting at the hotspots after finding them, that is how to determine the new mapping and initiate the necessary virtual machines migration. Therefore the problem for this particular discussion is to detect and mitigate the hotspots in a virtualized data center environment. So, again we will recall this word hotspot means that if the resource demands exceeds the provision capacity.

(Refer Slide Time: 12:10)

Hotspot Mitigation Problem

- Once a **hotspot has been detected** and new allocations have been determined for overloaded VMs, the migration manager invokes its hotspot mitigation algorithm.
- This algorithm determines which virtual servers to migrate and where in order to dissipate the hotspot.
- Determining a new mapping of VMs to physical servers that avoids threshold violations is NP-hard**—the multidimensional bin packing problem can be reduced to this problem, where each physical server is a bin with dimensions corresponding to its resource constraints and each VM is an object that needs to be packed with size equal to its resource requirements.
- Even the problem of determining if a valid packing exists is NP-hard.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, hotspot mitigation problem is that once we have to detect the hotspot that also is not trivial, we will see some of the methods how we can detect the hotspots and having detected the hotspots then we have to involve a new allocation strategy to deal with the overloaded virtual machines and maybe sometimes requires the virtual machine migration that is all contained in the hotspot mitigation algorithms.

So, mitigation of the hotspot algorithms will determine which virtual servers have that many that sufficient resources required by over provisioned virtual machines are required by the virtual machines, therefore it has to be migrated in order to mitigate the hotspots. So, determining a new mapping of virtual machine to the physical machine that avoids the threshold violations, specified as per the service level agreement is an NP hard problem. That means, there exist an NP complete problem that is called multidimensional bin packing problem, which can be reduced to the hotspot mitigation problem that we have just described.

So, if it is reduced that means multiple multidimensional bin packing problem can be reduced to the hotspot mitigation problem, where each server is a bin with the multiple dimension corresponding to the resource constraints and each virtual machine is an object that need to be packed with the equal with the size equal to its resource requirements. Even the problem of determining if a valid packing of multidimensional bin exist to determinate itself is a hard problem

(Refer Slide Time: 14:35)

Research Challenges

- **Hotspot Mitigation:** automatically detect and mitigate hotspots through virtual machine migration
- **When** to migrate?
- **Where** to move to?
- **How much** of each resource to allocate?



Sandpiper
A migratory bird

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, this will serve the research challenges for this particular discussion we will see some of the intricacies and we will also see what are the solutions available in the literature in this part of the discussion. So, the research challenges for hotspot mitigation is to automatically detect and mitigate hotspot through the virtual machine migration.

To do this we have to decide when to migrate where to migrate and how much of these resources will be needed to allocate after the migration, this all migration problem is now is published in a paper which has proposed the method which is called sandpiper, which is inspired from that bird called sandpiper which is a migratory bird.

(Refer Slide Time: 15:38)

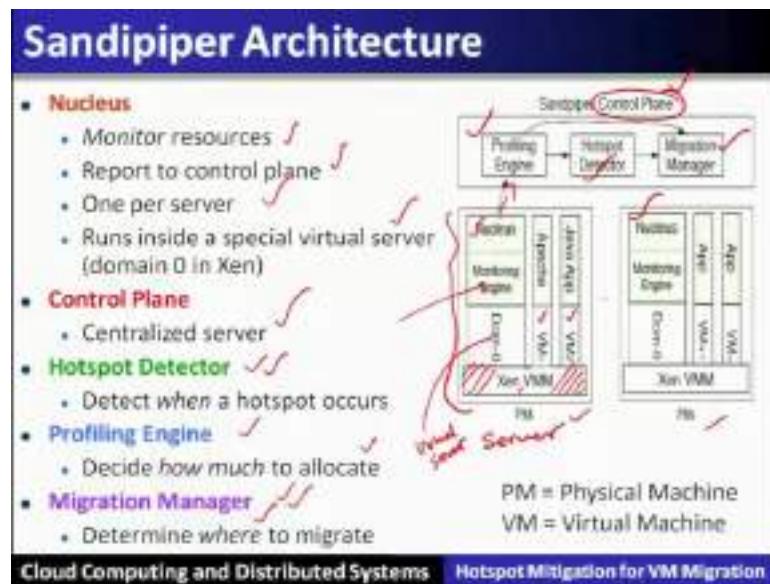
Background

- **Dynamic replication:**
 - Dynamic provisioning approaches are focused on dynamic replication, where the number of servers allocated to an application is varied.
- **Dynamic slicing:**
 - In dynamic slicing, the fraction of a server allocated to an application is varied.
- **Application migration:**
 - In the virtualization, VM migration is performed for dynamic provisioning.
 - Migration is transparent to applications executing within virtual machines.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, how much information is needed to make these particular decisions let us see some of the terminologies as a background. A dynamic replication means that a dynamic provisioning approaches which are focused on dynamic replication where the number of servers which are to be allocated to an application is being varied. A dynamic slicing in a dynamic slicing a fraction of the server is allocated to an application is varied, application migration in the virtualized scenario virtual machine migration is performed for dynamic provisioning and this migration is transparent to the applications executing within the virtual machine.

(Refer Slide Time: 16:37)



Let us understand the architecture of sandpiper scheme for hotspot mitigation problem. The entire architecture can be understood by looking at the physical machines that is nothing but the server, server is we assume that it is virtualized using Xen virtual machine monitor or an hypervisor; therefore, the entire server can run virtual machine on top of it which is shown over here as virtual machine 1 virtual machine 2. Now, there is another virtual machine or that is a control virtual machine which is called a nucleus, so nucleus will monitor the resources of that particular server this will report to the control plane this is the control plane that is a global control plane across the data center.

(Refer Slide Time: 17:44)



This particular nucleus will be per machine per server that we see that there are 2 different servers and every server is running a nucleus, this particular nucleus will run inside a special virtual server. So, this is a special virtual server which is called domain zero in the Xen hypervisor.

So, nucleus is nothing but a monitoring engine of that particular server or a physical machine. Next comes the control plane control plane is a global plane or you can say that it is a global intelligence which comprises of this particular data center. So, this intelligence will have the complete information about all the servers running in that particular data center and this information will be sent by special virtualized server which is called as a nucleus. Now, let us see the details of control plane which is the centralized server, this particular control plane has 3 different components the first 1 is called profiling engine.

This profiling engine will get the profile of all the resources, which are collected by the nucleus and then it will decide using that particular profile of the resources. How much of these resources are available and how much they are allocated the next one is called hotspot detector.

So, it will detect when the hotspots are formed we will see how this particular detection of hotspots are done in this part of the discussion, but let us see in the architecture where these algorithms will work. Finally, there is a manager which is called a migration manager, after detecting the hotspot the migration manager has to decide about how to mitigate that hotspot. Therefore, it has to choose a physical machine where sufficient resources required by the hotspot virtual machine can be made available for the migration plan. This all will be decided by the migration manager where to migrate that particular virtual machine to mitigate the hotspots.

This particular architecture has given also 2 different strategies or a schemes for hotspot mitigation, the first one is called black box method here in the black box method this particular operating system and the application is agnostic; therefore, only it has to collect the data from outside the virtual machine it cannot go and see what is happening or the resource profile within that virtual machine.

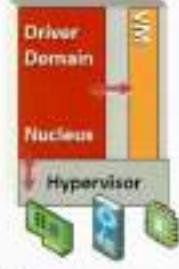
So, outside the virtual machine it will collect the data or the profile and therefore it is called a black box. The other method is called the grey box; here it is possible to look

into inside the virtual machine and to collect the information of operating system in statistics and application logs of that virtual machine. This particular information will generate a profile and also detect the request level data which will be required in the detection method.

(Refer Slide Time: 22:14)

Black-box Monitoring

- Xen uses a “Driver Domain”
 - Special VM with network and disk drivers
 - Nucleus runs here
- CPU
 - Scheduler statistics
- Network
 - Linux device information
- Memory
 - Detect swapping from disk I/O
 - Only know when performance is poor



Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Now, the question is that black box good enough to solve all the problem, if not then what do we gain from the grey box method. Let us understand these methods in more detail to answer this particular question. Black box scheme for monitoring of hotspots in this method since it is operating system and application agnostic; therefore, it uses the profile and the information from the outside that is it will depend on the Xen hypervisor.

So, Xen hypervisor will require to have a special virtual machine which will collect the data from network and disk drivers and the nucleus will run based on that particular information, this will be an outside monitoring without looking into the details of what virtual machine is doing.

Therefore hypervisor will collect the scheduler statistics it will collect the network device information statistics it will also collect the memory. But memory will be detected only through the swapping from the disk I/Os and this will only be known when the performance is poor why because it is only allowed to collect from the outside information.

(Refer Slide Time: 20:53)

Black-box Monitoring

- **CPU Monitoring:** By incrementing the Xen hypervisor, it is possible to provide domain-0 with access to CPU scheduling events which indicate when a VM is scheduled and when it relinquishes the CPU. These events are tracked to determine the duration for which each virtual machine is scheduled within each measurement interval I .
- **Network Monitoring:** Domain-0 in Xen implements the network interface driver and all other domains access the driver via clean device abstractions. Xen uses a **virtual firewall-router (VFR) interface**; each domain attaches one or more virtual interfaces to the VFR. Doing so enables Xen to multiplex all its virtual interfaces onto the underlying physical network interface.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, black box monitoring will perform the CPU monitoring that is why incrementing the Xen hypervisor it is possible to provide the virtual machine that is domain 0, a special control virtual machine with access to the CPU scheduling events which is available with the hypervisor. These events are tracked to determine the duration for which each virtual machine is scheduled and the measurements of that interval when the CPU was allocated or being used by running that particular virtual machine.

Similarly the network monitoring will be done by special control virtual machine that is domain 0, in the Xen hypervisor by collecting the data from the network interface driver for other domain accesses. So, Xen uses the virtual firewall router interface for this particular purpose this particular information will be collected.

(Refer Slide Time: 25:05)

Black-box Monitoring

- **Memory Monitoring:** Black-box monitoring of memory is challenging since Xen allocates a user specified amount of memory to each VM and requires the OS within the VM to manage that memory; as a result, the memory utilization is only known to the OS within each VM.
- It is possible to instrument Xen to observe memory accesses within each VM through the use of shadow page tables, which is used by Xen's migration mechanism to determine which pages are dirtied during migration.
- However, trapping each memory access results in a significant application slowdown and is only enabled during migrations. Thus, memory usage statistics are not directly available and must be inferred.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

(Refer Slide Time: 25:07)

Black-box Monitoring

- Black-box monitoring is useful in scenarios where it is not feasible to "peek inside" a VM to gather usage statistics. Hosting environments, for instance, run third-party applications, and in some cases, third-party installed OS distributions.
- Amazon's Elastic Computing Cloud (EC2) service, for instance, provides a "barebone" virtual server where customers can load their own OS images.
- While OS instrumentation is not feasible in such environments, there are environments such as corporate data centers where both the hardware infrastructure and the applications are owned by the same entity.
- In such scenarios, it is feasible to gather OS-level statistics as well as application logs, which can potentially enhance the quality of decision making.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

In black box for monitoring purposes, so black box monitoring is useful in the scenarios where it is not visible to peek inside the virtual machine together, the usage statistics that is possible in most of the applications. So, the hosting environments for instance runs the third party application and for example amazons elastic computing cloud easy to service provides a bare bone virtual server where the customer can load their operating system images. So, in such scenarios it is feasible together the operating system level statistics as well as the application logs to enhance the quality, therefore it is possible to do grey box monitoring.

So, grey box monitoring can be supported when feasible using a demon which can access to the operating level statistics and also the logs. So, this particular demon can process the logs of the applications and to derive the statistics such as request rate request drop and service times. These direct monitoring of application level statistics will enable the explicit detection of service level violation.

(Refer Slide Time: 26:30)

Gray-box Monitoring

- **Gray-box monitoring** can be supported, when feasible, using a light-weight monitoring daemon that is installed inside each virtual server.
- In Linux, the monitoring daemon uses the **/proc** interface to gather **OS level statistics of CPU, network, and memory usage**. The memory usage monitoring, in particular, enables proactive detection and mitigation of memory hotspots.
- The monitoring daemon also can process **logs of applications** such as web and database servers to derive statistics such as **request rate, request drops and service times**.
- Direct monitoring of such **application-level statistics** enables **explicit detection of SLA violations**, in contrast to the black-box approach that uses resource utilizations as a proxy metric for SLA monitoring.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So that means, in a grey box has more access inside the statistics of the virtual machine in contrast to the black box approach.

(Refer Slide Time: 26:40)

What is a Hotspot ?

- **A hotspot** indicates a resource deficit on the underlying physical server to service the collective workloads of resident VMs.
- Before the hotspot can be resolved through migrations, The system must first estimate how much additional resources are needed by the overloaded VMs to fulfill their SLAs; these estimates are then used to locate servers that have sufficient idle resources.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, what is the hotspot? So hotspot indicates a resource deficit on the underlying physical server to service the collective workloads of the resident virtual machine. So, if there is a resource deficit's then it will be called hotspot, why because that current server will not be able to allocate the physical resources to handle the workloads of the resident virtual machine. So, before the hotspots can be resolved through the migration, the service first determine the estimate how much of such additional resources are needed by the overloaded virtual machine to fulfill their service level agreements.

These estimates are then used to locate the servers with sufficient idle resources and then we will initiate the virtual machine migration for hotspot mitigation. So, let us see the hotspot detection how this is done this is very important algorithm.

(Refer Slide Time: 27:53)

Hotspot Detection

- The **hotspot detection algorithm** is responsible for signalling a need for VM migration whenever SLA violations are detected implicitly by the black-box approach or explicitly by the gray-box approach.
- Hotspot detection is performed on a per-physical server basis in the black-box approach—a hot-spot is flagged if the aggregate CPU or network utilizations on the physical server exceed a threshold or if the total swap activity exceeds a threshold.
- In contrast, explicit SLA violations must be detected on a per-virtual server basis in the gray-box approach—a hotspot is flagged if the memory utilization of the VM exceeds a threshold or if the response time or the request drop rate exceed the SLA-specified values.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

The hotspot detection algorithm is responsible for signaling a need for virtual machine migration, when service level agreements are violated this is are detected implicitly by black box approach or explicitly by the grey box approach. The hotspot detection is performed on a per physical basis in a black box approach.

That is a hotspot plan if the aggregate CPU network utilizations exceed the threshold or if the total swap activities exceed the threshold the explicit service level agreement violations must be detected on a per server basis in a grey box approach. A hotspot is flagged; if the memory utilization of virtual machine exceeds the threshold or if the response time or the request drop rate exceed the service level.

(Refer Slide Time: 28:46)

Hotspot Detection

- To ensure that a small transient spike does not trigger needless migrations, a hotspot is flagged only if thresholds or SLAs are exceeded for a sustained time. Given a time-series profile, a hotspot is flagged if at least k out of the n most recent observations as well as the next predicted value exceed a threshold. With this constraint, we can filter out transient spikes and avoid needless migrations.
- The values of k and n can be chosen to make hotspot detection aggressive or conservative. For a given n , small values of k cause aggressive hotspot detection, while large values of k imply a need for more sustained threshold violations and thus a more conservative approach.
- **In the extreme, $n = k = 1$** is the most aggressive approach that flags a hotspot as soon as the threshold is exceeded. Finally, the threshold itself also determines how aggressively hotspots are flagged; lower thresholds imply more aggressive migrations at the expense of lower server utilizations, while higher thresholds imply higher utilizations with the risk of potentially higher SLA violations.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Agreement is specified to ensure a small transient spike needless trigger the needless migration in the hotspot detection mechanism; therefore, at least k out of n most recent observation as well as the next predicted value exceeding that particular threshold is used. With this constraint we can filter out the transient spikes and avoid the needless migration. So, the value of k and n can we choose into make the hotspot detection aggressive or conservative. So, in the extreme when $n = k = 1$ is the most aggressive approach.

(Refer Slide Time: 29:27)

Hotspot Detection

- In addition to requiring **k out of n violations**, we also require that the next predicted value exceed the threshold.
- The additional requirement ensures that the hotspot is likely to persist in the future based on current observed trends. Also, predictions capture rising trends, while preventing declining ones from triggering a migration.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Let us see the hotspot detection that in addition to requiring k out of violations, we also require the next predicted value exceeds the threshold. The additional requirement ensures that the hotspot is likely to persist in the future based on the current observed trend also the predictions captures the rising trends while preventing the declining ones from triggering a migration.

(Refer Slide Time: 29:52)

Hotspot Detection

- Sandpiper employs **time-series prediction techniques to predict future values**. Specifically, the system relies on the auto-regressive family of predictors, where the n-th order predictor AR[n] uses n prior observations in conjunction with other statistics of the time series to make a prediction. To illustrate the first-order AR(1) predictor, consider a sequence of observations: u_1, u_2, \dots, u_k . Given this time series, we wish to predict the demand in the $(k+1)$ th interval. Then the first-order **AR(1)** predictor makes a prediction using the previous value u_k , the mean of the time series values μ , and the parameter which captures the variations in the time series. The prediction u_{k+1} is given by:

$$u_{k+1} = \mu + \phi(u_k - \mu)$$

(Refer Slides)

- As new observations arrive from the nuclei, the hot spot detector updates its predictions and performs the above checks to flag new hotspots in the system.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Let us understand the more details of hotspot detection; the sandpiper method will employ time series prediction techniques. To predict the future values this particular scheme prediction is based on auto regression family of predictors, where nth order predictor uses the prior observations in conjunction with the other statistics of the time series to make the prediction.

To illustrate this let us understand the first order predictor AR1 and then consider a sequence of observations let us say that u_1 to u_k . So, given this particular time series auto regressive collection of observations we use to predict the demand in the $(k+1)$ th interval.

Then using the first order prediction the predictor makes a prediction using the previous k values the mean of the time series value μ and the parameter which captures the variation in the time series. The prediction u_{k+1} is given by this particular formulation over here, it depends upon the mean of the time series value μ and also it will apply the

first order prediction on the values on the sequence of observations which are collected over here.

As the new observations arrive from the nucleus the hotspot detector update it is prediction perform the above checks to flag the new hotspot in the system. So, just see that this particular method we will use the predictions based on the mean and also the previous k different observations to predict the $(k + 1)^{\text{th}}$ possible value.

(Refer Slide Time: 32:18)

Hotspot Detection – When?

- **Resource Thresholds**
 - Potential hotspot if utilization exceeds threshold
 - Only trigger for *sustained overload* ✓
 - Must be overloaded for k out of n measurements
- **Autoregressive Time Series Model**
 - Use historical data to predict future values
 - Minimize impact of transient spikes

Utilization

Time

Utilization

Time

Utilization

Time

Not overloaded

Hotspot Detected!

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Using this it will detect it will conclude about the hotspot detection. Now when this hot spot detection will be done hotspot detection will be done when this particular resource requirements they will touch the thresholds. So, the potential thresholds if utilization exceeds the threshold then it will be flagged and the hotspot detection will be triggered it is not to be triggered during a transient spikes. Therefore, it will only trigger for sustained overloads and it must be overloaded for k out of n measurements to avoid the outliers of the spikes.

So, auto regression time series model we will see or we have been or it has been used here in hotspot detection, which will be based on the historical data to predict the future values and also minimize in this process the impact of transient spike. So, here in this particular example that if there is a transgender spy this will not trigger detection into the hottest part. But this particular k out of n observation we will is moved this and only

whenever there is a real sustained overload then only it will be triggered as an hotspot detection.

(Refer Slide Time: 33:55)

Resource Provisioning: Black-box Provisioning

- The provisioning component needs to estimate the peak CPU, network and memory requirement of each overloaded VM; doing so ensures that the SLAs are not violated even in the presence of peak workloads.
- **Estimating peak CPU and network bandwidth needs:** Distribution profiles are used to estimate the peak CPU and network bandwidth needs of each VM. The tail of the usage distribution represents the peak usage over the recent past and is used as an estimate of future peak needs.
- This is achieved by computing a high percentile (e.g., the 95th percentile) of the CPU and network bandwidth distribution as an initial estimate of the peak needs.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, let us see the resource provisioning in black box method. So, the provisioning components needed to estimate the peak CPU network and memory requirement for each overloaded virtual machine, to ensure that service level agreements are not violated even in the presence of peak workloads. So, estimating the peak CPU network bandwidth will need the distribution profiles to estimate the peak CPU network utilization needs of the virtual machines, that will be the tail of usage distribution which will represent the peak usage what the recent past and is used as an estimate for the future peak needs. Now, this particular estimation can be achieved by computing the high percentile of CPU and the network bandwidth distribution as the initial estimate of the peak needs.

(Refer Slide Time: 34:53)

Limitation of the black-box approach

- **Example:** Consider two virtual machines that are assigned CPU weights of 1:1 resulting in a fair share of 50% each. Assume that VM1 is overloaded and requires 70% of the CPU to meet its peak needs. If VM2 is underloaded and only using 20% of the CPU, then the work conserving Xen scheduler will allocate 70% to VM1.
- In this case, the tail of the observed distribution is a good indicator of VM1's peak need. In contrast, if VM2 is using its entire fair share of 50%, then VM1 will be allocated exactly its fair share. In this case, the peak observed usage will be 50%, an underestimate of the actual peak need. Since the system can detect that the CPU is fully utilized, it will estimate the peak to be $50 + \Delta$.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

To understand this there is a example shown over here, if there are 2 virtual machines that are assigned the CPU weights 1 is to 1 resulting in a fair share of 50 % each assume that VM 1 is overloaded and required 70 %. Now, to meet it is peak needs whereas virtual machine 2 is under loaded which requires only 20 % of the CPU, then the work conserving Xen scheduler will allocate seventy percent to the to the virtual machine 1. In this case the tail of the observed distribution is a good indicator of virtual machine ones peak needs, in contrast to the virtual machine 2 is using it is entire fair share of 50 % then virtual machine 1 will be allocated exactly it is fair share.

In this case the peak observed usage will only be 50 percent and underestimate of the actual peak needs. Since the system can detect the CPU as fully neutralized it will estimate the peak to be 50 plus delta.

(Refer Slide Time: 36:08)

Resource Provisioning: (i) Black-box Provisioning

- **Estimating peak memory needs:** Xen allows a fixed amount of physical memory to be assigned to each resident VM; this allocation represents a hard upper-bound that can not be exceeded regardless of memory demand and regardless of the memory usage in other VMs.
- Consequently, the techniques for estimating the peak CPU and network usage do not apply to memory. The provisioning component uses observed swap activity to determine if the current memory allocation of the VM should be increased.
- If swap activity exceeds the threshold indicating memory pressure, then the current allocation is deemed insufficient and is increased by a constant amount Δm .

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Similarly it is possible to estimate the peak memory needs, where Xen hypervisor allows fixed amount of physical memory to be assigned to each resident virtual machine this allocation represents a hard upper bound that cannot be exceeded regardless of the memory demand and regardless of the memory usage in other virtual machines. So, therefore, the swap activity exceeds the threshold indicating the memory pressure, when the current allocation is deemed to be insufficient and is increased by the constant amount of memory requirement.

(Refer Slide Time: 36:57)

Resource Provisioning: (ii) Gray-box Provisioning

- Since the **gray-box approach** has access to application level logs, information contained in the logs can be utilized to estimate the peak resource needs of the application.
- Unlike the black-box approach, the peak needs can be estimated even when the resource is fully utilized.
- To estimate peak needs, the peak request arrival rate is first estimated. Since the number of serviced requests as well as the number of dropped requests are typically logged, the incoming request rate is the summation of these two quantities.
- Given the distribution profile of the arrival rate, the peak rate is simply a high percentile of the distribution. Let λ_{peak} denote the estimated peak arrival rate for the application.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Let us see the resource provisioning in the grey box method, since the grey box method has access to the application level logs the information contained the logs can be utilized to estimate the peak resource requirement of the application. To estimate the peak needs, the peak request arrival rate is first estimated. Since, the number of service request as well as the number of dropped request are typically locked the incoming request rate is the summation of these 2 quantities. Given the distribution profile of arrival rate, the peak rate is simply a high percentile of the distribution which is modeled as the G/G/1 queuing system. The behavior of such queuing system can be captured using the formula which is available in the theory of queues.

(Refer Slide Time: 37:50)

Estimating peak CPU needs:

- An application model is necessary to estimate the peak CPU needs. Applications such as web and database servers can be modeled as **G/G/1 queuing systems**. The behavior of such a **G/G/1 queuing system** can be captured using the following queuing theory result:

$$\lambda_{\text{peak}} \geq \left[s + \frac{\sigma_a^2 + \sigma_b^2}{2 \cdot (d - s)} \right]^{-1}$$

- where d is the mean response time of requests, s is the mean service time, and λ_{peak} is the request arrival rate. σ^2_a and σ^2_b are the variance of inter-arrival time and the variance of service time, respectively. Note that response time includes the full queuing delay, while service time only reflects the time spent actively processing a request.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

(Refer Slide Time: 37:52)

Estimating peak CPU needs:

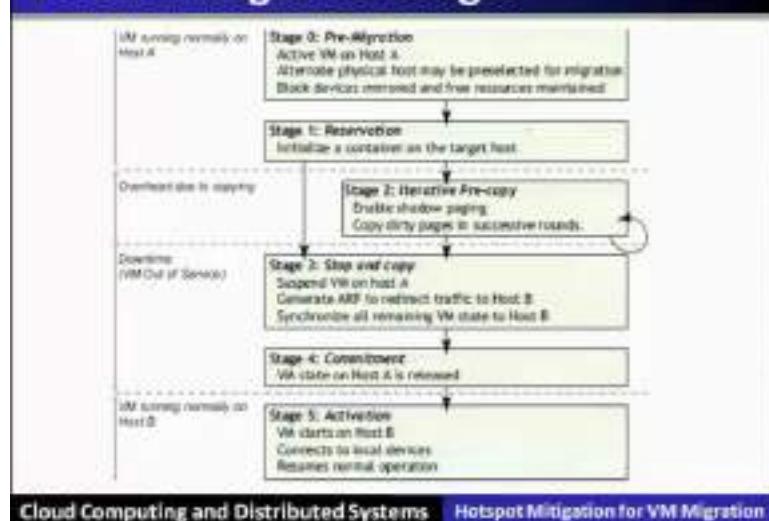
- While the desired response time d is specified by the SLA, the service time s of requests as well as the variance of inter-arrival and service times σ^2_s and σ^2_d can be determined from the server logs. By substituting these values into equation, a lower bound on request rate λ_{cap} that can be serviced by the virtual server is obtained.
- Thus, λ_{cap} represents the current capacity of the VM. To service the estimated peak workload λ_{peak} , the current CPU capacity needs to be scaled by the factor $\lambda_{peak} / \lambda_{cap}$.
- Observe that this factor will be greater than 1 if the peak arrival rate exceeds the currently provisioned capacity. Thus, if the VM is currently assigned a CPU weight w , its allocated share needs to be scaled up by the factor $\lambda_{peak} / \lambda_{cap}$ to service the peak workload.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

This way the peak CPU can be estimated. Similarly, the peak network also can be estimated in this manner. Now, having estimated all that requirement that is having detected the hotspot.

(Refer Slide Time: 38:09)

Live VM Migration Stages



Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Let us see the virtual machine migration will take place before that let us see what are the steps in a live virtual machine migration stages. So, stage 0 will say that during the pre migration when the virtual machine running normally on a host A; so that means, it will ensure that which are the active VMs on the host A and what are the alternative alternate physical host maybe preselected for the migration and what are the block devices which are mirrored and free resources which are maintained; this is basically a stage 0 that is

pre migration. The next stage will be stage 1, we will call reservation that is it will in a slice the container in the target host.

Now, the overhead due to the copying that is stage 2; what it will do? It will do iterative precopy, it will enable the shadow paging and then it will keep on copying the dirty pages in the successive rounds. Then stage number 3 is called stop and copy that way it will suspend the VM on the host A and it will generate the network traffic to redirect to the host B over the network and it will synchronize all the remaining virtual machine state to the host B that is called a stop and copy on the host B, stage 4 is called commitment.

So, virtual machine state on host A is released finally in stage 5, this is called activation. So, virtual machine will now start running on host B, it will connect to the local devices and resumes in local operations.

(Refer Slide Time: 40:11)

Live VM Migration Stages

- **Stage 0: Pre-Migration** We begin with an active VM on physical host A. To speed any future migration, a target host may be preselected where the resources required to receive migration will be guaranteed.
- **Stage 1: Reservation** A request is issued to migrate an OS from host A to host B. We initially confirm that the necessary resources are available on B and reserve a VM container of that size. Failure to secure resources here means that the VM simply continues to run on A unaffected.
- **Stage 2: Iterative Pre-Copy** During the first iteration, all pages are transferred from A to B. Subsequent iterations copy only those pages dirtied during the previous transfer phase.

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

So, these stages we can understand in more details in this particular figure.

(Refer Slide Time: 40:19)

Placement Algorithm

- First try migrations
 - Displace VMs from high Volume servers
 - Use Volume/RAM to minimize overhead
- Don't create new hotspots!
 - What if high average load in system?
- Swap if necessary
 - Swap a high Volume VM for a low Volume one
 - Requires 3 migrations
 - Can't support both at once

Swaps increase the number of hotspots we can resolve

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

(Refer Slide Time: 40:32)

Determining Placement – Where to?

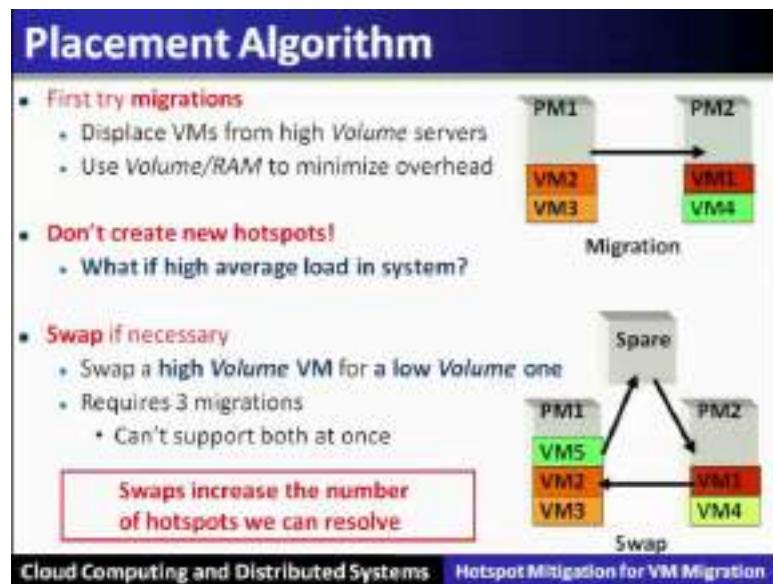
- Migrate VMs from overloaded to underloaded servers
- Volume = $\frac{1}{1\text{-cpu}} + \frac{1}{1\text{-net}} + \frac{1}{1\text{-mem}}$
- Use Volume to find most loaded servers
 - Captures load on multiple resource dimensions
 - Highly loaded servers are targeted first
- Migrations incur overhead
 - Migration cost determined by RAM
 - Migrate the VM with highest Volume/RAM ratio

Maximize the amount of load transferred while minimizing the overhead of migrations

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

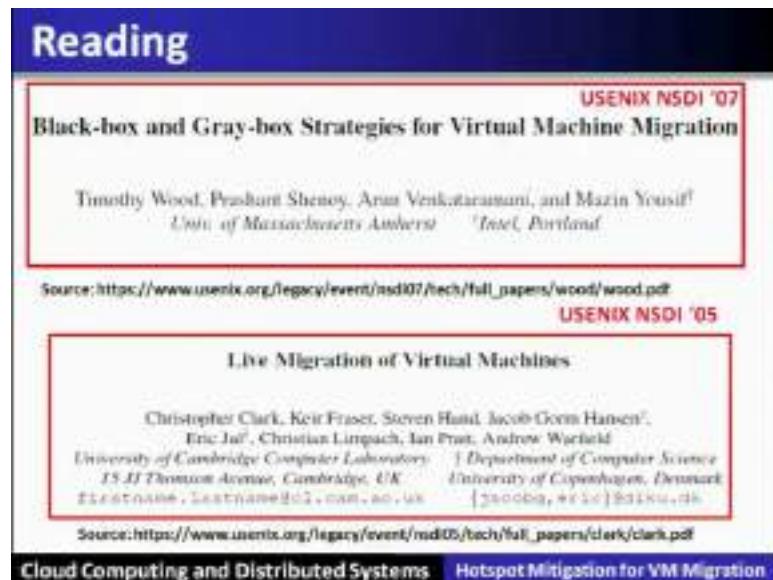
So, we have to determine the in this particular virtual machine migration the placement where the placement is to be done, finally which has the sufficient number of resources. So here in this particular method, we have to maximize the amount of load which is transferred while minimizing the overhead of migration.

(Refer Slide Time: 40:53)



So, the placement algorithm will first try the migration displays the virtual machine from the high volume servers, use the volume of the RAM to minimize the overhead. It will swap if necessary; swap a high volume virtual machine to a low volume 1 requires 3 different.

(Refer Slide Time: 41:17)



This entire paper is published in the form of black box and grey box strategies for virtual machine migration. This is published by the author Timothy Wood, Prashant Shenoy, Arun Venkataramani and Mazin Yousif of university of Massachusetts Amherst and Intel

Portland, this is published in NSDI in 2007. There is another paper which is referred here is called live virtual machine; live migration of virtual machines, this is by Christopher Clark and is published NSDI in 2005.

(Refer Slide Time: 42:08)

Conclusion

- Virtual Machine migration is a viable tool for dynamic data center provisioning.
- In this lecture, we have discussed an approach to rapidly detect and eliminate hotspots while treating each VM as a black-box.
- Gray-Box information can improve performance in some scenarios
 - Proactive memory allocations
- Future work:
 - Improved black-box memory monitoring
 - Support for replicated services

Cloud Computing and Distributed Systems Hotspot Mitigation for VM Migration

Conclusion: Virtual machine migration is a viable tool for dynamic data center provisioning, this lecture we have covered an approach to rapidly detect and eliminate hotspots while treating each virtual machine as a black box. A grey box method which wherein we can look inside the virtual machine statistics, grey box information can further be used to improve the performances in some of the scenarios. We can also do the proactive memory allocations in the grey box to make the virtual machines migration more efficient.

There are several future work possible in this particular direction and this is going to be very useful topic in automating the virtualized data center and for building the cloud computing system. The future work and tails as the improved black box memory based monitoring and also support for the replicated services.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 04
Server Virtualization

Server Virtualization; preface content of this lecture.

(Refer Slide Time: 00:18)

Preface

Content of this Lecture:

- In this lecture, we will discuss server virtualization and the need of routing and switching for physical and virtual machines.
- Then we will discuss the two methods of virtualization:
(i) Docker based and (ii) Linux container based and also address the problem of networking VMs and its **hardware based approach: SR-IOV, single-root I/O virtualization and software based approach: Open vSwitch**

Cloud Computing and Distributed Systems Server Virtualization

In this lecture, we will discuss server virtualization and the need of routing switching for physical and virtual machines. Also, we will see 2 other methods of virtualization that is the Docker based method and Linux container based methods. We will also see the problems of networking of VMs in this context we will cover 2 different ways one is hardware based approach that is SR-IOV, the other one is software based approach that is using open vSwitch.

(Refer Slide Time: 01:01)

Cloud Computing depends on Server Virtualization

- * In the context of server virtualization, how do we network the large number of VMs that might reside on a single physical server? Cloud computing depends heavily on server virtualization for several reasons:

(i) **Sharing of physical infrastructure:** Virtual machines allow multiplexing of hardware with tens to 100s of VMs residing on the same physical server. Also, it allows rapid deployment of new services.

(ii) **Spinning up a virtual machine in seconds:** Spinning up a virtual machine might only need seconds compared to deploying an app on physical hardware, which can take much longer.

(iii) **Live VM migration:** Further, if a workload requires migration. For example, you do physical server requiring maintenance. This can be done quickly with virtual machines, which can be migrated to other servers without requiring interruption of service in many instances. Due to these advantages today, more endpoints on the network are virtual rather than physical.

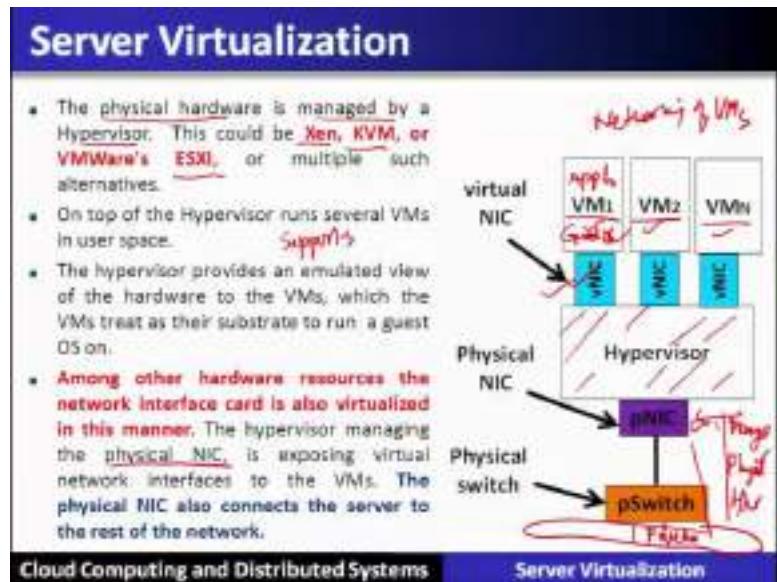
Cloud Computing and Distributed Systems Server Virtualization

The cloud computing heavily depends on the server virtualization. So server virtualization we will see how do we network the large number of VMs which are running on a single server machine. So, the cloud computing heavily depends on the server virtualization because of these several reasons, the first reason is sharing of physical infrastructure. So, virtual machine allows multiplexing of hardware to that hundreds of VMs residing the same physical server. So, therefore if such a large number of VMs are spanning on a same physical machine. So, now the issue comes of it is rapid deployment whenever a new services are coming.

So, the spinning of virtual machines need to be done in a fraction of seconds, so spinning of virtual machines might only needs some fraction of seconds compared to the deploying an application on a physical hardware, so this is another reason of the server virtualization.

Third most important reason is the maintenance, whenever there is a maintenance in the physical server then all the VMs need to be migrated to some other server that is called live VM migration. So, this will allow that the services which are running through the virtual machines are not disrupted in a state they will be migrated that is called live VM migration. So, due to this feature the server virtualization is also going to be very important in realizing the cloud computing or in realizing the today's cloud.

(Refer Slide Time: 03:23)



Let us see what is a server virtualization? Now we see that in this picture that hypervisor manages the physical hardware, so this is the hypervisor and it manages. Now this particular hypervisor could be Xen hypervisor or it could be Linux kernel based virtual machine that is KVM hypervisor or VMware's ESXI hypervisor or many other hypervisors are available.

So, this hypervisor is managing the hardware physical hardware resources using this managed management of physical hardware resources it also support several virtual machines. So, it supports several virtual machines which runs on top of these hypervisor. So, this hypervisor will provide an emulated view of the hardware to these virtual machines, which the virtual machines treat as their substrate to run on the guest operating systems. So, these virtual machines they have the guest operating system and this is the application. So, this guest operating system will have the view as if it is owning all the physical resources to run it is application and that is being supported by the hypervisors.

Now, among other hardware resources the network interface card is also virtualized in this particular manner. So, these VMs are need to be networked and thus physical network interface card NICs also need to be virtualized. So, that all the VMs can be networked together, so the hypervisor which is managing the physical network interface card is exposing the virtual network interface card that is called VNICs virtual NICs to the VMs. So, every VM will be having a virtual network interface card and this allows

the networking of all the VMs. So, the physical NIC also connects the servers this server to the rest of the world.

So, in this particular server virtualization we see that how the networking of these VMs are to be done through the virtual NICs, which basically is managed by the hypervisor which is managing the physical NICs and it will provide the support to the virtual NICs which will intern network all the VMs. You know that NICs are responsible to connect the servers to the outside world, so this way all the VMs also will get the connectivity in this form.

(Refer Slide Time: 07:33)

Networking of VMs inside the Hypervisor

- The hypervisor runs a virtual switch, this can be a simple layer tool searching device, Operating in software inside the hypervisor.
- vSw is connected to all the virtual NICs, has them as the physical NIC, and moves packets between the VMs and the external network.

Alternate Methods of virtualization:

- (i) Using Docker
- (ii) Using Linux containers

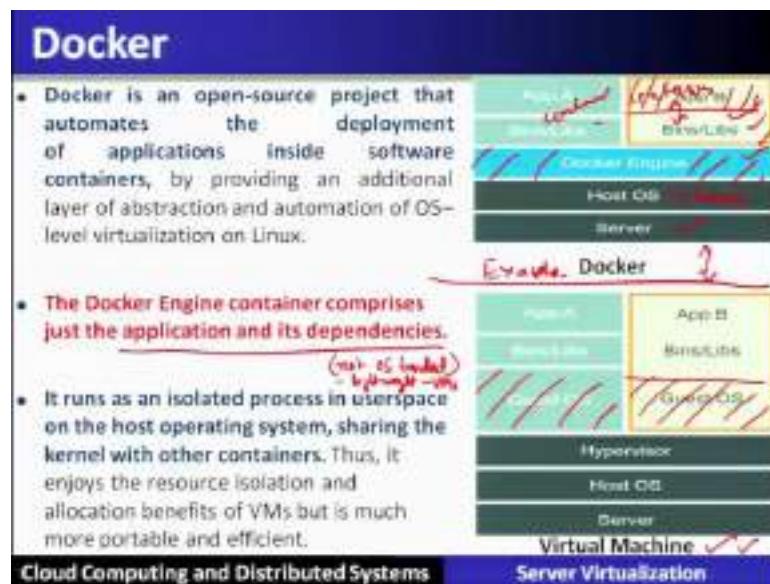
Cloud Computing and Distributed Systems Server Virtualization

So, that working of VMs inside hypervisors is done we will see the more detail of it. The hypervisors runs a virtual switch, so inside the hypervisor you can see here the virtual switch is represented. So, the hypervisor runs the virtual switch and this can be a simple layer tools of a switching device and the operating system in the software inside the hypervisor. So, virtual switch is connected this virtual switch is connected to all the virtual NICs and also it is connected to the physical NIC. So, virtual switch will basically which is managed by the or which runs inside the hypervisor provides the connectivities to the virtual NICs by managing the physical NICs.

So, whenever a packet of a particular VM comes through the virtual NICs. So, it is the virtual switch which will basically use the physical NIC and therefore these packets are routed to the external world. Now we will see this particular aspect in more details, but

before that let us see some more alternative methods of virtualization. We have seen the virtual machines as the virtualization, where in every application is bundled with its corresponding operating system and it will have the virtual resources this is called a virtual machine. This allows the sharing of the physical hardware resources or let us say the server this type of virtualization we have already seen. Now, there is an alternative to it using the Docker and the Linux container let us see these further approaches and then we will come back again to the virtual switch.

(Refer Slide Time: 10:24)



So, Docker is a method for virtualization as I told you. So, Docker is an open source project that automates the deployment of application inside the software container by providing additional layer of abstraction and automation of OS level virtualization on the Linux. Take this example on the figure here there is a Docker engine on top of it, it runs the containers this is container this is also container. This Docker engine is just above the host operating system this is nothing, but a Linux kind of operating system runs over the server.

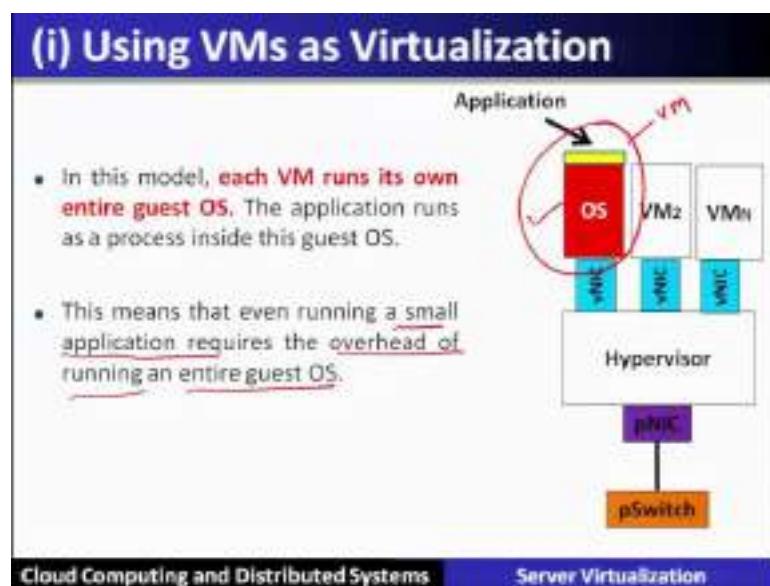
So, the Docker engine here it basically figures out in the picture contains Docker engine container comprises just the application and its dependencies. So, unlike in virtualization we have seen the application with its corresponding operating system bundled together, but here the application corresponds to the application and its dependencies. Dependencies mean the binaries and the libraries which require to run

that application not the entire operating system, so it is a some part of which was used earlier and this together is called the container.

So, now here the application and its corresponding dependencies are called as the container, we have shown the container over here and this is the simple virtual machine which also has the operating system. So, you can see the comparison between the Docker container based virtualization and the simple virtual machine where the entire guest operating system is also bundled with the application. So, the Docker engine container comprises just an application and its dependencies not; so that means, it is not having the OS bundled. So, hence it is a lightweight in comparison to the virtual machines based virtualization.

Now, it runs as an isolated process in the user space that you see because it is running above the operating system. So, it runs as a user space on the host operating system sharing the kernel and other container, thus it enjoys the resource isolation and allocation benefits of the virtual machine, but is much more portable and efficient why because it is managed by the Linux operating system all the containers.

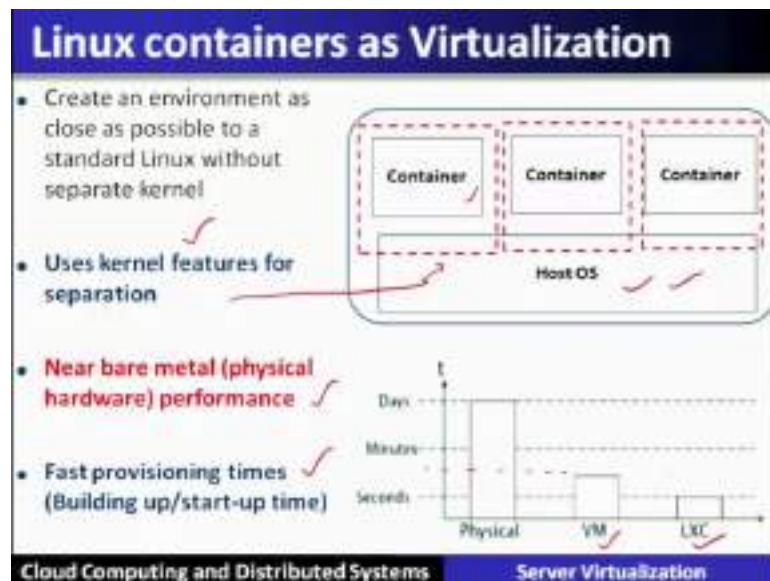
(Refer Slide Time: 13:40)



So, let us see without Docker and container that is simple virtual machine based virtualization, in this model you see that each VM runs its own guest operating system. So, just see that the application bundled with the operating system with the virtual NIC is called virtual machine. So, this means that even running a small application it requires to

be bundled with the entire guest operating system. So, it becomes a big overhead if the applications are a little or quite small, so this is the drawback of virtualization if it is done using virtual machines.

(Refer Slide Time: 14:38)



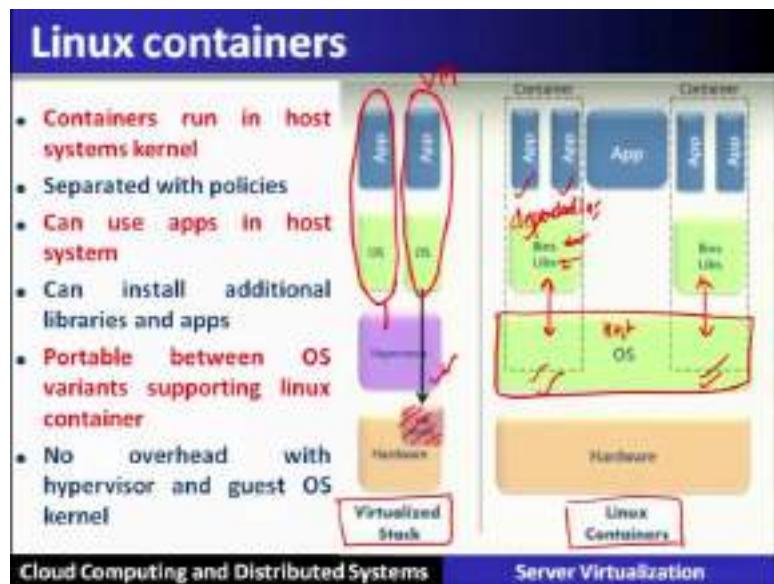
On other hand if Linux containers are used as the virtualization, then you see the container is different than the host operating system and container does not contain the operating system. So, this will create a environment as close as possible to the standard Linux without the separate kernel. So, there is a same operate host operating system which will now manage or which will give the support to all the different containers.

So, it uses the concept of Linux kernel based features for the separation or the isolation which are shown over here as dotted blocks which provides an separation and isolation. So, this will ensure that the container the applications which are there inside container running they will run having the performance near bare metal performance that is very fast.

Also it has the advantage that it has a fast provisioning times that is since it is a light weighted container do not contain the operating systems within, but it uses the host operating system. So, the provisioning time that is building up or the startup time is also very little, in this particular case take the comparison between the virtual machine and the Linux based containers.

So, the fraction of seconds here it is much more than or near to the minutes as for as a VMs are concerned. Therefore, the containers as the virtualization is one of the important features of virtualization, which is used in the server virtualization to support large number of applications running in the same server, compared to the to the VMs which will not be that many number of applications compared to the same capacity.

(Refer Slide Time: 17:08)



So, Linux containers run in the host system kernel and it is also having the separated with the policies; that means, every container it is own policies and can install it is additional libraries and binaries which are required to run this app and the same host operating system will support running these containers. So, they are portable between operating system variants supporting the Linux containers and there is no overhead with the hypervisor and the guest operating system kernel.

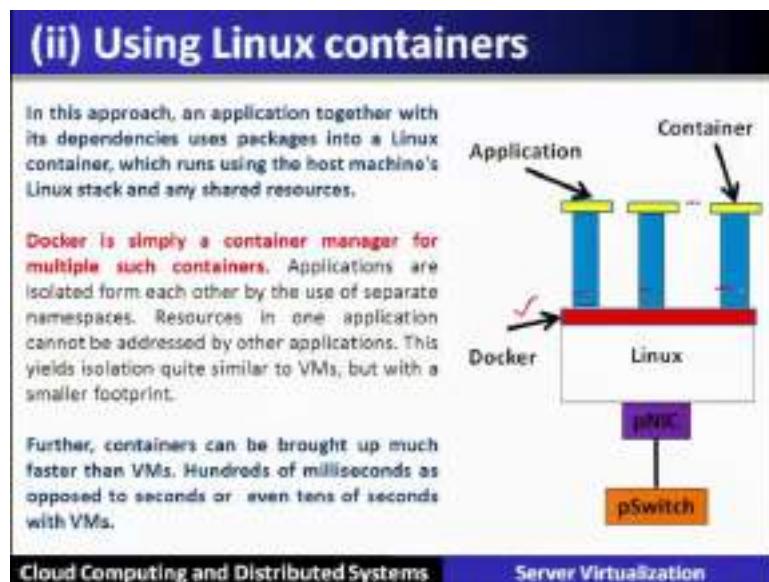
So, you can see here this is the example of a virtualized stack, in the virtualized stack there is a hypervisor and every application with it is own operating system is called virtual machine and there is a special hardware required to support to support the hypervisor to run these virtual machines.

On the other hand you can see the right side this Linux containers, this Linux containers are light weighted they do not have bundled or the containers running their applications requires only the dependencies within it. That is the binaries and the libraries which

required by the application during the runtime and together is called container and this is going to be used using the host operating system.

So, the operating system is basically one operating that is the host operating system will be meant will be supporting all the containers in contrast to every application having the separate operating system.

(Refer Slide Time: 19:24)



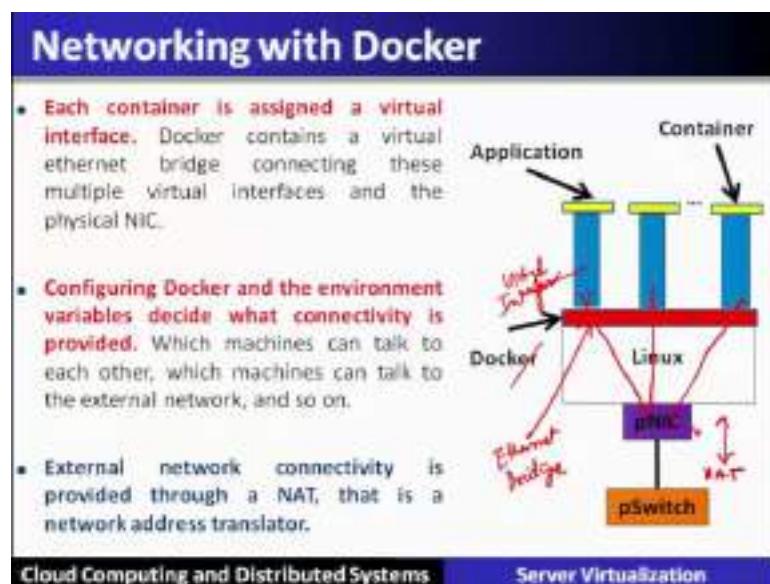
Now, coming to the more detail if when using the Linux container as the virtualization. Now in this approach an application together with its dependencies uses packages into the Linux containers that we have already seen which runs using the host machines Linux stack and any shared resources.

So, the Docker is so that was the container; what is the Docker? So, Docker is simply the container manager. So, it manages all the containers running on that particular server such that it has to provide the applications with the isolation or the separations by way of giving separate namespaces. So, resources in one application cannot be addressed by the other applications why because, they are in a different namespace and which is being given or which is being supported by the Docker.

So, this yields the isolation quite similar to the virtual machines but with a smaller footprint. So, without having the separate operating system yet it is able to provide all support to the containers and that is being managed by the Docker. Further containers

can be brought up much faster than VMs 100 of milliseconds as opposed to the seconds or even 10 of seconds with the virtual machines and that is why the Linux containers and the Dockers are becoming popular way of virtualization.

(Refer Slide Time: 21:11)



Now, comes the networking so each container is assigned about a virtual interface and Docker contains the virtual ethernet bridge. So, Docker contains a ethernet bridge connecting these multiple virtual interfaces, all these virtual interfaces are to be connected to the physical interface using this particular bridge which is been provided by the Docker.

Now, configuring the Docker and the environment variable we will decide what kind of connectivity is going to be provided to the container that is which machine can talk to each other which machine can talk to the external network and so on. So, external network connectivity external network connectivity is provided through a NAT that is called the network address translator.

(Refer Slide Time: 22:40)

Improving networking performance

- The hypervisor runs a virtual switch to able to network the VM's and CPU is doing the work for moving the packets.

Virtual Switch connects VMs

CPU does the work!

VM1 VM2 VM3

vSwitch

hypervisor

NIC

pSwitch

Cloud Computing and Distributed Systems Server Virtualization

Now, let us see the performance of this kind of networking and how it is going to be affected or improved. So, the hypervisor runs a virtual switch that we have already seen, but we will go in the great detail. So, the hypervisor runs a virtual switch which is able to network the VMs, so this is the location of virtual switch which is basically running in the hypervisor and this will give the networking to the virtual NICs or to the VMs while managing the physical NIC. Now, who will be responsible for moving the packets, obviously it will be the CPU which will be responsible to move the packets.

(Refer Slide Time: 24:15)

Packet processing on CPUs

Flexible slow, CPU-expensive: Now packet processing on CPUs can be quite flexible because it can have general purpose forwarding logic. You can have packet filters on arbitrary fields run multiple packet filters if necessary, etc. But if done naively, this can also be **very CPU-expensive and slow**.

$10\text{Gbps} - 84\text{ bytes} - \frac{1}{125\mu\text{s}} \approx 67.4\text{ ns}$

Packet forwarding: The packet forwarding entails: At 10Gbps line rates with the smallest packets, that's 84 Bytes. We only have an interval of 67ns before the next packet comes in on which we need to make a forwarding decision. Note that ethernet frames are 64 bytes, but together with the preamble which tells the receiver that a packet is coming and the necessary gap between packets. The envelope becomes 84 bytes. For context a CPU to memory access takes tens of nanoseconds. So, 67-nanoseconds is really quite small.

Cloud Computing and Distributed Systems Server Virtualization

Here, in this case; so the packet processing will be done here in the CPU. Now CPU will be now busy doing the packet processing now the goal should be that CPU should be relieved out of this packet processing. So that means, very less attention of CPU should be required here in the packet processing, so that CPU should do more of computation job.

So, let us see all these aspects now this particular approach is flexible slow and also CPU expensive. Now the packet processing is done by the CPU it acts the flexibility because it can have the general purpose forwarding logic in place. So that means, you can have the packet filters on arbitrary fields which will run multiple packet filters. If necessary therefore, lot of flexibility as far as packet processing is concerned is done, but with the cost of this flexibility is now born by the CPU expensive operations.

So, hence the CPU will be expensive most of the time CPU we will be doing the packet processing and also it becomes slow. Now another aspect which is called a packet forwarding the packet forwarding entails let us take an example, then we will understand the intricacies of packet forwarding. Now with the ten gigabits per second line rates and the size of the smallest packet let us assume of 84 bytes. Let us see what is the intervals of time to process the packets by the CPU. So if you find out how much time is required to process the 84 bytes packet. So, you will come out to be $(84 * 8) / 10$ that comes out to be this 10 GBPs that comes out to be 67.4 nanosecond. So that means, 64 nanoseconds is required by the CPU to process the packet. Now let us see that what is the time which CPU take for the memory access that also is in the range of 10 of nanoseconds.

So, most of the time the CPU will be busy reading from the memory these particular packets into the buffer and processing it why because reading from the memory that is I O also takes 10 of nanoseconds and packet processing also takes 67 also in the range of nanoseconds. So, hence packet forwarding is going to be looped into it, why because within a very small gap. That means, how the packet is to be processed and forwarded, so more insight is required so that this packet forwarding has to be dealt properly by the CPU.

(Refer Slide Time: 28:22)

Packet processing on CPUs

Need time for Packet I/O: Moving packets from the NIC buffers to the OS buffers, which requires CPU interrupts. Until recently a single X86 core couldn't even saturate a ten gigabits per second link. And this is without any switching required. This was just moving packets from the NIC to the OS. After significant engineering effort, packet I/O is now doable at those line rates. However for a software switch we need more.

Remedy - userspace - kernel - simple cache etc - Smart

Userspace overheads: If any of the switching logic is in userspace, one incurs the overhead for switching between userspace and kernel space.

Packet classification: Further, for switching we need to match rules for forwarding packets to a forwarding table. All of this takes CPU time. Also keep in mind that forwarding packets is not the main goal for the CPU. The CPU is there to be doing useful computation.

Cloud Computing and Distributed Systems Server Virtualization

Now, as I told you that it needs the time for packet I/O that is moving the packets moving from NIC buffer to the operating system also requires CPU interrupt. Since this requires 10 of nanoseconds and with also interrupt adding to it makes this particular the entire packet processing is quite slow. So, the gap between the packets is very small it is very difficult to process the packet at this speed. So, let us see how this particular problem will be solved if the packet processing is to be done by the CPU.

Now, here we can see that most of the tasks if it is done inside the kernel which is not possible, so most of the logic has to be in the user space. So, the user space overhead. So, if any of the switching logic is in the user space it occurs incur the overhead for switching between the user space and the kernel space. So, what is basically the remedy that instead of doing everything within by the CPU, so we have to divide this packet processing into 2 parts one is the user space related processing that is called different policies classification etc and this can be incorporated more advanced using a smart way of doing it.

The second is to be done inside the kernel that is the packet classification, this can be the simple packet lookup using cache and therefore this packet processing can be speed up. So, let us see the packet classification further for switching we need to match the rules for forwarding the packet to the forwarding table, all this takes the CPU time also keep in

mind that the forwarding packets is not the main goal of the CPU. So, CPU has to be there doing this other useful computation.

So, this particular problem will be resolved here in this case this the entire computation of a packet processing is divided into 2 parts some part can be shifted to the user space and the remaining part can be there in the CPU. So, can be there in inside the kernel this way the packet processing can be speed up and this problem can be resolved.

(Refer Slide Time: 31:38)

Approaches for Networking of VMs

There are two different approaches to address the problem of networking VMs:

- (i) One, using specialized hardware:
 - SR-IOV, single-root I/O virtualization
- (ii) Other using an all software approach:
 - Open vSwitch

Cloud Computing and Distributed Systems Server Virtualization

So, there are 2 different approaches to address this problem of networking the virtual machines. As I told you so one is using the specialized hardware which is called SR IOV that is single root I O virtualization and the other part is using the software based approach which is called a open switch. So, these are the networking these are the approaches for networking of VMs.

(Refer Slide Time: 32:08)

(i) Hardware based approach

- The main idea behind the hardware approach is that CPUs are not designed to forward packets, but the NIC is.
- The naive solution would be to just give access to the VMs, to the NIC. But then problems arise. How do you share the NIC's resources? How do you isolate various virtual machines?
- SR-IOV, single-root I/O virtualization, based NIC provides one solution to this problem.

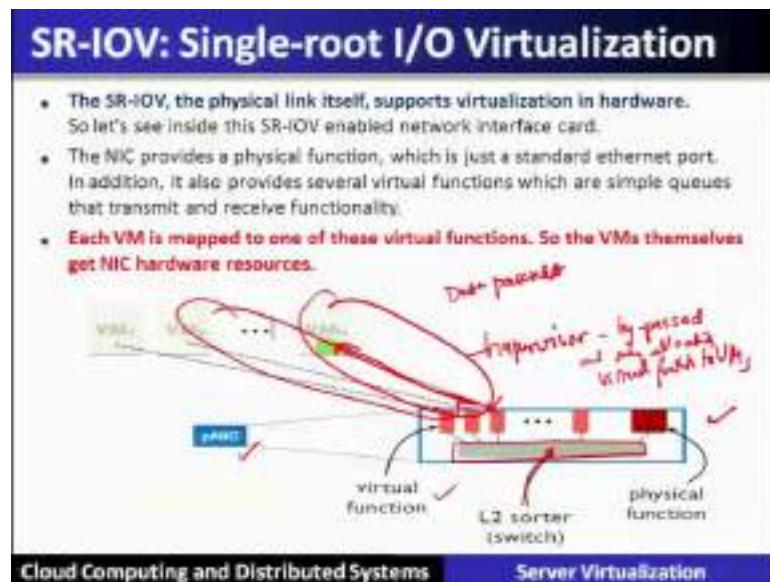
Cloud Computing and Distributed Systems

Server Virtualization

Let us see in more detail these 2 approaches, so the first one is hardware based approach the main idea behind the hardware based approach is that CPUs are not designed to forward the packets. But NICs are designed for it so the naive solution would be to just give the access to the VMs to the physical NICs, but then the problem arise how do you share the NIC resources. How do you isolate various virtual machines different packets.

So, the solution comes out to be in the form of SR IOV enabled NIC cards. So, SR IOV full form is single root IO virtualization based NIC cards are there which will provide this kind of solution, that it will provide the access to the virtual NIC to the physical NICs access to the VMs to the physical NICs and also ensures the isolation and other properties which are required in the hardware based approach.

(Refer Slide Time: 33:18)



Let us see in more detail about this how SR IOV does provide the access of NICs to the virtual machines. So, SR IOV the physical link itself supports the virtualization in the hardware, so let us see inside the SR IOV enabled network interface card. So, let us see in more detail what the physical NIC comprises of. So, physical NIC comprises of a physical function that is the port and it also provides the virtual functions in the form of buffers which are linked to each virtual machines and these virtual functions intern are connected by an L2 based switch which will basically start the incoming packet to these virtual functions which in turn will be delivered to this virtual machines.

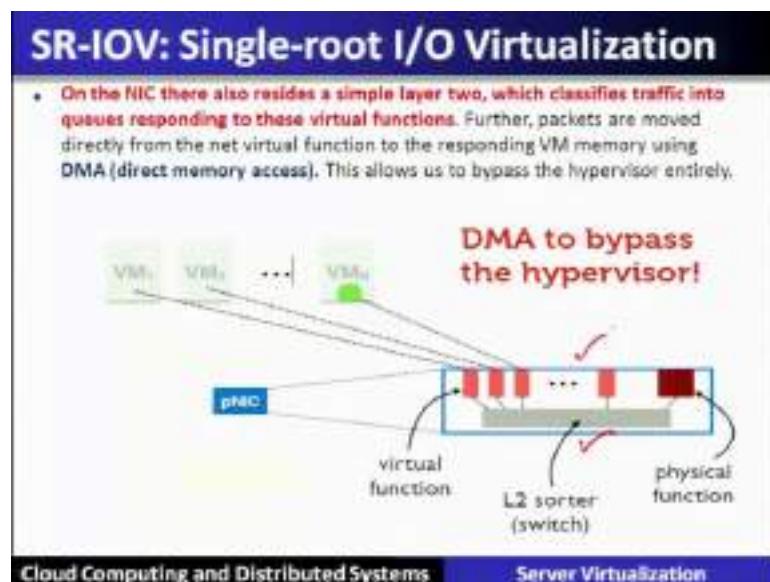
This way this virtual machines can directly be able to access, the network interface cards NICs with the way with the help of virtual functions and hypervisor is bypassed and only allocates the virtual functions to the virtual machines. So, this particular mapping is done by the hypervisor, virtual machine and it is corresponding virtual functions. So, this mapping will be done by the hypervisor and then hypervisor will be bypassed for the data packets. So, the data packets processing will not be done by the hypervisor it can directly be done by the NICs. So, it becomes a bit faster that is at the wire speed.

So, NICs provides the physical function which is just a standard ethernet port, in addition it also provides several virtual functions which are the simple queues that transmit and receive the functionality or these virtual functions are mapped to the virtual machines.

So, whenever the packets are coming and going from the virtual machine they are basically mapped to a; it is corresponding virtual function.

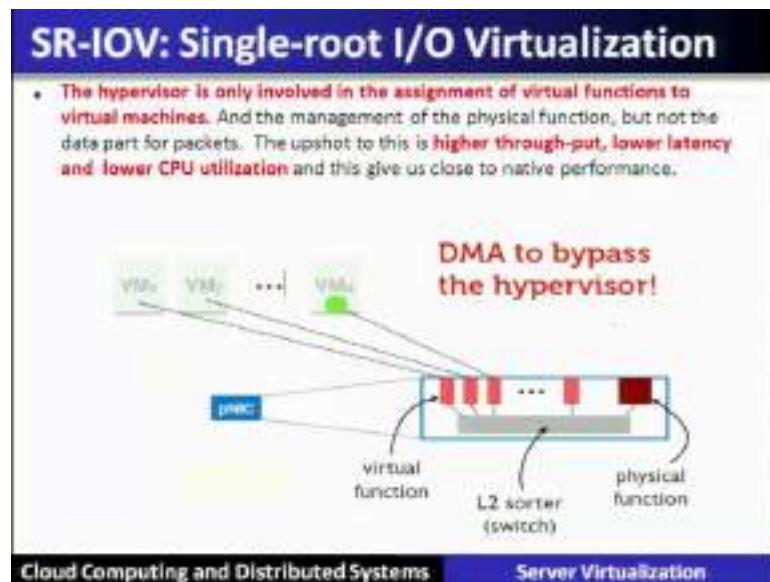
So, each virtual machine is mapped to these virtual functions. So, the virtual machine themselves get the NIC hardware resources directly without the hypervisors intervention. So, hypervisor will only allocate the virtual machines to the virtual functions and then the virtual machine can directly access the physical NICs for packet processing, so you can see that it can work or it can run at the wire speed.

(Refer Slide Time: 36:50)



On this, NICs there resides a simple layer 2 switch, which classifies the traffics into the queues corresponding to these virtual functions. So, this is the L2 switch which sorts the packets to the corresponding buffers and which in turn will be mapped to the virtual machines. Further packets are moving directly from the network virtual functions to the corresponding virtual memory using the direct memory access, so this allows to bypass the hypervisor entirely.

(Refer Slide Time: 37:36)



So, the hypervisor is only involved in assignment of virtual functions to the virtual machines and the management of physical functions not the data part of the packets is done by the hypervisor. So, there data packets are directly dealt with the physical NICs and this physical NICs can ensure with the help of L2 switch and virtual functions which will connect or which will network to the virtual machines. So, this may a higher through put will be achieved and latency can be reduced and also the CPU utilization can be will be lowered. So, CPU utilization will be lowered for packet processing; that means, CPU will be free to do it is own computation using SR-IOV. So, that is why SR-IOV is widely used for networking of VMs.

(Refer Slide Time: 38:46)

SR-IOV: Single-root I/O Virtualization

- There are downsides to this approach though. For one, **live VM migration becomes trickier**, because now you've tied the virtual machine to physical resources on that machine. The forwarding state for that virtual machine now resides in the layer two switch inside the NIC.
- **Second, forwarding is no longer as flexible.** We're relying on a layer two switch that is built into the hardware of the NIC. So we cannot have general purpose rules and we cannot be changing this logic very often. It's built into the hardware. In contrast, software-defined networking (SDN) allows a much more flexible forwarding approach.

The diagram illustrates the SR-IOV architecture. At the top, multiple virtual machines (VM1, VM2, VM3, VM4) are shown. A blue arrow labeled "DMA to bypass the hypervisor!" points from the VMs to a NIC card. The NIC card is divided into four colored segments: red, green, yellow, and blue. Each segment contains a "virtual function". Below the NIC card, a "physical function" is shown. An "L2 sorter (switch)" is positioned between the NIC and the physical function. Arrows indicate data flow from the VMs through the NIC to the physical function, with a specific path highlighted in red. The bottom of the slide features two horizontal bars: "Cloud Computing and Distributed Systems" on the left and "Server Virtualization" on the right.

Now, there are downsides to this approach that is when it comes to the live VM migration it becomes trickier, why because all the VMs are now tied up to the physical NICs through the virtual functions and layer 2 switch of the physical NICs, so when they are tied up then the live migration.

That means, migrating from one server to another server will become a problematic, now the question is how this networking or VMs is to be ensured so that it can detach from the physical NICs, so that the live VM migration becomes easier. So, the forwarding state of that virtual machine now resides in the layer 2 switch inside NICs and therefore live VM migration becomes problematic. Second thing is that forwarding is no longer as flexible why because, the forwarding is relying on layer 2 switch that is built into the hardware of NIC, which does not have much smart logic to be in to be dynamically changing.

So, we cannot have the general purpose rules and we cannot be changing this logic very often and do dynamically, why because it is built into the hardware therefore, to elevate from these 2 problems a software defined networking allows a much flexible forwarding approach that we will see later on.

(Refer Slide Time: 40:38)

(ii) Software based approach

The software based approach that addresses;

(i) Much more flexible forwarding approach

(ii) Live VM migration

Cloud Computing and Distributed Systems

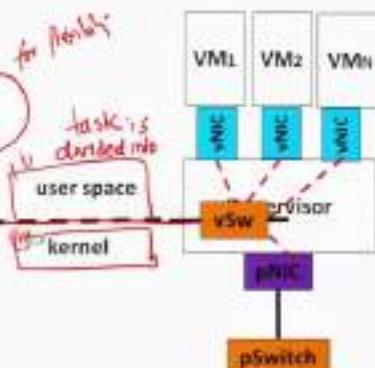
Server Virtualization

Now another approach which can solve this particular problem these 2 problems; that means, to provide more flexible forwarding approach and also to enable the live migration we will see the software based approach for this particular problem.

(Refer Slide Time: 40:58)

Open vSwitch

- Open vSwitch design goals are flexible and fast-forwarding.
- This necessitates a division between user space and kernel space task. One can not work entirely in the kernel, because of development difficulties. It's hard to push changes to kernel level code, and it's desirable to keep logic that resides in the kernel as simple as possible.



Cloud Computing and Distributed Systems

Server Virtualization

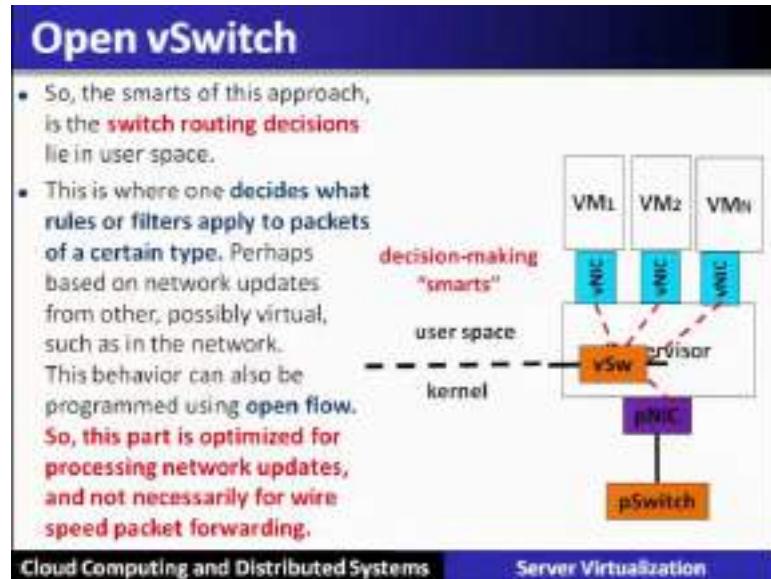
Open vSwitch; open vSwitch design issues are flexible and fast forwarding packet processing. This necessitates a division between the user space and the kernel space task as we have seen in the previous slides, that is we have already built the background to understand this particular division of user space and the kernel space tasks. Because one

cannot work entirely in the kernel space if you want to get the flexibility you have to come out from the kernel space and those flexible rules and filters can be implemented in the user space.

So, if at all if you want to provide the flexibility for that sake such tasks are to be shifted which is one of the goals. So, some of the tasks will be shifted to the user space and the kernel will be relieved with the simple task. So, that the packets forwarding can be speed up now let us see how this particular division between user space and kernel space of this packet forwarding task we have to do.

So, in this particular example here we can see that the open vSwitch, activities are now divided into the user kernel space and user space and the kernel space. So that means, that task is divided into user space and the kernel space. You will see more details what task goes into the user space and what task of virtual switch will fall into the kernel space by putting some of the task in to the user space is to ensure the flexibility. Whereas the relieving those operations from the kernel will increase the fast processing or the fast forwarding, so both these goals is going to be achieved by this particular division of work.

(Refer Slide Time: 43:53)

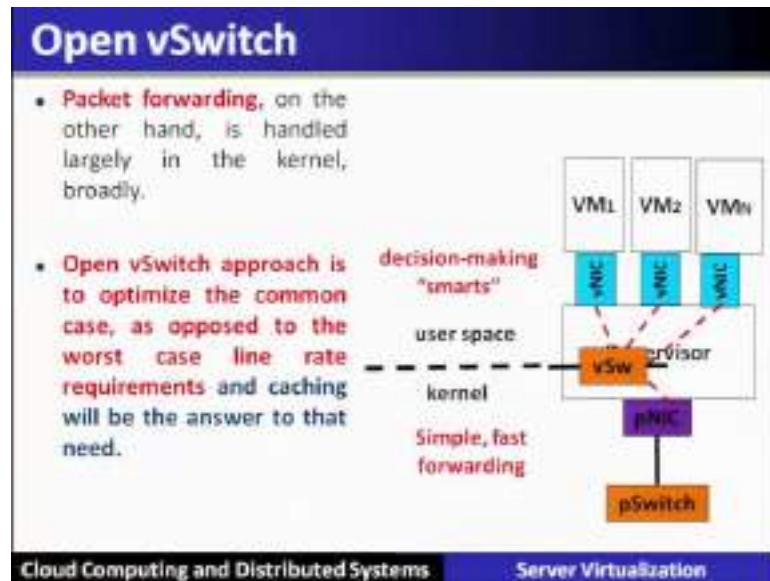


So, the smarts of this approach is the switch routing decisions that will lie into the user space, this is where one decide what rules or the filters applied to the packet of a certain type; perhaps based on the network updates from the other such vSwitch information

which gathers this information and propagates among all other vSwitch. So, that they may learn and may update the rules and the filters, so that it will have the smart decision making of packet processing.

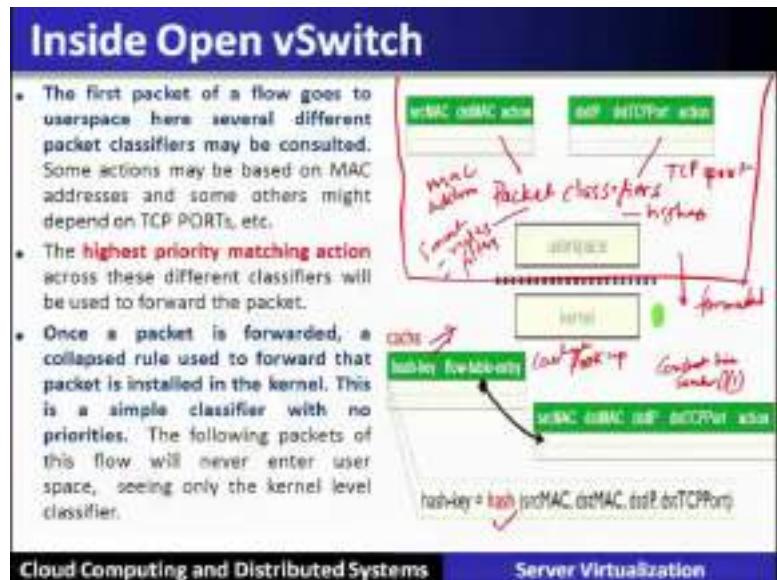
So, this behavior can also be programmed using open flow. So, this part is optimized for processing the network updates and not necessarily for wire speed packet forwarding that is to be taken care by the kernel.

(Refer Slide Time: 44:54)



So, the packet forwarding is handled largely into the kernel, so let us see what the kernel will be doing. So, open vSwitch approach is to optimize the common case as opposed to the worst case line rate requirement. So, basically the alternative for the first packet forwarding is to be done using caching.

(Refer Slide Time: 45:28)



So, let us see the entire division of the task between user space and the kernel space. So, we have only drawn the user space and the kernel space. So, the first packet flow goes to the user space here the several different packet classifiers are consulted. So, some action will be taken based on the MAC address and some maybe depending upon the TCP port address. So, these classifiers will classify the packet based on these type of addresses and the highest priority matching action across this classifier will be used to forward the packets.

So, here all the smart logic in terms of rules and filters are applied here in the different classifier which is residing in basically the user space. Having done this classification once the packet is forwarded this packet will be forwarded that is installed in the kernel. So, this is a simple classification with no priorities the following packets of this flow will never enter the user space seeing only the kernel level classifier. In the kernel level classifier this particular lookup will not be consulted through a big tables, but rather a cache based lookup entry will be there based on the hash key.

So, the hashing is basically the constant time search operation or a lookup of the order one can be implemented into the inside the kernel and once it is entered into the table that is in into the cache or a hash table then further accessing will be very fast that is of the order one to those particular packets. So, this way the packet processing into the kernel becomes the cache based hashing scheme which becomes very fast in contrast to

the previous scheme, where the CPU used to consult many tables and the search becomes inefficient and the packet processing takes lot of time.

(Refer Slide Time: 49:11)

Inside Open vSwitch

- The problem though is when it's still running a packet classifier in the kernel in software. What this means is for **every packet that comes in, you are searching in this table for the right entry that matters and using that entry for forward the packet. This can be quite slow.**
- Open vSwitch** solves this problem is to create a simple hash table based cache into the classifier. So instead of looking at this entire table and finding the right rule. You'll hash what fields are used to match packet, and the hash key is now your pointer to the action that needs to be taken. And, these hash keys and their actions can be cache.

Cloud Computing and Distributed Systems Server Virtualization

So, the problem tough is when it is still running the packet classifier in the kernel, what this means is that for every packet that comes in you are searching the table for the entry that matters and using that particular entry for forwarding the packet. So, that has gone now, so in open vSwitch is to create a cache based simple hash table classifier, so that means here the entire table is not consulted.

(Refer Slide Time: 49:50)

Introduction to Mininet

- Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command:

- It is a **network emulator** which creates realistic virtual network
 - Runs real kernel, switch and application code on a single machine
 - Provides both Command Line Interface (CLI) and Application Programming Interface (API)
 - CLI: interactive commanding
 - API: automation
 - Abstraction**
 - Host: emulated as an OS level process
 - Switch: emulated by using software-based switch
 - E.g., Open vSwitch, SoftSwitch

Cloud Computing and Distributed Systems Server Virtualization

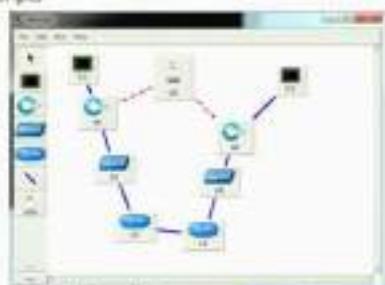
Now, the next thing which we are now going to discuss is a Mininet. So, Mininet is a emulator for the virtual network, so Mininet creates a realistic virtual network running kernel switch and application code on a single machine and that too in a very efficient manner. So, that is why Mininet is used for experimentation and for the learning also, so if you are interested to implement these concepts that is virtual machine networking or a networking of a virtual machine using emulation then mininet is a useful tool.

So, Mininet is a network emulator which creates a realistic virtual network. So, you can create your own network virtual network topology using a simple command which is called mn. So, when you run an mn it will create the network emulator for the virtual network. Now this mininet provides you the command line interface and application programming APIs using that you can create your own virtual network topologies. So, here the host is emulated as the operating system level process and also you can implement various switches, which is emulated using software based switch that is you can also implement open vSwitch and soft switches.

(Refer Slide Time: 51:57)

Mininet Applications

- **MiniEdit (Mininet GUI)**
 - A GUI application which eases the Mininet topology generation
 - Either save the topology or export as a Mininet python script
- **Visual Network Description (VND)**
 - A GUI tool which allows automate creation of Mininet and OpenFlow controller scripts



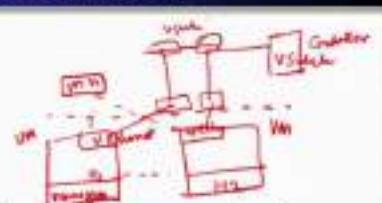
Cloud Computing and Distributed Systems Server Virtualization

So, this is the visual description you can see using tool you can create your own topology in Mininet.

(Refer Slide Time: 52:08)

Important Links for Mininet

- mininet.org
- github.com/mininet
- github.com/mininet/mininet/wiki/Documentation
- reproducingnetworkresearch.wordpress.com



Cloud Computing and Distributed Systems Server Virtualization

So, Mininet provides you the way you can create virtual networks you can create hosts, you can attach let us see through the diagram. So, the moment you say Mininet you launch you can create the virtual machines with the namespace, let us say this is H1 and you can create another virtual machine with let us say the name H2.

This will be having the ethernet virtual ethernet port and here there will be a physical ethernet which you can connect in the Mininet and intern you can connect to the scenario for control purpose and you can create a vSwitch which can basically create the network topology and the packet processing will be done through those vSwitch. So, all this features are provided in Mininet, so these are the links you can see the details and you can do the experimentation whatever we are discussing here in the server virtualization.

(Refer Slide Time: 54:11)

Comparison

- The hardware based approach the SR-IOV takes **sacrifices flexibility and forwarding logic** for line rate performance in all scenarios. Virtually hitting native performance.
- While the software based approach Open vSwitch, the compromise made is to avoid targeting worst case performance, and **focusing on forwarding flexibility**.

Cloud Computing and Distributed Systems

Server Virtualization

Now, if we compare these 2 schemes hardware based and software based networking schemes which we have seen SR-IOV and open vSwitch. So, SR-IOV is giving a line rate performance, which is nothing but it is touching up the native performance of a packet forwarding, but it sacrifices the flexibility and also the forwarding logic where as the software based approach that is called vSwitch gives the flexibility and also ensures the fast packet forwarding.

(Refer Slide Time: 55:05)

References

- W. Feitler, A. Ferreira, R. Rajamony and J. Rubio, "**An updated performance comparison of virtual machines and Linux containers**," 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 171-172.
- Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martin Casado, "**The design and implementation of open vSwitch**" In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15). USENIX Association, Berkeley, CA, USA, 2015, pp. 117-130.

Cloud Computing and Distributed Systems

Server Virtualization

These are the references which we are used in this part of the lecture conclusion.

(Refer Slide Time: 55:14)

Conclusion

- In this lecture, we have discussed **server virtualization** and **also discuss the need of routing and switching for physical and virtual machines.**
- We have discussed two methods of virtualization:
(i) Docker based and (ii) Linux container based.
- To address the problem of networking VMs, two approaches are discussed: **(i) One, using specialized hardware: SR-IOV, single-root I/O virtualization and (ii) Other using an all software approach: Open vSwitch**

Cloud Computing and Distributed Systems

Server Virtualization

We have covered the server virtualization and we have also covered how the networking of the virtual machines are done, we have also covered 2 different type of virtualization that is Docker based and Linux container based and for that we have also covered the networking virtualization that is through SR-IOV and open vSwitch.

Thank you.

Cloud Computing and Distributed systems
Prof. Rajiv Mishra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

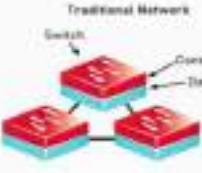
Lecture – 05
Software Defined Network Application to Networking in the Cloud

(Refer Slide Time: 00:18)

Preface

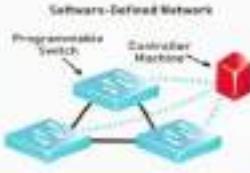
Content of this Lecture:

- In this lecture, we will discuss the architecture of software defined networking and its applications to networking in the cloud.
- We will also discuss the network Virtualization in multi-tenant data centers with case study of VL2 and NVP



Traditional Network

Switch
Control Plane
Data Plane



Software-Defined Network

Programmable Switch
Controller Machine

Cloud Computing and Distributed Systems Software Defined Network

Software Defined Network Application to Networking in the Cloud. Preface, content of this lecture, we will discuss the architecture of software-defined networking, and its application to the networking in the cloud. We will also discuss network virtualization in multi-tenant datacenters with two different case studies of VL2 and NVP.

(Refer Slide Time: 00:42)

Need of SDN

The traditional networking problem that SDN (software defined networking) is addressing as:

I) Complexity of existing networks

- Networks are complex just like computer system, having system with software.
- But worst than that it's a distributed system
- Even more worse: No clear programming APIs, only "knobs and dials" to control certain functions.

II) Network equipment traditionally is proprietary

- Integrated solutions (operating systems, software, configuration, protocol implementation , hardware) from major vendors

RESULT: - Hard and time intensive to innovate new kinds of networks and new services or modify the traditional networks more efficiently.

Cloud Computing and Distributed Systems Software Defined Network

The need of software-defined network requires to understand complexity and difficulty of traditional networks. And therefore, it becomes the motivation to understand the software-defined network. So, software-defined network in contrast to the traditional networks will overcome these difficulties in the traditional networking problems, such as complexity of existing networks.

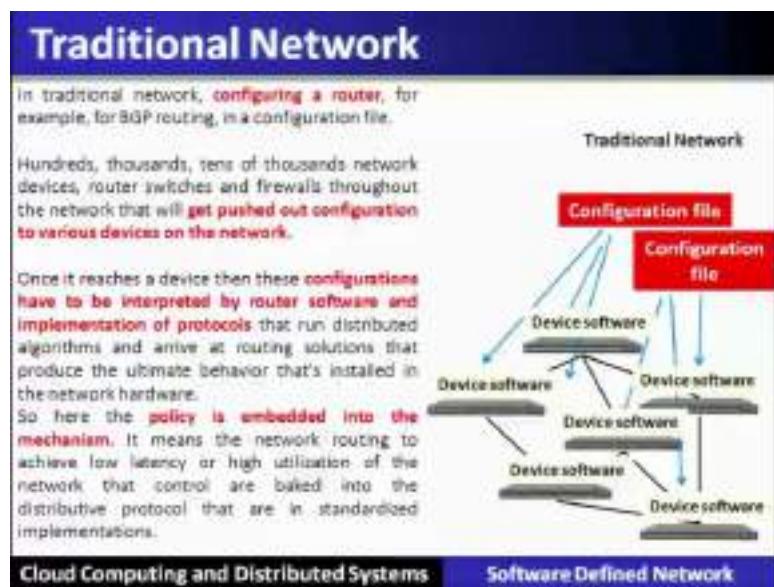
So, the current networks are quite complex; in the sense, it comprises of several 100 networking devices, which are having the embedded software within it, and the supplier has already configured the protocols. So, all the hardware and the software of the network devices are provided by the vendors, they are all proprietary in nature, worst than this it becomes a complete distributed system, wherein the traffic flow and other traffic and other policy decisions over the network regarding the traffic engineering and all these are done in the form of distributed algorithms.

And some aspects are being configured by the vendors. Even worst is that, there is no programming APIs available, only very few control functions are configured in such a network. So, in nut shell, this particular network is very very complex. And if a particular user want the network to be designed regarding the throughput and latency and all these aspects, it is very difficult to program such a so, complexity of the network is known to everyone in the traditional networking.

Now, another aspect in the traditional networks is that network equipments are traditionally proprietary in nature that means, the vendors provides the integrated solution in terms of operating system also is a part of that networking devices, software also is a part of networking devices, the configuration, the protocol implementation, hardware, all are integrated, and that comes from the major vendors. So, therefore, it is very little scope to understand and do the programming of these particular network devices.

Now, why it is required? It is required, because it is the time, when different users like for example, the cloud requires to innovate new kind of rules and methods to configure the network and run the services as per the requirements. But, such a traditional network is very difficult, due to these two problems to modify and give the programming control at the user level.

(Refer Slide Time: 04:28)

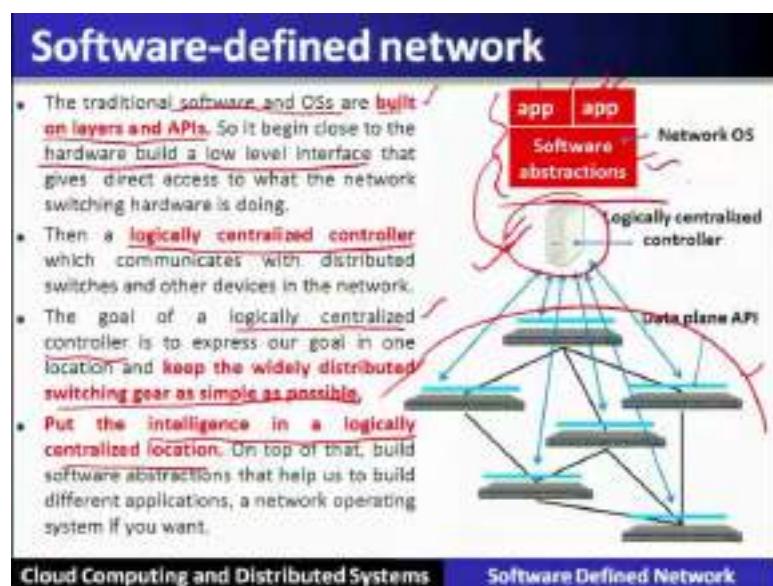


Let us see this difficulty of the traditional network in this scenario here. In the traditional network you can see here, the configuration of a router. For example, if the routers are configured for external routing, that is called BGP routing, and that is done in the configuration files. So, these particular router configured files are then send to or pushed out to the various devices switches. And once it reaches to these devices, then basically the router software and its protocol will run the distributed algorithm based on

that configuration to arrive into a routing solutions to basically produce the ultimate behavior that is installed in the network hardware.

Therefore, you can see in such traditional networks, the policies and the mechanisms both are embedded into the devices, and the distributed algorithm runs to deal with the traffic flow. Now, if you want to achieve the low latency and high throughput in this scenario, the controls are not given in the form of programming level, but it is already embedded into the distributed protocol, which is basically a very standard implementation, very hard to program and change as far as the requirements of applications are therefore.

(Refer Slide Time: 06:15)



To get this particular flexibility, there comes a rescue in the terms of software-defined networks. So, software-defined networks will deal with these complexities and these problems, so that it can be useful or usable in a different environment that is in the cloud scenario.

Now, let us see: what is the software-defined network, and how it does overcome from the problem of traditional networks. So, analogy is to be taken from the traditional software. So, the traditional software and the operating systems if we see how they are built, they are built on the layers and through the APIs, the programmers can program them that is the software and the operating systems are built in the form of systematically in the form of the layers and APIs are there. Therefore, this particular environment that is

the network environment can we do as per the traditional software and operating system inside this network devices.

If it is then, we are going to see the software-defined network provides this particular separation of policies and mechanisms separated it out in the terms of this hardware of that particular network devices can be accessed in the form of low level interface. So, directly it can be accessed as far as the policies are concerned, they are separated out from this particular logic. Therefore, in software-defined network, there exist a logically centralized controller, which is shown over here, which communicates with the distributed switches and other devices. And all the smart logic is embedded here in the centralized controller. So, as far as the user is concerned it can express all its policies centrally through this logically centralized controller.

On the other hand, the switches and the other networking devices becomes very simple, because all the decisions are basically the policy decisions and its programming is now taken out from them, and it is basically the pure hardware with the data plane APIs being provided. So, the switching gear is made as simple as possible. So, any vendor can provide its hardware and the logical centralized controller will embed with the software within it. These two separations are there in software-defined network.

So, all intelligence is put in the centralized location. And on top of it is built the software abstractions over which different applications can be made. So, the programming of this logically centralized controller is managed just like the softwares, which we have seen that it is in the form of layers and APIs, which can be programmed as per the need of the organization to run the networks.

So, you can see here, there is the logically centralized controller, and these particular devices are simply the hardware, which provides the data plane APIs, and it gets the instruction from logically centralized controller and work accordingly. So, now the control is all the policies are with the logically centralized controller, which can be programmed using these two layers that is in the form of software abstractions and different applications can run on top of it, which can use this network the way they want to. This software abstraction can be the network operating system and different applications can be programmed as per the requirement of the network. So, this is the

brief overview of a software-defined network. Let us see how this software-defined network will be useful for cloud networking in this discussion.

(Refer Slide Time: 11:17)

Key Ideas of SDN

Key ideas software-defined networking architecture:

Division of Policy and Mechanisms-

- Low-level interface and programmatic interface **for the data plane**
- **Logically centralized controller** that allows us to build software abstractions on top of it.

Cloud Computing and Distributed Systems Software Defined Network

So, again let us summarize the same thing, key idea of a software-defined networking architecture is to divide the policy and mechanism. So, that the low level interface and programmatic interface is provided for the data plane, and the policies are controlled in the form of logically centralized controller that allows to build software abstractions and users can program it.

Now, in the diagram, we have shown only one logically centralized controller, it is not a centralized controller; it is a logically centralized controller. In the sense, there may be more than one controller or the controllers are also run in a form of a distributed system, they may communicate with each other, they may be synchronizing their clocks with each other. And together they may solve or may communicate with each other to enforce the mechanisms and the policies given or programmed, so that all the network devices can work accordingly that means, all the devices can be programmed using the data plane APIs.

(Refer Slide Time: 12:35)

Example: NOX

NOX is a very early SDN controller. So to identify a user's traffic, a particular user or computer to send traffic through the network, and that traffic through the network is going to be tagged with an identifier, a VLAN and that identifies that user.

So to instruct the network we match a specific set of relevant packets and look at the location where this traffic comes in from as well as the MAC address. And we're going to construct an action that should happen, in this case, tagging, adding that VLAN tag to the traffic.

And then we're going to install that action on the specified set of packets in a particular switch. So we're basically telling the switch, if you see this, do that.

In addition, commonly SDN controllers have some kind of topology discovery, the ability to control traffic and monitor the behavior in the network.

The screenshot shows the NOX controller's graphical user interface. On the left, a 'Flow Table' window displays a list of flows with columns for 'Priority', 'Match', 'Actions', and 'Statistics'. A red circle highlights the 'Actions' column for the first flow. On the right, a 'Configuration' window shows a tree structure for defining actions based on match criteria like 'in_port' and 'eth_src'. A red circle highlights the 'actions' section under 'Matched Actions'.

Cloud Computing and Distributed Systems Software Defined Network

Let us see an example of the very earlier system of SDN controller, which is called a NOX. The example, which we will give is that how the NOX controller will identify a particular users traffic, and then enforce some action on it, in traditional network that is not possible to identify a particular users traffic and take action on it. Let us see how software-defined network in controller that is NOX controller does that. So, to identify a users traffic, a particular that is a particular user or the computer to send the traffic through the network, then that particular traffic is tag with an identifier in a virtual LAN, that will identify the users.

Now, the other aspect is to instruct the network that once this particular traffic comes out from that particular user, so that means, there is a match on a particular set of packets originated from that tagged user based on the MAC addresses and some other identifications. Then what action has to be taken up. So, the actions are also installed, which will specify what to do with those particular packet, when it arrives a particular switch, whether it is to be dropped or it is to be forwarded all these aspects can be programmed here in this particular scenario. Therefore, using SDN controller, many things can be done for example, monitoring the behavior of the entire network and ability to control some traffic are also topology discovery can be done through this particular way of software-defined networks.

(Refer Slide Time: 14:56)

Key Ideas of SDN

- A programmatic low level interface with the data plane
- Centralized control
- Higher level abstractions that makes easier control.

Cloud Computing and Distributed Systems

Software Defined Network

So, again the key idea is that programmatic low level interface is being provided with the data plane, so that the hardware switches can be used as APIs, so any vendor can pick in providing their hardwares in the form of switches, routers and so on. The entire control is done in a centralized manner that is the logically centralized controller existed. And to program this, there is a high level abstractions, which are provided just like a software, which can be programmed using the simple languages like python. So, the entire network can be programmed as per the requirement.

(Refer Slide Time: 15:47)

Evolution of SDN: Flexible Data Planes

- Evolution of SDN is driving towards making the network flexible.
- Label switching or MPLS (1997) i.e. matching labels, executing actions based on those labels adding flexibility:-
- Lay down any path that we want in the network for certain classes of traffic. — Traffic Engineering
- Go beyond simple shortest path forwarding, —
- Good optimization of traffic flow to get high throughput for traffic engineering, —
- Setting up private connections between enterprises and distributed sites.



Cloud Computing and Distributed Systems

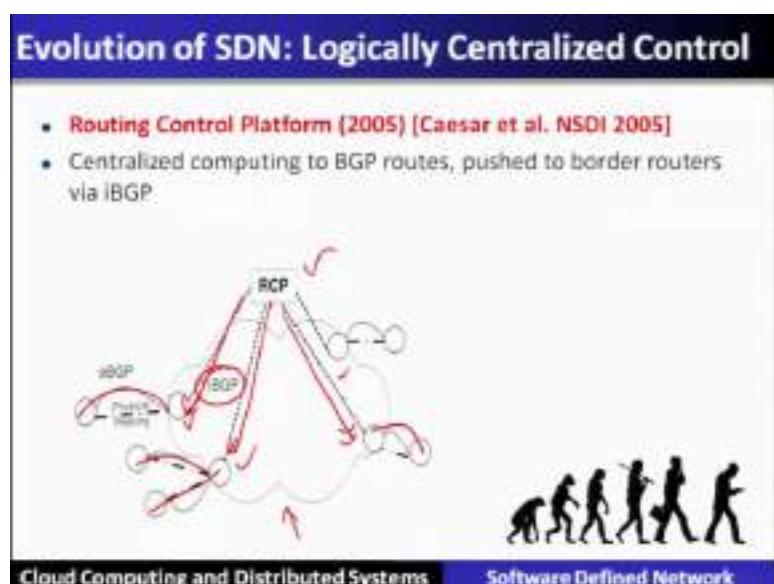
Software Defined Network

Let us see the evolution of software-defined network, which will provide the flexible data planes. So, the evolution of software-defined network, as we have just seen is to provide the flexibility in the network operations, so that different users can program as per their requirement the entire network, and will overcome from that proprietary in nature. So, the evolution that means, let us trace through the evolutions, which has reached to this particular stage that is software-defined network.

The first one in terms of the idea was called MPLS multiprotocol label switching in 1997, what it does it, matches with the labels, the traffic, which can be used for example, to setup a virtual private network, and then perform various actions on such particular traffic. So, this way one can lay down any path in the network to classify that particular certain traffic class, and do the traffic engineering on it.

So, MPLS provides these particular features. So, it is going beyond the shortest path algorithm, because here lot of traffic engineering can be performed. Similarly, lot of optimizations on a network flow can be done, so that there is a possibility of achieving the throughput. And also this is used to setup virtual private network connections across the different enterprises, and this is all possible using MPLS. So, MPLS has started giving this kind of provisions, and this is by start point of evolving the software-defined network.

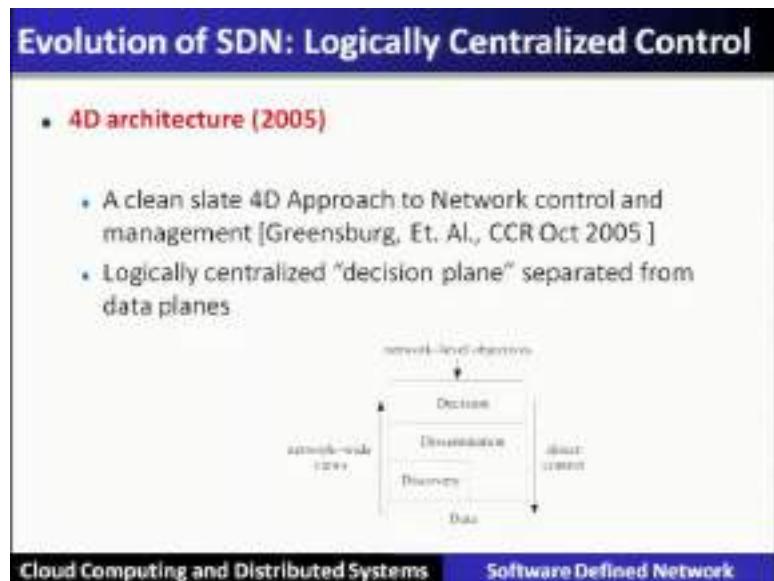
(Refer Slide Time: 18:10)



The next one in this evolution is called routing control platform in 2005. What it provides is that, within an organization also there are many number of border gateway protocols, there are various routers installed with BGP protocol to interact with the outside world. Normally in a routing control platform, that means, at the centralized control, the policies are being pushed to these border routers via iBGP. So, this iBGP intern will propagate to the other BGPs, which are installed externally or which are connected to the other BGP.

So, here we can see that the explorer traffic, that means, when the organization is communicating its messages with the other networks, that it can be programmed through the central control, through the centralized computing for computing the BGP routes and following the routing information. So, this is also the aspect, which is required here in the evolution of software-defined network.

(Refer Slide Time: 19:56)



Another, architecture is 4D architecture of 2005. Now, here it gives a approach to the network control and management. So, it also defines logically centralized decision plane separated from data plane. These ideas are also there in software-defined network.

(Refer Slide Time: 20:16)

Evolution of SDN: Logically Centralized Control

- Ethane (2007) [Casado et al., SIGCOMM 2007]
 - Centralized controller enforces enterprise network Ethernet forwarding policy using existing hardware.

Cloud Computing and Distributed Systems Software Defined Network

Ethane is just a precaution to the SDN; ethane was given in 2007. It is a centralized controller, which enforces the enterprise network Ethernet forwarding policy using existing hardware. So, again it is separating out policies and mechanisms.

(Refer Slide Time: 20:36)

Evolution of SDN: Logically Centralized Control

- OpenFlow (2008) [McKeown et al. 2008]
 - Thin standardized interface to data planes.
 - General purpose programmability at control.

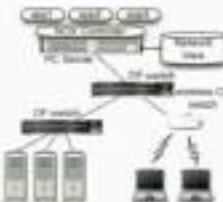
Cloud Computing and Distributed Systems Software Defined Network

Now, comes after that is open flow in 2008. This is a thin standardized interface to the data planes; and it also provides a general programmability at the control level.

(Refer Slide Time: 20:49)

Evolution of SDN: Logically Centralized Control

- Routing Control Platform (2005)
- 4D architecture (2005)
- Ethane (2007)
- OpenFlow (2008)
- **NOX (2008) [Gude et al. CCR 2008]**
 - First OpenFlow (OF) controller: centralized network view provided to multiple control applications as a database.
 - Handles state collection and distribution.
- **Industry explosion (~2010+)**



Cloud Computing and Distributed Systems Software Defined Network

Now, the NOX we have already seen uses the first open flow controller, which is having the centralized view, which is provided to the multiple control applications just like a database it handles. So, there is a central controller, which will have the complete view or it can be programmed as a database does. And what it does it collects by state of the entire network, stores in the database, do the analysis, and provides control back to those particular devices. And after that almost every networking industry has adopted the software-defined network in one form or the other.

(Refer Slide Time: 21:43)

SDN Opportunities

- **Open data plane interface**
 - **Hardware**: with standardized API, easier for operators to change hardware, and for vendors to enter market
 - **Software**: can more directly access device behavior
- **Centralized controller**:
 - Direct programmatic control of network
- **Software abstraction of the controller**
 - Solves distributed systems problem only once, then just write algorithm.
 - Libraries/languages to help programmers write net apps
 - Systems to write high level policies instead of programming

Cloud Computing and Distributed Systems Software Defined Network

What are the options or opportunities, which software-defined networking is giving us to the networking of the organizations and the cloud as well as. Open data plane interface provides the separation of hardware and software. In the hardware, with standardized APIs, if it is provided, then basically it is easier for the operators to change the hardware, and it is not proprietary hardware, and different vendors can enter into the hardware market. As far the software is concerned software can be programmed to directly control the behavior of such devices. For that, the policies are given from the centralized controller, which also is having a direct programmatic control of the network.

Software abstractions are also supported or provided for the input to the controller. This way, it is all (Refer Time: 23:01) distributed problem only once and then those policies are now generated in the form of distributed algorithm and being pushed to those hardwares, which will use its software to run the mechanisms. For that controls, the libraries, and languages, are also there, and it can be programmed to write down the network operations or applications.

So, here a very high level policies can be implemented in the form of the network controller, and the organizations through the simple programming language, so this way the SDN has given lot of opportunities to program and control the networks the way is required in the applications or in the organization.

(Refer Slide Time: 24:06)

Challenges of SDN

Performance and Scalability

- Controlling the network through devices to respond quickly with latency concerns i.e. capacity concerns.

Distributed systems challenges still present

- Network is fundamentally a distributed system
- Resilience of logically centralized controllers
- Imperfect knowledge of network state
- Consistency issues between controllers

Cloud Computing and Distributed Systems SoftwareDefined Network

Now, another challenge of software-defined network is the performance and scalability. So, the controlling such a network through the devices to respond quickly with the latency concerned, and also with the capacity concerned deals are related with the performance and scalability.

So, network is a distributed system. And resilience of logically centralized controller is very much required. So, imperfect knowledge of network system states due to the distributed system is already there, and lot of consistency related issues between the controllers are also there. For example, it is not a centralized one controller; it is a logically centralized controllers several controllers are there. So, all these challenges are still present in the software-defined network.

(Refer Slide Time: 25:14)

Architectural Challenges of SDN

- **Protocol to program the data planes**
 - OpenFlow ? NFV function ? WhiteBox switching ?
- **Devising the right control abstraction ?**
 - Programming OpenFlow : far too low level
 - But what are the right level abstractions to cover important use cases ?

Cloud Computing and Distributed Systems Software Defined Network

Let us see some more architectural challenges of software-defined network, how to program the data planes. So, that it can run different protocols, so different mechanisms are there, whether open flow should be used or network function virtualization, NFV can be used or white box switching is there. There are various different alternatives.

And this particular space is open for the research between (Refer Time: 25:47) and the industry. That is why, the devising the ride control abstraction in the form of whether the open flow is already very low level way of solving this particular problem. Instead of that, what kind of abstraction to cover the most important use cases is going to be important development for the future applications?

(Refer Slide Time: 26:20)

The First Cloud Apps for SDN

- **Virtualization of multi-tenant data centers**
 - Create separate virtual network for tenants
 - Allow flexible placement and movements of VMs
- **Inter-datacenter traffic engineering**
 - Trying to achieve maximum utilization near 100% if possible.
 - Protect critical traffic from congestion.
- **Key-characteristics for above use cases**
 - Special purpose deployments with less diverse hardware.
 - Existing solutions aren't just inconvenient and don't work.

Cloud Computing and Distributed Systems Software Defined Network

First cloud application for software-defined network. We will see here, the virtualization of multi-tenant datacenters, which is an application of software-defined network into the cloud system. Here we will see that how to create separate virtual functions for tenants, and allow the flexible placement and movements of virtual machines. So, to understand this, in a cloud datacenters, there are many tenants, which are running their VMs. And there is a requirement of the isolation separation among their, the traffic and also the different VMs not shared, but it has to be secured.

So, the virtualization of multi-tenant data center is going to be an important issue. And that is how using software-defined network this particular problem is going to be addressed, that we will see. We will also see the how inter data center traffic engineering can be used. So, that there should be a maximum utilization of nearly 100 percent to achieve the profit of running the data center or the cloud services for the public or maybe for the internal purpose. So, in that scenario, we will see how to protect the traffic, how to monitor the traffic and how to protect it from reaching into a state, which is called a congestion.

We will also see some of the key characteristics for different use cases. For example, this special purpose deployment with a less diverse hardware sometimes is required. Similarly, another use cases existing solutions are not just inconvenient and do not work.

So, how special use cases are going to be addressed in this particular way of software defined networks.

(Refer Slide Time: 28:40)

Multi-tenant Data Centers : The challenges

Cloud is shared among multiple parties and gives economy of scale. To share the cloud among multiple tenants, there's bit more work to do. So the key needs for building a multi-tenant Cloud data center are:

- (i) Agility**
- (ii) Location independent addressing**
- (iii) Performance uniformity**
- (iv) Security**
- (v) Network semantics**

Cloud Computing and Distributed Systems

Software Defined Network

So, multi-tenant data center, let us see the challenges. So, cloud is shared among multiple parties to give the economy of scale that means, the more are the parties, which are sharing the data center, the profit runs according to that scale, that is called economy of scale in the cloud. Now, to share the cloud among the different multiple tenants, lot of work has to be done start (Refer Time: 29:11) forward.

So, the key needs for building multi-tenant cloud data center are, it has to ensure or provide these five different requirements or we can say that the multi-tenant data center has five different challenges; let us see one by one, agility, location independent addressing, performance uniformity, security and network semantics.

(Refer Slide Time: 29:43)

(i) Agility

- **Use any server for any service at any time:**
 - Better economy of scale through increased utilization: Pack compute as best we can for high utilization. If we ever have constraints then it's going to be a lot harder to make full use of resources.
 - Improved reliability: If there is a planned outage or an unexpected outage, move the services to keep running uninterrupted.
- **Service or tenant can means:**
 - A customer renting space in a public cloud
 - Application or service in a private cloud as an internal customer

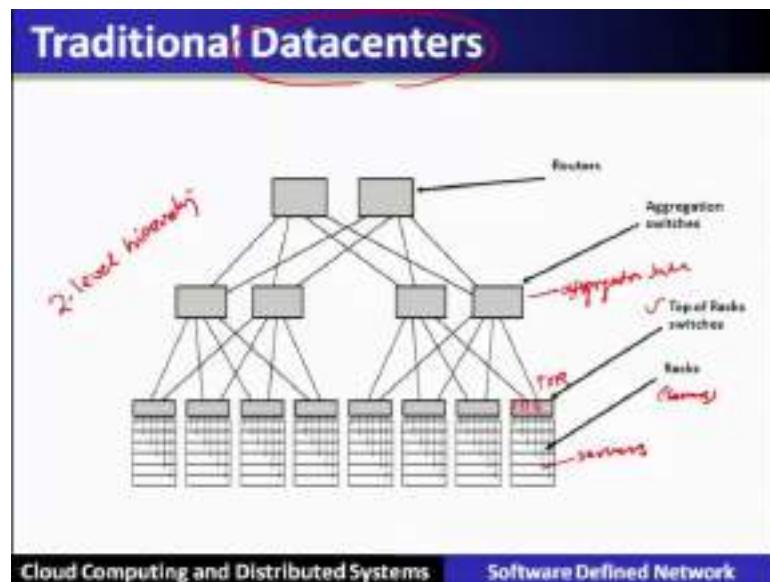
Cloud Computing and Distributed Systems Software Defined Network

Agility means, that use any server for any service at any point of time. If there is a constraint, obviously, economy of scale will not be achieved, and hence realization of multi-tenant cloud system will not be possible. So, agility is first and foremost important requirement for multi-tenant cloud data center. So, to provide the use of any server for any service at any point of time, means that it will provide the better economy of scale through increased utilization means to pack compute as best as we can for higher utilization. If you ever have the constraints, then it is going to be a lot harder to make the full use of the resources.

The other thing, the agility will support improved reliability, for example, if there is a tenant, which experiences the outage which is a planned outage let us say or maybe sometimes an unexpected outage, there in order to run its services without interruption, it can move out its services to some other data center. That is only possible due to the agility, that is use of any server for any service at any point of time, it can move to at any place, which is which can continue to give that particular service, that is called agility.

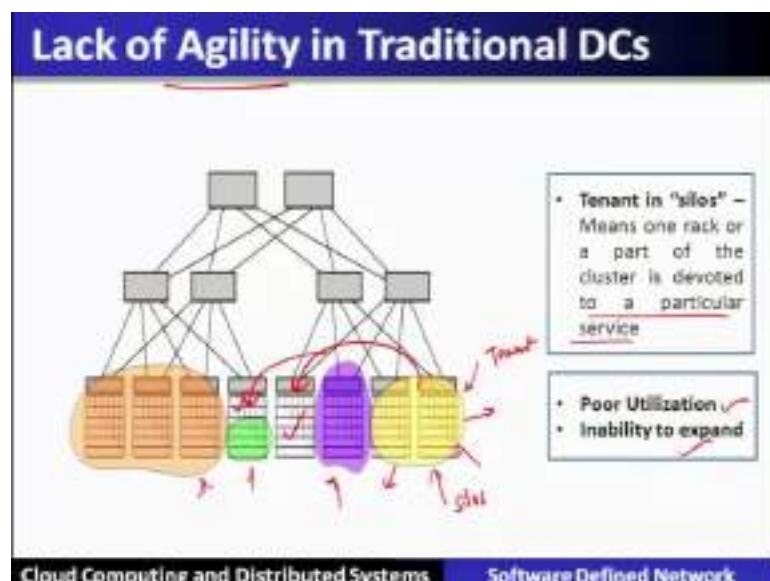
So, the service or the tenant can mean, the customer renting a space in a public cloud, and customer or the service in a private cloud as an internal customer. Either, so either it is a public cloud or internal or a private cloud, multi tenancy can be supported in the form of so, the first requirement is called agility.

(Refer Slide Time: 31:50)



Let us see, what are the issues. This is the picture of traditional datacenter, which comprises of the racks full of the servers, every rack has top of the rack switch. And different racks are connected through an aggregation switch, and they are being connected over the routers. So, it is a hierarchy of 2-level hierarchy, which you can see in any the traditional data center. So, hundreds and thousands of servers are deployed in a particular datacenter, which are connected in this network form.

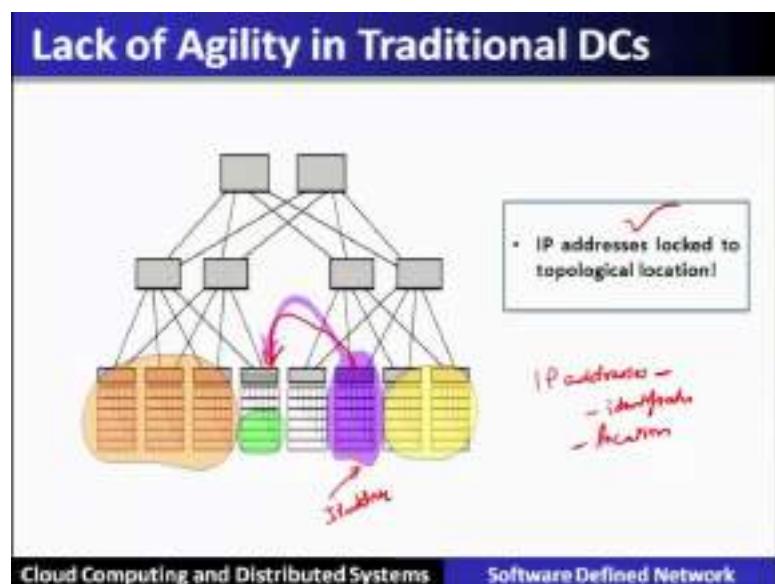
(Refer Slide Time: 33:03)



Now, in this particular traditional datacenter, whether it lacks the agility, let us see why. So, tenants in the datacenter is just like the silos that means, a particular tenant is using let us say these two racks, which are shown here in different colors, so that means, these servers are devoted to that particular tenant service.

Now, if let us say this particular tenant want to expand its services, which is already full, so that means, which our rack having an empty servers, so how it can be further moved to use that empty scenario, so that is not possible in the traditional datacenters. Why, because it lacks the agility. So, this will lead to a poor utilization, and this inability to expand. Similarly, there are some unused servers, due to this there is a requirement of expansion, expansion is not possible, and also it results into a poor utilization. These are all possible these are all possible, due to the reason that it lacks the agility.

(Refer Slide Time: 34:43)



Why this particular problem is there, why, because every rack and its servers, they are locked they are tied with this IP address. So, moving it to some other rack that means, the applications will breakaway with this particular IP addresses and will disrupt the application in its continuation. So, therefore, this IP addresses are locked to the topological locations. So, IP addresses are over used, that means, it is not only went for the identification, but it also identifying the location; so, it is overly used and not supporting the agility.

(Refer Slide Time: 35:50)

Key needs: Agility

- **Agility**
 - **Location independent addressing:** Racks are generally assigned different IP Subnets, because subnets are used as topological locators so that we can route. To move some service over there, we're going to have to change its IP address and it is hard to change the IP addresses of live running services.
 - **Tenant's IP address can be taken anywhere:** Tenant's IP address to be taken anywhere, independent of the location and the data center without notify tenants that it has changed location. Large over subscription ratio i.e. 100 % or greater if there's a lot of communication between both sides will be about a hundred times lower throughput than communicating within the rack.

Cloud Computing and Distributed Systems Software Defined Network

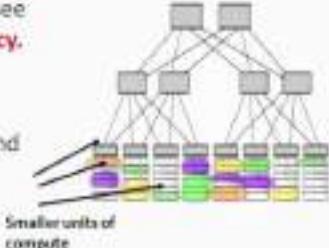
So, here let us see the details of requirements of the agility that is the location independent addressing, so that means, the addressing should be location independent, that is the racks are generally assigned different IP subnets, because subnets are used as topological locators, so that we can route. To move some service over there, we are going to change the IP address and it is hard to change the IP address of a live running service. So, there will be a possibility of disruption, that is not possible.

Now, tenant's IP addresses can be taken anywhere, if this is detached with the location. So, tenant's IP address can be taken to anywhere, independent of the location and the data center without notifying the tenants that has changed the location. This is the aspect of the agility, which is to be ensured.

(Refer Slide Time: 37:11)

Key needs: Performance Uniformity

- **Performance Uniformity**
 - Wherever the VMs are, they will see the **same performance and latency**.
 - **Smaller units of compute** that we've divided our services into, and put them anywhere in the data center, and may be on the same physical machine.



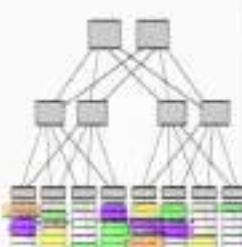
Cloud Computing and Distributed Systems Software Defined Network

Now, another key requirement is called performance uniformity. We say that wherever the VMs are, they will see the same performance and latencies because, when agility is being provided, so they will be the traffic order data will be segregated. So, all the VMs, all the servers, will be properly used, and they will experience in the same set of same performance and latency. So, smaller units of compute that we divided into the services into, and put them anywhere in the data center, may be on the same physical machine.

(Refer Slide Time: 37:55)

Key needs: Security

- **Security:** Untrusting applications and users : sitting right next to each other and can be inbound attacks. So to protect our tenants in the data center from each other in both the public data center as well as in the private cloud and you don't want them to have to trust each other.
- **Micro-segmentation:** separation of different regions of a network,
 - Much finer grained division and control of how data can flow.
 - Isolate or control just the data flow between pairs of applications, or tenants that should be actually allowed.



Cloud Computing and Distributed Systems Software Defined Network

Now, third requirement is called a security, that is untrusted applications and users sitting next to each other can face the inbound attacks. So, to protect the different tenants into the data center from each other, you require the security is to be enforced or implemented. So, there are various techniques, for example, micro segmentation. The separation of different regions of the network; and much finer grained division and control of how the data can flow and isolate the control just the data flow between the pairs of application, or the tenants. So, there are different ways are there, but the key accept here is to ensure the security.

(Refer Slide Time: 38:56)

Key needs: Network semantics

- Network semantics:
 - Match the functional service of a traditional data center
 - Not just Layer 3 routing services but also, Layer 2 services i.e. discovery services, multicast, broadcast etc. have to be supported.

Cloud Computing and Distributed Systems Software Defined Network

Finally, the last requirement of multi-tenant datacenter virtualization is called network semantics. So, it will match the functional service of the traditional datacenter, that is the traditional data services which are supported either in the layer 2 or layer 3 services in the form of discovery services, multicast, broadcast also has to be supported.

(Refer Slide Time: 39:30)

Network Virtualization in Multi-tenant Data Centers Case Study: VL2

VL2: A Scalable and Flexible Data Center Network

Adam Greenberg, Silvano Autore, David A. Mazi, Jeffrey F.阳, Deepak Narayanan, Ravindra Jain, Parag Khaitan, Sudipto Guha, Ravi Sankar Krishnamurthy, Michael Freedman

[ACM SIGCOMM 2009]

Cloud Computing and Distributed Systems Case Study: VL2

So, how this particular four five aspect of multi-tenant datacenters are to be virtualized that we will see in two different case studies, one is in the paper, which is described as VL2 method, that is the scalable and flexible data center design.

(Refer Slide Time: 39:53)

Network Virtualization Case Study: VL2

Key Needs:

- (i) **Agility**
- (ii) **Location independent addressing**
- (iii) **Performance uniformity**
- (iv) **Security**
- (v) **Network Semantics**

Cloud Computing and Distributed Systems Case Study: VL2

Let us see how the VL2 provides the multi tenancy virtualization for data center. [FL] let us take a break. Network virtualization in VL2; Again, we will see how the five different key needs for multi tenancy datacenter virtualization is being architect or being designed in VL2, that is we will see how the agility is supported, how the location independent

addressing is provided, how the performance uniformity is guaranteed, and how the security is ensured, how the network semantics is being allowed.

(Refer Slide Time: 40:49)

Motivating Environmental Characteristics

Increasing internal traffic is a bottleneck

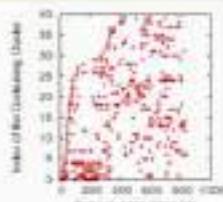
- Traffic volume between servers is 4 times larger than the external traffic

Rapidly-changing traffic matrices (TMs)

- i.e. Take traffic matrices in 100 second buckets and classify them into 40 categories of similar clusters of traffic matrices and see which of the clusters appear in the measurements
- So over time rapidly changing and no pattern to what the particular traffic matrix is.

Design result: Nonblocking fabric

- High throughput for any traffic matrices that respects server NIC rates.
- The fabric joining together all the servers, we don't want that to be a bottleneck.



[Greenberg et al.]

Cloud Computing and Distributed Systems Case Study: VL2

Before we go ahead into the details of the mechanisms, we will see: what was the motivating environment that led to the development of VL2 mechanisms. They have analyzed the internal traffic, and found that increasing internal traffic is a bottleneck. Why, because the traffic internal traffic within the data center between the servers is four times larger than the external traffic that is the traffic outside the data center. So, this particular internal traffic, which is becoming a bottleneck, has to be at rest, let us see that.

Now, they have also analyzed this internal traffic through the traffic matrices, and they found that this particular traffic matrices is rapidly changing, there is no fix pattern. How they have analyzed is they have taken the traffic matrices in 100 seconds bucket and classify them into 40 different categories of clusters of traffic matrices, and see which of the clusters appear in the measurements. If you see the diagram, it is all is scattered that means, there is no way a particular cluster is appearing again. So, over the time rapidly changing and no pattern to what the particular traffic matrix is, so that means, you cannot fix up a particular traffic matrix, it is basically rapidly changing even in the internal traffic, which is a bottleneck into a data center.

So, therefore, the approach for the design should be that the internet or a data center network should be non-blocking fabric. And it should basically be a high throughput for any traffic matrix irrespective of the server NIC rates. The fabric joining together all the servers, we do not want that it to be bottleneck in any case, so that brings up the motivating point that how the non-blocking fabric, which will provide the high throughput for any traffic matrices being supported as per the design is concerned.

(Refer Slide Time: 43:29)

Motivating Environmental Characteristics

Failure characteristics:

- Analyzed 300K alarm tickets, 36 million error events from the cluster
- 0.4% of failures were resolved in over one day
- 0.3% of failures eliminated all redundancy in a device group (e.g. both uplinks)

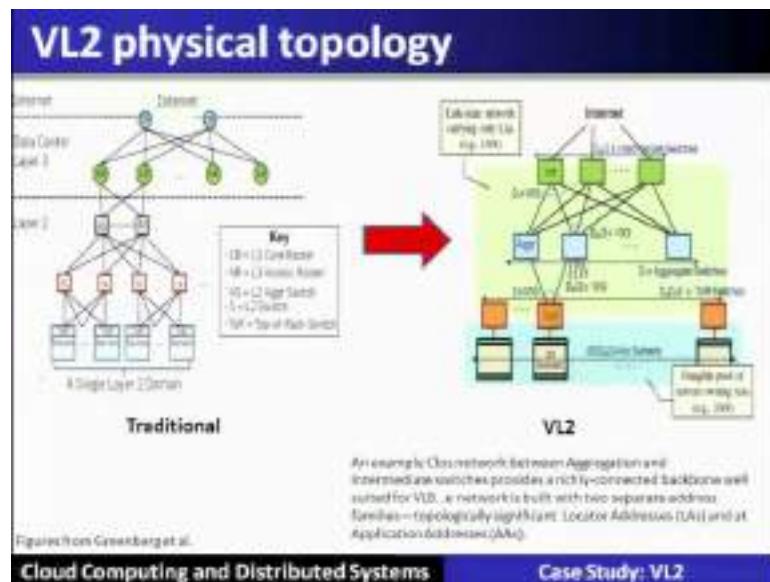
Design result: Clos topology:

- Particular kind of non blocking topology
- "Scale out" instead of "scale up"

Cloud Computing and Distributed Systems Case Study: VL2

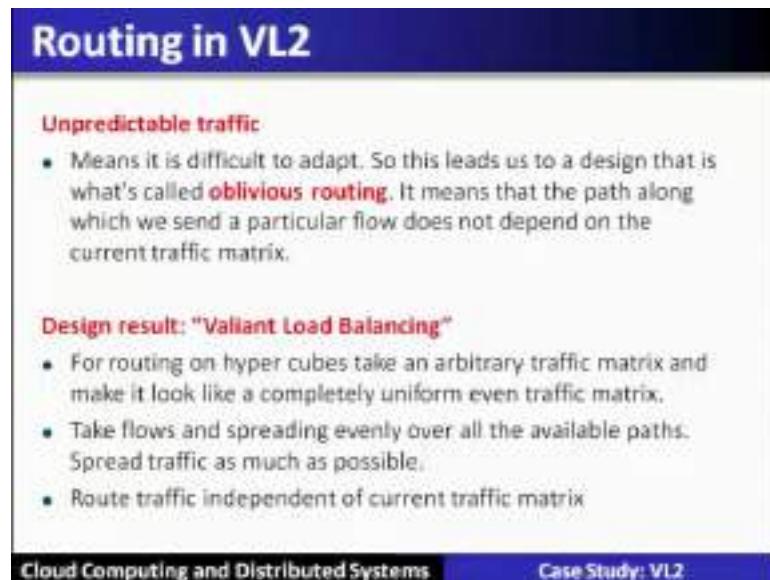
Now, another aspect, which is being seen here is about the failure characteristics. There are several error events occurring that is 0.4 percent of those failures can be resolved in a particular day, 0.3 percent of the failures eliminated all the redundancy in a device groups. So, as far as these failures and their handling, this is specific characteristics has shown that the design requirement should be like clos topology that is a particular kind of non-blocking topology, which will be scaled out instead of scaling up, so that means, large number of devices should be provided in such a topology.

(Refer Slide Time: 44:24)



Let us see the VL2 physical topology. Now, in the VL2 physical topology, we will see that different devices, which are connected that means, using different kind of devices are there, hierarchically networking the entire data center that is the racks and the servers.

(Refer Slide Time: 44:46)

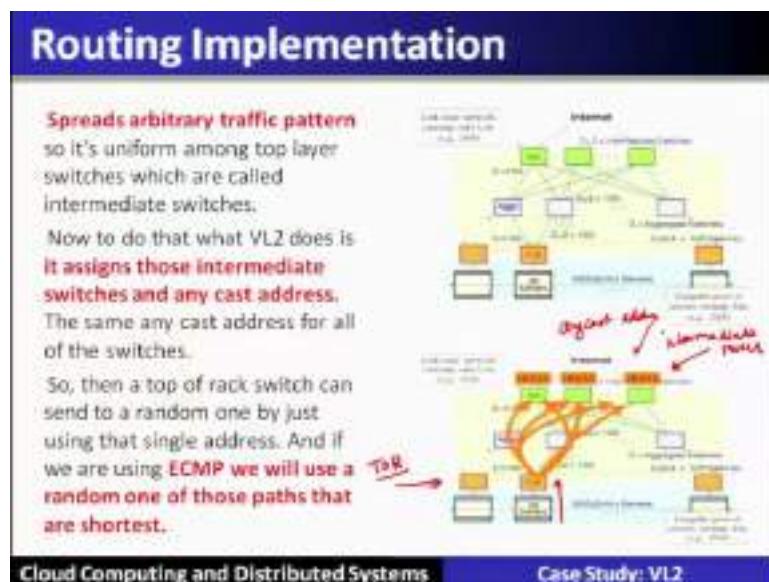


Now, let us see in this particular scenario, how the routing is supported here in VL2. Since, you know that we have already seen is a unpredictable traffic matrix, so best thing is to use the oblivious routing. Oblivious routing will basically not deal with the

particular traffic flow, rather than it is independent to the traffic flow and that means, the path along which we send the particular flow does not depend on the current matrix. So, oblivious routing is basically supported in this scenario.

The other designed result is a valiant load balancing. So, valiant load balancing also does not take into account of a current traffic matrix. So, for an arbitrary traffic matrix, the routing on a hypercube will give the valiant load balancing. So, take the flows and spreading evenly over all the available paths, so that becomes a multipath kind of routing a valiant load balancing will be done, so that means, is spreading the traffic as much as possible in the valiant load balancing. And route the traffic independent of the current traffic matrix.

(Refer Slide Time: 46:05)

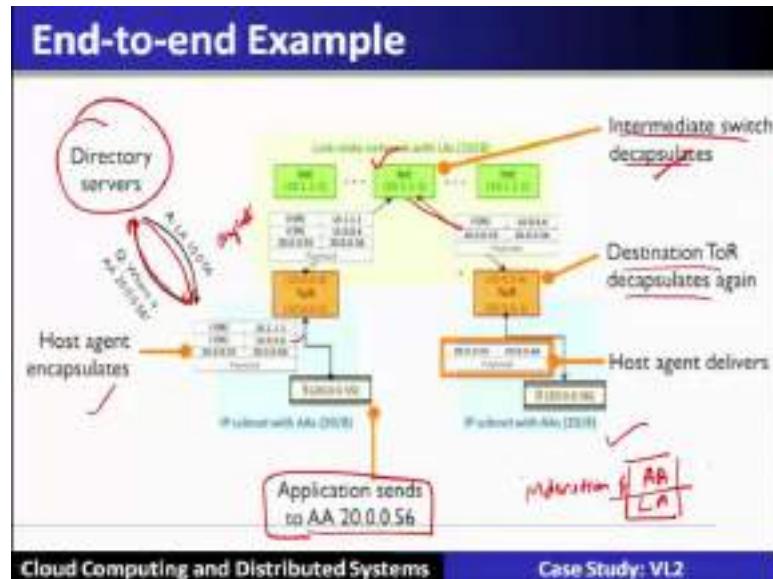


So, it is so after spreading arbitrary the traffic pattern, so it is a uniform among all the top layers, which are called intermediate switches. Now, to do that what VL2 does is that it assigns those intermediate switches with any cast address. So, then the top of the rack switch can send to a random one by just using a single address. We are using ECMP; we will use the random one of those paths that are shortest. Let us understand these concepts.

Now, here they are called intermediate switch. And any cast address is being used here, when a packet reaches up to this point. As far as the top of the rack switch, when a

packet originates from the server goes to the top of the rack switch, then top of the rack switch will change this particular address.

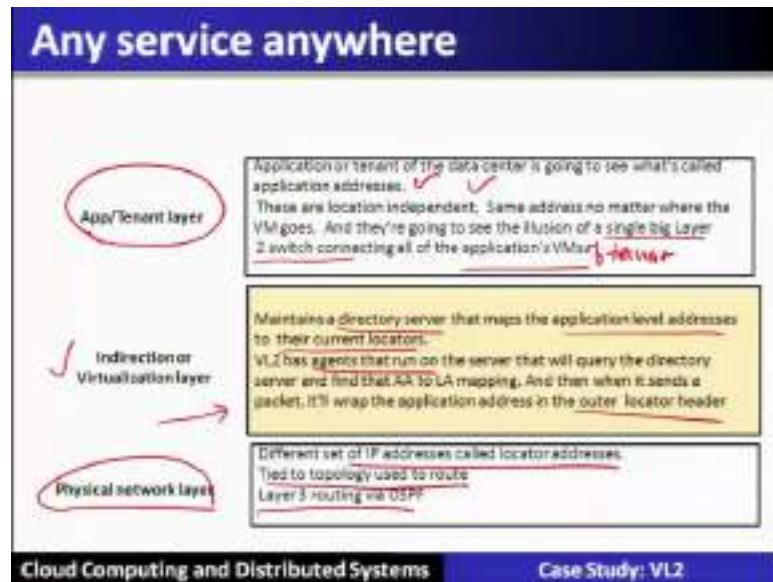
(Refer Slide Time: 47:18)



Let us take this end-to-end example. So, whenever a packet comes, then it will encapsulate into an application address like this. So, it will be wrapped or encapsulated with that address. To know this address, the top of the rack switch will basically consult a directory server, and the directory server will give the application address, which will be used up. So, this particular application address will be the any cast address, and will reach to the intermediate switch like this. Now, this intermediate switch will decapsulate the application or any cast address, and it will send to the destination top of the rack switch here in this case, which decapsulates the header of a top of the rack switch, and finally, it will be delivered to that application, which they want to interconnect.

So, using the directory service, this indirection of the application address with the logical address is being separated out. Therefore, the IP address is now being used for internal communication as you have seen when it reaches to the top of the rack switch within that IP addresses will be used; But, across the communication, application addresses are used and that is the dissolved using the directory service. So, therefore, this is going to be supporting the agility that means, it is not tied up with a particular rack address or IP address, so that the VMs can move from one server to other server or one rack to other rack in the data center.

(Refer Slide Time: 49:35)



So, this particular aspect is being supported by the virtualization layer. So, therefore, there are three different layers. Now, earlier we have seen there will be a physical layer, which will be nothing but the IP addresses, which are also called the locator addresses, and they are tied up on the topology used, and it also follows a particular routing scheme that is called OSPF at the physical network layer.

Similarly, at the application layer or the tenant layer, so the application or tenant of the data center is going to see what is called the application addresses. So, that tenants, they can communicate through the application addresses, which is quite different than the IP addresses. So, these are location independent addresses. So, same addresses no matter, where the VM goes, and they are going to see the illusion of a single big layer to switch connecting all the application VMs. So, now therefore, all the tenants can communicate with each other, and they will not cross across to the other tenant traffic. So, therefore, the illusion will be that, it will be at the application or tenant layer will give an illusion that it is a very big layer to switch, which is connecting all the applications VM of or particular tenant. Similarly, different tenants will have this kind of view.

Now, to translate the physical addresses to the physical addresses to the application addresses, there is a requirement of a indirection or a virtualization layer, which is maintained in the form of a directory server that maps the application level addresses to their to the current locators. So, VL2 has the agents that runs on the server that will query

to the directory service and find out the application address to the location address mapping. And then, when it sends the packet, it will be wrapped the application address in the outer locator header.

So, it is encapsulation and decapsulation will be done. So, encapsulation that means, when the packet is build, it will be encapsulated. And on the outer header, the application address will be and will be put. Similarly, at the time of delivery, all these decapsulation will be done, and it will be delivered in this particular manner. So, this will support a virtualization layer will support any service, anywhere, that is called agility, that we have already explained.

(Refer Slide Time: 52:45)

Did we achieve agility?

Location independent addressing

- AAs are location independent

L2 network semantics

- Agent intercepts and handles L2 broadcast, multicast
- Both of the above require "layer 2.5" shim agent running on host; but, concept transfers to hypervisor-based virtual switch

Cloud Computing and Distributed Systems **Case Study: VL2**

So, did we achieve agility? Let us review this. Now, we have seen that it is a location independent addressing, we have achieved with the help of application addresses. And application addresses are location independent. As far as L2 network semantics are concerned. These agents will intercept and handles the L2 broadcast, multicast, and so on. So, both of these above issues, will require a layer 2.5 that is shim agent running on the host, but the concept transfers to the hypervisor based virtual switch.

(Refer Slide Time: 53:26)

Did we achieve agility?

Performance uniformity:

- Clos network is nonblocking (non-oversubscribed)
- Uniform capacity everywhere.
- ECMP provides good (though not perfect) load balancing
- But performance isolation among tenants depends on TCP back off to the rate that the destination can receive.
- Leaves open the possibility of fast load balancing

Security:

- Directory system can allow/deny connections by choosing whether to resolve an AA to a LA
- But, segmentation not explicitly enforced at hosts

Cloud Computing and Distributed Systems

Case Study: VL2

Now, let us see the performance uniformity. So, clock network is non-blocking, and non-oversubscribed. Uniform capacity is everywhere. ECMP provides a good do not perfect load balancing. But, the performance isolation among the tenants depends on the TCP back off rates. And leaves open possibility for the fast load balancing. Security, let us review VL2, whether how it ensure the security. So, directory service can allow and deny connections by choosing whether to resolve an application address to a locator address. So, security is enforced at the level of directory system, but the segmentation not explicitly enforced at hosts.

(Refer Slide Time: 54:16)

Where's the SDN?

Directory servers: Logically centralized control

- Orchestrate application locations
- Control communication policy

Hosts agents: dynamic “programming” of data path

Cloud Computing and Distributed Systems

Case Study: VL2

So, here we have seen that software defined networks in the form of director service has been used as the logically centralized control. Why, because it used to deny or accept the new incoming requests of the traffic. So, therefore, the software-defined networks will orchestrate, the application locations will control the communication policies across different directory services. And this host agent will perform the data path, using a dynamic programming approach.

(Refer Slide Time: 55:04)



The next case study, the next type of implementation, you will see in the network virtualization using NVP protocol that is network virtualization in multi-tenant datacenters.

(Refer Slide Time: 55:20)

NVP Approach to Virtualization

- The network virtualization platform that was introduced in the paper "**Network virtualization in Multi-tenant Datacenters**" by Teemu Koponen et al. in NSDI 2014.
- Product developed by the Nicira startup that was acquired by VMware.

Cloud Computing and Distributed Systems

Case Study: NVP

Let us see, how it supports, what is his view. There is another approach to the virtualization of multi-tenant datacenters in the form of NVP approach that is Network Virtualization in Multi-tenant Datacenters, published in 2014. This particular approach later on was taken up by the VMware as a product.

(Refer Slide Time: 55:45)

Service: Arbitrary network topology

- Replicate arbitrary topology
- Any standard layer 3 network
- Network hypervisor
- Virtual network tenants want to build

Network hypervisor

Physical/Network; Any layer standard3 network

Cloud Computing and Distributed Systems

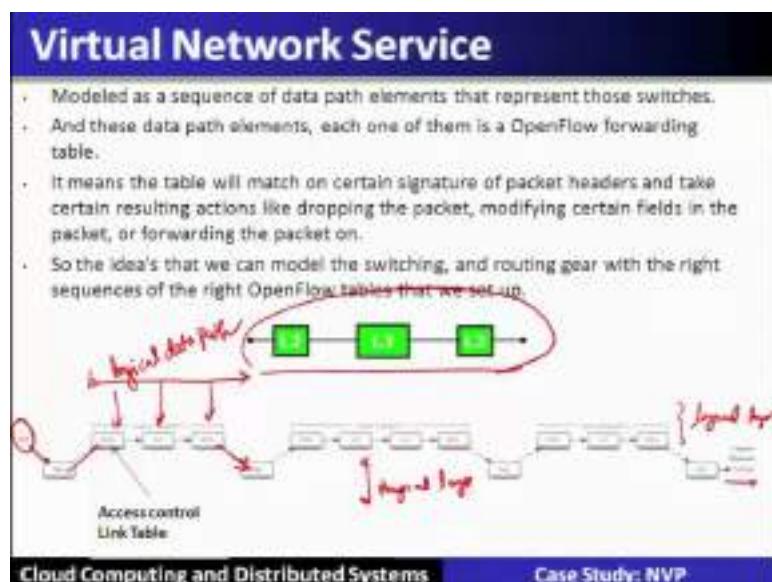
Case Study: NVP

Now, here it will consider for the service any arbitrary network topology that means, it considers the multi tenants to be using different tenants to be using an arbitrary network topology. What do you mean by arbitrary topology, means any combination of layer 2

switch, layer 3 switch, and the networks out of formed out of them is called an arbitrary network topology. It will use a concept of a network hypervisor; just we have seen in the virtualization of the servers that different hypervisors are there that is similarly, here the network hypervisors will be introduced for the first time.

So, therefore, what this NVP provides, it is provides a virtual network for the tenants, the way they want to build. Let us see, how it does. So, we have shown over here that the below is the physical network, which can be any standard layer 3 network, which is being virtualized that is called network hypervisor.

(Refer Slide Time: 57:15)

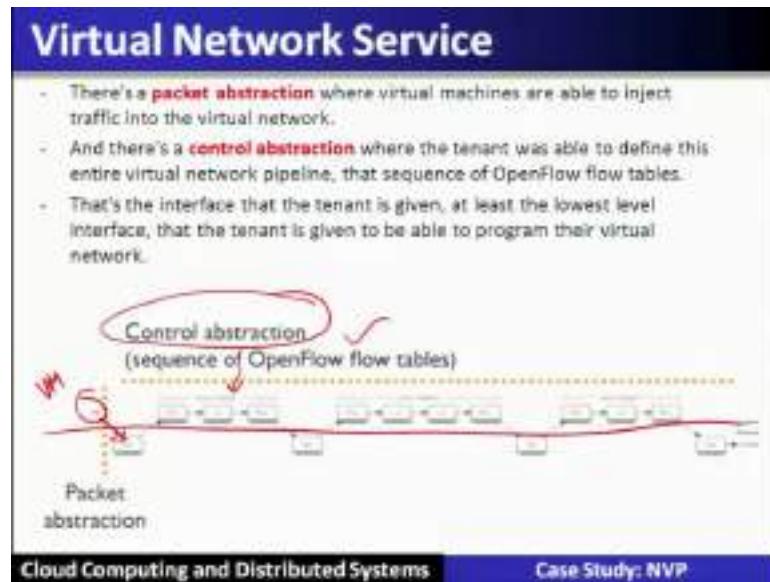


And above that, you will see a network virtual network service; so, modeled as the sequence of data path element that represents these switches. And these data path element, each one of them is OpenFlow forwarding table. So, it means that table will match on a certain signature of the packet headers and take certain resulting actions like dropping the packet, modifying certain fields, and so on. So, using open flow tables, you can form the network a virtual network in this way, which is shown as L2, L3, and so on.

So, this particular virtual network service, we can understand here. If there is a virtual machine will be mapped, which intern will mapped to a particular virtual network in the form of a logical data path that is nothing but an access control link table, followed by an L2 switch, followed by another access control table. And this will form a logical data path. And then, again it will come for the mapping to the physical layer. So, this is called

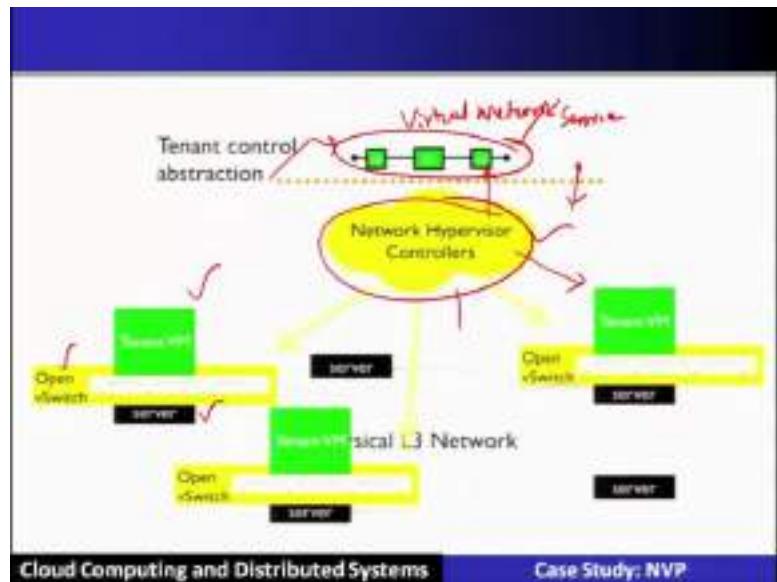
physical layer. And this is the logical layer. So, the virtual network service can be provided in this particular form of a logical layer. And finally, this particular or a packet will be send through the tunnel.

(Refer Slide Time: 59:23)



Now, here there will be a separation that there is a packet abstraction, where the virtual machines are able to inject the traffic into the virtual network that we have shown. And there is a control abstraction. So, control abstraction that is let us see this view. So, virtual machine can inject the packet, and the packet abstraction will be that the control abstraction will be provided in the form of a OpenFlow tables that we have seen in access control link tables. So, that is the interface that the tenant is given, at least the lowest level interface that the tenant is given to be able to program their virtual network.

(Refer Slide Time: 60:28)



So, let us see the virtualization in more detailed of the virtual network. So, this is the abstraction, which tenant can see its virtual network. So, this can be formed or tenant can specify its network in this form of virtual connections. And the tenant can control it over this particular topology. Now, this particular control topology or control abstraction will be supported by the network hypervisor controllers. And they will manage all the tenant virtual machines.

And this tenant virtual machine is being supported by an Open vSwitch, which manages the server. So, just see that, these network hypervisor controllers will provide an abstraction to the tenants in the form of a virtual network, which intern will manage the tenant virtual machine. The tenant virtual machine in turn will be supported by the Open vSwitch. And Open vSwitch manages the servers.

So, Open vSwitch is managing the devices and gives the support to the tenant virtual machine. And the network hypervisor will support to the tenant control to the tenant virtual network. So, this way every tenant can establish or can be specified through the abstraction its virtual network which in turn is virtualized by a network hypervisor, and using vSwitch all the network devices are being managed. So, therefore, NVP has given the idea of the network virtualization. So, it has given the concept of network hypervisor, just like we have seen the hypervisors or virtualization of the servers.

(Refer Slide Time: 63:21)

Challenge: Performance

Large amount of state to compute

- Full virtual network state at every host with a tenant VM!
- $O(n^2)$ tunnels for tenant with n VMs
- **Solution 1:** Automated incremental state computation with nlog declarative language
- **Solution 2:** Logical controller computes single set of universal flows for a tenant, translated more locally by "physical controllers"

Cloud Computing and Distributed Systems **Case Study: NVP**

Now, the challenge is let us see the performance. Here it requires large amount of state to be computed. Full virtual network state at every host with a tenant virtual machine, it requires order n square different tunnels for tenants with n virtual machines. So, the 1st solution says that automatic incremental state computation with nlog declarative languages is possible. 2nd solution says that logical controller computes single set of universal flows for a tenant, translated locally by physical controllers.

(Refer Slide Time: 64:06)

Challenge: Performance

Pipeline processing in virtual switch can be slow

- **Solution:** Send first packet of a flow through the full pipeline; thereafter, put an exact-match packet entry in the kernel

Tunneling interfaces with TCP Segmentation Offload (TSO)

- NIC can't see TCP outer header
- Solution: STT tunnels adds "fake" outer TCP header

Cloud Computing and Distributed Systems **Case Study: NVP**

So, therefore, the performance can be handled through the different programming of pipeline processing in the virtual switch can be slow. So, it is a tunnel interface with TCP offloading can be there.

(Refer Slide Time: 64:24)

Conclusion

- In this lecture, we have discussed the **need of software defined network, key ideas and challenges of software defined network.**
- We have also discussed the challenges in multi-tenant data centers i.e. (i) **Agility**, (ii) **Location independent addressing** (iii) **Performance uniformity**, (iv) **Security** and (v) **Network semantics**.
- Then we have discussed the **network Virtualization in multi-tenant data centers** with a case study of VL2 and NVP

Cloud Computing and Distributed Systems **Case Study: NVP**

So, conclusion, in this lecture, we have discuss the need of software-defined networks, and how this software-defined network has overcome from different challenges of the traditional network. We have also seen the use of software-defined network for virtualization in a multitenant datacenters to support the agility, location independent addressing, performance uniformity, security and network semantics. We have then discussed there are two different ways of network virtualization in multi-tenant datacenters in the form of VL2, and another way of design that is called NVP.

Thank you.

Cloud Computing and Distributed Systems
Prof. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 06
Geo – distributed Cloud Data Centers

(Refer Slide Time: 00:17)

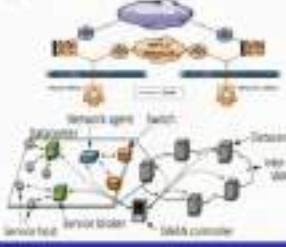
Preface

Content of this Lecture:

- In this lecture, we will study the **Geo-distributed cloud data centers**, interaction of data centers with users and with other data centers.
- We will also describe the data center interconnection techniques such as (i) Traditional schemes such as **MPLS**, (ii) **Cutting edge** such as Google's **B4** and (iii) Microsoft's **Swans**



A Warehouse-Scale-Computer (WSC) Network



Service gate switch
Server blade
Switch controller
Datacenter
Host GW

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Geo-distributed Cloud Data Centers; prefix, content of this lecture, we will discuss geo-distributed cloud data centers, the interaction with other data centers, and interaction of data centre with the users are the prime point for the discussion in this lecture. We will also cover data center interconnection techniques such as the traditional schemes of MPLS, versus the current edge schemes, which are applied in Google's B4, and Microsoft's Swans, and the current trends in the data center networking.

(Refer Slide Time: 00:59)

Inter-Data Center Networking: The Problem

- Today, the virtual use of any popular web application means to communicate with a server in a data center.
 - However the connectivity for this service depends on the internet
 - Internet becomes crucial in application service's performance
- Data centers also communicate with each other over the internet.
Example: replicating client data across multiple data centers.
- In cloud scenario, the wide area connectivity or the internet is as crucial as the data center infrastructure.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, the inter-data center networking, what is it and what are the different problems, let us see and understand it. And then we will discuss the trends, and what are the traditional methods to overcome from this particular problems. So, in today's virtual use of any web based applications means that you are communicating with the data center in some form or the other, so that means if this is the user of web based application through the internet, this particular user will be communicating with the data centers. This is data center number 1 and so on; this is the data center number 2, which are at different locations. So, they are called geographically distributed data centers.

So, therefore the users of any web based applications are connecting or interacting with the geographically distributed data centers through the internet. Now, this particular service, which the user is trying to get through the web application highly depends upon the internet connectivity. Now, you know that from the background of the network, this internet is also having an underlying network, which is called a wide area network. Now, as far as these data centers are concerned, these data centers, which are at different locations they are also communicating with each other with the help of internet. Therefore, the wide area network that means to provide a good service, the performance of the service depends highly on the performance of the wide area network in this particular scenario.

So, in the clouds scenario, we will see here the importance of wide area connectivity or the internet, which becomes a crucial as the data center infrastructure. So, the problem is that there are two kinds of interaction, which we will focus that is from data center to the user. And the other is between to the data center with respect to the wide area network. Now, what is the issue, what are the problems this wide area network, which is nothing but the internet is highly dependent on the bandwidth.

And there is a huge cost if lot of communication is done on this particular WAN or the internet, then there will be a huge cost of this bandwidth. And if it is not utilized, then that cost will be overhead or a burden or a waste. So, how this particular bandwidth, which is primarily essential for the performance of running web applications is going to be optimized, so that the performance with the cost or with the low cost, it can be provided as the service in the cloud scenario.

(Refer Slide Time: 05:22)

Why Multiple Data centers ?

- Why does a provider like Google need such an extensive infrastructure with so many locations across a wide expanse of the globe? *Geo-distributed DCs*
- **Better data availability:** If one of the facilities goes down, due to a natural disaster, you could still have the data be available at some other location if it is replicated.
- **Load balancing:** Multiple facilities can spread incoming and outgoing traffic over the internet across a wider set of providers, over a wider geographic regions.
- **Latency:** If present in multiple paths of the globe then can reach clients in different locations at smaller distances, thus reduces latency.
- **Local data laws:** Several authority might require that companies store data from that country in that jurisdiction itself.
- **Hybrid public-private operation:** Can handle the average demand for service from the private infrastructure, and then offload peak demand to the public cloud.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, the question is why the multiple data centers are existing in the previous picture that is also called why the geographically distributed data centers are there in the cloud. So, this question we will see, why we need the geographically distributed data centers, and then will see the networking part, which is also an essential infrastructure for providing the data center to the cloud. Now, the question is why does the provider like Google needs such an extensive infrastructure, so that at many locations across the wide

globe, these data centers are residing, and they are also called geographically or geo-distributed data centers.

Let us see this particular question, so the first region of having the geographically distributed data center is to provide the data availability, so that means, to provide the better data availability means, that if one of this data center facility is down, due to the calamities earthquake or some other reason, then you will continue to get the data or the availability of the data. If it is replicated at more than one geographic location elsewhere, which is maybe active at that point of time.

The second region is called load balancing. So, the second region is due to the load balancing, so that means, if multiple facilities can spread the incoming and outgoing traffic over the internet across the wide set of providers, over the wide geographic regions, that means the entire traffic is uniformly distributed. Hence, the load balancing will give the more performance better performance, and also it is a cost effective way. So, load balancing is possible if you support the geographically distributed data center, that is why, the providers like Google has its data centers geographically, it spreaded over twelve different location across three different continents on the globe.

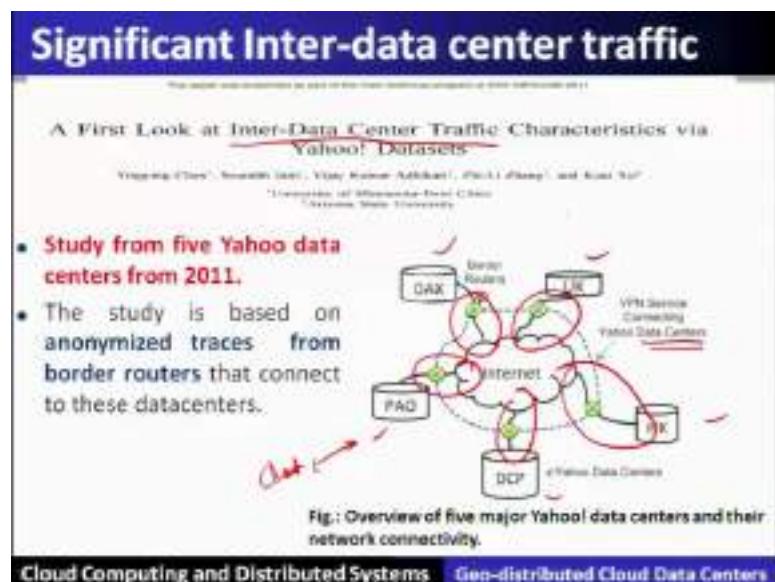
Third one is the latency. Now, if present in a multiple paths of the globe then can reach the clients in different location at a smaller distance. Hence, it reduces the latency meaning to say that if a client is located in the country India, and it is being served with the nearest data center located in India, then the traffic need not have to travel across a wide distance located in US or in some other country. Hence are a smaller distance is possibility, which will basically reduce the latency in accessing the application. So, hence the latency will be reduced, if the data centers are maintained in the form of geographically distributed sites.

Fourth one is called the local data laws. Now, there are different countries, which have different laws of for the companies, who stores that the data as per their jurisdiction of that country. So, therefore the data and its axis has to follow the laws of the land, wherever the data centers are maintaining those particular data sites. So, it depends upon the different applications, and how this laws can be useful in maintaining the geographically distributed data center to provide the services are also going to play an important role, that is why, this geographically distributed data centers are being made to

ensure the services, and also using the laws of maintaining the data as per the jurisdiction of different nation or the country.

Fifth one is called hybrid public-private operations. In the sense, that if there is a geographically distributed data centers, and also the private data centers are being maintained, then what people can do is that the average demand for the data or the service through the private infrastructure can be provided with the lower cost. And whenever there is a high demand in the peak axis or in peak period, then that kind of load can be shifted to public, hence it is called hybrid public private operations, can be supported with the help of geographically distributed data centers or multiple data centers, here in this case such as private and public clouds.

(Refer Slide Time: 11:38)

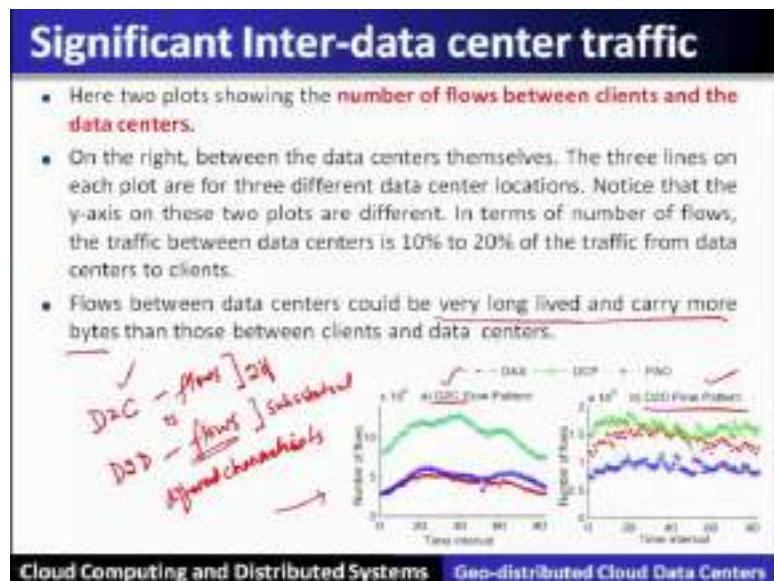


Now, we will see here that in inter-data center, what kind of traffic characteristics is being observed by different service providers, and what are the techniques to ensure the performance and efficiency. So, it was study of 2011, Yahoo data sets or data centers, which are maintained at five different locations. So, the study reveals that in the picture, five different locations are shown over here, which are nothing but, they are the data centers, these data centers are connected to the internet through the border routers, these are the border routers.

Now, as far as the inter-data center traffic is concerned that means, the traffic across the data centers. It is shown that these particular traffic, which will basically are going to be

a major data traffic between or among these data centers in contrast to the traffic, which is originated from the client to these data centers. If we compare, then this particular traffic is quite substantial, hence we will focus on the inter-data center traffic management, so that a better performance, and a lower cost can be achieved in this part of discussion.

(Refer Slide Time: 13:35)

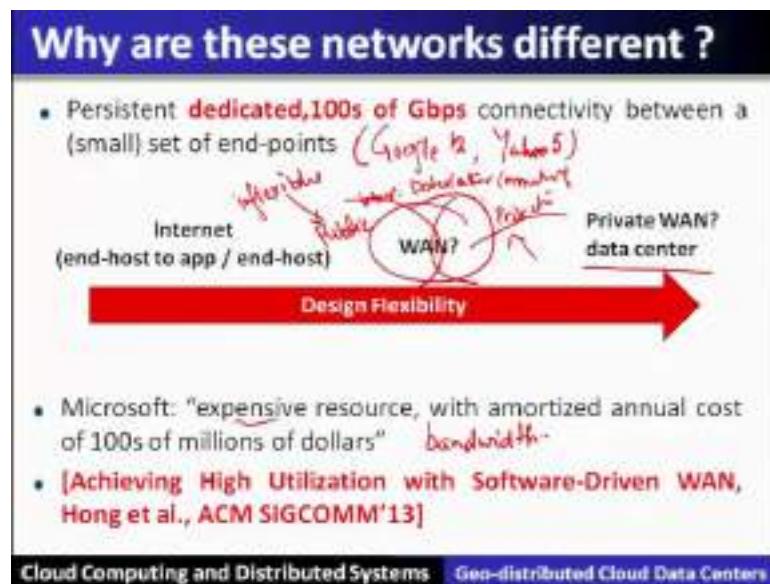


So, therefore a significant inter-data center traffic is observed in that particular study of the data sets of five different Yahoo data centers, that shows that the number of flows between the client and the data center is plotted in the two figures. So, this figure number 1 shows the traffic or a flow from between the data center and the client. Second figure shows the flow pattern of inter-data center traffic. Now, here you can see on the y axis the number of flows, which are being pointed. Now, one thing we can observe here is that the total number of flows that is the traffic between the data center is 10 to 20 % of the data of the traffic from data center to the client.

So, here one more thing has to be observed that the flows between the data center could be very long lived and live and carry more bytes than the flow between the client and data center, so that means the client and data center flows versus the flows, which basically has between data center is having a different characteristics. And this is quite substantial, this is only 20 percent of the total traffic, so 80 percent that is quite substantial is the flow or inter-data center flows. Now, as far as the characteristics of

these flows, if we can see here the traffic or the flows, which are there in the data center are very long lived and carry more bytes, that means, the traffic is a consistent between this data center or inter-data center traffic compared to the traffic or a data flow between the client and the data center.

(Refer Slide Time: 16:13)



So, now let us see that these particular flows, which are more substantial in case of inter-data center traffic has to access the wide area network. And this particular wide area network is nothing but 100s of Gbps connectivity between by small set of points, meaning to say that in the Yahoo only 5 different data center sites are there, Google there are 12 site, so hence or a set of end points basically will observe a huge basically traffic.

So, as far as we will see here, this particular WAN, which will provide the connectivity to the inter-data center connectivity through the wide area network. Now, the question is how much this of this part of WAN belongs to a private WAN, that means, the data center how much the data centers can lay their own fiber or can own that part of this WAN and rest, which is not owned by them they can be a public, public versus the private, so that becomes a question.

So, public is as good as, private is as good as, it is owned by that particular company, but as far as the public is concerned the cost of usage is going to be paid and also, it is inflexible. Inflexible in the sense, the protocols, and algorithm, and the techniques, which are used in the public wide area network, that is in the internet is inflexible that means, it

cannot be much cannot be done. Whereas, if it is private then all the protocols, can be can be redesigned policies can be implemented.

Therefore, Microsoft has said that the expensive resource, with amortized annual cost of 100s of millions of dollars is being is spend in the bandwidth. So, therefore we will see, how to achieve the high utilization with software driven wide area networks, so that is being published in ACM SIGCOMM conference in 2013.

(Refer Slide Time: 19:06)

(i) MPLS: Traditional WAN approach

- The traditional approach to traffic engineering in such networks is to use **MPLS (Multiprotocol Label Switching)**
- Network with several different sites spread over defined area, connected to each other perhaps over long distance fiber links.

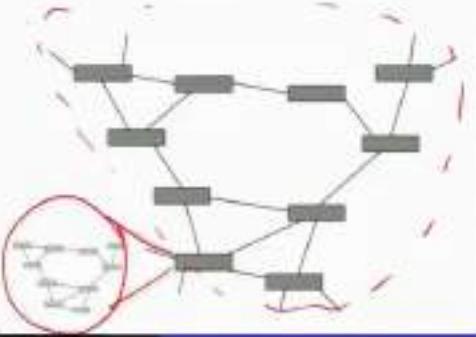
The diagram illustrates a Wide Area Network (WAN) connecting multiple Geo-distributed Data Centers (DCs). It shows a network of nodes (represented by rectangles) interconnected by lines representing fiber links. A red label 'WAN' is written diagonally across the top left of the network. Another red label 'Geo-distributed DCs' is written at the bottom right. Below the diagram, there is a footer bar with the text 'Cloud Computing and Distributed Systems' and 'Geo-distributed Cloud Data Centers'.

So, before we go into the details, let us see what are the approaches in this regard, that is what are the approaches to utilize the wide area network in a traditional way; so, traditional way of utilizing the wide area network efficiently was nothing but MPLS-Multiprotocol Label Switching, which ensures or which provides a traditional approach to do a traffic engineering in such networks using MPLS. Here, in this is key, this is the network with several different sites is spread over the defined area, so this is a geo-distributed data centers. So, they are connected over the long distance of fiber links either through the wide area network.

(Refer Slide Time: 20:12)

1. Link-state protocol (OSPF / IS-IS)

- Use link-state protocol (OSPF or IS-IS) to flood information about the network's topology to all nodes.
- So at the end of such a protocol, every node has a map of the network.



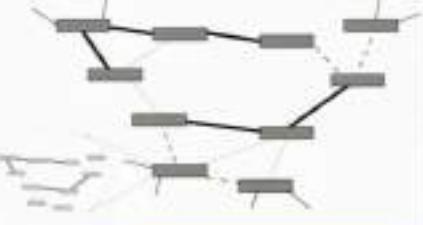
Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, the information about this particular topology can be collected with the help of link-state protocol that is OSPF-Open Shortest Path Flooding, and IS-IS to flood the information about the network topology to all the nodes and when the protocol terminates, so every node will have the map of the network, which is shown over here. So, after the flooding of link-state, so at the end every node will have the complete picture of the network within it.

(Refer Slide Time: 21:03)

2. Flood available bandwidth information

- For traffic engineering, also spread, information about the bandwidth usage on these different links in the network.
- Given that there's already traffic flowing in this network, some links will have spare capacity and some won't.
- Both IS-IS and OSPF have extensions that allow the flooding of available bandwidth information together with their protocol messages.



Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, let us see what are the uses of this information. Now, for the traffic engineering requires to spread the information about the bandwidth usage on these links in the network, that means, how much is the bandwidth available required to be known at all the points. So, given that there is already traffic flowing in the network, some links will have the spare capacity and some would not. So, both IS-IS and OSPF have extensions to allow the flooding of available bandwidth information together with their protocol messages. So, besides topology the information about the available bandwidth is also spreaded and now known at each point.

(Refer Slide Time: 21:57)

3. Fulfill tunnel provisioning requests

- Knowing the set of the network, when the router receives a new flow set of requests, it'll set up a tunnel along the shortest path on which enough capacity's available. It sends protocol messages to routers on the path setting up this tunnel.
- Further, **MPLS also supports the notion of priorities**. Thereby if a higher priority flow comes in with the request for a path, lower priority flows might be displaced. These flows might then use higher latency or higher cross paths through the network.



Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, third is to fulfill the tunnel provisioning request. Now, here knowing the set of networks, where the router receives the new flow requests, it will set up a tunnel along the shortest path on which enough capacity that is the bandwidth is available for supporting the data traffic. So, it sends the protocol messages to the router on the path setting up this particular tunnel. Further, MPLS also supports the notion of priorities. Therefore, even if the tunnel is established, and if a higher priority flow comes in with the request for a path in the lower priority flows will be displaced. And this lower priority flows and then use some other higher latency path to support the high priority flows, so that is all the possible using MPLS that is to fulfill the tunnel provisioning requests.

(Refer Slide Time: 23:02)

4. Update network state, flood information

- After a flow is assigned a terminal, the routers also update the network state.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, the fourth point here in the traditional way that is in the MPLS way is to update the network is steady, and also the flood information. Now, after the flow is assigned a terminal, the routers also update the network a state.

(Refer Slide Time: 23:23)

4. Update network state, flood information

- When a data packet comes into the ingress router, the router looks at the packet's header and decides what label, that is what tunnel this packet belongs to. Then it encapsulates this packet with that tunnel's label and sends it along the tunnel.
- The egress router then decapsulates the packet, looks at the packet header again and sends it to the destination. **In this scheme, only the ingress and egress routers read the packet header.** Every other router on the path just looks at the assigned label.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, when a packet comes into the ingress router for sample pack end is originated from here, so this becomes an ingress router. The router looks at the packet headers and decides what label, that is which tunnel this packets belongs to here, it will assign the

label. Label means, which tunnel in then encapsulates this packet with that tunnels label, and send it along this particular tunnel.

Now, egress router then decapsulates the packet, and looks at the packet header again and sends it to the destination. So that means only ingress router and egress routers are only looking inside the packet headers and rest are all using only the labels that means, they need not have to look into the routing tables and do not need much processing into the packet. Hence, once terminals are established the routing becomes quite simple, and therefore it is also safe and efficient also and also supports the virtual private network in this particular scenario. And using this particular tunnel lot of traffic engineering is possible in the traditional way.

(Refer Slide Time: 25:13)

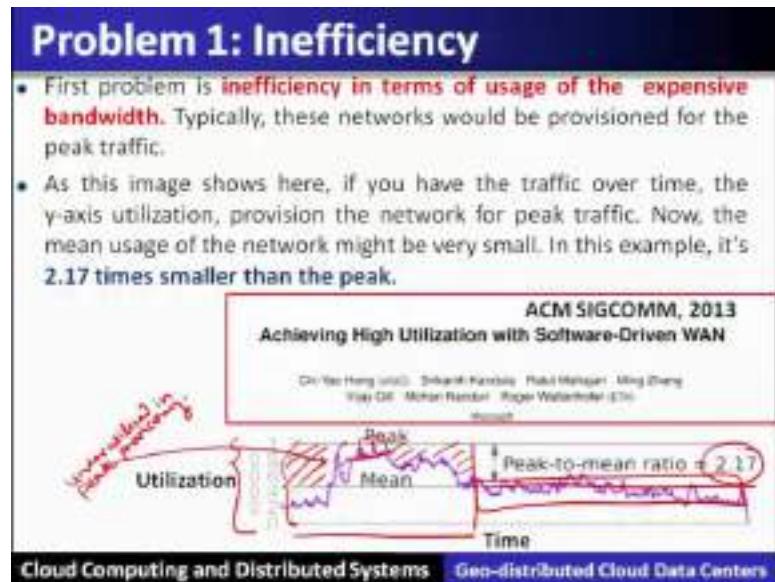
Simple forwarding along the path

- Making forwarding along the path very simple. This is the reason the protocol is called **Multi-Protocol Label Switching (MPLS)**.
- Also, MPLS can run over several different protocols, as long as the ingress and egress routers understand that protocol, and can map onto labels, that's why the name is: **multi-protocol label switching**.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, this simple forwarding along the path that is making forwarding along the path is now becomes very simple in Multi-protocol Label Switching. I have explained that only ingress and egress routers only. They have to basically consult their routing tables and rest of them are only in that tunnel, they have to only work on the labels. So, MPLS can run over several different protocols, as long as ingress and egress routers understand that protocol, and map on to the labels that is why the name is multi-protocol label switching.

(Refer Slide Time: 25:56)



Now, in the traditional approach, let us understand the inefficiencies with respect to the cloud inter-data center networking. And then, we will see what are the latest approaches are going to be solved. So, the first problem is about the inefficiency so, the inefficiency in terms of usage of the expensive bandwidth. So, in the sense that whether the bandwidth is utilized fully or it is not that much utilized, so that is called inefficiency. So, typically these networks would be provisioned for the peak traffic.

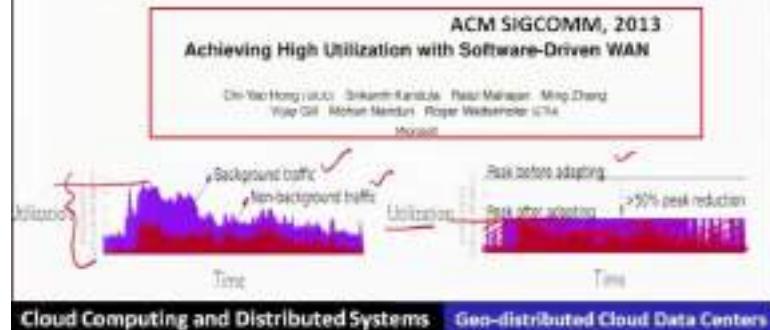
As shown here, in the figure below, you can see here the network is provisioned along this particular peak traffic. Now, you know that over a particular time, if you see how much is the percentage, which will be generating this particular peak traffic, so most of the time the traffic, which is generated is not peak. Therefore, this particular bandwidth of most of the time is underutilized. All the bandwidth is provided using the peak provisioning, but the peak traffic is not always consistent. It is coming in the form of a bus, so most of the time the bandwidth is basically underutilized.

So, now that can be seen here using the mean usage of the network. So, if you see the mean usage of the network, then we can see around the mean, the entire traffic is basically around the mean that is the mean the peak to the mean utilization is only 2.17. Hence, it is highly inefficient way of using the expensive resource that is the bandwidth.

(Refer Slide Time: 28:16)

Problem 1: Inefficiency

- Most of this traffic is actually **background traffic**, with some **latency sensitive traffic** as well.
- So you can provision for the peak of the **latency sensitive traffic**, and then fill the gaps with the **background which is not latency sensitive**.



So, most of the traffic, now we will see into the more details that reveals that most of the traffic is actually the background traffic, with some latency sensitive traffic as well in the peak traffic. Even in the peak, if you see in the details, we will see that it is having two types of traffic. One is the background traffic, the other is non-background traffic. So, non-background traffic is basically the latency sensitive traffic, so, this is the latency sensitive traffic is to be provisioned. And this particular bandwidth to support this latency sensitive traffic cannot be compromised, and only that much of provisioning should be ensured.

As far as, the background traffic is concerned, which is not latency sensitive and that can be basically fill the gap with the background, where the latency is not sensitive. If we see the traffic using these two classifications, then further there is a possibility of improving. So, here on the right side, if you see that the peak before the adopting, and we can further so that means, you can see here the peak is for the latency sensitive traffic. So, what we can see here, this is the provisioning, which is being allowed, and rest are all can be is filled by the background traffic, which is not latency sensitive. So, it can be spread, so this way there is a possibility of achieving high utilization with the software driven wide area network.

(Refer Slide Time: 30:21)

Problem 1: Inefficiency

- So unless you differentiate traffic by service, you cannot do such an optimization. This is not easy to do with the MPLS approach because **it does not have a global view of what services are running in the network**, what parts of the network they are using and such.
- Also, a related point is that regardless of whether there are multiple services or not, MPLS, the routers make local greedy choices about scheduling flows. So **traffic engineering is suboptimal**.
- For these reasons, such networks typically run around 30% utilization to have enough headroom for these inefficiencies, and this is expensive.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, unless you differentiate the traffic by the service, you cannot do much on the optimization of the bandwidth. So, this is not easy to do in the MPLS approach, because it does not have the global view of what services are running in the network, what part of the network they are using and such. So, also a related point is that regardless of whether they are multiple services or not, MPLS routers make greedy choices about the scheduling of flows. So, the traffic engineering is suboptimal with these reasons, such networks typically run around 30 percent utilization to have enough headroom for these inefficiencies and this is also going to be very expensive, due to the inefficiency.

(Refer Slide Time: 31:08)

Problem 2: Inflexible sharing

- Another big problem with the MPLS approach, is that it only provides link level fairness. So at any link, the flows can share capacity fairly. **But, this does not mean network wide fairness.**
- For example, we have the green flow sharing capacity across that length with a red flow. The blue flow also shares capacity with the red flow. But, the blue and green flows both get capacity half the red flow, because the red flow uses multiple paths. So we have link level fairness, but we do not have the network wide fairness.
- The network wide fairness is hard to achieve; **unless you have a global view of the network.**



Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Another problem is called inflexible sharing. For example, here in MPLS, you can see the link level sharing for example the link level sharing means, as far as the flow, the green flow is concerned is using these two kind of bandwidth sharing the common bandwidth across these two devices, with the red one. So, 50 percent of the bandwidth will be given to this one green one, and 50 percent of this bandwidth will be on this flow to the red one.

As far as red is concerned, it is also using another path. And in this path 50 percent, it will get as far as the bandwidth. As far as the blue is concerned, so as far as the red one is concerned; red one will get 100 percent bandwidth using multipath; and the green flow will get only 50 percent, due to the single path; and the blue also will get 50 percent, due to the single path.

Now, with the link level; link level does not ensure the global view. Hence, it does not cover the network wide fairness to ensure the network wide fairness. Here, it requires the complete information or the complete picture of the global view. So, the network wide fairness is hard to achieve unless, you have the global view of the network. Therefore, the sharing is inflexible that means, at means that the link level sharing is not achieving the network wide fairness, this is another problem.

(Refer Slide Time: 33:06)



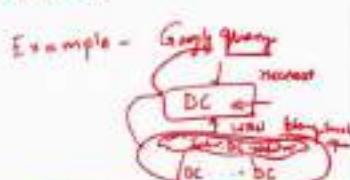
So, now with this traditional approach of using MPLS has two main problems. One is the inefficiency in the bandwidth utilization; the other is the global view of network wide

fairness is not ensured. Now, using this particular two problems, now let us see the cutting-edge solution of using wide area network traffic engineering in the modern times that it is the recent times. So, Google B4 has shown his way of solving these problems using software defined wide area network. We will see how these problems, Google's B4 is going to solve. We will also see the Microsoft swan, which will also showcase its particular software defined wide area network for achieving high utilization of the bandwidth.

(Refer Slide Time: 34:15)

1. Leverage service diversity: some tolerate delay

- To get very high bandwidth utilization in the WAN because of natural fluctuations over time. So, to leverage diversity in the services some services need a certain amount of bandwidth at a certain moment of time and they're inflexible and some other services can use to kind of fill in whatever room is left over.
- For example, latencies instead of queries.

Example - 

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, let us see in the newer approaches, what are the main points for the design, which they consider, and then how they are going to address in the implementation in their cloud data center or how they are managing the wide area network for inter-data center networking.

So, the first one is to leverage the service diversity, let us see what does this means to leverage the service diversity. Now, here we can see that to get very high bandwidth utilization wide area network. We have to basically see that there are some fluctuations, which are happening over the time in different diversity of the services. For example, some services requires a certain amount of bandwidth at a certain moment of time and they are inflexible, whereas some other kind of services, who are not very rigid about the requirement of the bandwidth over a particular time can fill in wherever the rooms are

available after being allocated to the services, which are very stringent requirement of the bandwidth at a particular moment of time.

Take the example that in a Google search engine, if there is a query, this particular query will now go to a data center, nearest data center. Now, that nearest data center; does not have the information about that particular query. So, this data center in turn will flood to the inter-data center network over other data centers. Now, this particular traffic, which is now generated to know or to satisfy the query has to be done very efficiently or in a within a particular time. So, this is latency sensitive operation over a wide area network.

So, this operation has to be immediately addressed, so that this particular response can be basically given. So, they are called latency sensitive queries, so which basically are need to be addressed they are inflexible. So, the bandwidth has to be allocated, whereas maybe some other kind of traffic may not be that latency sensitive. So, therefore there is a service diversity across different services, so that is to be leveraged for the bandwidth utilization in the wide area network. So, this is the first way.

Now, the second is called the tolerance that is the delay. So, once that means, different services will ensure how much tolerance is basically can be basically allowed for a particular application. And based on that different services can be classified, and the bandwidth can be put on the utilization according to the criteria.

(Refer Slide Time: 38:28)

2. Centralized TE using SDN, OpenFlow

- Software define networking approach gather information about the state of the network.
- Make a centralize decision about the flow of traffic and then push those decisions down to lower levels to actually implement them.
- But bringing all that information together in one place is a relatively complex decision.

Now, second one is called centralized traffic engineering using software defined networks. Now, here the software defined networking approach will gather the information about the state of the network. So, there will be a centralized decision about the flow of traffic, and then push these decisions down to the lower level actually, implement them done. But, bringing all this information in one place is a complex decision, why because in its distributed system.

(Refer Slide Time: 39:03)

3. Exact linear programming is too slow

- Traditionally, with a optimization technique like linear programming, which is a way to take a set of constraints on required amounts of data flow over parts of a network and come up with an optimal solution. But to apply it to the situation where we need to **make relatively quick decisions**.
- Part of the complexity comes from the multitude of services, the different priorities. If we have just one service, we could run it in flow algorithm, and that would be much faster.
- So it require something faster if it's not guaranteed to be exactly optimal.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, third one is called exact linear programming will be here, what is slow for example, if the information is collected and optimization with all the constraint is applied through the linear programming, it requires time for the optimization problem to be solved. So, here the situation is that it has to make a quick decisions. So, the part of the complexity comes from the multitude of the services, with different priorities. So, if we have just one service, we could run it in a simpler method. But, the scenario is quite complex, it requires a different kind of different schemes, that means a complicated optimization techniques are required. However, it is required something faster, if it is not guaranteed to be exactly optimal to ensure to make the quick decision.

(Refer Slide Time: 40:05)

4. Dynamic reallocation of bandwidth

- **The demands on the network change over time.** So to make continual decisions about what traffic is highest priority to move across which links at a given moment is a challenge with linear programming to make quick decisions.
- So these are online algorithms. But they're not online in the same way as things inside the data center might be.
- For example, Google runs its traffic engineering 500 or so times a day. So, it's not as fine grained as things we might need inside a data center. Traffic between these facilities is relatively stable it seems.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, the fourth one is called dynamic reallocation the bandwidth. The demands from the network changed over the time. So, to make the continual decisions about the traffic is the highest priority to move across, which link at a given moment is a challenge with the linear programming to make this particular decisions. These are online algorithms. But, with the data center they are not online they are not online in terms of fine grain. But, they are doing this traffic engineering 500 times in a particular day, and it is not the fine grained as the things inside the data center.

(Refer Slide Time: 40:47)

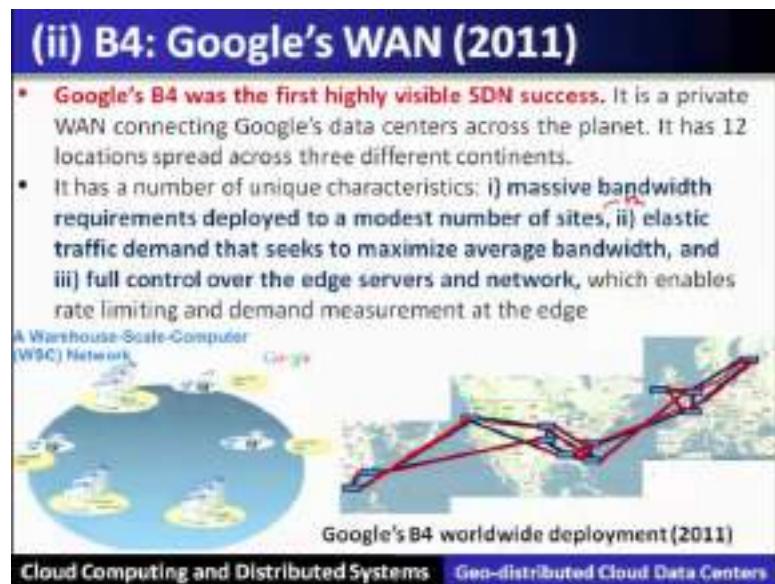
5. Edge rate limiting

- The commonality in the architecture is to implement an **enforcement of the flow rates**.
- So when the traffic enters the network will do that at the edge, rather than at every hop along the path.

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, fifth one is called edge rate limiting. The commonality in the architecture is to implement an enforcement of flow rates. So, when the traffic enters the network and will do that at the edge only, rather than at every half along the network; as, we have seen an MPLS that ingress and egress there are two routers, which has to deal with all the flow or a traffic engineering.

(Refer Slide Time: 41:16)



Now, let us see how these aspects are covered in Google's wide area network that is B4, as far as 2011 figure is concerned. So, Google's B4 was first highly visible software defined networking, which is applied to the wide area networks. So, it is a private wide area network connecting Google's data center across the planet, which is spread over twelve locations and in three different continents. It has number of unique characteristics first is the massive bandwidth requirements deployed to the modest number of sites, so that means, here how many sites are there the 12 different sites. So, lot of traffic here is seen across 12 different sites.

Now elastic traffic demand that seeks to maximize the average bandwidth and full control over the edge servers and the network; which enables the rate limiting and demand measurements at the edge; so, here you can see in the picture, there are 12 different locations, and they are connected with the high bandwidth links. There are few edges, which will basically see lot of traffic movement across inter-data center networks.

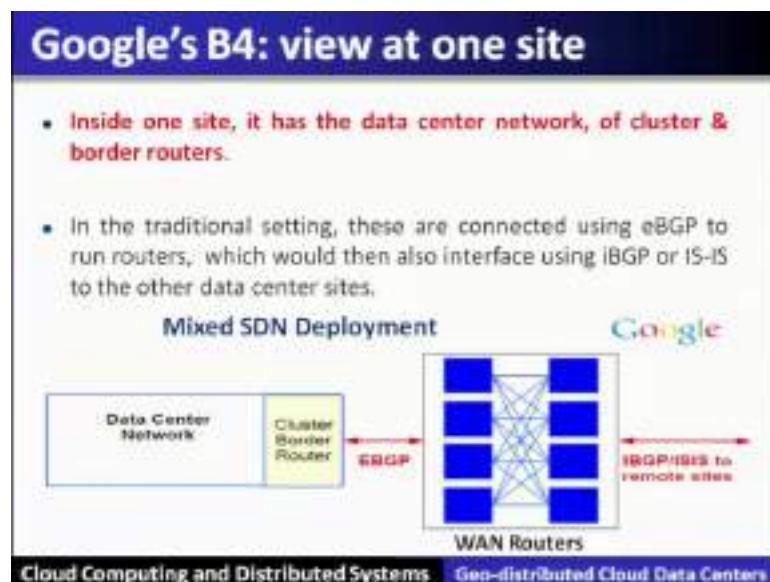
And this is all owned by the Google, so let us see how they have exploited this use of optimizing the wide area network.

(Refer Slide Time: 43:05)



Now, let us see what happens at inside, one data center. And that will be basically without laws of generality at all other data centers will manage.

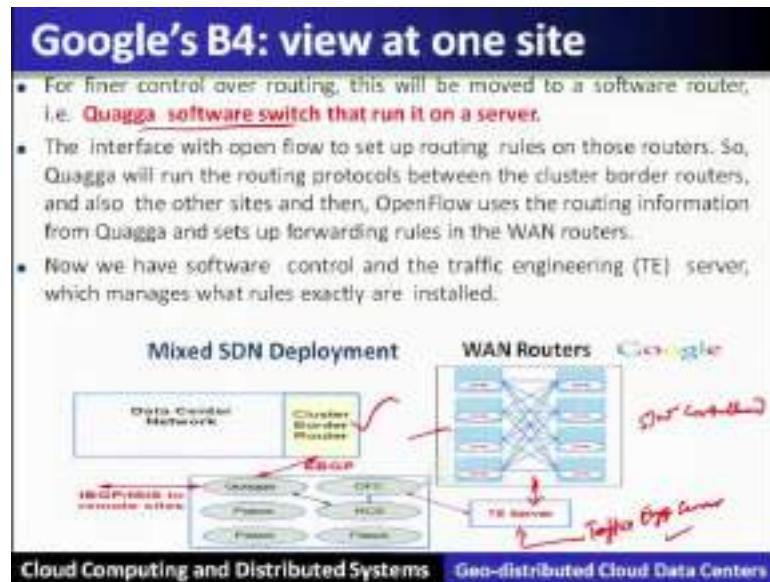
(Refer Slide Time: 43:19)



So, inside one data center nothing but a cluster and the border routers or a cluster border routers. So, this is a cluster border router in one data center, which is connected to the wide area network using EBGP. So, cluster border router using EGP is connected to the

wide area network routers, and which intern using IBGP and IS-IS, it is connected to the remote sites. So, in the traditional settings these are connected using EBGP to run the routers, which would then interface with IBGP or IS-IS for other data centers.

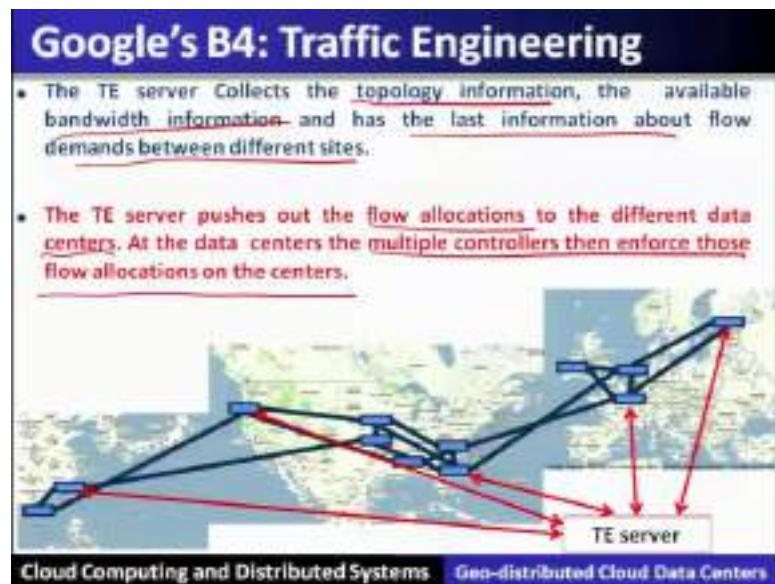
(Refer Slide Time: 44:09)



So, for final control over the routing, this will be move to the software router, meaning to say that this particular WAN router will be now the software controlled router. So, the software is basically pulled at one place and that is called a traffic engineering server, which will contains the entire logic. And so this particular software is are the Quagga software switch that runs on this particular server.

So, the interface with the open flow to setup the routing rules on these routers. So, this is an open flow routing which will; so the Quagga will run the routing protocols between the cluster border routers and also other sites. And open flow uses the routing information from Quagga and sets of the forwarding rule in the wide area network routers. Now, we have a software control and the traffic engineering server, which manages what exactly are installed over here.

(Refer Slide Time: 45:41)



So, let us see the details of the traffic engineering. So, traffic engineering server collects the topology information, available the available bandwidth and the last information about the flow demands between different sites. So, there are three different kind of information, which traffic engineering will collect from all different locations, they are the topology information and available bandwidth information that was already done in MPLS. But, besides that the last information about the flow demands between different sites that also is collected.

So, the traffic engineering server pushes out the flow allocation to different data centers, that means, after collecting it will compute the flow allocation, these flow elevations will be pushed back to the different data centers. So, at the data centers these multiple controller then force these flow allocation to the centers.

(Refer Slide Time: 46:49)

Google's B4: Design Choices

- **BGP routing as "big red switch":**
- They also keep available the BGP forwarding state because each of their switches allows them to have multiple forwarding tables at the same time they can afford to have BGP forwarding tables.
- Also, in addition to the traffic engineering, if the traffic engineering scheme does not work, they can discard those routing tables and use the BGP forwarding state instead. **So the BGP routing tables serve as a big red switch.**

The slide features a photograph of a custom-built switch on the left and a network topology diagram on the right. The topology diagram shows a 'Spine' layer at the top with four green nodes, connected via a 'Backplane' to a 'Leaf' layer below, which contains 128 nodes arranged in a grid. An arrow points from the Spine layer to the text '96 x 10G interchassis switches'. The diagram is labeled '128 x 10G' at the bottom.

Figure: A custom-built switch and its topology

Cloud Computing and Distributed Systems | Geo-distributed Cloud Data Centers

Now, if we see that the entire process of doing this is nothing but it looks like a big switch, which does this kind of BGP routing across all the data centers, that is the wide area network implementation using software; software for wide area network.

(Refer Slide Time: 47:08)

Google's B4: Design Choices

- **Hardware: custom, cheap, simple:**
- In 2011, there were not many software defined networking switches on the market. **So Google built their own using cheap commodity equipment.**

The slide features a photograph of a custom-built switch on the left and a network topology diagram on the right. The topology diagram shows a 'Spine' layer at the top with four green nodes, connected via a 'Backplane' to a 'Leaf' layer below, which contains 128 nodes arranged in a grid. An arrow points from the Spine layer to the text '96 x 10G interchassis switches'. The diagram is labeled '128 x 10G' at the bottom.

Figure: A custom-built switch and its topology

Cloud Computing and Distributed Systems | Geo-distributed Cloud Data Centers

Now, here we can see that doing this will not required to have to do this will require a cheap commodity equipments.

(Refer Slide Time: 47:24)

Google's B4: Design Choices

- Software smart with safeguard:
 - Most of these smarts then reside in the software.
 - Switches are simple in the software, which is the **open flow control logic** at each site replicated for fault tolerant using paxos.
 - Further, for scalability of this system they use a hierarchy of controllers. **The software solution achieves nearly 100% utilization and solves the traffic problem in 0.3 seconds.**

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Because, most of the things are done in the software that means, software all the controls are done through the software. So, the switches are or simple, which are nothing but and open flow logic at each site replicated for fault tolerant using paxos. Further, the scalability of the system is ensured by hierarchy of controllers and the software solution achieves 100 % utilization and solves the traffic problem traffic engineering problem in 0.3 seconds.

(Refer Slide Time: 47:58)

Google's B4: Design Choices

- **Hierarchy of controllers:**
 - At the **top level** we have the **global controller**, which is talking to an **SDN gateway**. The gateway can be thought of as a **super controller** that talks to the controller's at all these different data center sites.
 - Each site might itself have **multiple controllers**, because of the scale of these networks. **This hierarchy of controllers simplifies things from the global perspective.**



Figure: B4 architecture overview

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Let us see the hierarchy of controller means that at the top level, we have the global controllers. So, all these are the global controllers for different sites, which is talking to an SDN gateway. So, they are talking to the SDN gateway. So, the gateway can be thought of a super controller that talks to the controllers at all different sites of the data center so, this forms a hierarchy of controllers. Now, each site might itself have multiple control, because of the scale of the network. This hierarchy controller simplifies the things from global perspective.

(Refer Slide Time: 48:54)

Google's B4: Design Choices

- **Aggregation: flow groups, link groups:**
- Earlier, traffic engineering at this global scale is not at the level of mutual flows but of flow groups. That also helps scaling. Further, each pair of sites is not connected by just one link.
- These are massive capacity links that are formed from a trunk of several parallel high capacity links.
- All of these are aggregated and exposed to the traffic engineering layer as one logical link. **It is up to the individual site to the partition traffic, multiplex and demultiplex traffic across the set of links.**

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Now, then this particular hierarchy of controller will do the flow groups will form the flow groups and also will ensure the link groups. So, the traffic engineering will be confined or will focus on the flow groups not at a particular flow. So, the traffic engineering will be done at this globally scale and it is not at the level of mutual flows, but of the flow groups. So, these massive capacity link that are formed from the trunks of several high capacity links are now going to be utilized using this particular flow groups.

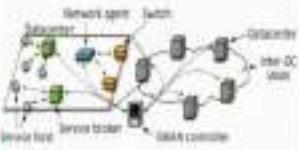
(Refer Slide Time: 49:46)

(iii) Microsoft's Swan

- Microsoft has publicly disclosed the design for optimizing wide area traffic flow in their WAN.
- Interesting feature in this design is the way **to make changes to the traffic flow without causing congestion.**

ACM SIGCOMM, 2013
Achieving High Utilization with Software-Driven WAN
Dix-Yan Hong (✉), Sanket Kandula, Ravi Ramamoorthi, Ning Zhang, Uday Sridhar, Michael Nonneng, Roger Wattenhofer
Institute for Data Communications

Architecture of Microsoft's Swan
Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

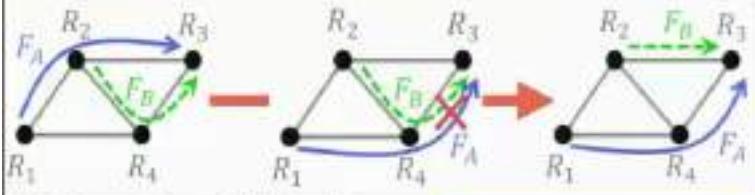


Now, here we will see another approach, which is called Swan approach in the Microsoft. How, they are going to utilize the bandwidth using software defined wide area network. Here is important feature is that to make the changes to the traffic flow without causing the condition.

(Refer Slide Time: 50:07)

Microsoft's Swan

- The network on the left have two flows, the blue one and the green one that are each using in this simple example 100% of the bandwidth of the links that they flow across.
- To move from the left scenario to the right scenario, if we do using any naive way, because we can't control exactly when every packet will flow across every link. There's going to be some period of time where, due to timing uncertainty, we have both of the flows sharing to some extent one of the links.



Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, let us see the broad idea over here is that there are two different flows F A and F B, which are shown with the green color. They might be sharing some of the common elements of the routing network. Now, instead of that we do not know at what time they

will be sharing the bandwidth. So, instead of that if these flows are routed through two different disjoint paths, then they may be independent, and can share the bandwidth as much as they can without.

(Refer Slide Time: 50:53)

Microsoft's Swan

- Can think of different ways to do this but we always going to run into some link that may get used by both flows at the same time. So, take a look at this design for SWAN there's an approach to making those updates with a certain amount of spare capacity so that congestion can be avoided.
- This approach that takes the optimization one step further **so that providing a pretty strong guarantee on lack of congestion even while the network data flow of changing.**

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

So, doing this kind of global flow control, it is also possible to reduce without means without congestion, it is going to solve the bandwidth utilization issue.

(Refer Slide Time: 51:13)

Conclusion

- In this lecture, we have discussed the **geo-distributed cloud data centers**, interaction of data centers with users and other data centers.
- Also discuss various data center interconnection techniques such as (i) **MPLS** (ii) **Google's B4** and (iii) **Microsoft's Swan**

Cloud Computing and Distributed Systems Geo-distributed Cloud Data Centers

Conclusion, in this lecture, we have discussed geo-distributed cloud data center; that is the interaction of data center to realizing to ensure the services of the applications to the

users. We have also covered the data center interconnection such as the traditional approaches used that is called MPLS and also the newer approaches, which is in the form of Google's B4 and Microsoft's Swan.

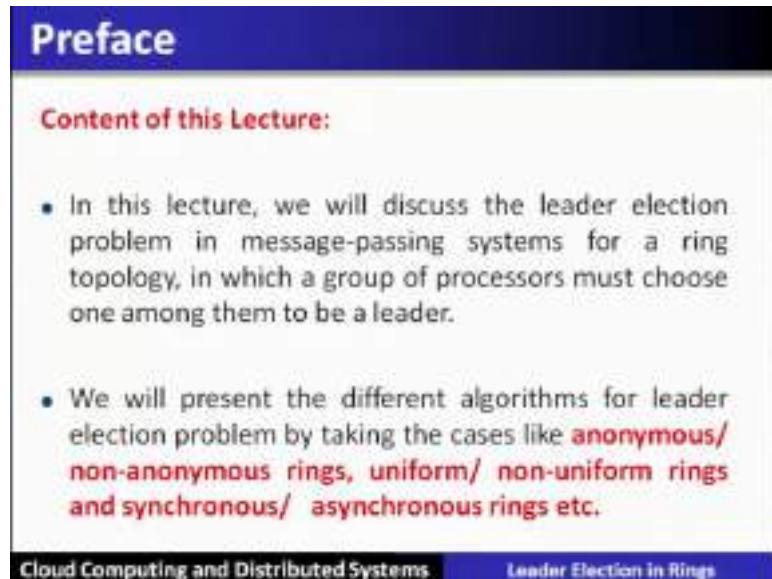
Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 07
Leader Election in Rings (Classical Distributed Algorithms)

Leader Election in the Rings Classical Distributed Algorithms. In this lecture we will discuss leader election problem in a message passing systems.

(Refer Slide Time: 00:24)



The slide has a blue header bar with the word "Preface" in white. Below it is a white content area. At the bottom, there is a blue footer bar with two items: "Cloud Computing and Distributed Systems" on the left and "Leader Election in Rings" on the right.

Preface

Content of this Lecture:

- In this lecture, we will discuss the leader election problem in message-passing systems for a ring topology, in which a group of processors must choose one among them to be a leader.
- We will present the different algorithms for leader election problem by taking the cases like **anonymous/ non-anonymous rings, uniform/ non-uniform rings and synchronous/ asynchronous rings etc.**

Cloud Computing and Distributed Systems Leader Election in Rings

Especially, for the ring topology in which the group of processors must choose one of them to be the leader. Different algorithms for leader election, different scenarios such as anonymous non-anonymous rings, uniform non-uniform rings, synchronous and asynchronous rings.

(Refer Slide Time: 00:49)

Leader Election (LE) Problem: Introduction

- The leader election problem has several variants.
- LE problem is for each processor to decide that either it is **the leader or non-leader**, subject to the constraint that exactly one processor decides to be the leader.
- LE problem represents a general class of **symmetry-breaking** problems.
- For example, when a deadlock is created, because of processors waiting in a cycle for each other, the deadlock can be broken by electing one of the processor as a leader and removing it from the cycle.

Cloud Computing and Distributed Systems

Leader Election in Rings

Leader election problem; the leader election problem has several variants, leader election is for each process to decide either it is the leader or a non-leader; subject to the constraint that exactly one processor decides to be a leader.

So, leader election problem represents a general class of symmetry breaking problems. For example, when a deadlock is created one of the processors waiting in a cycle for each other, the deadlock can be broken by electing one of the waiting processes as a leader and removing it from the cycle; that is breaking up the deadlock.

(Refer Slide Time: 01:23)

Leader Election: Definition

- Each processor has a set of **elected (won)** and **not-elected (lost)** states.
- Once an elected state is entered, processor is always in an elected state (and similarly for not-elected); i.e., irreversible decision
- In every admissible execution:
 - every processor eventually enters either an elected or a not-elected state
 - exactly one processor (**the leader**) enters an elected state

Cloud Computing and Distributed Systems

Leader Election in Rings

So, the leader election problem definitions, definition each process or each processor has a set of elected and a non-elected states. Once an elected state is entered the processor always in the elected state. And similarly for non-elected, that is irreversible decisions. So, in every admissible execution every processor eventually enters either as elected or non-elected state exactly one processors; are exactly one processor that is the leader enters into the elected state.

(Refer Slide Time: 01:59)

Uses of Leader Election

- A leader can be used to coordinate activities of the system:
 - find a **spanning tree** using the leader as the root
 - reconstruct a **lost token** in a token-ring network
- In this lecture, we will study the leader election in rings.

Now there are different uses of leader election algorithm. So, leader election can be used to coordinate the activities in the system. For example, to find out the spanning tree in a system requires a leader to be known which is called a root. Hence, if a root is given finding a spanning tree becomes easier.

Similarly, in a token ring system if a token is lost; that means, there is no leader so, it can elect a leader and restart the system with a token.

(Refer Slide Time: 02:42)

Definition: (1)Ring Networks

- In an **oriented ring**, processors have a consistent notion of left and right

1 = left = clockwise
2 = right = counter-clockwise

- For example, if messages are always forwarded on channel 1, they will cycle clockwise around the ring

Cloud Computing and Distributed Systems Leader Election in Rings

So, in this lecture we will study the leader election in a ring. So, let us define different terms which are used in the leader election problem; that is the ring problem. So, the first definition is about the ring networks.

So, in an oriented ring, the processors have a consistent notion of left and right. So, in this particular example, we can see here the label one called as a left side, if it is used to forward the message in this particular direction that is called a clockwise direction. Then this particular way the ring will be oriented in a clockwise manner. Similarly, if let us say that if p_0 and other processes they basically use the right side, that is the label number 2, always then it will be a counter clockwise and the ring will be oriented in that manner such rings are called oriented rings.

Now another definition is about anonymous rings.

(Refer Slide Time: 04:02)

Definition: (2) Anonymous Rings

- How to model situation when **processors do not have unique identifiers?**
- **First attempt:** require each processor to have the same state machine

So, if the processors do not have the unique ids; that means they are not given an ids, then that particular ring is called anonymous rings. So, in that situation each processor is like same running state machine there is no distinction.

(Refer Slide Time: 04:24)

Definition: (3) Uniform (Anonymous) Algorithms

- A **uniform** algorithm **does not use the ring size** (same algorithm for each size ring)
 - Formally, every processor in every size ring is modeled with the same state machine
- A **non-uniform** algorithm **uses the ring size** (different algorithm for each size ring)
 - Formally, for each value of n , every processor in a ring of size n is modeled with the same state machine A_n .
- Note the lack of unique ids.

Third definition is about uniform anonymous algorithms. So, uniform algorithm means that it does not use the information of the ring size that is n ; the number of nodes in the ring. So, formally every processor in every size ring is modeled with the same state

machine. So, the algorithm which is called a non-uniform algorithm we will use the size of the ring that is an n in the algorithm.

(Refer Slide Time: 05:03)

Impossibility: Leader Election in Anonymous Rings

Theorem: There is **no leader election algorithm for anonymous rings**, even if
algorithm knows the ring size (non-uniform) and synchronous model

Proof Sketch:

- Every processor begins in same state with same outgoing messages (since anonymous)
- Every processor receives same messages, does same state transition, and sends same messages in round 1
- Ditto for rounds 2, 3, ...
- Eventually some processor is supposed to enter an elected state. But then they all would.

Cloud Computing and Distributed Systems Leader Election in Rings

There is a impossibility result which says that about the leader election anonymous rings theorem. There is no leader election algorithm for anonymous rings even if the algorithm knows the ring size that is for the non-uniform. And also the model is synchronous model, why; because every processor begins in the same state with the same outgoing message. Since it is anonymous, and each processor when receive the message will also be in the same state transition, and sends the message in a round one.

(Refer Slide Time: 05:48)

Leader Election in Anonymous Rings

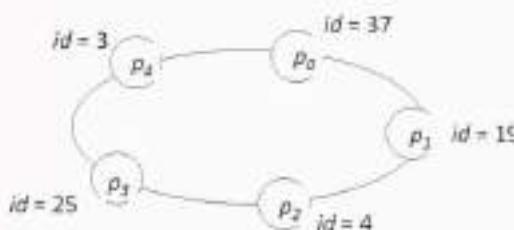
- Proof sketch shows that either safety (never elect more than one leader) or liveness (eventually elect at least one leader) is violated.
- Since the theorem was proved for non-uniform and synchronous rings, the same result holds for weaker (less well-behaved) models:
 - uniform
 - asynchronous

So, there is no distinction and there is all symmetry looking up in this particular way. So, it shows that either the safety or the liveness is violated here in this case. Hence the theorem was proved for non-uniform and synchronous rings. The same result hold for a weaker model that is for the uniform and the asynchronous model. So, rings with the ids. So, we will assume that each processor has the unique id, ok. Let us see a example of a ring.

(Refer Slide Time: 06:33)

Specifying a Ring

- Start with the smallest id and list ids in clockwise order.



- Example: 3, 37, 19, 4, 25

Here we see that every processor in the ring is having different ids for example, 3, 37, 19, 4 and 25.

(Refer Slide Time: 06:46)

Uniform (Non-anonymous) Algorithms

- **Uniform** algorithm: there is one state machine for every id, no matter what size ring
- **Non-uniform** algorithm: there is one state machine for every id and every different ring size
- These definitions are tailored for leader election in a ring.

Cloud Computing and Distributed Systems

Leader Election in Rings

So, uniform algorithms means that the algorithm is not using the size of the ring. So, no matter what is the size of a ring the algorithm in the algorithm. Similarly, non-uniform rings shows that it uses the size of the ring.

(Refer Slide Time: 07:04)

$O(n^2)$ Messages LE Algorithm: LeLann-Chang-Roberts (LCR) algorithm

- send value of own id to the left
- when receive an $id j$ (from the right):
 - if $j > id$ then
 - forward j to the left (this processor has lost)
 - if $j = id$ then
 - elect self (this processor has won)
 - if $j < id$ then
 - do nothing

Cloud Computing and Distributed Systems

Leader Election in Rings

Let us see the algorithm which is called a Lelann-Chang-Roberts algorithm; LCR algorithm for leader election problem. This is also called as a $O(n^2)$ leader election

algorithm. Here algorithm says that every processor will send its id to the left. When it receives an id from the right, and if $j > id$, then it will forward to the left. If $j < id$, then it will not do anything it will not forward it will swallow. And if $id = j$ then it will elect itself as the leader.

(Refer Slide Time: 07:53)

Analysis of $O(n^2)$ Algorithm

Correctness: Elects processor with largest id.

- message containing largest id passes through every processor

Time: $O(n)$

Message complexity: Depends how the ids are arranged.

- largest id travels all around the ring (n messages)
- 2nd largest id travels until reaching largest
- 3rd largest id travels until reaching largest or second largest etc.

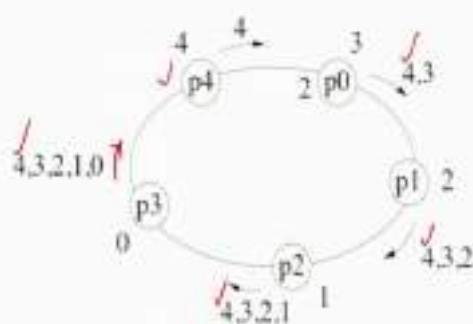
Cloud Computing and Distributed Systems

Leader Election in Rings

We will see through an example. As far as correctness is concerned, it will always elect a processor with a largest id in the time of the $O(n)$ and the message complexity is $O(n^2)$, let us see through an example.

(Refer Slide Time: 08:07)

Analysis of $O(n^2)$ Algorithm



Cloud Computing and Distributed Systems

Leader Election in Rings

Now let us see that the processor p3 will send its id to the left p4 on receiving 0 it will not forward it further. Whereas, p4 will send its own id, and when it reaches to p0, its id > 3 so, it will be forwarded further. When it reaches to p1 it will also be forwarded. And it will be forwarded, and when it reaches to this particular point because this is the highest. So, it will see its own message so, hence this will be elected as a leader.

Whereas, the other messages having lower ids will be absorbed so, let us see when this order n square algorithm will arise.

(Refer Slide Time: 09:21)

Analysis of $O(n^2)$ Algorithm

- Worst way to arrange the ids is in decreasing order:
 - 2nd largest causes $n - 1$ messages
 - 3rd largest causes $n - 2$ messages etc.
- Total number of messages is:
$$n + (n-1) + (n-2) + \dots + 1 = \Theta(n^2)$$



Cloud Computing and Distributed Systems Leader Election in Rings

Let us see that the messages are being forwarded here in this particular scenario. Let us see this particular method of analyzing the worst case scenario, that is of the order n square.

(Refer Slide Time: 09:46)

Analysis of $O(n^2)$ Algorithm

- Clearly, the algorithm never sends more than $O(n^2)$ messages in any admissible execution. Moreover, there is an admissible execution in which the algorithm sends $\Theta(n^2)$ messages; Consider the ring where the identifiers of the processor are $0, \dots, n-1$ and they are ordered as in Figure 3.2. In this configuration, the message of processor with identifier i is send exactly $i+1$ times. Thus the total number of messages, Fig 3.2 Ring with $\Theta(n^2)$ messages, including the n termination messages, is

$$n + \sum_{i=0}^{n-1} (i+1) = \Theta(n^2)$$

$\Theta(n^2)$ Clockwise Unidirectional Ring

Cloud Computing and Distributed Systems Leader Election in Rings

We will consider the ring where the identifiers of the processor 0 1 2 and so on up to $n-1$ and are ordered in this particular manner. Here the identifier i is send exactly i plus 1 times. So, for example, 0 will be send 0 + 1 that is only one time it will be observed here. 1 that is $i + 1$ times it will forwarded. 1 will be forwarded by 0; that means, it will be forwarded 2 times and so on. So, 2 will be forwarded 3 times, 3 will be forwarded 4 times and so on. So, this is the orientation of the ring.

How many number of messages are being forwarded? Here so, we will see that for i it will be forwarded $i + 1$ times. So, if we sum for the nodes from 0 to $n - 1$. So, these are the total number of messages which will be forwarded in this particular worst case scenario in the ring. And when a leader is elected then the highest id will circulate it is message that is n . Now, this particular summation is of the order n square.

(Refer Slide Time: 11:49)

Can We Use Fewer Messages?

- The $O(n^2)$ algorithm is simple and works in both synchronous and asynchronous model.
- But can we solve the problem with fewer messages?

Idea:

- Try to have messages containing smaller ids travel smaller distance in the ring

non-uniform $O(n^2)$ LE 

Cloud Computing and Distributed Systems

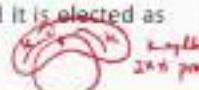
Leader Election in Rings

So, it requires n^2 number of messages in the worst situation we have shown in this analysis. Algorithm which is of the order n square algorithm is simple and works in both synchronous and asynchronous model. Another good thing about this algorithm is does not use the value of n or the size of n . Hence, it is non-uniform, hence it is uniform. So, as it is shown that it will work in both the situation either for the synchronous model or for asynchronous model, anonymous order n square leader election algorithm.

(Refer Slide Time: 12:44)

$O(n \log n)$ Messages LE Algorithm: The Hirschberg and Sinclair (HS) algorithm

- To describe the algorithm, we first define the ***k*-neighbourhood** of a **processor p** , in the ring to be the set of processors that are at **distance at most k from p , in the ring** (either to the left or to the right). Note that the ***k*-neighbourhood** of a processor includes exactly **$2k+1$** processors.
- The algorithm operates in **phases**; it is convenient to start numbering the phases with **0**. In the ***k*th phase** a processor tries to become a **winner** for that phase; to be a winner, it must have the **largest id** in its ***2k+1*-neighborhood**. Only processors that are winners in the ***k*th phase** continue to compete in the **$(k+1)$ st phase**. Thus fewer processors proceed to higher phases, until at the end, only one processor is a winner and it is elected as the leader of the whole ring.



Cloud Computing and Distributed Systems

Leader Election in Rings

Now, can we do with a lesser message complexity than order n square? We will see an algorithm which is given by Hirschberg and Sinclair HS algorithm, it is called it is order $n \log n$ leader election algorithm. This algorithm uses the concept of k neighbourhood for any processor p i in the ring. That is nothing but distance of at most k. It is (Refer Time: 13:09) and this is k this is k and this is one that is $2k + 1$ nodes for the processors in the k neighbourhood of a processor.

So, k neighbourhood of a processor includes exactly $2k + 1$ processors. This algorithm operates in the phases. Therefore, it is convenient to start numbering the phases starting from the phase 0. So, the kth phase of a process will try to become the winner, for that phase to be the winner it must have the largest id in its 2^k neighbourhood. So, only the processor that are the winners of kth phase will continue to participate in $k + 1$ th phase. Thus, fewer processor proceeds to a higher phase until at the end only one processor is the winner and is elected as the leader of the whole ring.

(Refer Slide Time: 14:29)

The HS Algorithm: Sending Messages

Phase 0

- In more detail, in phase 0, each processor attempts to become a phase 0 winner and sends a <probe> message containing its identifier to its **1-neighborhood**, that is, to each of its two neighbors.
- If the identifier of the neighbor receiving the probe is greater than the identifier in the probe, it swallows the probe; otherwise, it sends back a <reply> message.
- If a processor receives a reply from both its neighbors, then the processor becomes a phase 0 winner and continues to phase 1.

So, let us see the phase 0, in more detail phase 0 each processor attempts to become of a phase 0 winner, and sends a probe message containing its id to its one of neighbourhood to each of the 2 neighbours so for example if this is the neighbourhood and to one neighbourhood on both the sides in phase 0 if the [FL].

[FL].

[FL], If the id identifier of the neighbour receiving the probe is greater than the identifier in the probe, it swallows the probe, otherwise it sends back the reply.

So, if it gets the replies back from both the end; that means, it has become the winner of phase 0 and it will continue to the phase 1. So, in the phase 1 the neighbourhood size will basically double. If we say that it is phase k; that means, all the winners of phase k-1 will send its probe message with its id to its 2^k neighbourhood one in each direction. Each message traverses 2^k processor one by one and the probe is swallowed by the processor if it contains an id that is smaller than its own id.

If the probe arrives at the last processor on the neighbourhood without being swallowed, then the last processor will send back the reply as we have seen in the first or the 0th phase.

(Refer Slide Time: 16:17)

The HS Algorithm: Sending Messages

Phase k

- In general, in phase k, a processor p_i that is a phase **k-1** **winner** sends **<probe>** messages with its identifier to its **2^k -neighborhood** (one in each direction). Each such message traverses **2^k processors** one by one. A probe is swallowed by a processor if it contains an identifier that is smaller than its own identifier.
- If the probe arrives at the last processor on the neighbourhood without being swallowed, then that last processor sends back a **<reply>** message to p_i . If p_i receives replies from both directions, it becomes a phase k winner, and it continues to phase **k+1**. A processor that receives its own **<probe>** message terminates the algorithm as the leader and sends a termination message around the ring.

Cloud Computing and Distributed Systems Leader Election in Rings

So, this is the complete algorithm for the processor i and this algorithm will be for all the processor that is from 0 to n - 1.

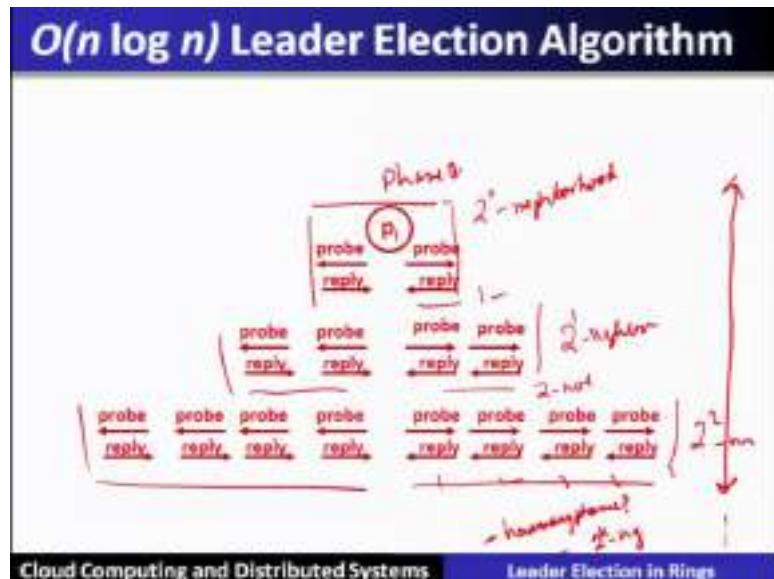
(Refer Slide Time: 16:28)

```
Algorithm 5 Asynchronous leader election: code for processor  $p_i$ ,  $0 \leq i < n$ .  
Initially,  $asleep = \text{true}$ .  
1: upon receiving no message:  
2:   if  $asleep$  then  
3:      $asleep = \text{false}$   
4:     send  $\langle \text{probe}, id(0,1) \rangle$  to left and right.  
5: upon receiving  $\langle \text{probe}, j, k, d \rangle$  from left (resp., right):  
6:   if  $j = id$  then terminate as the leader  
7:   if  $j > id$  and  $d < 2^k$  then // forward the message  
8:     send  $\langle \text{probe}, j, k, d+1 \rangle$  to right (resp., left) // increment hop counter  
9:   if  $j > id$  and  $d \geq 2^k$  then // reply to the message  
10:    send  $\langle \text{reply}, j, k \rangle$  to left (resp., right)  
11:   // if  $j < id$ , message is swallowed  
12: upon receiving  $\langle \text{reply}, j, k \rangle$  from left (resp., right):  
13:   if  $j \neq id$  then send  $\langle \text{reply}, j, k \rangle$  to right (resp., left) // forward the reply  
14:   else // reply is for own probe  
15:     if already received  $\langle \text{reply}, j, k \rangle$  from right (resp., left) then  
16:       send  $\langle \text{probe}, id, k+1, 1 \rangle$  // phase  $k$  winner
```

So, here we will see that the algorithm, it will send this particular probe to the left and right, and it will do the election it will initiate multiple elections in the k neighbourhood; that means, the entire ring is divided into k neighbourhood and this elections will parallelly run for the k th neighbourhood.

So, we see that that the size of the neighbourhood will double in each phase. So, if a probe reaches a node with a largest larger id the probe will stop.

(Refer Slide Time: 17:18)



If the probe reaches the end of the neighborhood, then the reply will be sent back to the initiator. So, this can be seen through this particular diagram that this is phase 0; that means this particular one half neighbourhood will perform the elections. And whose ever will be the winner then the size will be doubled that is 2^1 neighbourhood. So, that means, 2 this was earlier 1 node neighbourhood, now it is having 2 nodes in the neighbourhood.

And whosever and if it is able to win for 2^1 neighbourhood, then the neighbourhood size will double, that is 2^2 neighbourhood; that means, 1 2 3 4 different so, the neighbourhood size will keeps on growing in this particular manner. Now we will count what is the depth; that means how long how many different phases will be there, how many phases will be there and in each phase, you know that it will be 2^k neighbourhood. So, in each phase how many such elections will happen? And these particular information is required to compute the message complexity.

(Refer Slide Time: 19:10)

Analysis of $O(n \log n)$ Leader Election Algorithm

Correctness:

- Similar to $O(n^2)$ algorithm.

Message Complexity:

- Each message belongs to a particular phase and is initiated by a particular processor
- Probe distance in phase k is 2^k
- Number of messages initiated by a processor in phase k is at most $4 * 2^k$ (probes and replies in both directions)

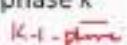


Cloud Computing and Distributed Systems Leader Election in Rings

So, the message complexity, that means, here the probe distance is in a phase k is 2^k . So, the number of messages initiated by a process in a phase k is at most $4 * 2^k$; that means if this is the k phase it is neighbourhood. So, the message will go one times, the reply will come back 2 times, then message will go on the right side third time and it will come back to the 4 times. So, $4 * 2^k$ in every phase 4 times these probes will be sent in counting the message complexity.

(Refer Slide Time: 19:55)

Analysis of $O(n \log n)$ Leader Election Algorithm

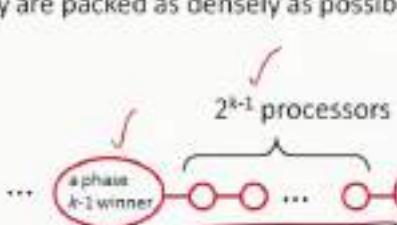
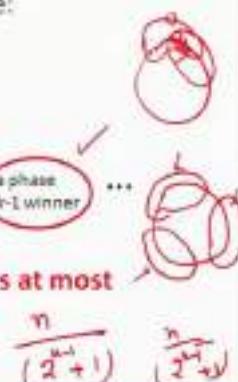
- How many processors initiate probes in phase k ? 
- For $k = 0$, every processor does 
- For $k > 0$, every processor that is a "winner" in phase $k - 1$ does
 - "winner" means has largest id in its 2^{k-1} neighborhood

Cloud Computing and Distributed Systems Leader Election in Rings

Now, the question is how many such processor will initiate the probe in the phase k . Meaning to say that if this is the k hop so, how many such packing of k hops will be initiated at a particular phase k . Now for $k = 0$ every processor will be participating in this particular election process when $k > 0$; that means, the winner of phase $k - 1$ only participate in k th phase.

(Refer Slide Time: 20:58)

Analysis of $O(n \log n)$ Leader Election Algorithm

- Maximum number of phase $k - 1$ winners occurs when they are packed as densely as possible:
- Total number of phase $k - 1$ winners is at most $\frac{n}{(2^{k-1} + 1)}$ 

Cloud Computing and Distributed Systems Leader Election in Rings

So, the winner of $k - 1$ th phase means that the largest id in 2^{k-1} neighbourhood which is being seen. So, the maximum number of phase $k - 1$ winner occurs when they are packed as densely as possible.

So, let us see that here it is phase $k - 1$ winner and another phase $k - 1$ winner. So, how many number of processors which are in between 2 extreme phase $k - 1$ winner is nothing but 2^{k-1} winner. We can see through an example that if this is phase 0 so, it will have the one neighbourhood. So, this particular 0, this particular another node will also have this kind of neighbourhood. So, the number of nodes involved is basically 2^{k-1} . So, the total number of phase $k - 1$ winner will be $n / (2^{k-1} + 1)$.

That means this is the packing, this is the packing of so many $k - 1$. So, $n / (2^{k-1} + 1)$, this will be total number of such $k - 1$ winners who are participating in k th round. Now the question is how many phases are there. Now you know that every phase the number of winners is cut in a half that is from $n / (2^{k-1} + 1)$.

(Refer Slide Time: 22:40)

Analysis of $O(n \log n)$ Leader Election Algorithm

- How many phases are there? ✓
- At each phase the number of (phase) winners is cut approx. in half
 - from $n/(2^{k-1} + 1)$ to $n/(2^k + 1)$
- So after approx. $\log_2 n$ phases, only one winner is left.
 - more precisely, max phase is $\lceil \log(n-1) \rceil + 1$

$\frac{n}{2^k+1} = 1$ $\log_2 \frac{n-1}{2^k+1} \rightarrow \log(n-1) + 1$
 \downarrow \downarrow

Cloud Computing and Distributed Systems Leader Election in Rings

It will now go to $n / (2^{k+1} + 1)$, in the k th phase. So, we will see that we will continue in this manner so that finally, there will be only one such neighbourhood remains that is the entire ring. And if we compute then it becomes it comes out to be $\log(n-1) + 1$ total number of phases.

So, again we can see $n / (2^{k-1} + 1)$. Let us say this becomes 1 if we take the log of n, then it will become $k - 1$. So, this $k = \log(n-1) + 1$. So, this is how this particular formula is being obtained.

(Refer Slide Time: 24:14)

Analysis of $O(n \log n)$ Leader Election Algorithm

- Total number of messages is sum, over all phases, of number of winners at that phase times number of messages originated by that winner:

$$\leq 4n + n + \sum_{k=1}^{\log(n-1)+1} 4 \cdot 2^k \cdot n / (2^{k-1} + 1)$$

$$< 8n(\log n + 2) + 5n$$

$$= O(n \log n)$$

msg for phases 1 to $\log(n-1)+1$

phase 0 msgs
termination msg
msg for phases 1 to $\log(n-1)+1$

Cloud Computing and Distributed Systems
Leader Election in Rings

Now plugging up all the values we will find out the total number of messages in all the phases, that comes out to be this is the phase 0 messages, $4n$ and this is the termination message. And as far as this formula is concerned, this formula says that how many different messages are there for the phases 1 to $\log(n - 1)$.

So, this is the summation of different phases, and every phase will require $4 \cdot 2^k \cdot n / (2^{k-1} + 1)$,.. So, if you compute it comes out to be $8n(\log n + 2) + 5n$. That comes out to be of the $O(n \log n)$. So, we have seen that this particular algorithm will work both for synchronous and asynchronous case. So, the question is can we reduce the number of messages even further than order $n \log n$.

So, we will have seen that not in asynchronous model you can do this better than $O(n \log n)$; that for that we can show that this is the lower bound for the leader election problem that is of the $O(n \log n)$.

(Refer Slide Time: 25:38)

Lower bound for LE algorithm

But, can we do better than $O(n \log n)$?

Theorem: Any leader election algorithm for asynchronous rings whose size is not known a priori has $\Omega(n \log n)$ message complexity (holds also for unidirectional rings).

- Both LCR and HS are comparison-based algorithms, i.e. they use the identifiers only for comparisons ($<$, $>$, $=$).
- In synchronous networks, $O(n)$ message complexity can be achieved if general arithmetic operations are permitted (non-comparison based) and if time complexity is unbounded.

So, the theorem says that any leader election algorithm for asynchronous rings whose size is not known a priori has the lower bound of $n \log n$ message complexity, holds also for the unidirectional rings. Both LCR and HS are comparison based algorithm; that is, they use identifiers only for the comparison. In synchronous algorithms $O(n)$ message complexity can be achieved if general arithmetic operations are permitted, non-comparison based and if the time complexity is unbounded.

(Refer Slide Time: 26:29)

Overview of LE in Rings with Ids

- There exist algorithms when nodes have unique ids.
- We have evaluated them according to their **message complexity**.
- Asynchronous ring:**
 - $\Theta(n \log n)$ messages
- Synchronous ring:**
 - $\Theta(n)$ messages under certain conditions
 - otherwise $\Theta(n \log n)$ messages
- All bounds are asymptotically tight.

So, overview of leader election in rings with the ids, there exist algorithm when the nodes have unique ids. We have evaluated them according to their message complexity, and we found out that in the case of asynchronous ring the message complexity of the algorithm the best known algorithm gives $n \log n$ messages. Similarly, for synchronous rings we have seen that of the order n messages under certain conditions. Otherwise, the complexity is the same that is $n \log n$ messages in the comparison based algorithms.

So, all these bounds are asymptotically tight conclusion. This lecture provided an in depth study of the leader election problem in the message passing system for the ring topology.

(Refer Slide Time: 27:23)

Conclusion

- This lecture provided an in-depth study of the leader election problem in message-passing systems for a ring topology.
- We have presented the different algorithms for leader election problem by taking the cases like **anonymous/non-anonymous rings, uniform/non-uniform rings and synchronous/asynchronous rings**

Cloud Computing and Distributed Systems Leader Election in Rings

We are also presented different algorithm for the leader election by taking different cases. Such as anonymous oblique non-anonymous rings, uniform oblique non-uniform rings, synchronous oblique asynchronous rings.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 08
Leader Election (Ring LE & Bully LE)

Leader election and Bully Leader Election algorithm. Preface; content of this lecture.

(Refer Slide Time: 00:21)

Preface

Content of this Lecture:

- In this lecture, we will discuss the underlying concepts of '**leader election problem**', which has been very useful for many variant of distributed systems including today's Cloud Computing Systems.
- Then we will present the different '**classical algorithms for leader election problem**' i.e. **Ring LE** and **Bully LE algorithms** and also discuss how election is done in some of the popular systems in Industry such as **Google's Chubby** and **Apache's Zookeeper system**.



Cloud Computing and Distributed Systems | Leader Election (Ring LE & Bully LE)

We will discuss the underlying concepts of leader election problem, which has been very useful for many variants of distributed systems, and in today's cloud computing systems, we will present the different classical distributed algorithm for leader election problem, namely ring leader election, Bully's leader election problem and also we will discuss how this election is done in some of the popular systems, used in the industry; such as in Google's Chubby and in Apaches Zookeeper system. So, the need of the leader election.

(Refer Slide Time: 01:09)

Need of Election

- **Example 1:** Suppose your Bank account details are replicated at a few servers, but one of these servers is responsible for receiving all reads and writes, i.e., it is the **leader** among the replicas
 - What if there are two leaders per customer?
 - What if servers disagree about who the leader is?
 - What if the leader crashes?

Each of the above scenarios leads to Inconsistency

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

Leader election is an important problem in a distributed system. To understand the use of leader election, let us see through some examples. Suppose your bank account details are replicated in a few servers and out of these many servers one of them is responsible for receiving all the request for reading and writing operations; that is out of several replicated servers, one of them is responsible that is called leader, that is coordinating the work on behalf of all other replicas.

Now, in this particular scenario what will happen if there are two different leaders per customer, then it will be unclear or a confusion across the different replicas. that who is the leader or who is the coordinator, who is responsible to control all the operations. what if the server disagree about who the leader is that means, if there is no agreement among all the replicas about the server then the application will not be able to be coordinated, to perform the operations, as per the specification of the application.

What if the leader is crashed or is down. So, these different situations will lead to inconsistency in this situation of the leader and that requires to be resolved in the leader election problem.

(Refer Slide Time: 03:15)

Some more motivating examples

- **Example 2:** Group of NTP servers: who is the root server?
- **Other systems that need leader election:**
Apache Zookeeper, Google's Chubby
- Leader is useful for coordination among distributed servers

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

Similarly, there is another example of the group of servers in NTP that is network time protocol that we have seen among them one of them is called the root server. So, the one who is the root is the leader among all other servers. Similarly in other systems that need the leader elections are such as Apache Zookeeper and Google's Chubby system. So, the leader is useful for the coordination across the distributed servers that too we have seen in the cloud computing also comprises of several servers, so the coordination across all the servers is to be done through a server. So, that is the requirement for distributed and the cloud computing system.

(Refer Slide Time: 04:09)

Leader Election Problem

- In a group of processes, elect a **Leader** to undertake special tasks
 - And **let everyone know** in the group about this Leader
- **What happens when a leader fails (crashes)?**
 - Some process detects this (using a Failure Detector!)
 - Then what?
- **Goals of Election algorithm:**
 1. Elect one leader only among the non-faulty processes
 2. All non-faulty processes agree on who is the leader

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, the leader election problem, in a group of processes elect a leader to undertake the a special task and that everyone knows in the group who the leader is. And if the leader crashes then some process will detect this failure of a leader, may be sometimes using the failure detector and then elect out of the non faulty servers, one of them as the leader. So, the goal of leader election is to elect one leader among the non faulty processes and all non faulty processes agree on who is the leader.

(Refer Slide Time: 05:03)

System Model

- N processes.
- Each process has a unique id.
- Messages are eventually delivered.
- Failures may occur during the election protocol.

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, let us understand the system module which will be the assumption under which this particular problem will be formulated. Let us consider there are N processes. Each process has the unique id and the message are eventually delivered in a message passing system and the failures may occur during the election. Now any process can call a leader election process.

(Refer Slide Time: 05:35)

Calling for an Election

- Any process can **call** for an **election**.
- A process can call for **at most one election at a time**.
- Multiple processes are allowed to call an election simultaneously.
 - **All of them together must yield only a single leader**
- The result of an election should not depend on which process calls for it.

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

That means anyone can initiate, and a process can call at most one leader election at a time, so the multiple processes are allowed to call the leader election simultaneously, but they should lead to only one leader elected. So, the result of an election should not depend on which process calls it, and how many process are calling that leader election. At the end there should be one leader among the non faulty processes and everyone else should know who the leader is.

(Refer Slide Time: 06:18)

Formally: Election Problem

- A run of the election algorithm must always guarantee at the end:
 - **Safety:** For all non-faulty processes p : $(p \text{ is elected} \Rightarrow (\exists q: \text{a particular non-faulty process with the best attribute value}) \text{ or Null}$)
 - **Liveness:** For all election runs: (election run terminates)
 - & for all non-faulty processes p : $p \text{ is elected is not Null}$
- At the end of the election protocol, the non-faulty process with the **best (highest)** election attribute value is elected.
 - Common attribute : leader has highest id
 - Other attribute examples: leader has highest IP address, or fastest cpu, or most disk space, or most number of files, etc.

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, the leader election problem has to guarantee two properties; the first one is called safety property, the other is called liveness property. Safety property says that nothing bad happens that is for all non faulty processes p. So, p is elected one having the best attribute among all the other non faulty processes, this is called a safety condition. The second one is called liveness that eventually something good happens; that is for all election runs, that is election process terminates and for all non faulty processes, the p s elected is not null that means, the algorithm eventually terminates and where a leader elected among the non faulty processes.

So, at the end of the election protocol the non faulty processes with the highest attribute defined is elected as a leader. So, the common attribute over which the leader is elected, is for example, the non faulty processor having the highest id becomes the leader. The other attributes for the leader election which can be used is the highest IP address or the fastest CPU or the most disk space etcetera. Let us understand the classical leader election in a ring; this is called a ring election algorithm.

(Refer Slide Time: 07:59).

(i) Classical Algorithm: Ring Election

The Ring:

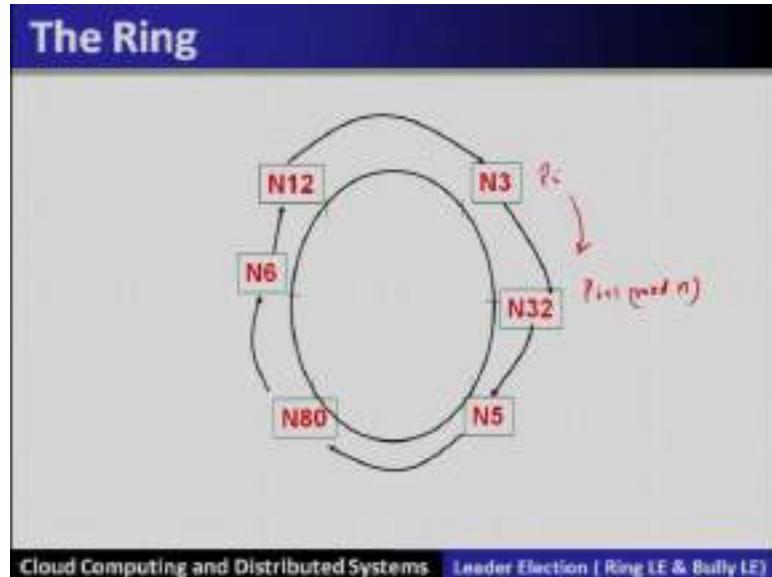
- N processes are organized in a logical ring
 - Similar to **ring in Chord p2p system**
 - i-th process p_i has a communication channel to $p_{(i+1) \text{ mod } N}$
 - All messages are sent clockwise around the ring.

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, the ring is found out of N processes, that is the logical ring which we have seen in the chord system. Here i th process that is p_i has the communication channel connection to $p_{(i+1) \text{ mod } N}$. So, this way every process p_i has its successor $p_{(i+1) \text{ mod } N}$ and therefore, defines a logical ring.

So, all the messages are sent in a clockwise around the ring. So, this is the ring which we have just defined that is p_i , will basically be able to communicate with $p_{(i+1)}$.

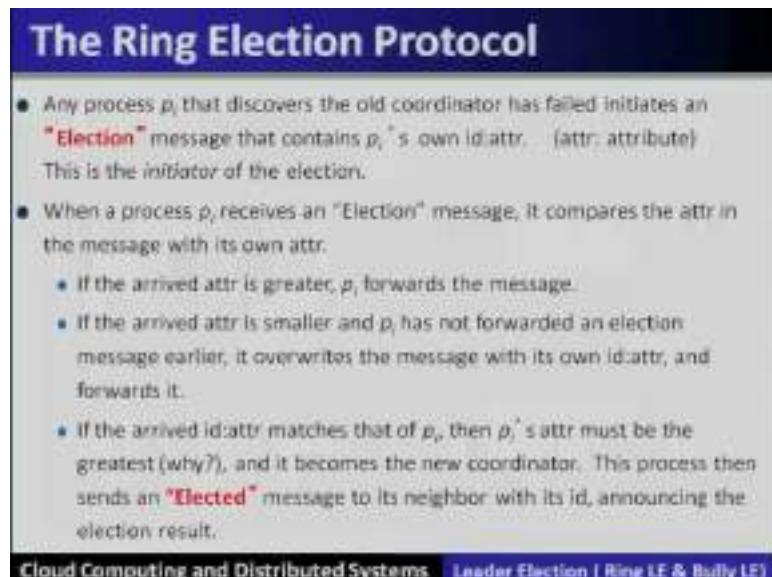
(Refer Slide Time: 08:50)



Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, N 3 will have its successor N 3 2 and N 3 2 is having a successor N 5 and so on. Finally, the node N 12 will have its successor N 3 and together they will form a logical ring.

(Refer Slide Time: 09:25)



Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, the ring election protocol; any process p_i that discovers that the old coordinator has failed will initiate a election algorithm, using a message called election, that contains the

p_i 's own id that is the attribute. So, this is the initiation of the election process. So, when p_i receives an election message it compares with attributes in the message with its own attribute, if the arrived message is greater than p_i 's, it forwards the message. If the arrived attribute is smaller and p_i has not forwarded an election message earlier, it overwrites the message with its own id and forwards it.

If the arrived id matches that of p_i then p_i 's attribute must be greater, and if it is happening at all the processes and comes back then to the same one, then it is the new coordinator, this process then sends the elected message to its neighbor with its id announcing the election result. Let us see through a working example here before that.

(Refer Slide Time: 11:07)

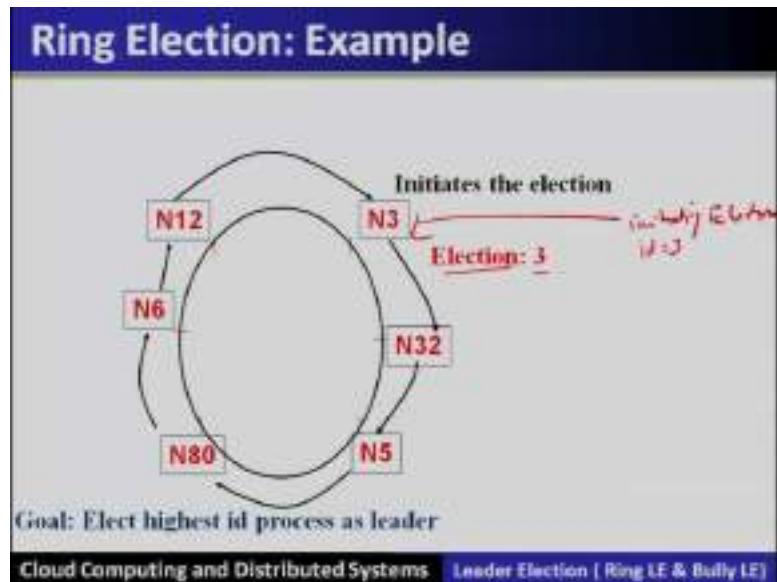
The Ring Election Protocol (2)

- When a process p_i receives an "Elected" message, it
 - sets its variable $elected \leftarrow$ id of the message.
 - fowards the message unless it is the new coordinator.

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

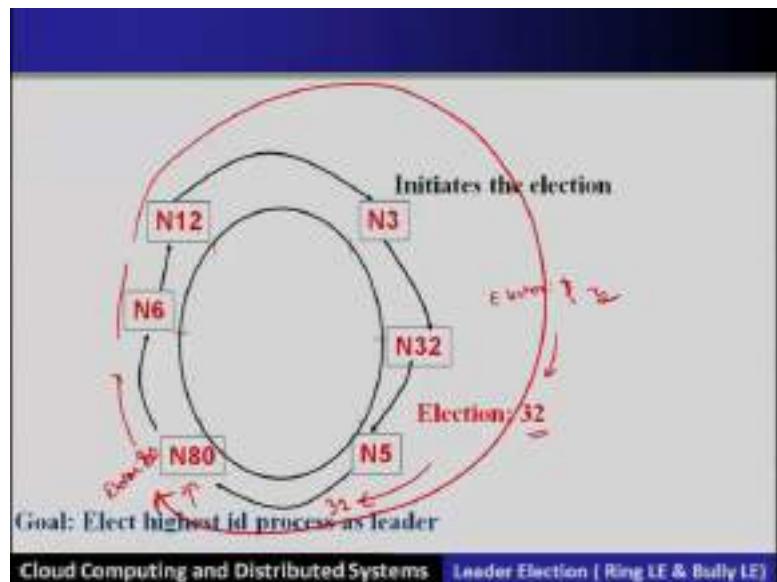
So, when a process p_i receives an elected message, it sets the variable $elected$ as its id of the message and forwards the message, unless it is the new coordinator. So, let us see how this particular leader election algorithm will work.

(Refer Slide Time: 11:30)



Let us say that N 3 is initiating the election with its id that is 3. So, the election message and with this id will go to N 32, N 32 will check the id of incoming message.

(Refer Slide Time: 12:03)



Now, N 32 is greater than 3, then it will overwrite in its message, 3 will be overwritten with 32 and it will be forwarded further and when it reaches at N 5 which is lower id, then it will forward N 32 and when it reaches at this particular N 80. So, N 80 will overwrite its id in the message and forwards that election N 80. Similarly it will forward from here and so on and it will come back again over here. So, then once that particular

message will come back again to the same highest id process, then it will change from election to elected message and this will be circulated so that, so that all other processes knows that who is the leader which is elected.

So, let us see the analysis. So, here we are assuming there are no failure during the election itself and there are N processes, so, we will count how much is the complexity in terms of messages.

(Refer Slide Time: 13:21)

Analysis

- Let's assume no failures occur during the election protocol itself, and there are N processes
- How many messages?**
- Worst case occurs when the initiator is the ring successor of the would-be leader

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

(Refer Slide Time: 13:28)



So, in the worst case situation happens, when the successor of would be, would be leader will become the initiator, the process it will go it will change at as 12 and here also 12 will continue, here it will change to 32 and it will continue, here it will change 80 and now it will continue and here then it will be changed to elected and then it will continue; likewise and here it will terminate. So, just see that how many times 1 2 3, 3 times the total messages will take around.

(Refer Slide Time: 14:46)

Worst-case Analysis

- $(N-1)$ messages for Election message to get from Initiator (N_6) to would-be coordinator (N_{80})
- N messages for Election message to circulate around ring without message being changed
- N messages for Elected message to circulate around the ring

- **Message complexity: $(3N-1)$ messages**
- **Completion time: $(3N-1)$ message transmission times**
- Thus, if there are no failures, election terminates **(liveness)** ✓ and everyone knows about highest-attribute process as leader **(safety)** ✓

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, let us say that $N - 1$ is basically the number of messages in a round. So, there are $(3N-1)$ messages, which are required in this particular algorithm in the worst case.

Now, here there are no failures, hence the election will terminate eventually and this ensures the liveness property and everyone will know that the highest attribute process is elected as a leader, and there is only one highest attribute process, hence the safety is ensured.

(Refer Slide Time: 15:23)

Best Case?

- Initiator is the would-be leader, i.e., N80 is the initiator
- **Message complexity:** $2N$ messages
- **Completion time:** $2N$ message transmission times

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, the best case is that when the would be leader will become an initiative in that case only $2N$ message complexities will be there. How about multiple initiators? So, each process in that case will remember in the cache, the initiator of each election message it receives at all point of time the process suppresses, the election messages of any lower id initiators.

(Refer Slide Time: 16:00)

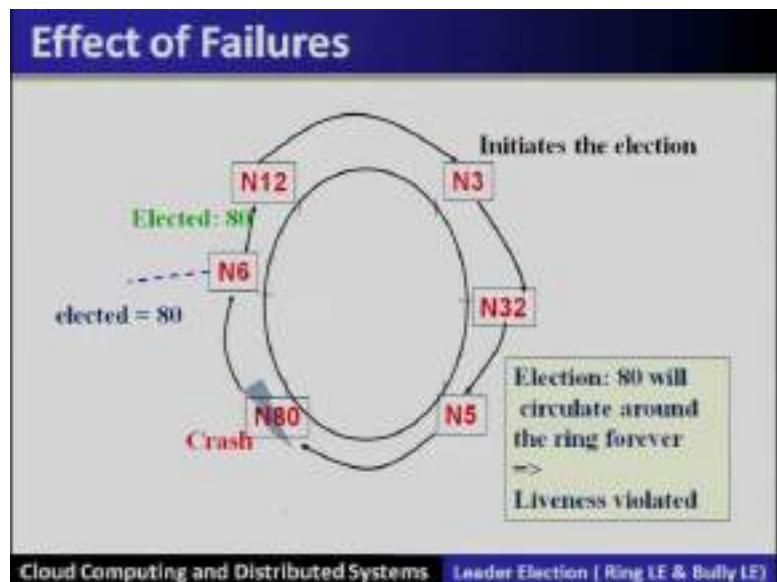
Multiple Initiators?

- Each process remembers in cache the initiator of each Election/Elected message it receives
- **(All the time)** Each process suppresses Election/Elected messages of any lower-id initiators
- Updates cache if receives higher-id initiator's Election/Elected message
- Result is that only the highest-id initiator's election run completes

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

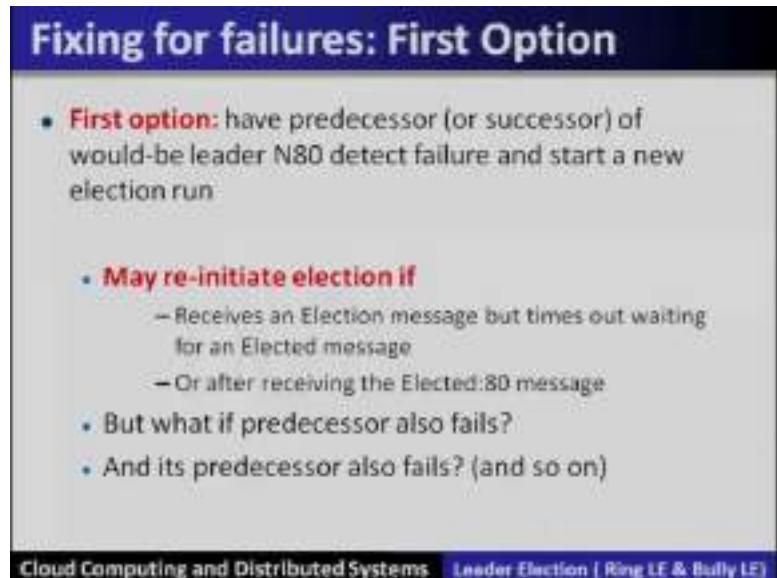
So, the result is that only highest id initiators election run will be completed in that case.

(Refer Slide Time: 16:07)



Now, let us see another situation where we allow the failures. Now what happens after the node releases the elected 80, the node crash. Then in that case this particular message keeps on circulating forever and it will never be consumed and stopped. Hence the liveness will be violated in this case.

(Refer Slide Time: 16:50)



Now to fix up this particular problem the first option is that, the predecessor or the successor of the would be leader will detect the failures and start a new leader election algorithm, may re initiate the leader if the received election message, but it becomes a

timeout for looking up the elected leader or after receiving the elected leader this becomes a timeout.

Hence it will re initiate that election in that case what if that predecessors are also fails and so on. So, the second option is to use a failure detector that any process after receiving the election message can detect the failure of the highest id via its own failure detector, if so then it will start the leader election, but there are issues with the failure detectors may not be complete and accurate.

(Refer Slide Time: 17:48)

Fixing for failures: Second Option

- **Second option:** use the failure detector (FD)
- Any process, after receiving Election:N₈₀ message, can detect failure of N₈₀ via its own local failure detector
 - If so, start a new run of leader election
- But failure detectors may not be both complete and accurate
 - **Incompleteness in FD** => N₈₀'s failure might be missed
=> Violation of Safety
 - **Inaccuracy in FD** => N₈₀ mistakenly detected as failed
 - => new election runs initiated forever
 - => Violation of Liveness

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, if it is incompleteness is there in the failure detector then it might be missed and hence it will violate the safety in that case.

Similarly, the inaccuracy is in failure detector by mistakenly detected as the failed message.

(Refer Slide Time: 18:05)

Why is Election so Hard?

- Because it is related to the consensus problem!
- If we could solve election, then we could solve consensus!
 - Elect a process, use its id's last bit as the consensus decision
- But since consensus is impossible in asynchronous systems, so is leader election!
- Consensus-like protocols such as Paxos used in industry systems for leader election

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

So, why the election is so hard problem? Election is hard because it is related to the consensus problem; that means, if you could solve the election process then we could also solve the consensus problem. For example, the leader if it is elected then let us say the last id, last bit in the id can be the consensus value.

Hence if the leader election is done, then basically it will also solve the consensus problem, but you know that consensus is impossible in asynchronous system so is the leader election. So, consensus protocols such as packs of which are used in the industry system for leader election that we will see in this discussion.

(Refer Slide Time: 19:09)

(ii) Classical Algorithm: Bully Algorithm

- All processes know other process' ids
- When a process finds the coordinator has failed (via the failure detector):
 - **if** it knows its id is the highest
 - it elects itself as coordinator, then sends a **Coordinator** message to all processes with lower identifiers. Election is completed.
 - **else**
 - it initiates an election by sending an **Election** message
 - (contd...)

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

First we will see the classical algorithm that is called bully algorithm. Here all processes know other processes id, so when a process finds the coordinator has failed via the failure detector, and if it knows its id is the highest, then it elects itself as the coordinator and then sends the coordinator message to all the processes with the lower id and the election is completed, else it initiates an election by sending an election message sends it to only the processes that have the higher than itself, if receives no answer within the timeout then calls itself as the leader and sends the coordinator message to all the lower id processes and the election is completed

(Refer Slide Time: 19:43)

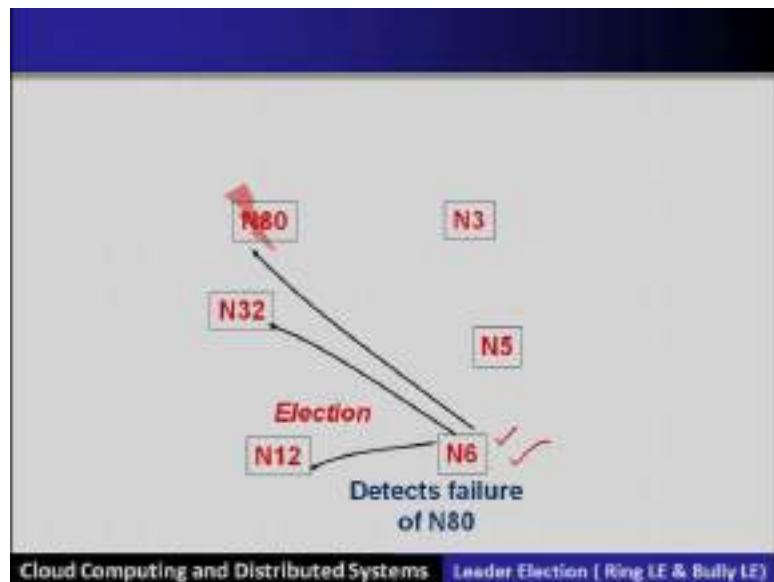
Bully Algorithm (2)

- **else** it initiates an election by sending an **Election** message
 - Sends it to only processes that have a *higher id than itself*.
 - **if** receives no answer within timeout, calls itself leader and sends **Coordinator** message to all lower id processes. Election completed.
 - **if** an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another timeout, start a new election run.
- A process that receives an **Election** message replies with **OK** message, and starts its own leader election protocol (unless it has already done so)

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

If an answer is received; however there is some non faulty higher id process. So, it will wait for the coordinator message, if none are received after another timeout then it will start a new election transfer. So, a process that receives an election message replies message and it starts its own leader election algorithm.

(Refer Slide Time: 20:27)

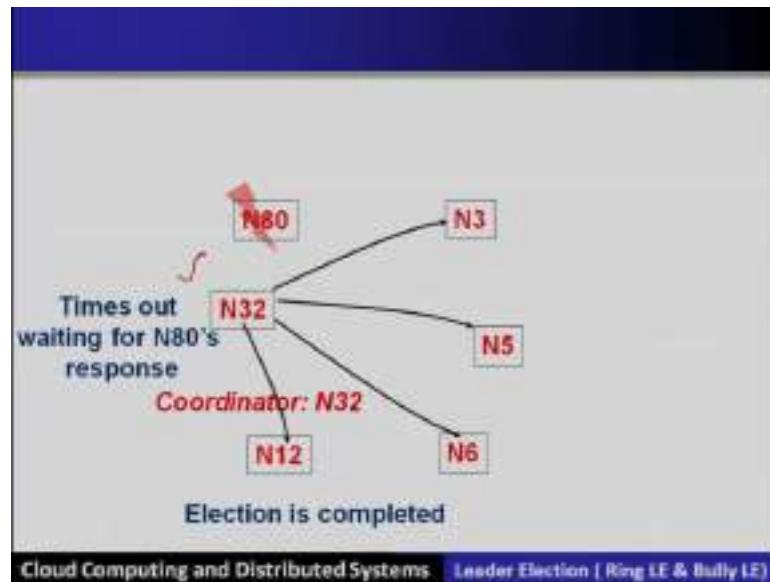


So, here let us see through an example. Now let us say that the highest id N 80 is failed and it is being detected by a process having the id N 6. So, it detects through a timeout failure detector at N 80 is down. So, it sends the message to the nodes who is having the higher id than its own id. For example, here N 6 the nodes which is have higher id than N 6 is N 12, N 32, N 80.

N 3 and N 5 they are having ids lower than N 6, so it will not be sent. So, having received these particular message, so N 12 is having higher id, so it will send a ok in the sense he will take care of since it is having the higher id, so he will initiate the leader election and also N 32 is having higher id than N 6 they will also send the message for N 6 and N 6 will now be waiting for the outcome.

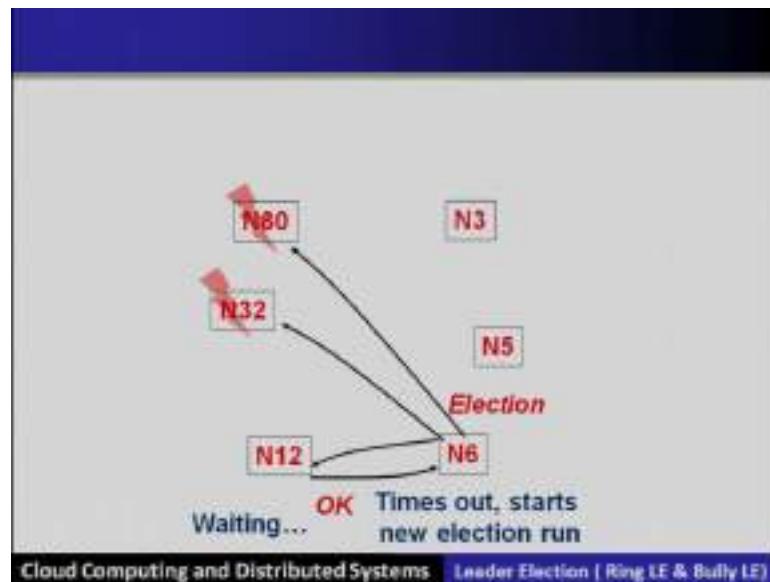
Then N 12 will send the election to its higher id that is N 32 and N 80 the same way and N 32 will send the ok, so N 12 will also be waiting. Since N 80 is faulty, so N 32 will not receive the, from N 80. So, N 32 will timeout and will assume as the coordinator. So, it will now send the coordinator message to all other nodes in the system and hence the election is complete

(Refer Slide Time: 22:11)



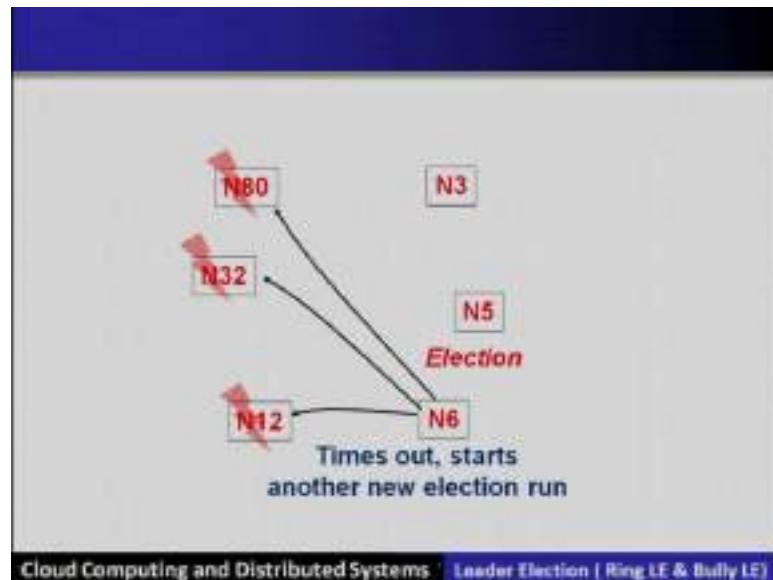
Now if N 32 also is failed, then these two waiting process they will timeout and N 6 will timeout before N 12.

(Refer Slide Time: 22:27)



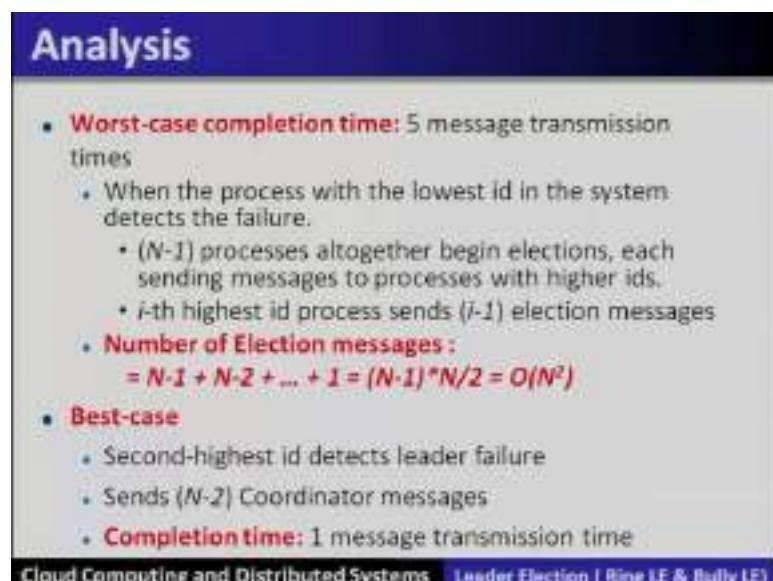
So, again it will run the election and send this messages, election message to the nodes of having higher id than N 6. Only N 12 is alive, so it will send the message. If N 12 is also crashed then it will timeout and it start another leader election where in N 6 will be elected then.

(Refer Slide Time: 22:55).



So, here irrespective of how many failures are there the algorithm will terminate with that we have seen in the Bully's algorithm. So, if the failures stop, eventually a leader will be elected. So, how do you set this timeouts will based on the worst case time to complete the election. So, the 5 message transmission times here, if there are no failure during the run that we have already seen. So, the analysis of this algorithm, if we see how many number of election messages are send.

(Refer Slide Time: 23:32).



So, using 5 different message transmission time, the total number of messages which are send is $N - 1 + N - 2$ and so on 1. So, if we some up it will be of the $O(N^2)$, the best case will occur when the second highest id detects the leader failure that we have seen earlier. So, it will send $N - 2$ coordinator messages and the completion time is only one message transmission here in this case.

(Refer Slide Time: 24:07)

Impossibility?

- Since timeouts built into protocol, in asynchronous system model:
 - Protocol may never terminate => Liveness not guaranteed
- But satisfies liveness in synchronous system model where
 - **Worst-case one-way latency can be calculated = worst-case processing time + worst-case message latency**

Cloud Computing and Distributed Systems | Leader Election | Ring LE & Bully LE

Let us see the impossibility since the timeouts built into the protocol in the asynchronous system model. So, the protocols may never terminate and the liveness is not guaranteed here in this case, but it satisfies the liveness in the synchronous model, they are the worst case one way latency can be calculated. We will see now the leader election in the industry system such as Google's Chubby and Apache Zookeeper.

(Refer Slide Time: 24:35).

Election in Industry

- Several systems in industry use Paxos-like approaches for election
 - Paxos is a consensus protocol (safe, but eventually live)
- Google's Chubby system
- Apache Zookeeper

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)

So, this leader election in this industry system is different than the approach, which we have seen using the process, sending a proposed value and reaching agreement.

(Refer Slide Time: 24:49)

Use of Consensus to solve Election

- One approach:
 - Each process proposes a value
 - Everyone in group reaches consensus on some process P_i's value
 - That lucky P_i is the new leader!

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)

Instead of that industry uses the systems like Paxos for the leader election. We have seen earlier that Paxos is a consensus protocol which is now being used in the industry for the leader election. The Paxos is safe, but it is not guaranteeing the liveness property, it is guaranteeing the eventual live. So; that means, there are some scenarios where in the algorithm will never terminate, but it says that eventually it is live that is most of the

cases everything goes fine, then basically the algorithm terminates. Let us see how the Paxos is used in the industry system; like Google Chubby and Apache Zookeeper for leader election.

(Refer Slide Time: 25:48)

Election in Google Chubby

- A system for locking
- Essential part of Google's stack
 - Many of Google's internal systems rely on Chubby
 - BigTable, Megastore, etc.
- **Group of replicas**
 - Need to have a master server elected at all times

Reference: <http://research.google.com/archive/chubby.html>

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)



The diagram shows five rectangular boxes arranged vertically, each labeled with a server name: Server A, Server B, Server C, Server D, and Server E. This visual representation corresponds to the list of servers mentioned in the slide content.

So, Google Chubby is a distributed locking system and is also a essential part of a Google's stack, and many of the Google's internal systems rely on this particular Google Chubby; such as BigTable and megastore.

In this particular system, there is a group of replicas. Here in this example we have shown server A B C D E, there are 5 different servers, they are all replicas and there is a need to elect a leader at all points of time among them. So, this particular example shows that here, let us say the server D is elected as a leader called master and this particular leader is elected using a election protocol.

(Refer Slide Time: 26:33)

Election in Google Chubby (2)

- **Group of replicas**
 - Need to have a master (i.e., leader)
- **Election protocol**
 - Potential leader tries to get votes from other servers
 - Each server votes for at most one leader
 - Server with majority of votes becomes new leader, informs everyone

Cloud Computing and Distributed Systems Leader Election (Chubby & Zookeeper)

Question 5 - Interactions

So, the potential leader tries to get the vote from the other servers. So, each server will vote at most one leader. So, the server with the majority of the vote, now become the leader and he informs about this decision. So, this particular election is based on that majority of the servers vote whose ever gets.

For example here there are 5 different servers, so majority becomes 3. So, if 3 or more any server gets the votes, then basically that particular server will be elected as the leader. Now why this majority not all? So, majority is basically equivalent to the quorums. So, in every quorum are the intersection of the quorums, there is exactly one common node between the quorums and you know that every server has to vote at most one leader; therefore, this particular election using the majority of the votes that is following the quorum principal.

Hence it ensure the safety that at most; this particular leader will elect only one coordinator or a one master.

(Refer Slide Time: 28:18).

Election in Zookeeper

- Centralized service for maintaining configuration information
- Uses a variant of Paxos called Zab (Zookeeper Atomic Broadcast)
- Needs to keep a leader elected at all times

Reference: <http://zookeeper.apache.org/>

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)

So, election in a zookeeper, zookeeper is a centralized service for maintaining the configuration information. So, zookeeper uses a variant of Paxos which is called Zab, zookeeper atomic broadcast that need to keep the leader elected at all points of time.

(Refer Slide Time: 28:37)

Election in Zookeeper (2)

- Each server creates a new **sequence number** for itself
 - Let's say the sequence numbers are **ids**
 - Gets highest id so far (from ZK(zookeeper) file system), creates next-higher id, writes it into ZK file system
- Elect the highest-id server as leader.

N12 N3
N6 N32
N80 N5
Master

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)

Let us see how the election in a zookeeper is done. Here each server will create a new sequence number for itself. Let us say the sequence numbers are the ids, it gets the highest id so far seen and that is written into a zookeeper file system. So, the highest id server will be elected as a leader in this particular situation regarding failures.

(Refer Slide Time: 29:10)

Election in Zookeeper (3)

- **Failures:**
 - One option: everyone monitors current master (directly or via a failure detector)
 - On failure, initiate election
 - Leads to a flood of elections
 - Too many messages

Cloud Computing and Distributed Systems Leader Election (Chubby & Zookeeper)

One option that everyone monitors the current master and on failure it will initiate again the election, and it may lead to the flood of elections and too many messages will be there.

(Refer Slide Time: 29:29)

Election in Zookeeper (4)

- **Second option:** (implemented in Zookeeper)
 - Each process monitors its next-higher id process
 - **If** that successor was the leader and it has failed
 - Become the new leader
 - **else**
 - wait for a timeout, and check your successor again.

Cloud Computing and Distributed Systems Leader Election (Chubby & Zookeeper)

So, there is second option which says that each process monitors its next higher id process, so if the successor who was the leader is failed and it will become a new leader. For example, the current leader was let us say N 80 if it failed, the successor N 32 successor will identify will monitor it and it will become the leader if phase successor is

not responding, else it will wait for a timeout and check the other successor and if that is also not responding then it will become the leader.

(Refer Slide Time: 30:10)

Conclusion

- Leader election an important component of many cloud computing systems
- **Classical leader election protocols**
 - Ring-based
 - Bully
- **But failure-prone**
 - Paxos-like protocols used by Google Chubby, Apache Zookeeper

Cloud Computing and Distributed Systems | Leader Election (Chubby & Zookeeper)

So, conclusion leader election is an important component in many cloud computing applications and systems, primarily it is used for the coordination. So, we have seen the classical leader election algorithm that is ring based leader election and bully leader election. We have also seen the industry uses the protocol Paxos like protocols, which are used in Google Chubby and Apache Zookeeper for leader election, which also works in the failure situations.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 09
Design of Zookeeper

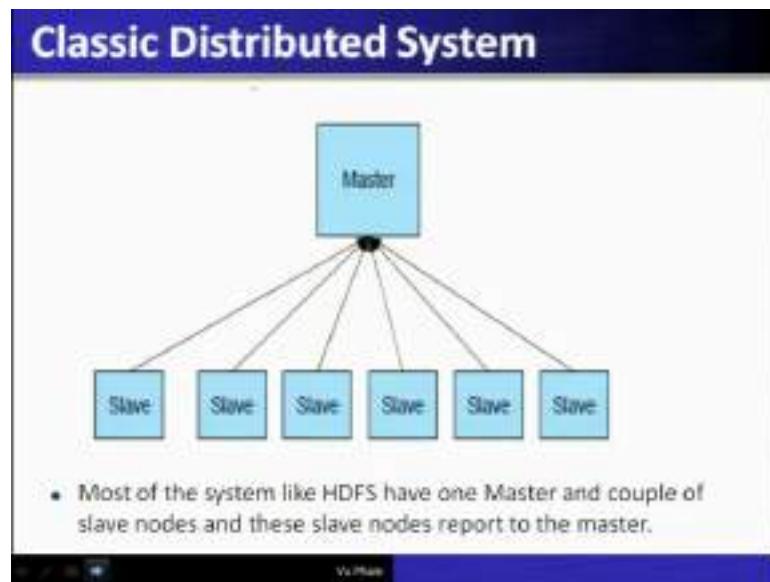
Design of Zookeeper. Preface, content of this lecture, we will discuss the Design of a Zookeeper, which is a coordination service or distributed application. We will discuss its fundamentals, its design goals, the architecture of apache zookeeper, and various uses its applications.

(Refer Slide Time: 01:29)



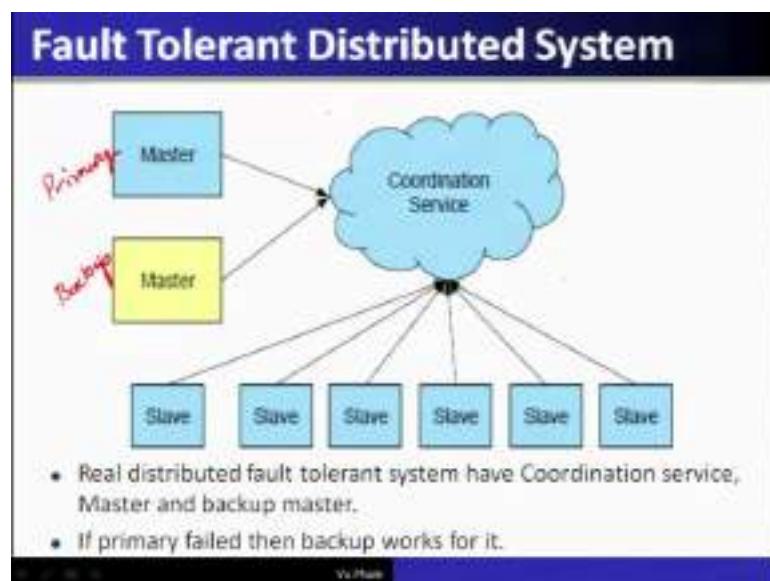
To understand what a zookeeper does, it does a very simple task, which is called coordination, but coordination is important and it is not so simple. In this figure, this shows a situation of a traffic condition in a particular city. This kind of traffic situation requires a coordination, so that the progress at all points of time can be there in the traffic flow. So, the that important, and we will see in our situation in our scenario of distributed systems and a cloud computing systems, how this particular coordination is useful in the applications.

(Refer Slide Time: 01:43)



This is a simple distributed system scenario, where there is a master and a couple of slaves. These slaves, they send their heart beats or their results to the master, all of them and master will then application issues. This particular kind of scenario, you can see you have seen in like HDFS, where there is one master and a couple of slaves, and these slaves reports these slave reports to the master.

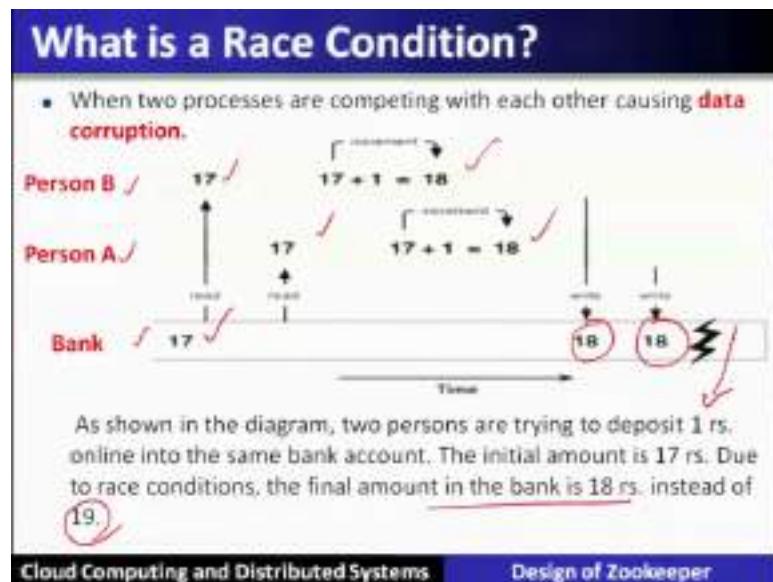
(Refer Slide Time: 02:32)



Now, if you require a fault-tolerant distributed system, then the scenario will be changed from the previous master that is the primary and the backup that means, the masters

should be more than one. So if one fails, automatically the other resume its role in there is a requirement of a coordination service, the slaves and the master to ensure the fault tolerant distributed system at all points of time. So, real distributed fault tolerant system have a coordination service, which always which ensures the existence of the master and a few backup masters. So, if a primary master fails, then the backup can he still continue to serve the application without any disruption.

(Refer Slide Time: 03:51)



Let us see some of the basics, which we are going to use in our discussion. What is a race condition? Here, it is shown that there is a bank account, having an amount 17 in a particular account, and two persons A and B; they want to increment this particular account value. So, they read both at the same point of time, so both of them will get the same value in the account, and then both of them will increment write, so where is the problem. So, if two people are incrementing on 17, the final value should be 19. But, due to the race conditions, the final value is not correct that is it comes out to be 18 that is 1 increment is lost, this is called a race condition.

(Refer Slide Time: 04:55)

What is a Deadlock?

- When two processes are waiting for each other directly or indirectly, it is called **deadlock**.

The diagram shows a traffic intersection with three cars. Process 1 is at the top, Process 2 is at the bottom right, and Process 3 is at the bottom left. Arrows indicate they are waiting for each other: Process 1 is waiting for Process 2, Process 2 is waiting for Process 3, and Process 3 is waiting for Process 1.

- Here, Process 1 is waiting for process 2 and process 2 is waiting for process 3 to finish and process 3 is waiting for process 1 to finish. All these three processes would keep waiting and will never end. This is called dead lock.

Cloud Computing and Distributed Systems Design of Zookeeper

Another problem, which we have seen in the traffic jams lights. In the system, when two processes are waiting for each other. So, for example, if there are three processes, who are circularly waiting for each other, then they may lead to a dead lock, because they are waiting, they are holding state and waiting for each other. And that will kept on waiting, and this situation is called deadlock. So, race condition and deadlocks are two important things; two important problems, which require to be solved, while the coordination is to be achieved.

(Refer Slide Time: 05:39)

What is Coordination ?

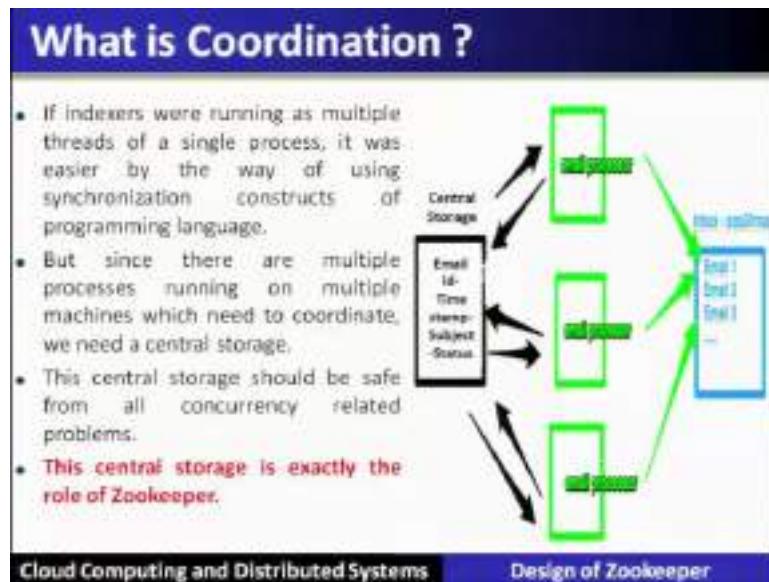
- How would Email Processors avoid reading same emails?
- Suppose, there is an inbox from which we need to index emails.
- Indexing is a heavy process and might take a lot of time.
- Here, we have multiple machine which are indexing the emails. Every email has an id. You can not delete any email. You can only read an email and mark it read or unread.
- Now how would you handle the coordination between multiple indexer processes so that every email is indexed?

The diagram shows three boxes labeled "index processor" connected by arrows pointing to a central box labeled "Inbox - pg27map". Inside the "Inbox" box, there is a list of emails: "Email 1", "Email 2", "Email 3", and "...".

Cloud Computing and Distributed Systems Design of Zookeeper

Now, one application, which requires the coordination is let us say the email processors. Suppose, there is an inbox, from which we need to index mails. Indexing is a heavy process has an id. So, now you would handle the coordination between multiple indexers, process, and the email indexing.

(Refer Slide Time: 06:15)



Now, if the indexers are running as multiple threads on a single process, then the most of the operating system and the programming constructs provides the synchronization in the programming language, and that can be solved. But, if multiple processes are running on multiple machine, which needs to be coordinated, we need a central for the coordination, this particular indexer or indexes are running over multiple machines. So, this particular central storage, which is responsible for the coordination, for all concurrent related issues will serve this purpose in the pure distributed system model. This particular central storage can be modelled or can be modelled as the zookeeper application.

(Refer Slide Time: 07:18)

What is Coordination ?

- **Group membership:** Set of datanodes (tasks) belong to same group
- **Leader election:** Electing a leader between primary and backup
- **Dynamic Configuration:** Multiple services are joining, communicating and leaving (Service lookup registry)
- **Status monitoring:** Monitoring various processes and services in a cluster
- **Queuing:** One process is embedding and other is using
- **Barriers:** All the processes showing the barrier and leaving the barrier.
- **Critical sections:** Which process will go to the critical section and when?

Cloud Computing and Distributed Systems Design of Zookeeper

So, we have seen enough of examples for the coordination. Let us see, what are the other use cases of the coordination. The first one is called group membership, so that means, are executing different tasks so for a particular application, and its corresponding task, they may form a group. These nodes may join, they may leave, therefore a group membership is required, and for that a coordination service is very much needed to ensure the group membership across several applications, which requires different tasks or different data nodes.

Similarly, another use case is about the leader election. So, we have seen in the fault tolerant distributed system, so there exist more than one master for tolerating any kind of failure that is called single point of failure. So, let us say that primary and backup masters are being provided out of several servers; one of them will be the primary that is called leader, so that particular leader will be elected through a election. So, if there are three different servers maintaining the coordination service, one another use case is about the dynamic configuration.

So, the multiple services if it is running on a particular cloud or a distributed system so for each services, the nodes or the servers which are joining and leaving, therefore a service look up registry is very much required. And that is to run each service a separate configuration that is the number of nodes, which nodes are running those services are

required that is called dynamic configuration. And the dynamic configuration is a coordination problem.

Similarly, about the status monitoring, that is the monitoring will of various processes and services in the cluster is also a coordination problem. Queuing is also a coordination problem to some extent that is one process is giving an output, which will be fade as the input to the other piped process. Therefore, forming a queue, this particular way of piping, several processes for the application or a pipeline for a application is nothing but a coordination problem.

Barriers is also an application of a coordination problem, so that means, all the process will join at a barrier, and then they may leave from the barrier after all the processes joins it, that is also a coordination problem at the barrier end. Similarly, the critical sections, that means, to coordinate that across since the particular critical section, it has to be only one process at a time can be executing into the critical section; it is also a coordination problem.

(Refer Slide Time: 10:49)

What is ZooKeeper ?

- **ZooKeeper is a highly reliable distributed coordination kernel,** which can be used for distributed locking, configuration management, leadership election, work queues,....
- Zookeeper is a replicated service that holds the metadata of distributed applications.
- **Key attributes of such data**
 - Small size
 - Performance sensitive
 - Dynamic
 - Critical
- **In very simple words,** it is a central store of key-value using which distributed systems can coordinate. Since it needs to be able to handle the load, Zookeeper itself runs on many machines.

Cloud Computing and Distributed Systems

Design of Zookeeper

So, how this, so, this is done through the apache zookeeper. Let us see how it achieves this kind of coordination, which is applied over wide variety of applications. So, zookeeper is a highly reliable distributed coordination kernel. Kernel in the sense, it is not providing all these services, such as locking, configuration management, leader election that we have already seen. But this is the core, which can be in sense; it is a

highly reliable distributed coordination kernel, so it provides lot of APIs. And using this API, this application can be build or can use the coordination service to build the applications.

Zookeeper is a replicated service that holds the metadata of the distributed applications. So, when we say metadata, that means, it is distributed application. It manages, hence the data is small, which zookeeper manages. Several key attribute of this matter is that it is of small size, and it is of performance sensitive, it is dynamic, it is critical. In very simple words, it is the central store of key value store using the distributed system to coordinate for different distributed applications. So, it need to be able to manage to handle the load, the zookeeper itself runs on many machines.

(Refer Slide Time: 12:34)

What is ZooKeeper ?

- Exposes a simple set of primitives
- Very easy to program
- Uses a data model like directory tree
- Used for
 - Synchronisation
 - Locking
 - Maintaining Configuration
- Coordination service that does not suffer from
 - Race Conditions
 - Dead Locks

Cloud Computing and Distributed Systems Design of Zookeeper

So, zookeeper will provide simple primitives, which are easy to program for the clients to use for their applications. It uses a very simple data model like a directory tree that we have seen in a many file system or a directory structure of a file system. Zookeeper is basically used for synchronization in a distributed application. It is used for locking that is called distributed locking the consistency, and also maintain the configuration that is dynamic configuration. Such a coordination service, which is provided by the zookeeper ensures that does not suffer from race conditions and deadlocks.

(Refer Slide Time: 13:24)

Design Goals: 1. Simple

- A shared hierachal namespace looks like standard file system
- The namespace has data nodes - znodes (similar to files/dirs)
- Data is kept in-memory
- Achieve high throughput and low latency numbers.
- **High performance**
 - Used in large, distributed systems
- **Highly available**
 - No single point of failure
- **Strictly ordered access**
 - Synchronisation

*(Large No read ops - by many servers
by many clients)*

(Master/Backup Master)

(Leader elected)

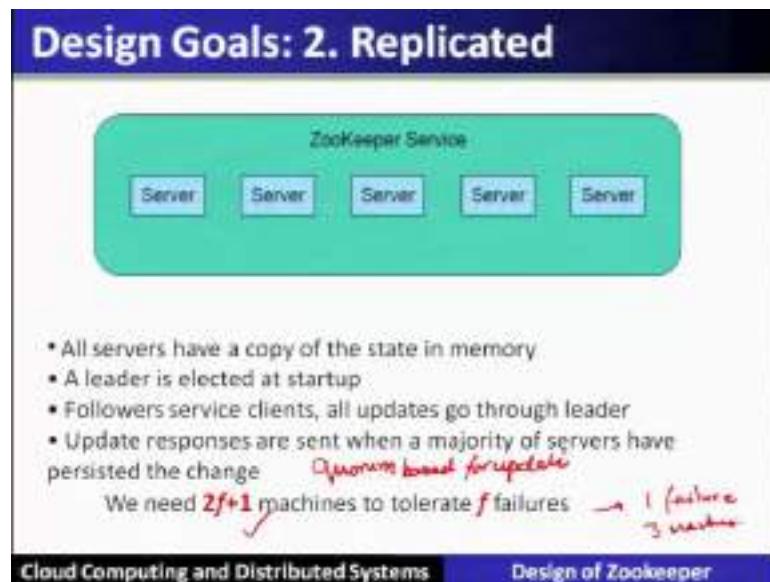
Cloud Computing and Distributed Systems Design of Zookeeper

Therefore, the design goal is to make to ensure the simplicity that is in the form of the hierachal namespace, which looks very similar to the standard file system hierarchy. However, the namespace has the data nodes, which are called znodes, similar to the files or directories. These data is kept in memory. And therefore, it achieves the high throughput and low latency numbers.

Since, this particular aspect that is the namespace is a metadata, therefore it can be kept in the memory because of its small size. This will give the high; that is it is used in the large scale distributed system; it is also to be highly available. Why, because there is no single point of failures, because it is replicated, and also it is strictly ordered; why, because it ensures the synchronization. So, in the zookeeper, all three things that is high performance, highly available, and strictly ordered access is supported.

So, high performance is basically used in a large scale, that means, it is supporting large number of reads questions, which are served by many servers that means, leaders and its slaves; therefore it achieves the high performance for its operations. It is highly available that is there is no single point of failure that means, so because there is a master and there is a backup master also, which ensures the failure tolerance. It is strictly ordered access that is synchronization is guaranteed, since there is a leader elected at all points of time. Therefore, this ensures the strictly ordered access and also ensures the sync rate environment.

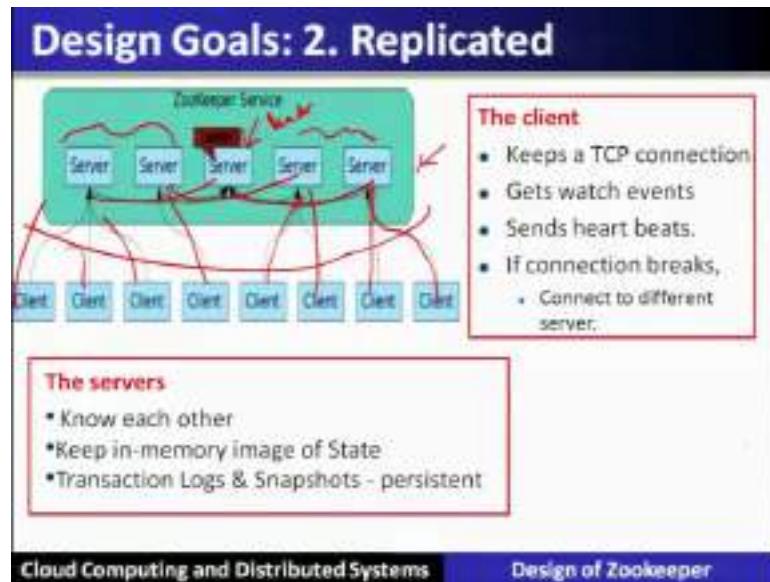
(Refer Slide Time: 16:13)



The 2nd design goal is about the replication. So, all the servers have a copy of the state in the memory. So, a leader is elected at the startup. Followers are those servers, who are not elected as a leader. So, they also have the updated state at all point of time, and it is the responsibility of the leader to ensure that all the servers. So, the clients for any update, it has to follow through the leader.

So, update responses are sent, when the majority of the servers have persisted the change. So, majority means, it is following the quorum based approach for updates. Therefore, for fault tolerance, we require $2f + 1$ machine to tolerate f failures. So, to tolerate one failure, so it required how many machines, 3 machines. So, we will see that minimum three machines are required to tolerate one failure. If two are failing in three, then basically it is not the majority, hence this will violate this condition.

(Refer Slide Time: 17:57)



So, the replicated structure, we have seen we will divide into two different classes; one is called client, we will keep the TCP connections and also gets a watch event, it will always through that it will be recognize that it is having a connection to the zookeeper. If the connection breaks, then basically it is no longer the client. Similarly, the severs will know each other; and server will keep in memory image of the state; it will also use the transaction logs and takes the snapshot in a persistent storage.

Here in this particular figure, we have seen all the servers, which are knowing each other. And through the election, one of them will be the leader, and others are the followers. Clients can connect to any of this particular leader or the follower, and keep on sending the heartbeat, so that they will know that they are basically the clients of these services.

(Refer Slide Time: 19:31)

Design Goals: 3. Ordered

- ZooKeeper stamps each update with a number
- **The number:**
 - Reflects the order of transactions.
 - used implement higher-level abstractions, such as synchronization primitives.

Cloud Computing and Distributed Systems Design of Zookeeper

The 3rd design goal is the ordering. So, zookeeper stamped each update with a number. So, the number reflects the order of the transaction; and used to implement the higher level abstraction, such as synchronization primitives.

(Refer Slide Time: 19:55)

Design Goals: 4. Fast

Performs best where reads are more common than writes, at ratios of around 10:1.

At Yahoo!, where it was created, the throughput for a ZooKeeper cluster has been benchmarked at over 10,000 operations per second for write-dominant workloads generated by hundreds of clients

Cloud Computing and Distributed Systems Design of Zookeeper

4th goal is about the speed. So, this achieves, when it performs the best. Performance, when the reads are more than the writes, at the ratio let us say 10 is to 1. Then in that scenario, it is benchmark that it basically has achieved 10,000 operations per second, generated by hundreds of clients.

(Refer Slide Time: 20:27)

Data Model

- The way you store data in any store is called **data model**.
- **Think of it as highly available fileSystem:** In case of zookeeper, think of data model as if it is a highly available file system with little differences.
- **Znode:** We store data in an entity called znode.
- **JSON data:** The data that we store should be in JSON format which Java script object notation.
- **No Append Operation:** The znode can only be updated. It does not support append operations.
- **Data access (read/write) is atomic:** The read or write is atomic operation meaning either it will be full or would throw an error if failed. There is no intermediate state like half written.
- **Znode:** Can have children

Cloud Computing and Distributed Systems

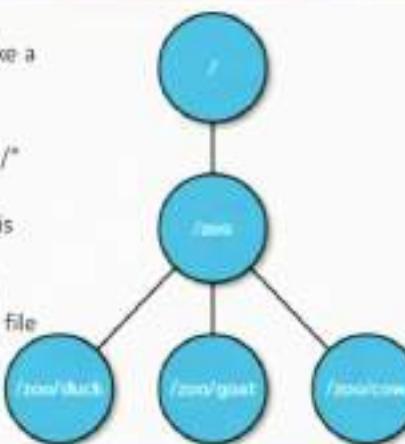
Design of Zookeeper

Let us see the data model used in the zookeeper. The way you store the data in any data store is called a data model. Now, here in the it uses a file system like interface, so zookeeper will think of data model, as if it is highly available file system where the nodes are called znodes, which will store the data in an entity. The data is in JSON format. And it does not supports the append operation, it can only be updated. Now, data access is done through the read and write operation, which is to be done in an atomic manner; either it will be full or it will give (Refer Time: 21:29) znodes can have the children.

(Refer Slide Time: 21:32)

Data Model Contd...

- So, znodes inside znodes make a tree like heirarchy.
- The top level znode is "/".
- The znode "/zoo" is child of "/" which top level znode.
- duck is child znode of zoo. It is denoted as /zoo/duck
- Though ". " or ".." are invalid characters as opposed to the file system.



Cloud Computing and Distributed Systems

Design of Zookeeper

So, this is a typical data model, which is called here the nodes are called z. So, this particular hierarchy of znodes will stop, and then followed by a znode slash zoo; goat and cow, they are the child of slash zoo znode.

(Refer Slide Time: 22:01)

Data Model – Znode - Types

- **Persistent**
 - Such kind of znodes remain in zookeeper until deleted. This is the default type of znode. To create such node you can use the command: `create /name_of_myznode "mydata"`
- **Ephemeral**
 - Ephemeral node gets deleted if the session in which the node was created has disconnected. Though it is tied to client's session but it is visible to the other users.
 - An ephemeral node can not have children nor even ephemeral children.

Cloud Computing and Distributed Systems Design of Zookeeper

There are three different type persistent, ephemeral, and sequential (Refer Time: 22:08) until deleted. And this is a default type of znode. Ephemeral node get to the client users also. Ephemeral nodes cannot have in their names.

(Refer Slide Time: 22:20)

Data Model – Znode - Types

- **Sequential**
 - Creates a node with a sequence number in the name
 - The number is automatically appended.

<code>create -s /zoo v</code> Created /zoo0000000008	<code>create -s /zoo/ v</code> Created /zoo/0000000003
<code>create -s /xyz v</code> Created /xyz0000000009	<code>create -s /zoo/ v</code> Created /zoo/0000000004

- The counter keeps increasing monotonically
- Each node keeps a counter

Cloud Computing and Distributed Systems Design of Zookeeper

These numbers are automatically appended. The counters keep increasing monotonically, and each node keeps a counter. Example here, first time when a node v is created znode v is created; its number is 3. Then second time, they are sequential.

(Refer Slide Time: 22:47)

Architecture

- Zookeeper can run in two modes: (i) Standalone and (ii) Replicated.

(i) Standalone:

- In standalone mode, it is just running on one machine and for practical purposes we do not use standalone mode.
- This is only for testing purposes.
- It doesn't have high availability.

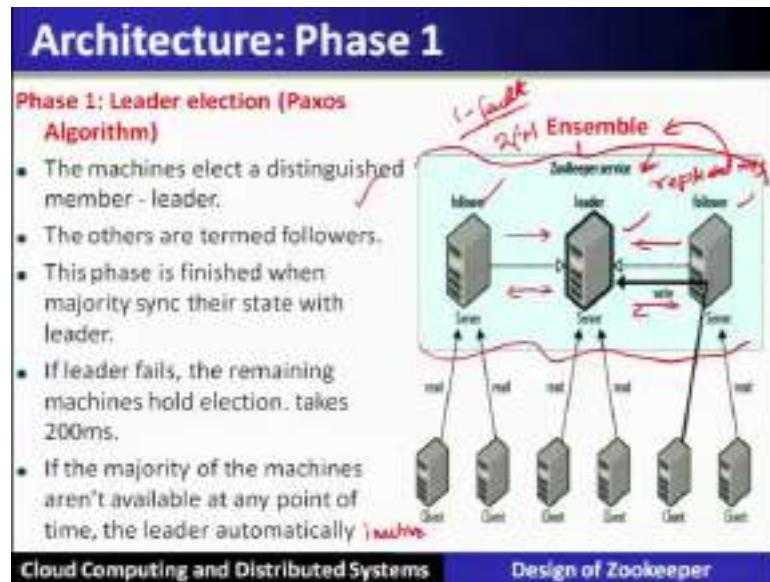
(ii) Replicated:

- Run on a cluster of machines called an ensemble.
- High availability
- Tolerates as long as majority.

Cloud Computing and Distributed Systems Design of Zookeeper

Let us see the architecture of zookeeper. So, zookeeper can run in two different modes for the testing purpose in into the lab. Therefore, it does not supports many important features like high availability, and so on and also it is not used in the in the practice. So, the other mode which is called a replicated mode, which will be launched it runs on a cluster of machines, which is called ensemble. It ensures high availability; and it tolerates as long as the majority of machine are responding.

(Refer Slide Time: 23:37)



See that this particular architecture is supported by two different phases; the phase one is called leader election, and it is done through the Paxos algorithm. Here, we can see that the set of machines in the zookeeper, we call it as ensemble. And here, there are number of clients are there, who are accessing the zookeeper service. So, among that servers or the machines, more than one machines are there, so among them, they will that is called ensemble. If we go to the previous slide, (Refer Time: 24:25) replicated service will run on a cluster of machines, which is called ensemble. So, this zookeeper service will run on which is called ensemble.

And among them, one of a leader, this leader election is done through the exchange of ping messages based on their id's, their heartbeats (Refer Time: 24:59) elect one of them to be the leader. And all other machines will become follower for the leader and followers. So, as you know, that three machines that means, it is able to tolerate 1 machine failure that is here, it is following $2f + 1$. This phase that is the leader election phase is finished, when the majority sync their state with the leader, that means, they will sync their states with the leader. And if the leader fails, the remaining machine can hold the election within 200 milliseconds. So, if the majority of the machines are not available at any point of time, the leader automatically will cease to work.

(Refer Slide Time: 26:07)

Architecture: Phase 2

Phase 2: Atomic broadcast

- All write requests are forwarded to the leader.
- Leader broadcasts the update to the followers.
- When a majority have persisted the change:
 - The leader commits the update.
 - The client gets success response.
- The protocol for achieving consensus is atomic like two-phase commit.
- Machines write to disk before in-memory.

Client → Write → Leader → Broadcast → Follower, Follower, Follower, Follower → 3 out of 4 have saved → Write Successful

Cloud Computing and Distributed Systems Design of Zookeeper

Write requests are forwarded to the leader, followers in this manner. Now, when the majority have persisted the change, let us say out of 4, 3 is the majority. When they have persisted the change, and they will inform it to the leader about it, then the leader will do the commit of this particular write operation. And for this the client will be informed about the success. The protocol for achieving the consensus is atomic like two-phase like two-phase commit.

(Refer Slide Time: 27:14)

Election Demo

- If you have three nodes A, B, C with A as Leader. And A dies. Will someone become leader?

Yes. Either B or C.

Cloud Computing and Distributed Systems Design of Zookeeper

Let us see, so if you have three different nodes, when A dies, then among B and C, because out of three, two are the majority. So, if A fails, either B or C can become (Refer Time: 27:39).

(Refer Slide Time: 27:40)

Election Demo

- If you have three nodes A, B, C And A and B die. Will C become Leader?

The diagram illustrates a three-node ensemble (A, B, C) connected in a ring. In the first state, all nodes are active. In the second state, nodes A and B are marked with a red 'X' and crossed out, indicating they have failed. Node C remains active. A question mark is shown to the right. Below the diagram, a checkmark indicates 'Yes' to the question of whether C becoming the leader is correct. A handwritten note says 'Why?' and '1 is not majority in 3 set ensemble'. The footer of the slide shows 'Cloud Computing and Distributed Systems' and 'Design of Zookeeper'.

And if two are failed, A and B let us say they are failed, remaining only one. What will happen, whether c will become a leader or not? No. Why? Because, because 1 is not the majority in 3 set ensemble, so that means, to work in this particular scenario, a majority should always be present, otherwise it will not have a leader in its operations.

(Refer Slide Time: 28:30)

Election Demo

- If you have three nodes A, B, C And A and B die. Will C become Leader?

The diagram illustrates a three-node ensemble (A, B, C) connected in a ring. In the first state, all nodes are active. In the second state, nodes A and B are marked with a red 'X' and crossed out, indicating they have failed. Only node C remains active. A question mark is shown to the right. Below the diagram, a box contains the text: 'No one will become Leader. C will become Follower. Reason: Majority is not available.' The footer of the slide shows 'Cloud Computing and Distributed Systems' and 'Design of Zookeeper'.

So, this will become a follower, there will not be any leader present in that case.

(Refer Slide Time: 28:35)

Why do we need majority?

- Imagine: We have an ensemble spread over two data centres.

The diagram illustrates a distributed system with two data centers. Data Center - I contains nodes A, B, and C, where A is designated as the 'Leader'. Data Center - II contains nodes D, E, and F. A horizontal double-headed arrow connects the two data centers. Each data center has a local network of nodes connected by single-headed arrows pointing towards the leader node.

Cloud Computing and Distributed Systems Design of Zookeeper

Let us say if you do not agree to it regarding the majority, let us see what problem will happen. Let us say that you have an ensemble two different data centers; data center one has three; another data center also has three. So, there are total how many, six are there.

(Refer Slide Time: 29:05)

Why do we need majority?

- Imagine: The network between data centres got disconnected.
If we did not need majority for electing Leader,
- What will happen?

This diagram shows the same setup as the previous slide, but with a pair of large scissors cutting through the horizontal link between the two data centers. This visualizes a network partition where the two data centers are no longer connected. Node A remains the 'Leader' in Data Center - I, and node D remains the 'Leader' in Data Center - II.

Cloud Computing and Distributed Systems Design of Zookeeper

Now, if this particular link is broken, both these data will elect the leader.

(Refer Slide Time: 29:14)

Why do we need majority?

Each data centre will have their own Leader.
No Consistency and utter Chaos.
That is why it requires majority. ✓

The diagram shows two separate data centers, Data Center - I and Data Center - II. Data Center - I contains nodes A, B, and C, with A being the leader. Data Center - II contains nodes E, F, and G, with F being the leader. A red circle highlights node A in Data Center - I and node F in Data Center - II. A pair of scissors is shown cutting a line between the two data centers, symbolizing they are disconnected. Handwritten annotations include: 'Leader' above node A, 'Leader' above node F, 'Data Center - I' below node B, 'Data Center - II' below node E, 'No master' above node G, '1 (majority) majority - 4' pointing to the 4 nodes in each center, and '2 (majority)' pointing to the 3 nodes in each center.

Cloud Computing and Distributed Systems Design of Zookeeper

They will elect their own leader in this particular scenario, when there is no link. This will non-ensure any consistency, and it will be utter chaos situation. That is why, a majorities. 4 is required to be in the majority number, (Refer Time: 29: 43) they basically are disconnected. Then they will be only 3 that means, not in the majority. Therefore, this particular leader cannot be there be that means, both of them should not have any leader the slave. So, the majority is very much important to ensure coordination at all points of time.

(Refer Slide Time: 30:19)

Sessions

- Lets try to understand **how do the zookeeper decides to delete ephemeral nodes** and takes care of session management.
- A client has list of servers in the ensemble
- It tries each until successful.
- Server creates a new session for the client.
- A session has a timeout period - decided by caller

Cloud Computing and Distributed Systems Design of Zookeeper

Now, then there is a concept of a session, let us try to understand how the zookeeper delete the ephemeral nodes and take care of the session management. So, a client has a list of servers in the ensemble, till it is successful. So, the server create a new session for a client; a session has a time period that is decided by the caller.

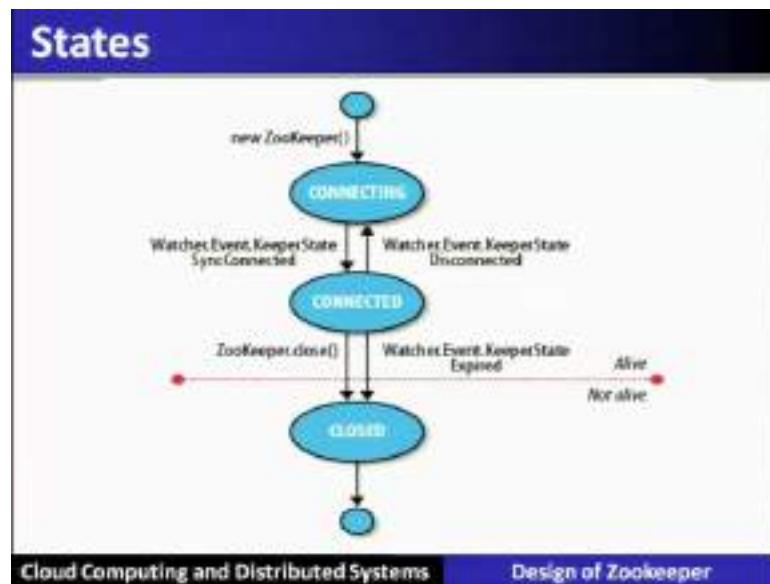
(Refer Slide Time: 30:51)

Contd...

- If the server hasn't received a request within the timeout period, it may expire the session.
- On session expire, ephemeral nodes are lost
- To keep sessions alive client sends pings (heartbeats)
- Client library takes care of heartbeats
- Sessions are still valid on switching to another server
- Failover is handled automatically by the client
- Application can't remain agnostic of server reconnections
- because the operations will fail during disconnection.

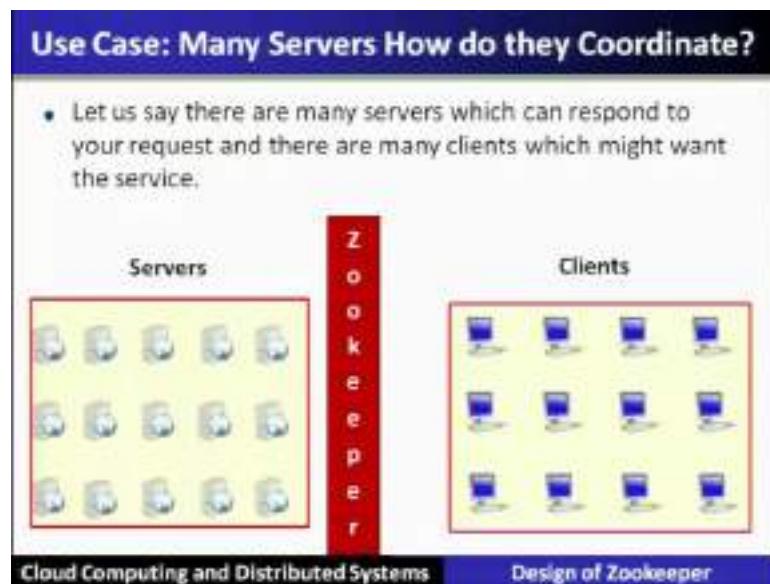
Server has not received the request within a particular timeout, it may expire the session. On session expiry, ephemeral nodes are lost. The client sends the server; and a failover is main agnostic of the server reconnections, because of the operations will fail during the disconnections.

(Refer Slide Time: 31:08)



So, these are the transitions of these states, which the watcher will keep track and keep on informing about the coordination.

(Refer Slide Time: 31:29)



Now, here we consider we have seen or we are looking in the slide that there are many servers available; similarly on the other side, we have many clients. Not all servers are responding to the client for a particular service. So, for a particular service, the client will contact to the zookeeper, and zookeeper will find out who are the servers. It will try to find out the ephemeral servers, who are active in by the zookeeper. So, zookeeper comes

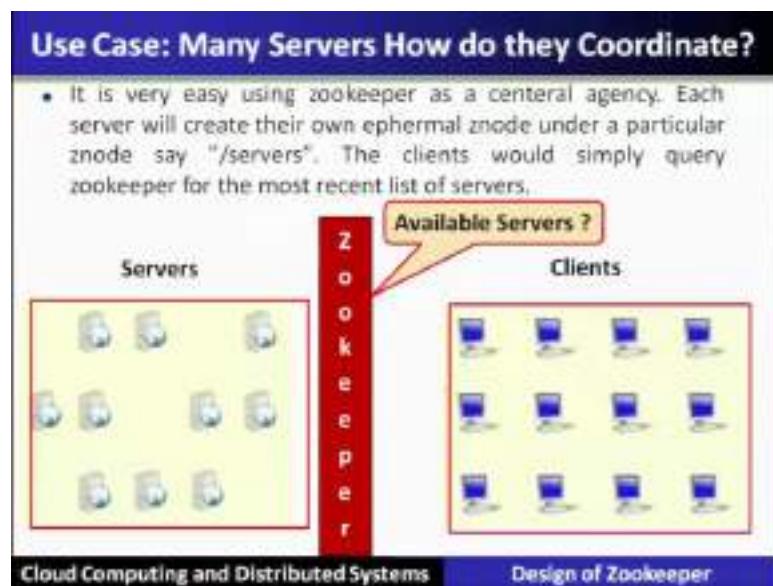
in between that is on one side, lot of servers are there; on the other side, lot of clients are there, and zookeeper is in between to coordinate between them for an applications.

(Refer Slide Time: 32:20)



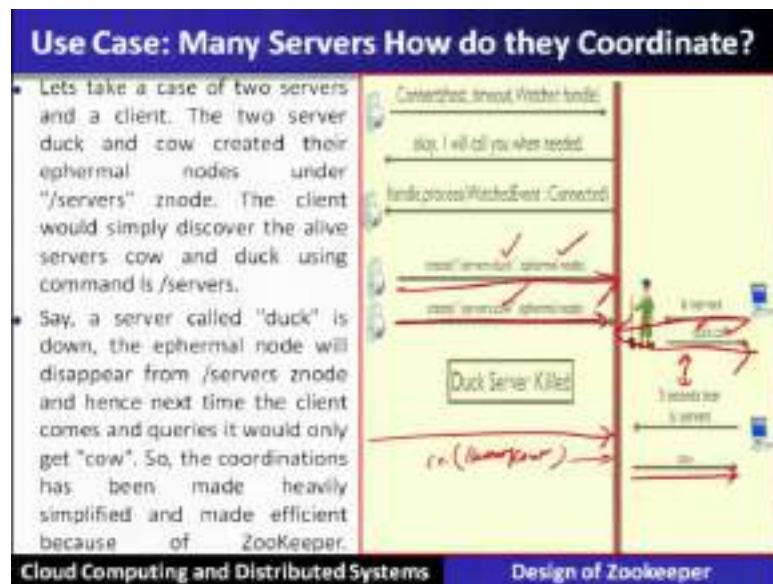
So, (Refer Time: 32:22) require, because the server are keep coming up and going down at different point of and the client, who are the available server (Refer Time: 32:33) who will basically do this coordination. To do this coordination, clients are provided with the very simple interface that is a file system kind of interface. So, client will think that it is writing on a particular file node or a znode. So, the operations to access large number of servers becomes quite simple.

(Refer Slide Time: 32:58)



So, zookeeper is becoming a central agency here. So, each server will create its own ephemeral node that is znode under a particular directory that is slash servers. So, the client would simply query for the most recent list of servers in this manner.

(Refer Slide Time: 33:21)



So, client will query about the servers. So, all the ephemeral servers who are basically active at that point of time, will get respond by the zookeeper in the form of a list. For example now, so this list now after some elapse after some point of particular servers are not active or let us say that (Refer Time: 33:56) the server that is called cow remains

visible at that point of time as an ephemeral node only the cow. So, the zookeeper will coordinate between the set of servers and the client to ensure the availability of the servers. And this is the coordination activity of zookeeper without more detail botheration at the client sent.

(Refer Slide Time: 34:32)

Guarantees

- **Sequential consistency**
 - Updates from any particular client are applied in the order
- **Atomicity**
 - Updates either succeed or fail.
- **Single system image**
 - A client will see the same view of the system. The new server will not accept the connection until it has caught up.
- **Durability**
 - Once an update has succeeded, it will persist and will not be undone.
- **Timeliness**
 - Rather than allow a client to see very stale data, a server will shut down.

Cloud Computing and Distributed Systems Design of Zookeeper

Therefore, there are guarantees in their coordination service. The first one is called it ensure the sequential consistency are applied through the leader, and which ensures the sequential consistency, which is the strongest form of consistency. The second guarantees is done, which is called atomicity that means, all this operation read and write, they are all atomic that is updates or either fail or successful the consistency and avoids the race condition.

Single system image the client will see only a view of the system as a file system; therefore it will provide a very simple interface to program in such a large scale distributed system. Durability that once an, that is whenever in memory update is done, it has to be made persistent and also it will be stored in the log, so that it can be recovered even inspite of the failures. Timeliness also is guaranteed that a client to see the stale data, the server will shut down here in this particular case.

(Refer Slide Time: 35:58)

Operations	
OPERATION	DESCRIPTION
create	Creates a znode (parent znode must exist)
delete	Deletes a znode (mustn't have children)
exists/ls	Tests whether a znode exists & gets metadata
getACL, setACL	Gets/sets the ACL for a znode getChildren/ls Gets a list of the children of a znode
getData/get, setData	Gets/sets the data associated with a znode
sync	Synchronizes a client's view of a znode with ZooKeeper

Cloud Computing and Distributed Systems

Design of Zookeeper

There are different set of operations, which are supported in the form of APIs. Create that we have already seen, delete also, we will delete the znode, exists or ls, getACL, getData.

(Refer Slide Time: 36:16)

Multi Update	
<ul style="list-style-type: none">• Batches together multiple operations together• Either all fail or succeed in entirety• Possible to implement transactions• Others never observe any inconsistent state	

Cloud Computing and Distributed Systems

Design of Zookeeper

There are (Refer Time: 36:16) for multiple updates, so that are batch together, and they are supported in the form of transactions.

(Refer Slide Time: 36:25)

APIs

- Two core: Java & C
- contrib: perl, python, REST
- For each binding, sync and async available.

Synch:

```
Public Stat exists (String path, Watcher watcher) throws KeeperException,  
InterruptedException
```

Asynch:

```
Public void exists (String path, Watcher watcher, StatCallback cb, Object ctx)
```

Cloud Computing and Distributed Systems Design of Zookeeper

Different APIs are there.

(Refer Slide Time: 36:29)

Watches

- Clients to get notifications when a znode changes in some way
- Watchers are triggered only once
- For multiple notifications, re-register

Cloud Computing and Distributed Systems Design of Zookeeper

Now, there is also a construct, called watches the clients to get notification, whenever there is a change in znode. So, watches are triggered only once.

(Refer Slide Time: 36:41)

Watch Triggers

- The read operations exists, getChildren, getData may have watches
- Watches are triggered by write ops: create, delete, setData
- ACL (Access Control List)** operations do not participate in watches

WATCH OF ... ARE TRIGGERED	WHEN ZNODE IS...
exists	created, deleted, or its data updated.
getData	deleted or has its data updated.
getChildren	deleted, or its any of the child is created or deleted

Cloud Computing and Distributed Systems Design of Zookeeper

So, watch triggers are being supported through the APIs, using access control list.

(Refer Slide Time: 36:46)

ACLs - Access Control Lists

ACL Determines who can perform certain operations on it.

- ACL is the combination**
 - authentication scheme,
 - an identity for that scheme,
 - and a set of permissions
- Authentication Scheme**
 - digest - The client is authenticated by a username & password.
 - sasl - The client is authenticated using Kerberos.
 - ip - The client is authenticated by its IP address.

Cloud Computing and Distributed Systems Design of Zookeeper

So, access control list will determine who can perform certain operations on that particular data that is znode. So, ACL is a combination of authentication scheme, an identity for that scheme, and a set of permissions. For example, the authentication schemes, which are used in the access control of the operations are digest that is the simplest form of authentication, where the client is authenticated only by the user and password. sasl is a authentication scheme, where the client is having the session token

that is it is being authenticated using Kerberos. And hence, that particular token is available, and he is authenticated already by the Kerberos to used in the system. IP based authentication the machine from which the client is accessing, that also is one of the authentication scheme used in ACL that is Access Control List.

(Refer Slide Time: 38:07)

Use Cases

Building a reliable configuration service

- A Distributed lock service
 - Only single process may hold the lock

Cloud Computing and Distributed Systems Design of Zookeeper

The reliable configuration service. Another one is called distributed lock service.

(Refer Slide Time: 38:13)

When Not to Use?

1. To store big data because:
 - The number of copies == number of nodes
 - All data is loaded in RAM too
 - Network load of transferring all data to all Nodes
2. Extremely strong consistency

Cloud Computing and Distributed Systems Design of Zookeeper

And when not to use them is for storing the big data, because number of copies all are stored in memory. Therefore, and also it is supporting the strong consistency, which is not to be used in a big data scenario.

(Refer Slide Time: 38:30)

ZooKeeper Applications: The Fetching Service

- **The Fetching Service:** Crawling is an important part of a search engine, and Yahoo! crawls billions of Web documents. The Fetching Service (FS) is part of the Yahoo! crawler. Essentially, it has master processes that command page-fetching processes.
- The master provides the fetchers with configuration, and the fetchers write back informing of their status and health. The main advantages of using ZooKeeper for FS are recovering from failures of masters, guaranteeing availability despite failures, and decoupling the clients from the servers, allowing them to direct their request to healthy servers by just reading their status from ZooKeeper.
- Thus, FS uses ZooKeeper mainly to manage configuration metadata, although it also uses ZooKeeper to elect masters (leader election).

Cloud Computing and Distributed Systems

Design of Zookeeper

Zookeeper applications. There are many zookeeper, let us scan through one by one. The first type of application, which we are discussing is about fetching service. Now, we have seen that the crawling is an important part of the search engine, so the different search engines let us say Yahoo! crawls billions of web documents. And the fetching service is a part of the crawler. Essentially, it has a master processes that commands page fetching processes.

So, the master provides that forming their status and the health. The main advantage are the recovering from the failures of the masters, guaranteeing availability, despite failures, decoupling the clients from the servers, allowing them to directly to direct their request to the healthy servers by just reading their status from zookeeper. Thus, fetching service uses the zookeeper mainly to manage the configuration metadata, although it also and the leader election.

(Refer Slide Time: 39:55)

ZooKeeper Applications: Katta

- **Katta:** It is a distributed indexer that uses Zoo-Keeper for coordination, and it is an example of a non-Yahoo! application. Katta divides the work of indexing using shards.
- A master server assigns shards to slaves and tracks progress. Slaves can fail, so the master must redistribute load as slaves come and go.
- The master can also fail, so other servers must be ready to take over in case of failure. Katta uses ZooKeeper to track the status of slave servers and the master (**group membership**), and to handle master failover (**leader election**).
- Katta also uses ZooKeeper to track and propagate the assignments of shards to slaves (**configuration management**).

[Cloud Computing and Distributed Systems](#) [Design of Zookeeper](#)

Another application of zookeeper is katta. So, in katta, it is a distributed indexer that uses zoo-keeper for the coordination. So, katta device the work of indexing into the shards, the master server assigns the shards, to the slaves and track the progress. The slaves can fail, so the master can redistribute also fail. Servers must be ready to take over katta uses zookeeper to track the status of the slave and the master that is it mention it ensures the group membership, and also handles the master failover that is using the leader election. So, zookeeper is used in two different forms that is as the group membership, and as the leader election. So, katta also uses zookeeper to track and propagate the assignment of the shards to the slaves that is configuration management. So, all three different use cases we can see.

(Refer Slide Time: 40:53)

ZooKeeper Applications: Yahoo! Message Broker

- **Yahoo! Message Broker (YMB)** is a distributed publish-subscribe system. The system manages thousands of topics that clients can publish messages to and receive messages from. The topics are distributed among a set of servers to provide scalability.
- Each topic is replicated using a primary-backup scheme that ensures messages are replicated to two machines to ensure reliable message delivery. The servers that makeup YMB use a shared-nothing distributed architecture which makes coordination essential for correct operation.
- YMB uses ZooKeeper to manage the distribution of topics (configuration metadata), deal with failures of machines in the system (failure detection and group membership), and control system operation.

[Cloud Computing and Distributed Systems](#) [Design of Zookeeper](#)

The next application yahoo message is a distributed publish subscribe system. The system manages thousands of topic that the client can publish messages and receive the messages from the topics are distributed among the set of servers to provide scalability. Each that ensures messages are replicated to two machines to ensure the reliable message delivery. The servers that message broker use a shared in a coordination essential for correct operation;, zookeeper to manage the distribution of topics that is called configuration management, the failures of the machine that is the failure detection through the zookeeper and also the group membership, and control system operations in detail, the example shown over here about this message broker.

(Refer Slide Time: 41:46)

ZooKeeper Applications: Yahoo! Message Broker

- Figure, shows part of the znode data layout for YMB.
- Each broker domain has a znode called nodes that has an ephemeral znode for each of the active servers that compose the YMB service.
- Each YMB server creates an ephemeral znode under nodes with load and status information providing both group membership and status information through ZooKeeper of YMB.

```
graph TD; root["root_znode"] --> proxy["proxy"]; root --> role["role"]; root --> replica_joined["replica_joined"]; root --> topic["topic"]; root --> topic_name["topic_name"]; proxy --> electro1["electro"]; proxy --> electro2["electro"]; electro1 --> load["load"]; electro1 --> status["status"]; load --- loadBox["load"]; status --- statusBox["status"]; topic --> zmq["zmq"]; topic --> kafka["kafka"]; zmq --- zmqBox["zmq"]; kafka --- kafkaBox["kafka"];
```

Figure: The layout of Yahoo! Message Broker (YMB) structures in ZooKeeper

Cloud Computing and Distributed Systems Design of Zookeeper

(Refer Slide Time: 41:51)

More Details

ZooKeeper: Wait-free coordination for Internet-scale systems

Patrick Hunt and Mahadev Konar
Yahoo! Grid
[phunt, mahadev]@yahoo-inc.com

Flavio P. Junqueira and Benjamin Reed
Yahoo! Research
[fpj, breed]@yahoo-inc.com

See: <https://zookeeper.apache.org/>

Apache ZooKeeper™

Cloud Computing and Distributed Systems Design of Zookeeper

Now, for more details about the zookeeper, wait-free coordination from the people at Yahoo. So, for more details about the zookeeper, you are welcome to see this particular paper.

(Refer Slide Time: 42:09)

Conclusion

- **ZooKeeper** takes a wait-free approach to the problem of coordinating processes in distributed systems, by exposing wait-free objects to clients.
- **ZooKeeper** achieves throughput values of hundreds of thousands of operations per second for read-dominant workloads by using fast reads with watches, both of which served by local replicas.
- In this lecture, the basic fundamentals, design goals, architecture and applications of ZooKeeper.

Cloud Computing and Distributed Systems Design of Zookeeper

Conclusion. Zookeeper takes a wait-free approach processes in the distributed systems, by exposing wait-free objects to the clients. Zookeeper achieves the throughput of operation per second for read-dominant workloads by using fast reads with watches, both of which are the importance of the coordination in different distributed applications. We have also seen many applications, which require zookeeper for either the leader election; or the group membership; or for the failure detection particular lecture. Zookeeper is now being used at different services, which are running either by the blue chip companies uses this zookeeper as a service.

So, in this lecture, we have covered the basic fundamentals that is how the failure toll the zookeeper. This Particular (Refer Time: 43:36), if it is going to be tolerant with one fault, then at least $2f + 1$ number of servers are required. So, we have seen that to tolerate the failure of one master, at least three different servers are required. To tolerate two failures, it requires seven different servers.

We have also seen the role of majority in the zookeeper; majority is also a kind of quorums. So, if majority is not available, then there is no leader and all of the servers will become the followers. Hence, to ensure that the zookeeper is doing its coordination service for the application, it is very much required that the availability of majority ensemble is very much needed for the correct behavior or the working of a zookeeper.

The zookeeper is not done in one server; at least three servers are required because of the fault tolerance. So, the scaling of zookeeper is very much needed as the size of the application grows. So, also the tolerances many servers are required to run the zookeeper service. Design goals; and also seen the architecture that is there are two different phases; one is called the leader election, where in it will elect the leader out of the ensemble, and the second phase atomic broadcast, we have seen that for doing the updates or the writes the clients have to contact through the server through the leader.

So, the leader will now update the slaves. And if the majority has agreed, then only the update is finalized. Why the majority or the quorum that we have already seen that it will ensure the consistency that means, if we take the intersection of any two majority or the quorums, there must be one at least one common member, who has already updated. So, for the writes, if it is followed the majority or the quorum that means, they will support the strong consistency without asking from call.

So, all the concepts, which we have seen is being applied here in the zookeeper and we have also seen different applications of the zookeeper, and this gives a good fundamentals, how the cloud system and distributed system in a fault tolerant way are used in the production scenarios.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 10
Time and Clock Synchronization

Time and Clock Synchronization in cloud data centers.

(Refer Slide Time: 00:19)

Preface

Content of this Lecture:

- In this lecture, we will discuss the fundamentals of clock synchronization in cloud and its different algorithms.
- We will also discuss the causality and a general framework of logical clocks and present two systems of logical time, namely, lamport and vector timestamps to capture causality between events of a distributed computation .



The diagram shows two rectangular nodes representing cloud servers. Each node has a digital clock at the top and a vertical timeline below it. A green line labeled t_1 connects the top of the left node's timeline to the top of the right node's timeline. A red line labeled t_2 connects the bottom of the left node's timeline to the top of the right node's timeline. A blue line labeled t_3 connects the bottom of the left node's timeline to the bottom of the right node's timeline. This visualizes how clock synchronization allows nodes to coordinate their internal timelines despite physical differences.

Cloud Computing and Distributed Systems Time and Clock Synchronization

Preface; content of this lecture. We will discuss the fundamentals of clock synchronization, in cloud and also it is different algorithms. Before that we will discuss the importance of time in the distributed system and the cloud systems. We will also cover causality a general framework to avoid the physical clock synchronizations using logical clocks. And we will cover the different logical clock such as Lamport clock, vector clocks to capture the causality between the events in the distributed system and the cloud system.

(Refer Slide Time: 01:21)

Need of Synchronization

- You want to catch a bus at 9.05 am, but your watch is off by 15 minutes
 - What if your watch is Late by 15 minutes?
 - You'll miss the bus!
 - What if your watch is Fast by 15 minutes?
 - You'll end up unfairly waiting for a longer time than you intended
- Time synchronization is required for:
 - Correctness
 - Fairness

In Distributed Systems, Cloud Systems
Time Correctness & fairness - Synchronize Clocks
Ensuring it is Fairly!

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, let us see the importance of time. So, time is one of the most crucial in the distributed system, and as also in the cloud system. But the support for the time and the clock synchronization is nontrivial, it is difficult. In this particular discussion, we will first understand why the timing is required to be maintained in the distributed systems and in particular the cloud computing systems.

Let us see that you want to catch a bus at 9. 05 am, but your watch is off by 15 minutes. In case your watch is running late by 15 minutes and the bus will depart at 9 o clock. So, definitely you are going to miss the bus. On the other hand, if your watch is running fast by 15 minutes, then you will reach earlier and unfairly waiting for a longer time.

So, this brings up the synchronization of the clock. And we have also seen that if the clock is running late, then you are going to miss a bus; that means, to run the applications and that to ensure the correctness, the clocks need to be synchronized. Hence the correctness is required using the time synchronization of the clocks in the distributed systems. Similarly, if your watch is fast by 15 minutes, then you will reach earlier and you will be unfairly waiting for a longer time. This leads to a fairness that is the compromise with the fairness.

So, time synchronization here also requires that to ensure the fairness in distributed system and a cloud computing system. So, therefore, to ensure the correctness and

fairness in the distributed systems and the cloud systems, to ensure correctness and fairness, there is a need to synchronize the clocks running in these systems.

Now, the question is we will see how this particular synchronization is to be done in such a system.

(Refer Slide Time: 05:17)

The slide has a dark blue header bar with the title "Time and Synchronization". Below the header, there are three main bullet points:

- **Time and Synchronization**
("There's is never enough time...")
- **Distributed Time**
 - The notion of time is well defined (and measurable) at each single location
 - ✓ But the relationship between time at different locations is unclear
(Distributed Sys, Cloud System)
- Distributed Clock Synchronization
- **Time Synchronization is required for:**
 - Correctness
 - Fairness

At the bottom of the slide, there are two tabs: "Cloud Computing and Distributed Systems" and "Time and Clock Synchronization". The "Time and Clock Synchronization" tab is highlighted with a blue background.

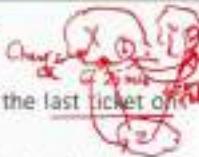
Now, synchronization of a clock in a single system is a quite trivial, but if there is a distributed system and a cloud computing system where the time is distributed, then the synchronization of the clocks to have a consistent distributed timing at all different location is nontrivial is very, very difficult. So, we are going to see the second case here in our model, which is nothing but a distributed system and the cloud computing system, to ensure a distributed clock synchronization.

So, as we have already seen that time synchronization in this environment that is in the distributed model ensures the correctness and fairness in the different services and applications.

(Refer Slide Time: 06:42)

Synchronization in the cloud

Example: Cloud based airline reservation system:



- Server X receives a client request to purchase the last ticket on a flight, say PQR 123.
- Server X timestamps the purchase using its local clock as 6h:25m:42.55s. It then logs it. Replies ok to the client.
- That was the very last seat, Server X sends a message to Server Y saying the "flight is full".
- Y enters, "Flight PQR 123 is full" + its own local clock value, (which happens to read 6h:20m:20.21s).
- Server Z queries X's and Y's logs. Is confused that a client purchased a ticket at X after the flight became full at Y.
- This may lead to full incorrect actions at Z.
Clocks at X, Y are not synchronized.

Cloud Computing and Distributed Systems Time and Clock Synchronization

For example, let us say that we are running an airline reservation system using cloud. And we will see the scenario where the synchronization of the clocks are required. So, assume that there is a server x on the cloud and there is another server y and a client request for a reservation. So, the client will send a request to purchase the ticket, which is the last ticket on a particular flight which is maintained at the server x. Server x timestamp the purchase as 6 hours and 25 minutes. That was the last ticket then after the purchase of the ticket and that timestamp at which time this particular ticket was sold out to the client, it will be logged and it will send the reply ok, back.

Since that was the last seat, server x responds to the client and when the next request now comes from the through the server y about booking an another ticket for the same flight which is already full. So, the server x will send the message that it is full to the server y. Now y will enter that the flight is full and will store this information with its own local time. So, his clock will give another time that is 6 hour 20 minutes. Now another server z queries to x and to y regarding this particular flight seats, but when it gets this information that server x has booked the ticket, the last ticket at 6 hours 25 minutes and y at the time 6 hour 20 minutes the flight was full.

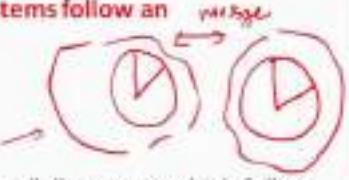
Then in that case the server z will be confused, why because the earlier time that is 6. 20 the server y is showing that the all the seats are filled whereas, later time that is 6. 25 the server x could book a ticket that is the last ticket. Why this problem is there because both

these server x and y their clocks are running different timings. Hence their clocks are not synchronized. So, this particular problem arises because the clocks at x and y are not synchronized.

(Refer Slide Time: 10:47)

Key Challenges

- End-hosts in Internet based systems (like clouds)
 - Each have its own clock
 - Unlike processors (CPUs) within one server or workstation which share a system clock.
- Processes in internet based systems follow an asynchronous model.
 - No bounds on
 - Messages delays ✓
 - Processing delays ✓
 - Unlike multi-processor (or parallel) systems which follow a synchronous system model ✓



Cloud Computing and Distributed Systems Time and Clock Synchronization

Now, what are the key challenges in clock synchronization problem in distributed systems and the systems like cloud?

Now, here the hosts which are the part of a cloud system and which are internet based connection with each other. Each host in this scenario has its own local clock and unlike the CPUs within one server or a workstation, which share a common clock here there is an internet based connectivity. So, they have the network and through the network they can communicate they do not have any shared global clock. So, that becomes a problem. So, the processes in the internet based system follow an asynchronous distributed system model. Asynchronous model we have seen in the earlier lecture that they are messages when they come communicate.

For example, these 2 clocks are running in 2 different cloud systems. They can only communicate through the messages. And these messages the delays are not predictable or are not bounded similarly the processing delays are also not bounded in such a system. This asynchronous model is different than the synchronous model where they were in such a system there is a common shared memory where the messages delays are bounded. So, in this scenario the clock synchronization that is in the

cloud system, the clock synchronization across different hosts is nontrivial and we will see that in the asynchronous model how the clock synchronization will create a problem and then you will see the technologies or techniques to solve it.

(Refer Slide Time: 13:18)

Definitions

- An asynchronous distributed system consists of a number of **processes**.
- Each process has a state (**values of variables**).
- Each process takes **actions** to change its state, which may be an **instruction** or a **communication action** (**send, receive**).
- An **event** is the occurrence of an action. (internal, send, receive)
- Each process has a large clock – events within a process can be assigned **timesteps**, and thus **ordered linearly**.
- But- in a **distributed system**, we also need to know the time order of events **across** different processes.

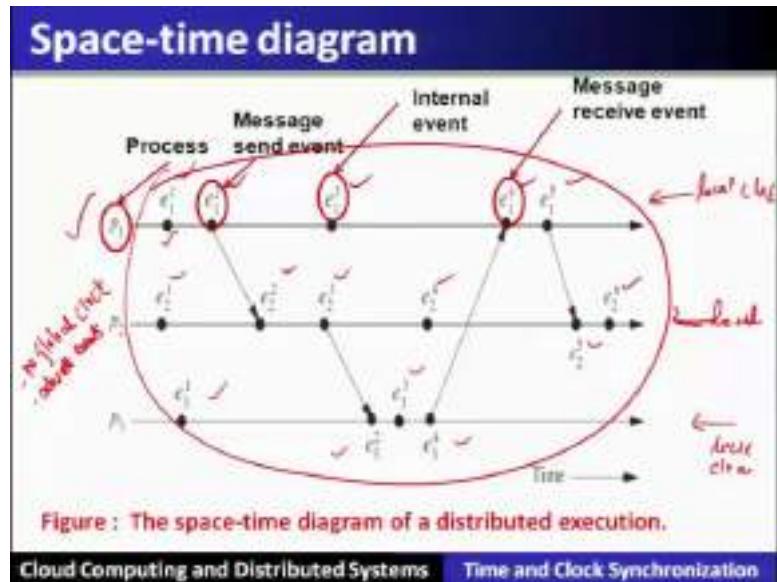
Cloud Computing and Distributed Systems Time and Clock Synchronization

So, let us see the definitions of this asynchronous distributed system model. So, let us consider that the distributed system which is an asynchronous model consists of number of processes and each processes has its defined state. So, state of a process comprises of its data, its stack, its heap, its program counter and different registers. So, it is all collapsed and we call it as the state of a process. Now these process takes different actions to change its states which maybe a instruction or a communication messages so; that means, there are 3 different ways by which states can change the first is when the instruction is executed, the second one when a communication takes place that is in the form of a send or a receive of a message

So, internally we define there are 3 different events in the state transitions of such a model, they are the internal events in the form of instructions; then send of a message is an event and a receive of a message is an event. So, each process has a local clock and the events within a particular process which is called a internal events can be assigned with a timestamp of its local clock and all the events which are called internal events ordered linearly, but in a distributed systems where there are a large collection of such host

which runs their own clocks. So, we need to know the order we need to know the time to order the events across the different processes

(Refer Slide Time: 15:35)



Let us see the example using this particular diagram that a particular process one, it is events 1 2 3 4 5; they are being time through our local clock. Similarly this event P 2 also has its own clock and can order the events 1 2 3 4 and 5, similarly P 3 also has the local clock which can time this events and using that time all these events can be linearly ordered that is shown here in this space time diagram; however, if we take all the events and want to order them, then there is no global clock which can order all the events there is no global clock which can even which can order all the events in the distributed system.

(Refer Slide Time: 16:58)

Clock Skew vs. Clock Drift

- ✓ • Each process (running at some end host) has its own clock.
- When comparing two clocks at two processes:
 - ✓ • Clock Skew = Relative difference in clock values of two processes.
 - Like distance between two vehicles on road.
 - ✓ • Clock Drift = Relative difference in clock frequencies (rates) of two processes
 - Like difference in speeds of two vehicles on the road.
- ✓ • A non-zero clock skew implies clocks are not synchronized → ~~drift away~~
- ✓ • A non-zero clock drift causes skew increases (eventually).
 - ✓ • If faster vehicle is ahead, it will drift away.
 - If faster vehicle is behind, it will catch up and then drift away.

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, to do this to see how this, in the absence of a global clock how the events are to be ordered in a distributed system; we will see through a systematic theory.

So, let us see some more definitions in this respect. First we will see the 2 definitions one is the called clock skew the other is called clock drift. Now as you know that each process is running it is own local clock. So, when you compare the clocks of 2 different processes. So, there are 2 things which will immediately be; one is the current timing of both the clocks. Now if these times are not same they are different then it is called a clock skew. So, clock skew is nothing but the difference in the timings across 2 different clocks running in different processes, just like on the road 2 vehicles are running. So, that distance between these 2 vehicles is called the skew similarly the clocks are running with a different timings. So, the difference in the timings is called a clock skew.

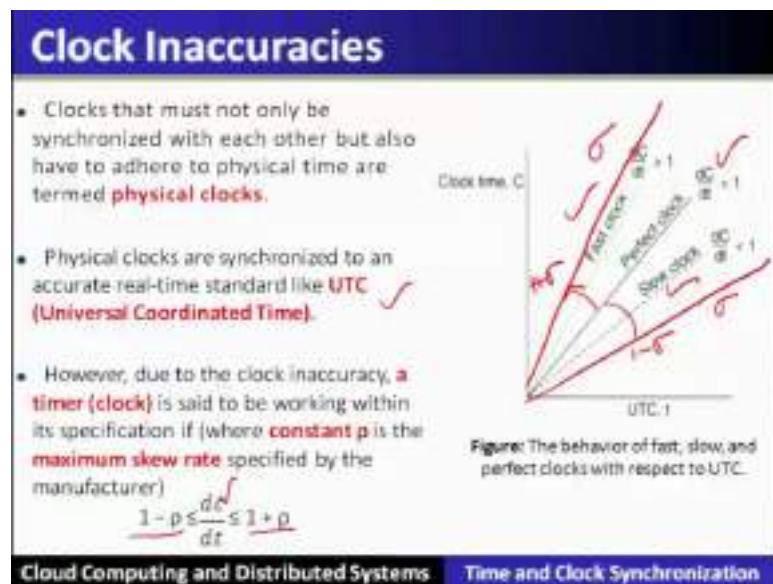
Now, second thing second definition is about the clock drift. Now these clocks runs with a different speed. Speed of a clock 1 speed of a clock 2. Sometimes it is also called as a rate and some people also called as a frequency by which these clocks are running. If they have the same speed or a rate or the frequency, then there is no clock drift, but if the frequencies or the speed is different, then it is called a clock drift. So, that relative difference in the clock frequencies of the rate or the rates of the 2 processes is called a clock drift. For example, 2 vehicles which are running on the road with a different speed;

let us say it is 80 kilometer per hour, and the other one is quite slow that is 40 kilometer per hour.

So, their speeds are different. So, although the 80 kilometer per hour that is a fast running vehicle is behind the slow running vehicle, but after sometime it will cross and then will run away then; that means, drift apart drift away. So, the speed is also important. So, the timing the difference in the timing is called a skew and the difference in the speed is called the drift. So, in this particular clocks or multiple clocks, where more than 2 clocks are there, then these 2 factors are going to play important role in the synchronization.

So, therefore, a non 0 clock skew implies that the clocks are not synchronized; that means, they are giving different timings. Just like in the previous example we have seen one clock was giving the time 6:25, the other clock was giving the time at 6:20. So, they are running at different timings. So, non-zero clock skew implies that the clocks are not synchronized. A non-zero clock drift causes this skew; that means, the difference in their timing to increase eventually. For example, if a faster vehicle ahead, then it will drift away. And if the faster vehicle is behind, then it will catch up and then it will drift away.

(Refer Slide Time: 21:58)



So, having understood the clock skew and clock drift, let us see the clock inaccuracies. So, as far as the physical clocks are concerned that we use; so, not only they have to synchronize with each other, but also has to adhere that physical time, that is the physical

time a common time or accurate time. That is called physical clocks. So, how to get that particular correct time so, that you can synchronize your physical clocks? So, physical clocks are synchronized to the accurate real time, with a standard like UTC universal coordinated time which is supposed to maintain the correct time at all point of time; however, due to the clock inaccuracies a timer of the clock is said to be working within the specifications. So, let us see what are the specifications.

Now, if a constant rho is the maximum skew rate, specified by the manufacturer. It has to be bounded within $1 + \rho$ and $1 - \rho$. So, let us see that dC / dt , if it is 1, then it is the perfect clock; that means, it is synchronized with the standard time. Now if dC / dt is less than 1, then it is a slow clock, and if it is fast, then it is a fast clock. And if it is bounded by these values rho; that means this is $1 + \rho$ and this is $1 - \rho$. Then this is acceptable as far as manufacturer is concerned, but beyond that it need to designated as clock inaccuracies.

(Refer Slide Time: 24:04)

How often to Synchronize

- Maximum Drift rate (MDR) of a clock
- Absolute MDR is defined to relative coordinated universal Time (UTC). UTC is the correct time at any point of time.
 - MDR of any process depends on the environment.
- Maximum drift rate between two clocks with similar MDR is $2 * MDR$.

$$\text{Max drift} = 2 * MDR$$
- Given a maximum acceptable skew M between any pair of clocks, need to synchronize at least once every:

$$M / (2 * MDR)$$
 - Since time = Distance/ Speed.

M = Max acceptable skew
MDR = Clocks to be synchronized

Cloud Computing and Distributed Systems Time and Clock Synchronization

Now, the question is how often you need to synchronize the physical clocks that depends upon a parameter which is called a maximum drip rate MDR of a clock. The absolute MDR is defined to be the relative to the coordinated universal time UTC. So, MDR of any process depends on the environment. So, the maximum drift rate between the 2 clocks with the similar MDR = $2 * MDR$. Because if we consider 2 clocks, maybe one clock is running behind with the rate MDR and the other clock is running ahead of MDR,

then total drift rate here in this case will be $2 * MDR$. That is the maximum drift rate. Now given the maximum acceptable clock skew that is m , as you know this is the time, between any pair of the clocks. So, therefore, the amount of time duration when they are required to be synchronized will be this m divided by the script that is $2 * MDR$. So, this is the interval. When the clock needs to synchronize; after every such interval, the clocks need to be synchronized.

(Refer Slide Time: 26:26)

External vs Internal Synchronization

- Consider a group of processes
- **External synchronization**
 - Each process $C(i)$'s clock is within a bounded D of a well-known clock S external to the group
 - $|C(i) - S| < D$ at all times
 - External clock may be connected to UTC (Universal Coordinated Time) or an atomic clock.
 - Example: Christian's algorithm, NTP
- **Internal Synchronization**
 - Every pair of processes in group have clocks within bound D
 - $|C(i) - C(j)| < D$ at all times and for all processes i, j
 - Example: Berkley Algorithm, DTP

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, having seen this kind of equation, now you will consider two different types of synchronizations. Now if we consider a group of processes in our model of a distributed system, and the cloud system there are 2 kinds of synchronization possible. One is called external synchronization; that means, if the processes used to synchronize their clocks with an external clock; that means, not in a group that is external clock which maintains the correct time then it is called external synchronization.

For example, if a process C i 's clock is within a bound D of a well-known clock S external clock S then C i minus S ; that means, in the absolute form that clock drift is to be bounded by D at all points of time; that means, it has to be synchronized, that is C i 's clock has to be synchronized whenever there is a violation of this condition. This external clock S is nothing but, it is always synchronized at any point of time using universal timing which is also called as a universal coordinated time UTC or the timing which is basically maintained as per the atomic clock.

There are two algorithm for this external synchronization, we will cover in this discussion. The first one is called Christian's algorithm the second one is called network time protocol NTP. On the other hand, the group of processes will synchronize their clocks within bound D without the external clock. So; that means, between any 2 process i and j if their clock values is having the difference or skew, then it has to be bounded by D at all points of time. This synchronization can be achieved using the algorithm which is called the Berkley's algorithm. And we will also see the algorithm which is called a data centre time protocol DTP.

(Refer Slide Time: 29:24)

External vs Internal Synchronization

- External synchronization with $D \Rightarrow$ Internal synchronization with $2*D$.
- Internal synchronization does not imply External Synchronization.
 - In fact, the entire system may drift away from the external clock SI

Cloud Computing and Distributed Systems Time and Clock Synchronization

Now, when a small note, to see here is that external synchronization with the bound D will basically implies that, it is the internal synchronization with a $2 * D$ where as if it is external synchronization that does not implies anything with the external synchronization. Because of the fact that entire system may drift away from the external clock; basic fundamentals of external time synchronization.

(Refer Slide Time: 30:02)

Basic Fundamentals

- External time synchronization
- All processes P synchronize with a time server S.

The diagram shows two horizontal timelines for Process P and Time Server S. At the top, a red arrow labeled 'Set clock to t' points from S to P. Above P, a question 'What's the time?' is followed by a red arrow pointing to S. Below S, a red arrow labeled 'Check local clock to find time t' points to P. A red handwritten note 'message latency' is written above the timelines near the arrows. The timelines are labeled 'Time' at the right end.

- What's Wrong:
 - By the time the message has received at P, time has moved on.
 - P's time set to t is inaccurate.
 - Inaccuracy a function of message latencies.
 - Since latencies unbounded in an asynchronous system, the inaccuracy cannot be bounded.

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, let us see through this example. So, all the processes let us say that P will synchronize using a standard time server S which maintains the accurate time. So, such process P sends a message to S what is the time. So, S will check the local time to find out the time let us say t, put in the message and send the message back to the process P here is the time t. So, when the message received by P, it will set its clock as time t.

So, in this particular manner what is the problem, what is the wrong? So, the problem is that, by the time the message is received at P, the time has moved on; take this particular example. So, this is the time let us say at this instant of S the time is t, but when the message is sent to t, this particular time, when this is set as time t the time of the server S will be now t 1; that means, it is a different time, but here the time which is said by P will be the old time hence by the time the message is received at P the time has moved on. So, P's time if it is set to t will become inaccurate.

Now, the inaccuracy is a function of message latency. So, inaccuracy is this particular function of a message latency; that means, if you know this latency and inform to P. So, P can set this particular value. So, that it can also be accurate or it can reduce the level of inaccuracy. Since latencies are unbounded in an asynchronous system the inaccuracies also cannot be bounded. Hence this particular simple method of synchronizing the clock through a time server S using an asynchronous system to using message passing will have an inaccurate time set.

(Refer Slide Time: 33:09)

(i) Christians Algorithm

- P measures the round-trip-time RTT of message exchange
- Suppose we know the minimum P \rightarrow S latency min1 ✓
- And the minimum S \rightarrow P latency min2
 - Min1 and Min2 depends on the OS overhead to buffer messages, TCP time to queue messages, etc.
- The actual time at P when it receives response is between $[t+min2, t + RTT-min1]$

The diagram illustrates the Christian's algorithm for time synchronization. It shows two nodes, P (Client) and S (Server), connected by a network link. A horizontal arrow labeled 'RTT' indicates the round-trip time between them. At node P, a message is sent to node S. A red bracket labeled 'Set clock to t' indicates the time at which the message is sent. Node S receives the message and sends a response back to P. A red bracket labeled 'Check local clock to find time t' indicates the time at which the response is received at P. The time interval between the sending of the message and its reception is labeled 't+min2'. The time interval between the reception of the message and the sending of the response is labeled 't + RTT - min1'. A red box highlights the interval $[t+min2, t + RTT-min1]$, which represents the actual time at P when it receives the response.

Cloud Computing and Distributed Systems Time and Clock Synchronization

Let us see how different algorithms resolves this particular inaccuracy. Christian's algorithm, P measures the round trip time which is called a RTT for the message exchange. So, suppose, we know the minimum latency from sending the message from P to S, let us say this is minimum latency. And the minimum latency if it is travelling from S to P is called as min 2. So, min 1 and min 2, these latencies highly depends upon the operating system overheads to buffer the messages then TCP time involved to queue the messages etcetera.

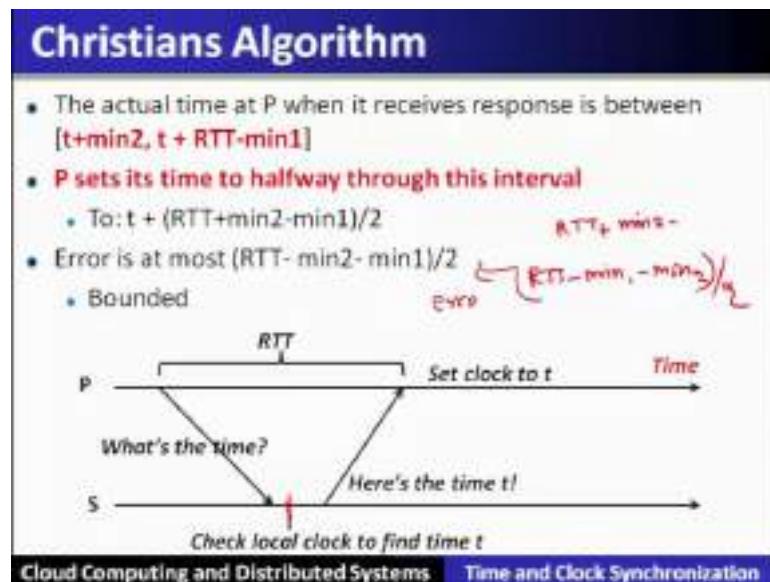
So, the actual time at P, when it is received the response is between $t+min 2$ and $t+RTT-min1$. So, let us see the what is the minimum response. So, minimum response is. So, the actual time at P when it receives the response, the minimum of that bound is t plus because, here this is the time t which is stored and send. So, this particular latency will be added. This is the minimum of latency which is involved and what is the maximum of these values?

So, maximum of these values will be the time t plus this is the round trip time RTT. That is this particular entire time is RTT. So, out of RTT let us say this particular time which is called min 2. If we abstract out of min 1 RTT - min 1 will be this much of the maximum time. So, $t+RTT-min1$ will be this particular time and this will be added here. So, that will be $t+RTT-min1$. So, this is the bound; that means, the inaccuracies. So, the

actual time at P when it receives the response false, the minimum of this particular time $t + \text{min } 2$.

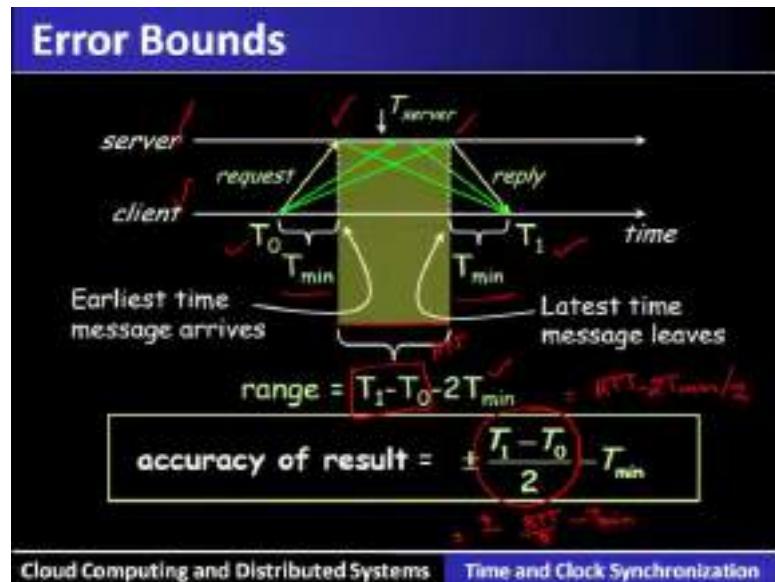
And the maximum that is upper bounded by this particular time $t + \text{RTT} - \text{min } 1$. So, P will set its time to the halfway through this particular interval; that means, $t + \text{RTT}$. So, what is that interval? If we recall that interval will be $t + \text{RTT} - \text{min } 1 - t - \text{min } 2$. That is $\text{RTT} - \text{min } 1 - \text{min } 2$. So, this is the interval and half way of this will be divided by 2. So, let us see.

(Refer Slide Time: 37:28)



So, half way will be that $\text{RTT} + \text{min } 2$. So, the error will be at most $(\text{RTT} - \text{min } 1 - \text{min } 2)/2$ that is this halfway. So, this will be the error and it is bounded by this corresponding value.

(Refer Slide Time: 38:00)



So, again let us see through this particular illustration that this is a server, which maintains the time. And this is the client which want to synchronize it is clocks. Assuming that minimum 1 minimum 2 is equal to minimum. Both are equal in that case without loss of generality; for the sake of simplicity let us understand this. So, the earliest time when the client message will be received is that this start from T_0+T_{\min} . Similarly, the server will response and the reply will be received the latest time will be when the server sends plus T_{\min} . So, that particular range will be that $T_1 - T_0$ is the round trip time. And 2 times that minimum value because both are same. It is same as the equation which we have seen. And if we divide by 2 this becomes as this is RTT minus this is t_{\min} in. So, this is the accuracy of the result of Cristian's algorithm.

(Refer Slide Time: 39:39)

Christians Algorithm: Example

- Send request at 5:08:15.100 (T_0) ✓
- Receive response at 5:08:15.900 (T_1) ✓
 - Response contains 5:09:25.300 (T_{server}) ✓
- Elapsed time is $T_1 - T_0$
 - + $5:08:15.900 - 5:08:15.100 = 800 \text{ msec}$ ✓
- Best guess: timestamp was generated
 - + 400 msec ago ✓
- Set time to $T_{server} + \text{elapsed time}$
 - + $5:09:25.300 + 400 = 5:09:25.700$ ✓

$$\text{Error} = \pm \frac{900 - 100}{2} - 200 = \pm \frac{800}{2} - 200 = \pm 200$$

If best-case message time=200 msec
 $T_0 = 5:08:15.100$ ✓
 $T_1 = 5:08:15.900$ ✓
 $T_{server} = 5:09:25.300$
 $T_{min} = 200 \text{ msec}$

Cloud Computing and Distributed Systems

Time and Clock Synchronization

So, here as far as error bounds are concerned, we know that in the synchronization of the clocks we always increase the clock value, but never decrease it. So, if you ever found to be the requirement to decrease the value then you do not do it leave it as it is.

So, it is allowed to increase or decrease the speed of the clock then. So, if the error is too high, it will take multiple readings and average them. Let us understand through another example. Here the server sends this is let us say the client will send at t_0 the request, t_0 is 5:08. It will receive the response at 5:08:15:900. The response contains the timing of the t server. So, it is t_0 and t_1 and t_{server} time 5:09:25:300. Total elapsed time is in that particular message that is a difference of timing t_1 minus t_0 that comes out to be 800 milliseconds.

So, if you guess as per the Christian's time so; that means, RTT / 2; that means, the guess says that around 400 milliseconds, the message was being originated by the t server. So, the time which will be set by the client will be the time which is sent by the t server, plus this particular average elapsed time that is 400. So, the server time is 5:09:25:300. This is the t server plus this offset value is added, comes out to be this much time. So, the clock will be set the client will set the clock value as this one.

Here we can see that t_{min} value is 200 milliseconds why because it is a 400. So, it is 200 and 200 becomes 400 we are in this particular case.

(Refer Slide Time: 42:22)

(ii) NTP: Network time protocol

- (1991, 1992) Internet Standard, version 3: RFC 1305
- **NTP servers organized in a tree.**
- Each client = a leaf of a tree.
- Each node synchronizes with its tree parent

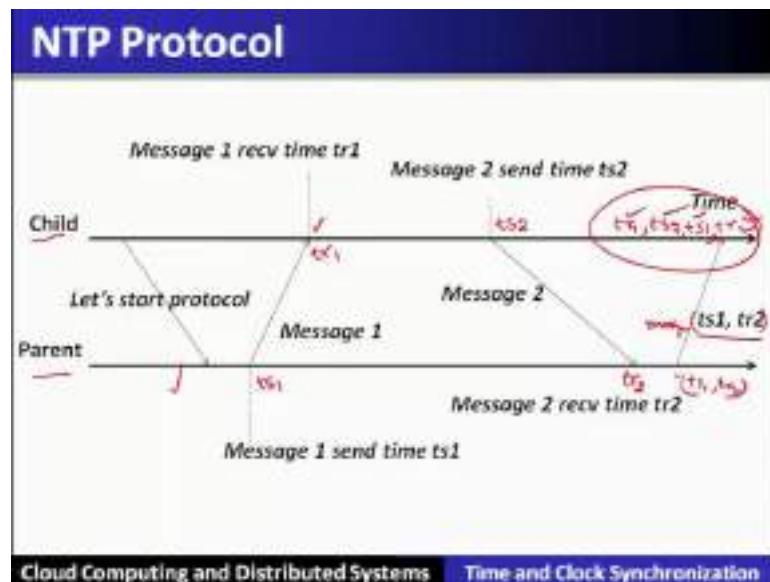
```
graph TD; PS1[Primary servers] --- SS1[Secondary servers]; PS1 --- SS2[Secondary servers]; SS1 --- TS1[Tertiary servers]; SS1 --- TS2[Tertiary servers]; SS2 --- TS3[Tertiary servers]; Client[Client] --- TS2;
```

Cloud Computing and Distributed Systems Time and Clock Synchronization

Now, we will see another time synchronization protocol which is called a network time protocol, NTP, which is a 1991 and 92 it becomes an internet standard, RFC 1305. Here it is assumed that the network time servers, they are organized in a form of a tree at each client will be at the leaf of a tree. This particular tree hierarchy of the servers are such that every child will synchronize with its parent server.

So, on the top let us see you have a primary server which is assumed to be synchronized with its time. And secondary server will synchronize their time with the primary server and tertiary server will synchronize their time with the secondary server and client, will synchronize their time from tertiary server and so on. Let us see the details of network time protocol.

(Refer Slide Time: 43:34)



Here let us see that so, every child will now synchronize its time with the parent. So, child will send a message. Let us start the protocol. And at time ts_1 the parent will send a message 1 and this particular message 1 will be received by the child at time tr_1 . Then child will send another message 2 at the time ts_2 and this particular message will be received by the parent or the server is at tr_2 .

Then this particular parent will pack these 2 messages tr_1 and tr_2 here and send in a message, which will be received by the child. So, child will now have its own timing (tr_1, ts_2) . And these 2 messages which is received from the message that is tr_2 . So, (tr_1, tr_2, ts_1, ts_2) all 4 values it has now received. Let us see how this network time protocol uses all 4 values to synchronize the clocks.

(Refer Slide Time: 45:04)

Why $o = (tr1 - tr2 + ts2 - ts1)/2$?

- Offset $o = (tr1 - tr2 + ts2 - ts1)/2$
- Let's calculate the error.
- Suppose real offset is oreal
 - Child is ahead of parent by oreal.
 - Parent is ahead of child by -oreal.
- Suppose one way latency of Message 1 is L1.
(L2 for Message 2)
- No one knows L1 or L2!
- Then
 - $tr1 = ts1 + L1 + oreal$
 - $tr2 = ts2 + L2 - oreal$

$oreal = o + \frac{(L2 - L1)}{2}$

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, using these 4 values it will calculate the offset. So, using this formula $(tr1 - tr2 + ts2 - ts1) / 2$. Let us see how this offset value is calculated.

So, for that let us calculate the total error. Let us assume the real offset is oreal. And we also assume that the child is ahead of the parent by this value oreal. And the parent is behind the child by oreal, which is written as minus oreal; that means, parent is a head of child by minus oreal. Suppose one-way latency of a message 1 is L1 and the message 2 is L2 then let us calculate the tr1. So, here let us calculate tr1. Tr1 is equal to tr1 plus let us say this particular latency is L1 plus, the child is ahead of the parent by oreal. Similarly, we can calculate tr2. Tr2 is equal to ts2 plus this latency is L2, minus oreal; that means, the parent is ahead of a child by minus oreal.

So, this is equation number 1 this is equation number 2. Now we subtract 2 from 1, we will get $tr2 - tr1 = ts2 - tr1 + L2 - L1 - 2oreal$. So, it comes out to be oreal is equal to this will go on the other side. And $ts2 - tr1$ then $- tr2 + tr1$. This is one factor plus L2 minus L1. And this will be divided by 2. So, this particular value if we rearrange it comes out to be $tr1 - tr2$. Then plus $ts2 - tr1 / 2$, that we have already obtained this value plus $(L2 - L1) / 2$. Now this is an offset. So, oreal is equal to offset $+ (L2 - L1) / 2$.

(Refer Slide Time: 48:44)

Why $\hat{o} = (tr_1 - tr_2 + ts_2 - ts_1)/2$?

- Then

- $tr_1 = ts_1 + L_1 + o_{real}$.
- $tr_2 = ts_2 + L_2 - o_{real}$.

- Subtracting second equation from first

- $o_{real} = (tr_1 - tr_2 + ts_2 - ts_1)/2 - (L_2 - L_1)/2$
- $\Rightarrow o_{real} = \hat{o} + (L_2 - L_1)/2$
- $\Rightarrow |o_{real} - \hat{o}| < |(L_2 - L_1)/2| < |(L_2 + L_1)/2|$

• Thus the error is bounded by the round trip time (RTT)

So, here, we have obtained up to this stage. $o_{real} = \hat{o} + (L_2 - L_1)/2$. So, if offset we will take on the left side. So, $|o_{real} - \hat{o}| < |(L_2 - L_1)/2| < |(L_2 + L_1)/2|$. So, this is the error bound. So, this particular error is bounded by the round trip time that is RTT.

(Refer Slide Time: 49:24)

(iii) Berkley's Algorithm

- **Gusella & Zatti, 1989**

Internal synchronization

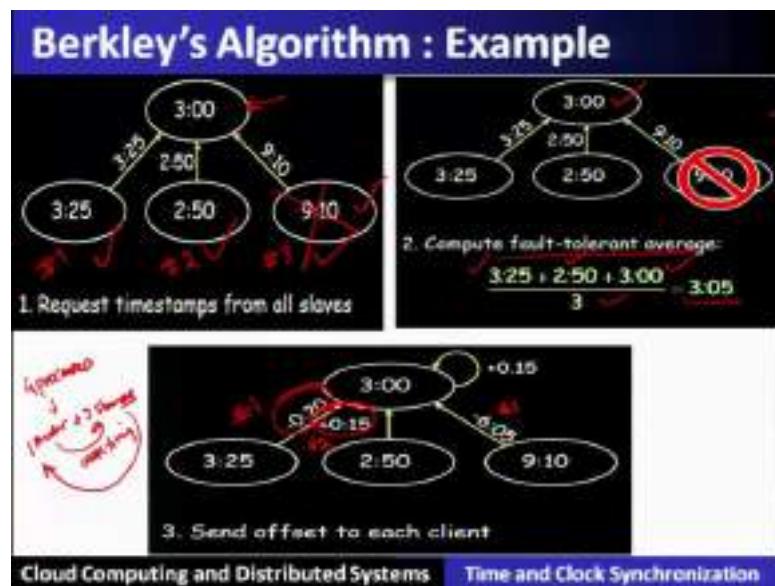
Process

- Master polls each machine periodically
 - Ask each machine for time ✓
 - Can use Christian's algorithm to compensate the network's latency.
- When results are in compute, ✓
 - Including master's time.
- **Hope: average cancels out individual clock's tendency to run fast or slow**
- Send offset by which each clock needs adjustment to each slave
 - Avoids problems with network delays if we send a time-stamp.

Now, we will see another algorithm that is called Berkley's algorithm. And this is the algorithm for internal synchronization where as the previous algorithm which we have seen was an external synchronization. Here out of the group of processes, one of these processes will be elected as the master using any leader election algorithm.

And this master will poll each other machines periodically. That is it will ask each machine for its time. Now as far as the message latency is concerned which basically will be obtained through the previous algorithm, which is the Christian's algorithm. Now when these results are computed, then using the master's time it will be computed as the offsets. Why because the average cancels out the individual clocks tendency to run fast or slow. So, only the offset will be calculated by which is clock needs to be adjusted to each slave.

(Refer Slide Time: 50:56)



Let us take this particular example, to understand this let us see 1, 2, 3, 4. In the system there are 4 processes. Out of them one is the master plus 3 slaves. So, the master will ask their timings, master will ask the timings of the slaves. So, slaves will send the timing to the master in this particular way.

So, for example, slave one is having time 3:25, slave 2 is having time 2:50, slave 3 is having time 9:10. When all these values will reach to the server is having time 3 o'clock. So, server can find out the skew of these particular timings. Now 3:25 and 2:50 according to server's time is good enough; that means, around that around his timing they are running their clocks maybe with some skew. As far as 9:10 is outlier. So, it will not consider in averaging this particular time like here. Now then it will calculate the average; that means, it will perform the averaging using his value and this average comes

out to be 3 or 5. So, with this 3 or 5 this particular time requires this particular offset for server for the slave number one, this outside for the slave number 2 that is plus 15.

And this offset will be calculated for the slave number 3 and this will send to the each client. And hence the Berkley algorithm will synchronize will do the internal synchronization.

(Refer Slide Time: 53:18)

(iv) DTP: Datacenter Time Protocol

Globally Synchronized Time via Datacenter Networks

H. Bahl, L. Eren, Weng, Yuhai Li, Kishan Mehta et al.
Computer Science Department
Cornell University
hans.bahl@cornell.edu, eren@cs.cornell.edu

ACM SIGCOMM 2016

- DTP uses the physical layer of network devices to implement a decentralized clock synchronization protocol.
- Highly Scalable with bounded precision!**
 - ~25ns (4 clock ticks) between peers
 - ~150ns for a datacenter with six hops
 - No Network Traffic
 - Internal Clock Synchronization ✓
- End-to-End: ~200ns precision!

Cloud Computing and Distributed Systems Time and Clock Synchronization

Now, we will see another time protocol which is also the internal synchronization. That is called data centre time protocol. So, this particular protocol is published in a paper that is called globally synchronized time via datacenter network, which is published in ACM SIGCOMM 2016. Here it considers the layers of the network also into an account for calculating different parameters. So, DTP uses the physical layer of the network.

So, the for example, humans use causality at all points of time. Take the example that I enter the house only if I unlock. So, it that means, unlocking of an event the time of that should always be less than the time when you enter the house. So; that means, unlocking happens before the entering into the house.

(Refer Slide Time: 58:39)

Ordering events in a distributed system

- To order events across processes, trying to synchronize clocks is an approach.
 - What if we instead assigned timestamps to events that were not absolute time ?
 - As long as those timestamps obey **causality**, that would work
- If an event A causally happens before another event B, then $\text{timestamp}(A) < \text{timestamp}(B)$
- Example: Humans use causality all the time
- I enter the house only if I unlock it
 - You receive a letter only after I send it
- ↑ (unlocking) < ↑ (enter house)
↓ unlocking → entering house
↑ (sending) < ↑ (receiving)
↳ sending happens before receiving
↳ happens before*

Cloud Computing and Distributed Systems Time and Clock Synchronization

The second example is that you receive a letter only after I send it. So, the sending of a letter happens before the receive of a letter so; that means,. So, this particular way humans are basically using the physical clock and now they are basically ordering the event, for example, here unlocking first and then entering the house or sending letter first and then receiving it later. So, without using physical clock, if you can obey this causality, then that concept can be used without even.

(Refer Slide Time: 60:07)

Logical (or Lamport) ordering

- Proposed by Leslie Lamport in the 1970s.
 - Used in almost all distributed systems since then
 - Almost all cloud computing systems use some form of logical ordering of events.
-
- Leslie B. Lamport (born February 7, 1941) is an American computer scientist. Lamport is best known for his seminal work in distributed systems and as the initial developer of the document preparation system LaTeX. Leslie Lamport was the winner of the 2013 Turing Award for imposing clear, well-defined coherence on the seemingly chaotic behavior of distributed computing systems, in which several autonomous computers communicate with each other by passing messages.



Cloud Computing and Distributed Systems Time and Clock Synchronization

The synchronization of physical clock; this concept was called as logical ordering given by the or it is also called as a lamport ordering, which was proposed by Leslie lamport in 1970. Almost all distributed system and the cloud system since then are applying this concept.

(Refer Slide Time: 60:29)

Lamport's research contributions

- Lamport's research contributions have laid the foundations of the theory of distributed systems. Among his most notable papers are
 - "Time, Clocks, and the Ordering of Events in a Distributed System", which received the PODC Influential Paper Award in 2000;
 - "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", which defined the notion of Sequential consistency;
 - "The Byzantine Generals' Problem";
 - "Distributed Snapshots: Determining Global States of a Distributed System" and
 - "The Part-Time Parliament".
- These papers relate to such concepts as logical clocks (and the happened-before relationship) and Byzantine failures. They are among the most cited papers in the field of computer science and describe algorithms to solve many fundamental problems in distributed systems, including:
 - the Paxos algorithm for consensus;
 - the bakery algorithm for mutual exclusion of multiple threads in a computer system that require the same resources at the same time;
 - the Chandy-Lamport algorithm for the determination of consistent global states (snapshot); and
 - the Lamport signature, one of the prototypes of the digital signature.

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, these are the details of the lamport's contribution. And for which he has got the Turing award that is time clocks and the ordering of events in a distributed systems.

(Refer Slide Time: 60:45)

Logical (or Lamport) Ordering(2)

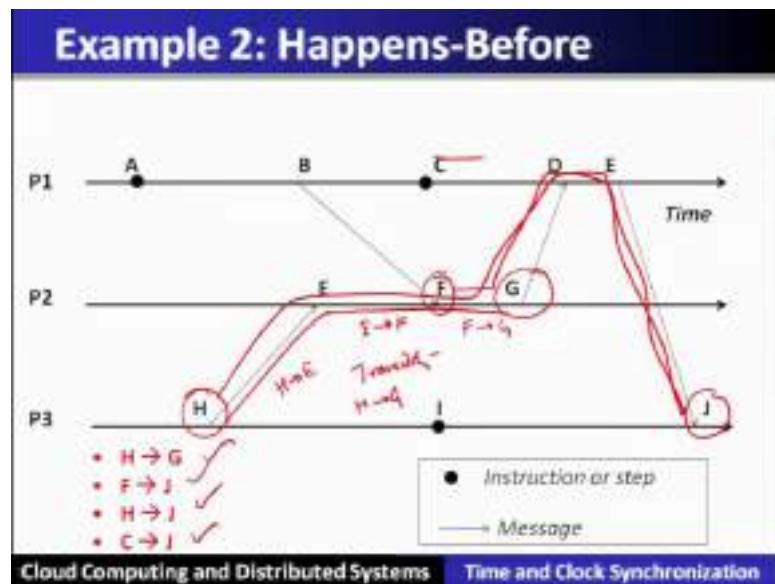
- Define a logical relation **Happens-Before** among pairs of events
- **Happens-Before** denoted as " \rightarrow "
- **Three rules:**
 1. On the same process: $a \rightarrow b$, if $time(a) < time(b)$ (using the local clock)
 2. If p1 sends m to p2: $send(m) \rightarrow receive(m)$
 3. **(Transitivity)** If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- Creates a partial order among events
 - Not all events related to each other via \rightarrow

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, let us see the logical ordering or a Lamport ordering, which defines the logical relation which is called happened before relation between the pair of events and is denoted by an arrow. So, this happens before relation is denoted by an arrow. This happens before relation follows 3 different rules. The first rule says that on the same process a has happened before b if the time(a) < time(b), using the logical clock.

The second rule says that if p sends a message m to p , then send off message m has happened before the receive of a message m . Third rule is the transitivity which says that if a has happened before b and b happened before c then a is happened before c . These 3 rules will create a partial order among all the events in the distributed system. Not all events are related to each other that is why it is a partial order via happened before relation.

(Refer Slide Time: 61:59)



Let us see through an running example. Here we have P 1, P 2, P 3. P 1 is having the events A B C D E, where A is internal event, B is a sendoff a message. F is a receive of a message at the process P 2. Similarly, event j is the received of a message from P 1 at P 3. Let us see that A is happened before B. So, A has happened before B, this we can get through the timing of the internal clock, why? Because A and B they are all they are internal event. Hence the time of A is less than time of B. So, it follows A is happened before B relation.

The second one is b has happened before F. So, b is a send off an event. So, the time of the sending off an event which is maintained by P 1 is less than the time when the message is received as far as the rule 2 is concerned. Hence, B is also happened before f, similarly A is happened before F. So, by transitivity property; that means, A is happened before B and B is happened before F therefore, A is happened before F through the transitive property. So, there is a causal path, this is also called a causal path. So, there exist causal path, it will connect all the events using this happened before relation.

Similarly, let us say that H has happened before G. Here this is the H has happened before G. So, there must exist a causal path which connects H to G; that means, H is a send off event and e is a receive of a message. So, H has happened before E Similarly E and F they are internal events of P 2. So, E has happened before F and F and G their internal events of process P 2, F has happened before G. So, using transitivity property H has happened before G. Similarly, F has happened before J. So, F F has happened before J. So, there must exist a causal path.

Similarly, H has happened before J. So, H has happened before j there exist a causal path. C has happened before J. So, from C there exist a causal path which connects C and J.

(Refer Slide Time: 65:15)

Lamport timestamps

- Goal: Assign logical (Lamport) timestamp to each event
- Timestamps obey causality
- Rules
 - Each process uses a local counter (clock) which is an integer ✓ ✓
 - initial value of counter is zero ✓ ✓ ✓
 - A process increments its counter when a **send** or an **instruction** happens at it. The counter is assigned to the event as its timestamp. ✓ ✓ ✓
 - A **send (message)** event carries its timestamp
 - For a **receive (message)** event the counter is updated by
 $\max[\text{local clock}, \text{message timestamp}] + 1$

Cloud Computing and Distributed Systems Time and Clock Synchronization

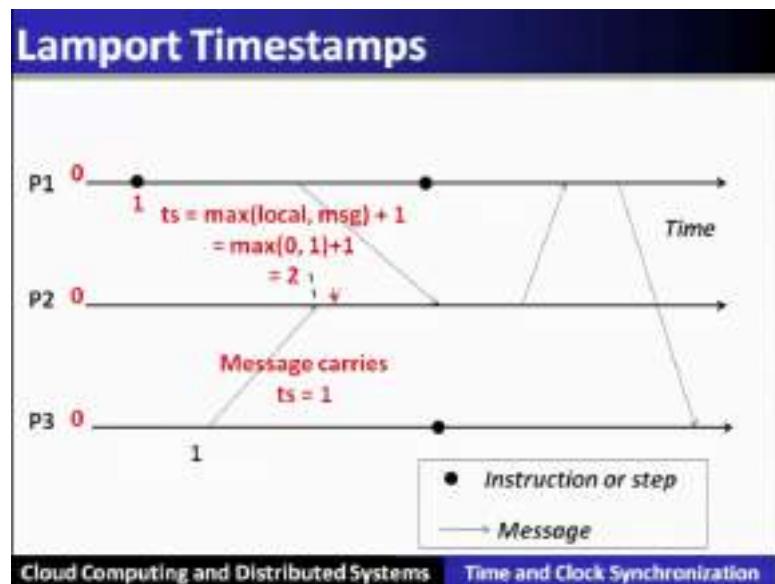
And hence this that particular lamport clock is able to capture the causal relation. Now, the implementation of these timestamps to each event; so that let us, assign a logical

timestamp without a physical clock which will obey the causality. Let us see the implementation using a logical clock which follows the 2 different rules the first rule says that each process uses a local counter which is called a logical clock, which is nothing but an integer which is in a slice 0. A process increments it is counter when it executes the send off a message or the internal execution of an event happens within that particular process.

The counter is assigned to the event as it is timestamp. So, the process increments its counter whether it is an internal event or it is a send off a message event. And this particular timestamp will be carried with the send off a message. So, this is the rule number one. The rule number 2 says that when a message is received, receive of a message event will update the counter in this form. First of all, the timestamp which is carried in the message and it is local clock, the maximum of these 2 values will be taken up and it will be incremented, that is rule number 2.

So, this will provide the Lamport's timestamp.

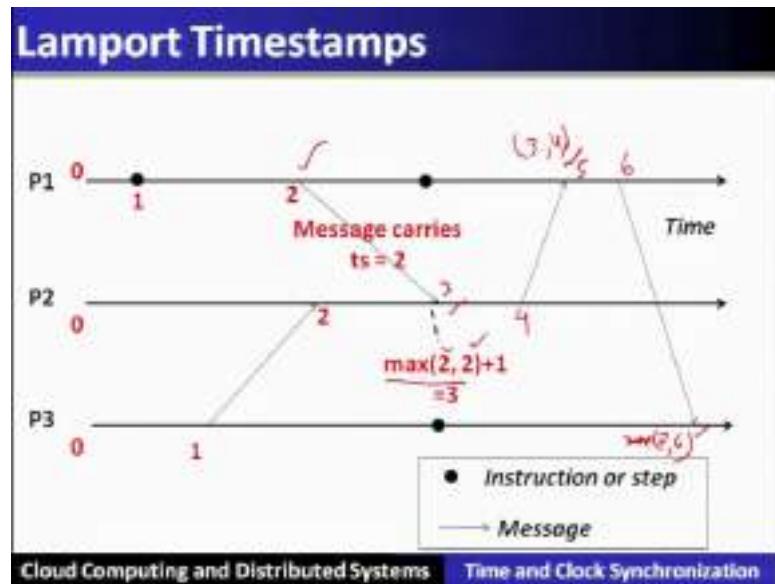
(Refer Slide Time: 67:09)



Let us see through an example, here all the counters are in a slice to 0 as we have seen in the implementation. Then process P1 will have its local clock will be in a slice to one or will be incremented from 0 to 1 because it is an internal event. Similarly, as far as process 3 is concerned, the send off a message hence its clock will be incremented from 0 to 1, and this will be contained in the message and will be sent.

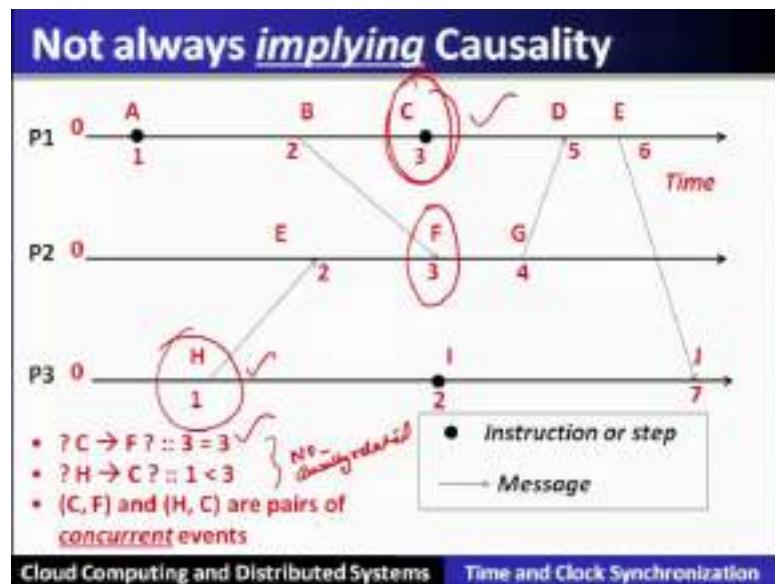
Similarly, when the message will be received, then we will follow the rule R 2 which says that the message which is carrying 1 and the local clock is 0 the maximum(0,1) will become 1. And when it will be incremented the timestamp here will be equal to 2.

(Refer Slide Time: 68:16)



Now, when this particular message will be received, here we can take the maximum of 2 and 2 that become 2 and plus 1 that become 3. This particular event 3 + 1 it will become 4. And 4 when it will reach over here, 4 and this is 3 the maximum is 4 + 1 that becomes 5 and 5 + 1 this will become 6. And when you will it will reach over here; so, the maximum of 6 and 1 2 2 that becomes 6 + 1; that is 7.

(Refer Slide Time: 69:05)



So, all these we have already explained. And they are obeying the causal properties for example, A has happened before B. So, the timestamp also is less in case of A. B has happened before F. So, the time is A B is less than F. A is happened before F. So, the timestamp A is less than F. Now we have seen the example of the events which are casually related they are following the Lamport's logical clock. But not all events are causally related; that means, not always implies the causality among all the events that is why it follows the partial order relation.

Let us see through the example C and F although they are timestamp is 3. So, do we infer that C has happened before F? Similarly, H and C H and C; so, the time of H is 1 and the time of C is 3. So, can we infer that H has happened before C. No, we cannot, why because they are not causally related. Hence they do not follow this happened before relation. Hence they are called as concurrent events. So, C F and H C they are the pairs which are not obeying the causality relation. Hence they are called concurrent events here in this case.

(Refer Slide Time: 70:55)

Concurrent Events

- A pair of concurrent events doesn't have a causal path from one event to another (either way, in the pair)
- Lamport timestamps not guaranteed to be ordered or unequal for concurrent events
- Ok, since concurrent events are not causality related!
- Remember:
 $E1 \rightarrow E2 \Rightarrow \text{timestamp}(E1) < \text{timestamp}(E2)$, BUT
 $\text{timestamp}(E1) < \text{timestamp}(E2) \Rightarrow \{E1 \rightarrow E2\} \text{ OR } \{E1 \text{ and } E2 \text{ concurrent}\}$

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, a pair of concurrent even does not have the causal path from one event to another. So, lamport timestamp not guaranteed to be ordered or unequal for concurrent events. This is since the concurrent events are not causally related. But what we can see here as I take away that, if E 1 has happened before E 2, this implies that the timestamp of E 1 is less than the timestamp of E 2. This is what is known here in our logical clock given by the lamport.

But if we see the timestamp, only and can be inform whether the events are happened before or they are concurrent. That we cannot just by looking a timestamp we cannot infer whether the events are happened before or; that means, causally related or they are not causally related they are concurrent event. That is the drawback of the lamport's clock.

(Refer Slide Time: 71:57)

Vector Timestamps

- Used in key-value stores like Riak
- Each process uses a vector of integer clocks
- Suppose there are N processes in the group 1..N
- Each vector has N elements
- Process i maintains vector $V_i[1..N]$
- jth element of vector clock at process i, $V_i[j]$, is its knowledge of latest events at process j

$V_i[j] = P_i's \text{ knowledge}$
about events at
process j

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, the next clock is called a vector timestamp or a vector clocks; which is used in the key value stores like Riak. Each process uses the vector of integer clocks. Suppose there are n processes in a group one to n and each vector has elements. So, process I will maintain the vector i for each process one to N. Now jth element of the vector at process i is denoted by; this indicates that the process P i's knowledge about the events at process j.

(Refer Slide Time: 73:09)

Assigning Vector Timestamps

Incrementing vector clocks

- On an instruction or send event at process i, it increments only its ith element of its vector clock
- Each message carries the send-event's vector timestamp $V_{\text{message}}[1..N]$
- On receiving a message at process i:
$$V_i[i] = V_i[i] + 1 \quad (\text{as per Lamport logic})$$

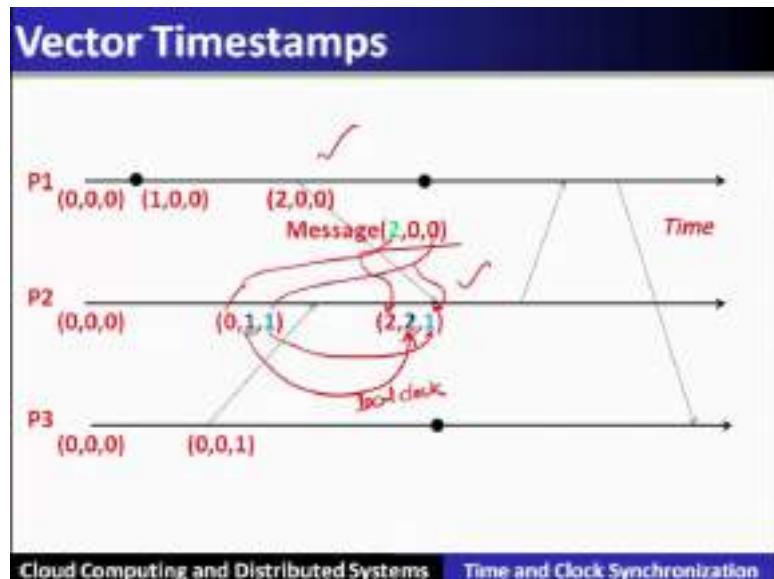
$$V_i[j] = \max(V_{\text{message}}[j], V_i[j]) \text{ for } j \neq i \quad (\text{update knowledge by process } j \text{ at } i)$$

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, this particular notation we are going to use further in this understanding of vector clocks. Assigning the vector timestamps; that is, on an instruction execution or send off an event at process i increments only i th element of it is vector clock. And this particular timestamp the entire vector will be timestamp. And so, the message carries the send events vector timestamp within it. When the message will be received at a process i , then process I will increment it is vector. As we have seen this is as per the as per the lamport's clock. And then it will take the maximum of the vectors of j and it is knowledge about other j excluding it is own vector; that means, the message the vector of the process j will be used to update it is own clock. So, this is a new step that is about update the knowledge of process j at i .

So, whenever there is a message exchange the clock values will be updated based on the information which is being sent the knowledge of other processes. So, let us see through an example. Here all the counters are in a slice to 0.

(Refer Slide Time: 75:17)



Now, with the event at P 1, only his clock value will be incremented, similarly at for P 3 only this value will be incremented. And whenever there is a message exchange, then these other values will also be updated.

For example, the receive of a message will increment it is local variable, but the message contains the information about the P 3's knowledge. So, this will be updated. So, here it will be updated for the other value. So, better timestamp will not only update it is

knowledge, but whenever a message exchange it will update it is vector based on the based on the information of the other processes. So, this is another example that if when P 2 will send this message with the timestamp 2,0,0. And it will be packed as a message when it will receive at P 2 then P 2 will increment, it is local clock as per the lamport's logical clock. Whereas, 0 and 2 they will update to the maximum of 2. So, 1 and 0 will be updated over here. So, this is not updated, same value will be because it has more knowledge than the pone about P 3.

Now, we will see that using vector clock we can perform some operations on the vectors. And this will be used to compare the vectors just by looking the vector we can infer whether the whether 2 events are causally related are there or not.

(Refer Slide Time: 77:14)

Causally-Related

- $VT_1 = VT_2$
iff (if and only if)
 $VT_1[i] = VT_2[i]$, for all $i = 1, \dots, N$
- $VT_1 \leq VT_2$
iff $VT_1[i] \leq VT_2[i]$, for all $i = 1, \dots, N$
- **Two events are causally related iff**
 $VT_1 < VT_2$, i.e.
 iff $VT_1 \leq VT_2$ &
 there exists j such that
 $1 \leq j \leq N \text{ & } VT_1[j] < VT_2[j]$

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, that means, if the vector $VT_1 < VT_2$. This will implies that there exist 2 vectors which are $VT_1 \leq VT_2$ and also there exist a particular value which is strictly less than, then basically this particular condition holds. And that shows that 2 events are causally related.

So, 2 events are concurrent if neither $VT_1 \leq VT_2$ and nor $VT_2 \leq VT_1$ hence they are causally related. Vector timestamp obeys the causality relation. This also has been identifying the concurrent events. Let us take an example. So, C and F so, vector C is 3 0 0 F is 2 1 1 and 2 1 1. Now this particular value is higher than this one where as the other elements they are less than this values. So, it is not less than; that

means, there is no neither it is neither C is vector of C neither it is less than or equal to vector of C. F nor vector of F is less than or equal to vector of C.

(Refer Slide Time: 78:01)

... or Not Causally-Related

- Two events VT_1 and VT_2 are **concurrent**
iff

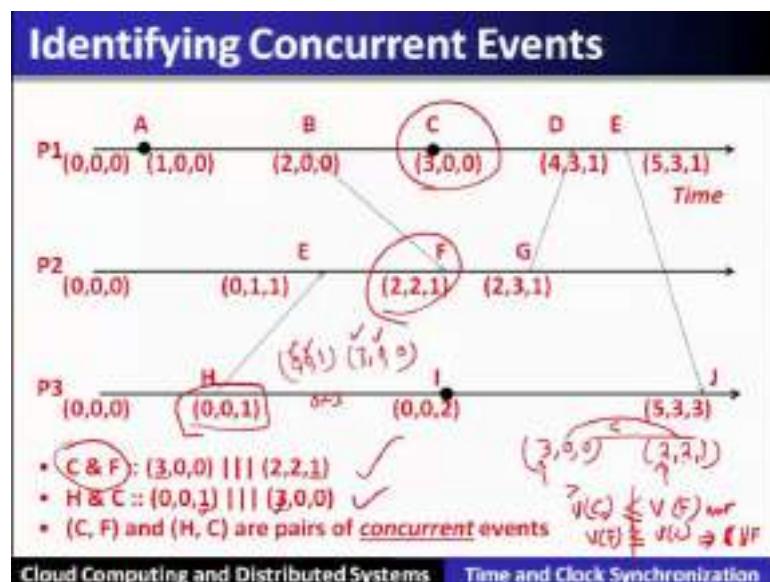
$$\text{NOT } (VT_1 \leq VT_2) \text{ AND NOT } (VT_2 \leq VT_1)$$

We'll denote this as $VT_2 \parallel\!\!\parallel VT_1$

Cloud Computing and Distributed Systems Time and Clock Synchronization

Hence they are C and F they are concurrent events. Similar H and C, you can see that this one is less than 3 H and C. So, 1 0 is less than 3 that is fine.

(Refer Slide Time: 78:14)



This also is fine, but one is greater than 0. Therefore, the same principle it is neither less than or equal to; so, that means it is not following obeying the causality relation. Hence they are concurrent events. Summary, we will see that lamport timestamp uses the

integer clocks to assign the event obeys the causality, but cannot distinguish the concurrent event.

(Refer Slide Time: 80:28)

Summary : Logical Timestamps

- **Lamport timestamp**
 - Integer clocks assigned to events.
 - Obeys causality
 - Cannot distinguish concurrent events,
- **Vector timestamps**
 - Obey causality
 - By using more space, can also identify concurrent events

Cloud Computing and Distributed Systems Time and Clock Synchronization

Vector timestamps obeys the causality by using more space and also can identify the concurrent event more space in the sense. Here the vector is required to be sent in the message is an additional overhead of using the space.

(Refer Slide Time: 80:52)

Conclusion

- Clocks are unsynchronized in an asynchronous distributed system
- But need to order events across processes!
- **Time synchronization:**
 - Christian's algorithm
 - Berkeley algorithm
 - NTP
 - DTP
 - But error a function of RTT
- **Can avoid time synchronization altogether by instead assigning logical timestamps to events**

Cloud Computing and Distributed Systems Time and Clock Synchronization

So, conclusion clocks are unsynchronized or the synchronization of clocks in a asynchronous distributed system using physical clock synchronization method which we

have seen has or having the inaccuracies. So, we have seen the methods for doing the clock synchronization in the distributed system. Let us say that they are Christian's algorithm Berkeley algorithm network time protocol DTP, but all these particular synchronization algorithms, they have the error which is the function of round trip time. So, we have seen also the method by avoiding the time synchronization and also ordering the events using the method such as Lamport's clock vectors clock.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 11
Global State and Snapshot

(Refer Slide Time: 00:17)

Preface

Content of this Lecture:

- In this lecture, we will discuss about the Global states (i.e. consistent, inconsistent), Models of communication and Snapshot algorithm i.e. Chandy-Lamport algorithm to record the global snapshot.

Cloud Computing and Distributed Systems Global State and Snapshot

Global State and Snapshot Recording Algorithms. Content of this lecture. We will discuss global states, models of communication snapshot algorithms. Especially, we will focus on Chandy-Lamport's algorithm for recording of a global state.

(Refer Slide Time: 00:33)

Snapshots

Here's Snapshot: Collect at a place



Distributed Snapshot
How do you calculate a "global snapshot" in this distributed system? What does a "global snapshot" even mean?



Cloud Computing and Distributed Systems Global State and Snapshot

Snapshots. So, we have seen here in this particular picture that all the heads of the nations, they have collected at one place and snapshot was taken up. Idea is that, everyone has to come and gather at one place. Whereas, we will see on the right side, the concept of a distributed snapshot, without coming at one place, how can you take the snapshot that is called a distributed snapshot.

So, snapshot in a distributed system that all, that scenario is also there in the cloud system, how this snapshot that is called a global distributed snapshot is taken up in such a scenario. So, this example is a geographically distributed cloud, which is nothing but a distributed system. And the distributed snapshot is a challenge here and this is the topic for our discussion. So, what does this global snapshot even mean all these things we will explore.

(Refer Slide Time: 01:48)

In the Cloud: Global Snapshot

- In a cloud each application or service is running on multiple servers
- Servers handling concurrent events and interacting with each other
- The ability to obtain a “global photograph” or “Global Snapshot” of the system is important
- Some uses of having a global picture of the system
 - **Checkpointing:** can restart distributed application on failure
 - **Garbage collection of objects:** objects at servers that don't have any other objects (at any servers) with pointers to them
 - **Deadlock detection:** Useful in database transaction systems
 - **Termination of computation:** Useful in batch computing systems

Cloud Computing and Distributed Systems Global State and Snapshot

So, in a cloud, each application or a service is running on multiple servers and the servers handling the concurrent event and interacting with each other. So, the ability to obtain the global snapshot of the entire system is important because of the following applications.

For example, sometimes check pointing requires a global snapshot or a global picture of the system. So, if a checkpoint is available, and if there is a failure, then system can restart without much lost that is called check pointing requires, the input of a global snapshot. Similarly, the garbage collection of the objects to find out about the objects, which are not even pointed or not being used by the other servers, they are to be removed off that also requires the global snapshot.

Deadlocks; in such a system is also useful that can only be done or that can only be analyzed, if the global snapshot of the entire system is available. Similarly, in a batch computing system, the termination of the computation is important and that can be analyzed from the input of a global snapshot.

(Refer Slide Time: 03:21)

Global State: Introduction

- Recording the global state of a distributed system on-the-fly is an important paradigm.
- The lack of globally shared memory, global clock and unpredictable message delays in a distributed system make this problem non-trivial.
- This lecture first defines consistent global states and discusses issues to be addressed to compute consistent distributed snapshots.
- Then the algorithm to determine on-the-fly such snapshots is presented.

Cloud Computing and Distributed Systems Global State and Snapshot

So, to understand, this let us see some of the preliminaries and some definitions, then only we will see the recording of a global snapshot algorithm. So, recording the global state of a distributed system on-the-fly, that means, without stopping the system, how the snapshot is to be taken is an important paradigm.

Now, the challenge over here is that the model of a distributed system that is the geographically distributed cloud system lacks a global shared memory, and also a global common clock and it also has the messages, which have the finite, but unpredictable delays making this particular problem of global snapshot a non-trivial problem. So, as I told you that we will first build up the basics, to understand the global snapshot problem. And then, we will see the algorithm of global snapshot or snapshot recording global state recording.

(Refer Slide Time: 04:38)

System Model

- The system consists of a collection of n processes p_1, p_2, \dots, p_n that are connected by channels.
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
- C_{ij} denotes the channel from process p_i to process p_j and its state is denoted by SC_{ij} .
- The actions performed by a process are modeled as three types of events: internal events, the message send event and the message receive event.
- For a message m_{ij} that is sent by process p_i to process p_j , let $send(m_{ij})$ and $rec(m_{ij})$ denote its send and receive events.

Cloud Computing and Distributed Systems

Global State and Snapshot

System model. So, the system consist of the collection of n processes p_1 to p_n , we assume that they are all connected by the channels. We also assume that there is no globally shared memory; we also assume that there is no global common physical clock. Therefore, the processes communicate only by the means of messages using communication channels.

Let C_{ij} denotes the channel from a process i to j ; and that is denoted by state of a channel SC_{ij} . The actions performed by the by the process are modeled as three different types; internal events, message send event, and message received events. For message m_{ij} that is sent by a process i to process j , let $send(m_{ij})$ be the send of an event; similarly, $rec(m_{ij})$ denotes the receive of an event.

(Refer Slide Time: 05:43)

System Model

- At any instant, the state of process p_i , denoted by LS_i , is a result of the sequence of all the events executed by p_i till that instant.
- For an event e and a process state LS_j , $e \in LS_j$ iff e belongs to the sequence of events that have taken process p_j to state LS_j .
- For an event e and a process state LS_j , $e \notin LS_j$ iff e does not belong to the sequence of events that have taken process p_j to state LS_j .
- For a channel C_{ij} , the following set of messages can be defined based on the local states of the processes p_i and p_j

Transit: $transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \wedge rec(m_{ij}) \notin LS_j\}$

Cloud Computing and Distributed Systems Global State and Snapshot

At any instant, the state of a process p_i is denoted by LS_i , is a result of the sequence of all the events executed by p_i , till that instant. So, for an event e and a process state LS_i , e is an LS_i , if e belongs to the sequence of events that have taken in the process p_i to the state of LS_i . For event e and a process state LS_i , is not in LS_i , if e does not belong to the sequence of event, which has taken place in a process to their state. Similarly, for a channel C_{ij} , the following set of messages can be defined based on the local state of a process.

So, the state of the channel is called transit. If given two different states of two different processes i and j , there exist a message m_{ij} . Such that the send of that particular message is part of the state of a channel i . And the received of m_{ij} is not recorded here in LS_j that means, the messages message is in the channel or that is called in the transit.

(Refer Slide Time: 07:18)

Consistent Global State

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Notationally, global state GS is defined as,
$$GS = \{U_i LS_i, U_{ij} SC_{ij}\}$$
- A global state GS is a consistent global state iff it satisfies the following two conditions:
 - C1: $\text{send}(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus \text{rec}(m_{ij}) \in LS_i$
(\oplus is Ex-OR operator)
 - C2: $\text{send}(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \wedge \text{rec}(m_{ij}) \in LS_j$

Cloud Computing and Distributed Systems Global State and Snapshot

Consistent global state definition. So, a global state of a distributed system is a collection of the local state of the processes and the corresponding channel. Notationally, global state GS is basically the collection of the local state of all the processes that is the union of LS_i 's; and the union of state of channel SC_i 's.

So, the global state the GS, which we have just defined is a consistent global state, if it only satisfies two conditions; C1, and C2. C1 says that if the send of a particular message m_{ij} is in the state of a channel state of a local channel i, this implies that the message is in state of a channel or it is received by a process. So, this is an exclusive or operation. Similarly, if the message is not in the any of the local states, this will imply that neither it is in the state of a channel, nor it is received at any of the local states that is then only it is called a consistent state.

(Refer Slide Time: 08:49)

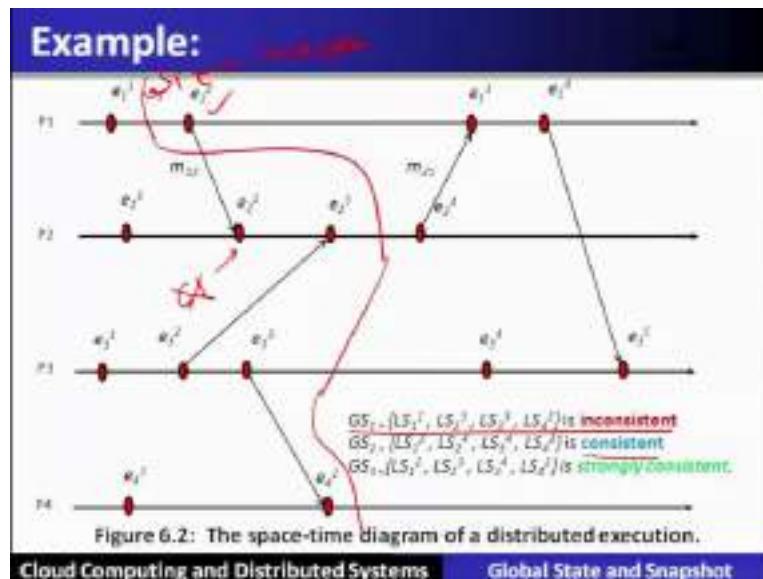
Global State of a Distributed System

- In the distributed execution of Figure 6.2:
- A global state GS_1 consisting of local states $\{LS_1^1, LS_2^1, LS_3^1, LS_4^1\}$ is **inconsistent** because the state of p_1 has recorded the receipt of message m_{12} , however, the state of p_1 has not recorded its send.
- On the contrary, a global state GS_2 consisting of local states $\{LS_1^2, LS_2^2, LS_3^2, LS_4^2\}$ is **consistent**; all the channels are empty except c_{21} that contains message m_{21} .

Cloud Computing and Distributed Systems Global State and Snapshot

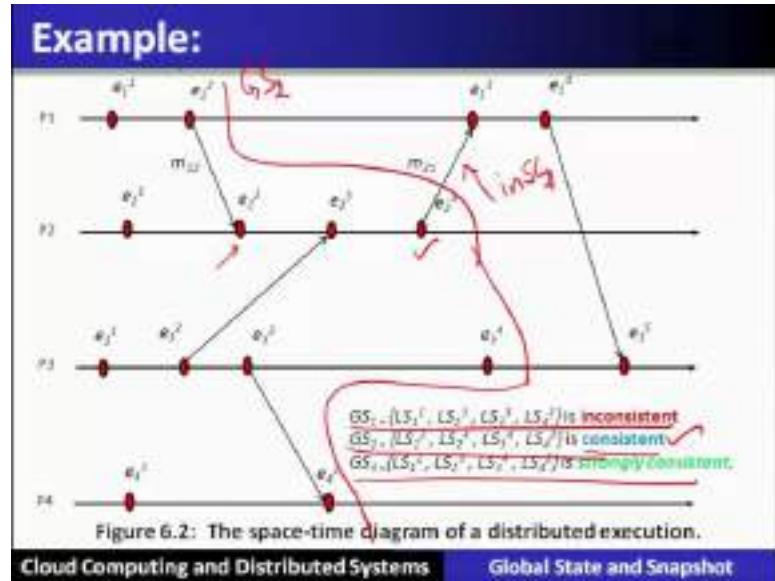
We will see the example of a local state, which is inconsistent that means, it does not follow the property C1 and C2. Let us see here in this particular example.

(Refer Slide Time: 09:02)



So, global state 1 is inconsistent. Let us figure out in this particular diagram, where is local state 1, that is 1 that is p_1 up to 1; here this is line. Then for p_2 up to 3, this is the line; then 3 and 3, this is the line and 4 and 2, this is the line. So, this is a global state 1 and this is inconsistent. Why, because of this particular message is recorded as I received, but it does not record as the send, so that means, C1 is violated.

(Refer Slide Time: 10:16)



Now, we will see another one, which is called consistent that is LS, LS_1^2 , this; then LS_2^4 , this; then LS_3^4 this, and this one. This is the consistent global state. You can see here, the send of message is recorded, but the receive is not recorded that is fine, but send is recorded. So, this is called consistent global state. So, this message must be in the in the state of a channel that is 2 1.

Similarly, there is another global state, which is called GC GS 3; and this is called strongly consistent. Let us trace 3 2, and then 3 4, and then this is the 2. So, you just see that all the messages, which are sent is being received. So, there is no message, which is in the transit. Hence, it is called strongly consistent global state.

(Refer Slide Time: 11:15)

Issues in Recording a Global State

- The following two issues need to be addressed:

I1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

- Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot [from C1].
- Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot [from C2].

I2: How to determine the instant when a process takes its snapshot.

- A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

Cloud Computing and Distributed Systems

Global State and Snapshot

So, let us see the definitions, we have already understand through the diagram and examples. So, global state is transitless that means, there is no message in the channels that means, all the channels are empty that means, whatever message is sent, they are being received. If that is the condition, then that state is called strongly consistent state.

Now, let us see the issues in recording of a global state in a distributed systems. So, the first issue I1 says that how to distinguish between the messages to be recorded in a snapshot from those, which or not to be recorded. So, that means, any message that is sent by a process before recording its snapshot, must be recorded in a global snapshot that is it has to follow the condition C1. Any message that is send by a process after recording its snapshot must not be recorded that is the condition C2. So, the issue 1 is to differentiate between the messages, which are to be recorded; and the messages, which are not to be recorded that is the issue number 1.

Issue number 2 says that how to determine the instant, when the process takes the snapshot. A process p_j must record a snapshot before processing a message m_{ij} that was sent by a process i after recording its snapshot. So, these two issues are important. And let us see how it is going to be taken care, while we discuss the recording of a snapshot.

(Refer Slide Time: 13:10)

Example of Money Transfer

- Let S_1 and S_2 be two distinct sites of a distributed system which maintain bank accounts A and B, respectively. A site refers to a process in this example. Let the communication channels from site S_1 to site S_2 and from site S_2 to site S_1 be denoted by C_{12} and C_{21} , respectively. Consider the following sequence of actions, which are also illustrated in the timing diagram of Figure 6.3:
 - Time t_0 : Initially, Account A=\$600, Account B=\$200, $C_{12}=50$, $C_{21}=0$.
 - Time t_1 : Site S_1 initiates a transfer of \$50 from Account A to Account B.
 - Account A is decremented by \$50 to \$550 and a request for \$50 credit to Account B is sent on Channel C_{12} to site S_2 . Account A=\$550, Account B=\$200, $C_{12}=$50, $C_{21}=0$.$

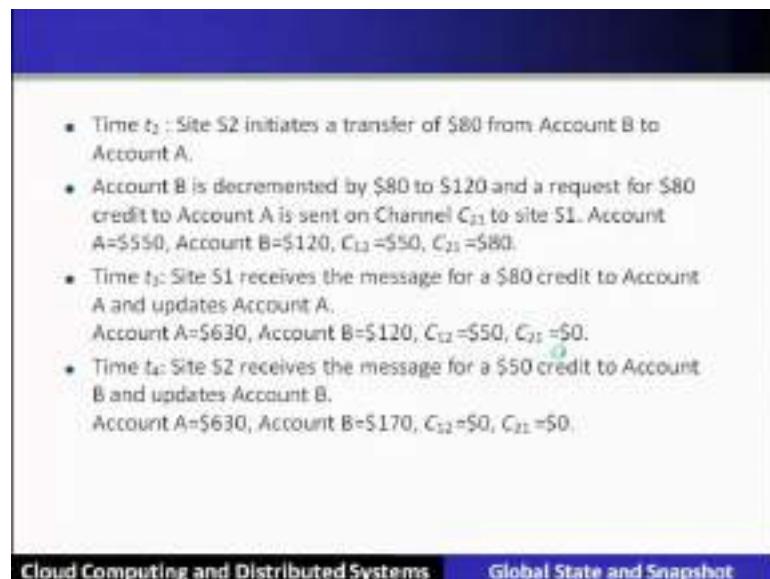
Cloud Computing and Distributed Systems

Global State and Snapshot

There is a example of a money transfer that we will take up, when we will discuss the algorithm. Let us understand this example of a money transfer. Let there are two sites; S_1 , and S_2 in a distributed system, and which maintains the bank account A and B, respectively that is site S_1 maintains bank account A, and bank account B is maintained at site S_2 . Let the communication channel from site S_1 to S_2 and from site S_2 to S_1 be denoted by C_{12} and C_{21} , respectively.

Now consider the following sequence of actions, which are illustrated. At time t_0 , the account of A is 600 dollars, account B is 200 dollars and the channels are empty. At time t_1 , site S_1 will initiate a transfer of 50 dollars from account A to account B. So, accounting A is decremented by 50 dollars and a request to credit 50 dollars is being sent the message as a message to the account B over a communication channel C_{12} . So, the account A becomes 550, whereas the account B is not changed. And the request to credit that 50 dollar is in the message, which is in the communication channel C_{12} , C_{21} is 0.

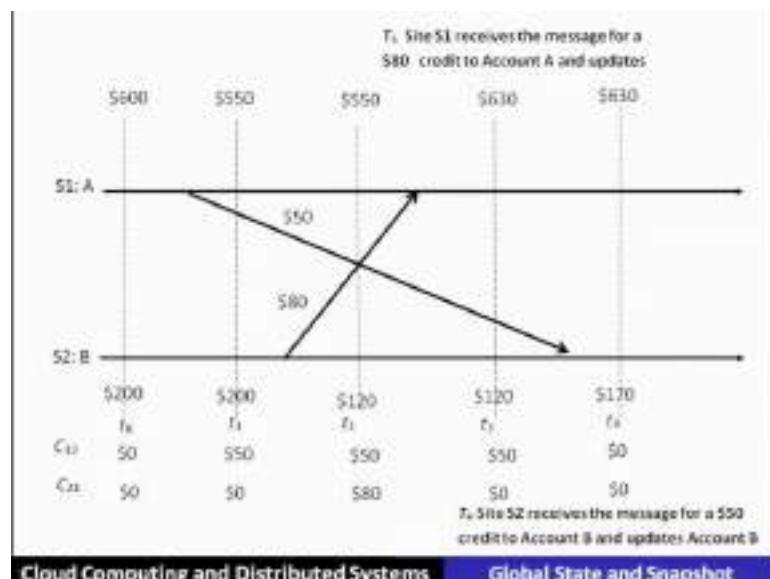
(Refer Slide Time: 15:01)



Cloud Computing and Distributed Systems Global State and Snapshot

And so on, we will see this example here in this particular way.

(Refer Slide Time: 15:05)



Cloud Computing and Distributed Systems Global State and Snapshot

So, here as I told you that 50 dollar is being sent, and it will be debited, similarly here in at time t_1 . Before time t_2 , 80 dollar is being sent from B to A and that corresponding things are being reflected in this state time diagram.

(Refer Slide Time: 15:35)

Cloud Computing and Distributed Systems Global State and Snapshot

- Suppose the local state of Account A is recorded at time t_0 to show \$600 and the local state of Account B and channels C_{12} and C_{21} are recorded at time t_1 to show \$120, \$50, and \$80, respectively. Then the recorded global state shows \$850 in the system. An extra \$50 appears in the system.
- The reason for the inconsistency is that Account A's state was recorded before the \$50 transfer to Account B using channel C_{12} was initiated, whereas channel C_{12} 's state was recorded after the \$50 transfer was initiated.
- This simple example shows that recording a consistent global state of a distributed system is not a trivial task. Recording activities of individual components must be coordinated appropriately.

Now, here in this case, if we record something like that we are recording 600 dollar, and then 120 and we are taking this. At time t_0 , we are taking the snapshot of A; and at time t_2 , we are taking the values of time t_2 . Then this kind of recording will have inconsistent information. Why, because the total value should be 800 at any point in the recording recorded global state. So, if you see if you count this, it will be 750 dollars, and so on. So, this is not or this is a inconsistent snapshot, if we take.

(Refer Slide Time: 16:30)

Model of Communication

Cloud Computing and Distributed Systems Global State and Snapshot

- Recall, there are three models of communication: FIFO, non-FIFO, and Co.
- In **FIFO model**, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.
- In **non-FIFO model**, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.
- A system that supports **causal delivery** of messages satisfies the following property: "For any two messages m_{ij} and m_{kj} ,
if $\text{send}(m_{ij}) \rightarrow \text{send}(m_{kj})$, then $\text{rec}(m_{ij}) \rightarrow \text{rec}(m_{kj})$ "

Now, let us assume some of the models of communication. We will consider three different model of communication in this model, for the global snapshot algorithm FIFO non-FIFO, and the causal order.

(Refer Slide Time: 16:50)

Snapshot algorithm for FIFO channels

Chandy-Lamport algorithm:

- The Chandy-Lamport algorithm uses a **control message**, called a **marker** whose role in a **FIFO system** is to separate messages in the channels.
- After a site has recorded its snapshot, it sends a **marker**, along all of its outgoing channels before sending out any more messages.
- A marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.

Now, let us consider the Chandy-Lamport algorithm for the FIFO channels. So, Chandy-Lamport algorithm uses the control messages, which are called marker messages, and whose role in FIFO is to separate the messages, which are to be recorded from the messages, which are not to be recorded.

So, the marker is basically dividing the messages, which are to be recorded, which are not to be recorded. So, when a marker comes, all the message before the marker arrives are to be recorded; and after marker arrival, whatever messages are coming are not going to be recorded. So, marker is basically a separating point. So, marker will be used to decide what are things to be recorded, what are the messages which are not to be recorded.

(Refer Slide Time: 17:46)

Chandy-Lamport Algorithm

- The algorithm can be initiated by any process by executing the “**Marker Sending Rule**” by which it records its local state and sends a marker on each outgoing channel.
- A process executes the “**Marker Receiving Rule**” on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the “Marker Sending Rule” to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Cloud Computing and Distributed Systems

Global State and Snapshot

So, the Chandy-Lamport algorithm can be initiated by any process by executing the marker sending rule by which it records its local state and it will send the marker on each outgoing channels. So, a process executes the marker receiving rule on receiving the marker. If the process has not yet recorded its local state, will record the state of a channel on which the marker is received as empty and executes the marker sending rule to record its local state. Therefore, this algorithm is a recursive algorithm. Now, this algorithm will terminate after each process has received a marker on all of its incoming channels.

(Refer Slide Time: 18:28)

Chandy-Lamport Algorithm

Marker Sending Rule for process i :

- 1) Process i records its state.
- 2) For each outgoing channel C on which a marker has not been sent, i sends a marker along C before i sends further messages along C .

Marker Receiving Rule for process j

On receiving a marker along channel C :

if j has not recorded its state **then**

 Record the state of C as the empty set

 Follow the “Marker Sending Rule”

else

 Record the state of C as the set of messages
 received along C after j ’s state was recorded
 and before j received the marker along C

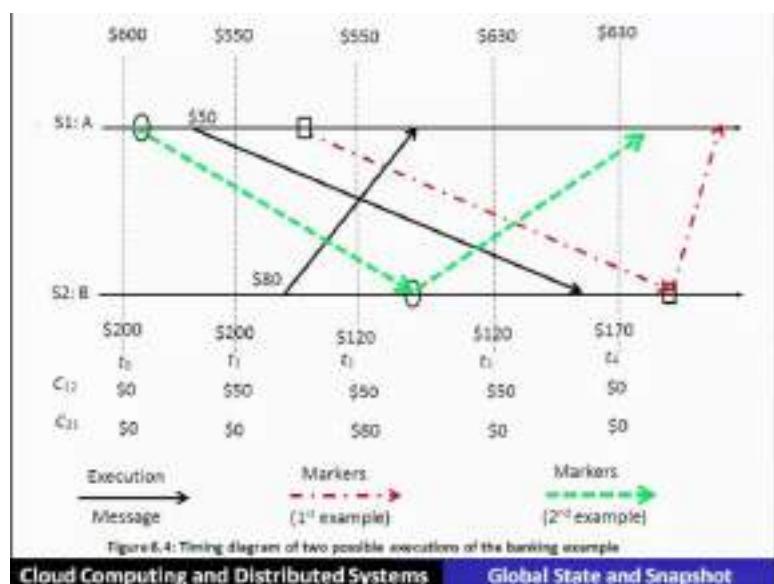
Cloud Computing and Distributed Systems

Global State and Snapshot

Let us see the algorithm in more details. This algorithm is called a Chandy-Lamport algorithm. So, marker sending rule for a process i has two steps; the first step says that process i will record its state, second rule says that for each outgoing channel C on which the marker has not been sent, i sends a marker along C before i sends the further messages along C .

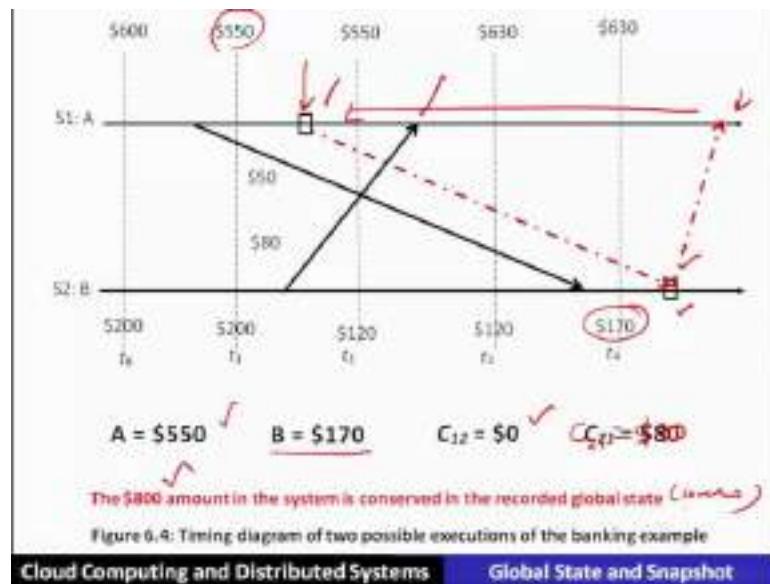
The marker receiving rule for a process j or receiving the marker along channel C , if j has not recorded its state, then record the state of a channel is an empty, and follow the marker sending rule that means, marker sending rule means again it will follow this; it is a recursive call. So, follow the marker sending rule that means, then it will record its state and send the marker to another level of outgoing channels, if it is not. Now, if j has been recorded, then it will record the state of a state of channel C as the set of messages received along C after j 's state was recorded, and before j received the marker along C .

(Refer Slide Time: 19:52)



So, we will see this particular working of Chandy-Lamport's algorithm through an example. The same set of the example, but there are two different initiation, which is shown here; this is one set of initiation, this is another set of initiation.

(Refer Slide Time: 20:11)



So, when this particular algorithm is initiated at this point of time, so as per the algorithm is concerned, it will record its state that is A will be recorded as 550, and it will send the marker on this channel. And when the marker is received, then what it will do, it will record its state as empty. And then, marker sending rule it will apply out of that, it will record its state as 170 and send the marker further when the marker is received at this point, then since it has already recorded its state. So, what it will do, it will check all the messages, since last time it has recorded and up to the receive the marker any messages, it has received in the communication channel. Yes, this message is received.

So, the state of a channel C_{21} that will be denoted as dollar 80, so that will be the recorded snapshot. So, if you sum up all the values, it basically will give you that 800 that is the total amount is preserved, hence it is a consistent global state recording algorithm, which is recording the state, which is global state, which is a consistent state.

(Refer Slide Time: 21:51)

Properties of the recorded global state

2. (Markers shown using green dotted arrows.)

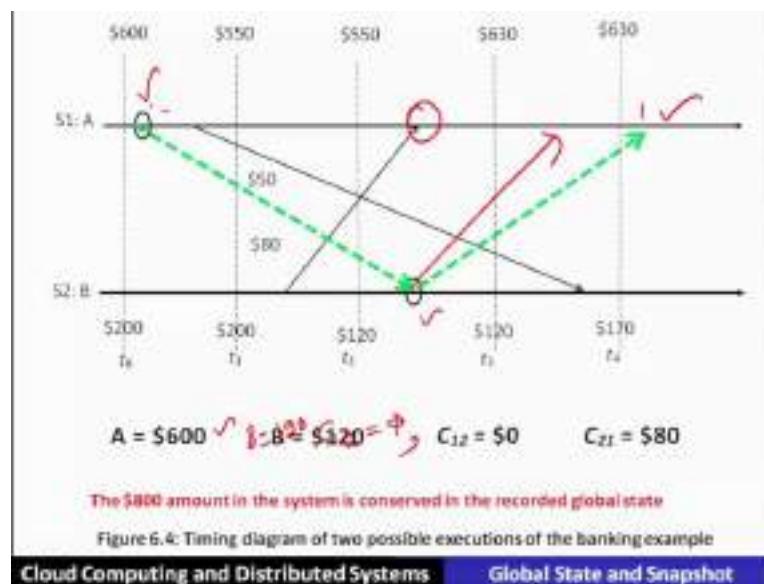
Let site S1 initiate the algorithm just after t_0 and before sending the \$50 for S2. Site S1 records its local state (account A = \$600) and sends a marker to site S2. The marker is received by site S2 between t_1 and t_2 . When site S2 receives the marker, it records its local state (account B = \$120), the state of channel C_{12} as 50, and sends a marker along channel C_{21} . When site S1 receives this marker, it records the state of channel C_{21} as 80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$600, B = \$120, C_{12} = 50, C_{21} = \$80$$

Q

Cloud Computing and Distributed Systems Global State and Snapshot

(Refer Slide Time: 21:53)



Now, let us see another example. In this example, the algorithm is initiated at this instant of time that is after just after t_0 . Now, as far as the marker, the rule one says that it will record its state that is A will be 600, and then it will send the marker. So, when the marker is received by a site S2, it will record its; it will record its state of a channel C_{12} as empty. And then, it will apply the marker sending rule, and it will then it will record its state that is B will be equal to 120, and it will send the channel; and it will send the marker on the further channels, like this.

Now, when the marker is received, since this state is already recorded, since the last time up to this marker, any message which is being arrived, that will be in the state of a channel. So, state of a channel will be recorded as the 80.

(Refer Slide Time: 23:11)

Properties of the recorded global state

- In both these possible runs of the algorithm, the recorded global states never occurred in the execution.
- This happens because a process can change its state asynchronously before the markers it sent are received by other sites and the other sites record their states.
- But the system could have passed through the recorded global states in some equivalent executions.
- The **recorded global state** is a valid state in an equivalent execution and if a stable property (i.e., a property that persists) holds in the system before the snapshot algorithm begins, it holds in the recorded global snapshot.
- Therefore, a recorded global state is useful in detecting stable properties.

So, this completes the illustration of this particular working of this algorithm. So, we have seen that it preserves all the properties, which we have defined for the consistent global snapshot in this particular algorithm.

(Refer Slide Time: 23:30)

Conclusion

- **Recording global state** of a distributed system is an important paradigm in the design of the distributed systems and the design of efficient methods of recording the global state is an important issue.
- This lecture first discussed a formal definition of the **global state of a distributed system and issues** related to its capture; then we have discussed the **Chandy-Lamport Algorithm** to record a snapshot of a distributed system.

Conclusion. Recording the global state of a distributed system is an important paradigm in the design of a distributed system and the cloud system. And the design of efficient method for recording global state is also an important issue. In this lecture, we have formally defined the global snapshot the state of a global state, and snapshot recording algorithm using Chandy-Lamport's algorithm.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Mishra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture-12
Distributed Mutual Exclusion

(Refer Slide Time: 00:17)

Preface

Content of this Lecture:

- In this lecture, we will discuss about the 'Concepts of Mutual Exclusion', Classical algorithms for distributed computing systems and industry systems for Mutual Exclusion.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Distributed Mutual Exclusion; Content of this lecture; in this lecture, we will discuss concepts of mutual exclusion used in the Cloud Computing Systems and also in the classical Distributed Systems and also we will see the industry systems which are using a different notion of mutual exclusion, the need of mutual exclusion in the cloud.

(Refer Slide Time: 00:40)

Need of Mutual Exclusion in Cloud

- **Bank's Servers in the Cloud:** Two customers make simultaneous deposits of 10,000 Rs. into your bank account, each from a separate ATM.
 - Both ATMs read initial amount of 1000 Rs. concurrently from the bank's cloud server
 - Both ATMs add 10,000 Rs. to this amount (locally at the ATM)
 - Both write the final amount to the server
 - What's wrong? 11000Rs. (or 21000Rs.)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see through an example that is the banks server in the cloud, let us consider 2 customers who makes simultaneous deposits of 10,000 rupees into your bank account each from the separate ATMs.

Both ATMs read initial values of amount, let us say 1000 concurrently from the banks cloud server. So, both the customers will now read the value 1000, second step; now both the ATM will add 10,000 to this amount locally at this ATM and then they both write the final amount to the server. Now what will be the final value which will be shown in your account? So, is it 11,000 or it is 21,000. So, what is wrong in this particular method?

(Refer Slide Time: 01:38)

Need of Mutual Exclusion in Cloud

- **Bank's Servers in the Cloud:** Two customers make simultaneous deposits of 10,000 Rs. into your bank account, each from a separate ATM.
 - Both ATMs read initial amount of 1000 Rs. concurrently from the bank's cloud server
 - Both ATMs add 10,000 Rs. to this amount (locally at the ATM)
 - Both write the final amount to the server
 - You lost 10,000 Rs.!
- **The ATMs need *mutually exclusive* access to your account entry at the server**
 - or, mutually exclusive access to executing the code that modifies the account entry

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Now, here we can say that if both write the final amount to the server, then one of them will be losing 10,000.

(Refer Slide Time: 01:52)

Need of Mutual Exclusion in Cloud

- **Bank's Servers in the Cloud:** Two customers make simultaneous deposits of 10,000 Rs. into your bank account, each from a separate ATM.
 - Both ATMs read initial amount of 1000 Rs. concurrently from the bank's cloud server
 - Both ATMs add 10,000 Rs. to this amount (locally at the ATM)
 - Both write the final amount to the server
 - What's wrong? 11000Rs. (or 21000Rs.)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

And this particular amount will be written as 11,000; that means, both will overwrite and the same amount will be shown and a loss of 10,000 will be let us see the illustration of this banks servers example. Now let us see that here this is the bank server and this is there in the cloud. Now this is an ATM 1 and this is an ATM number 2, it has initially 1000 in its account.

Now, both of them reads this value and they have the value balance as 1000, both will see the balance as 1000, they want to add or a deposit 10,000 to it, total becomes 11,000. Similarly, here it also want to deposit 10,000. So, total becomes 11,000, both they write these values to the server at concurrently, this is to be done concurrently what will happen over here both will be writing at the same point of time where the value will be 11,000 or 21,000 that is not known.

So, whether this is correct or this is correct. So, if the final value is comes out to be 11,000, then 10,000 will be lost. So, this condition is called a race condition and this can be solved with the help of accessing this particular section which is called a critical section which is nothing, but a code which allows the access to this particular value of your bank account that is the problem in this mode of operation. So, what is the issue what is the problem. So, the ATM need mutual exclusive access to your bank account entry which is located at the cloud so; that means, one at a time. So, one at a time means one customer will read your bank account then add 10,000 to it and then write back.

And once it is written back, then another; the next customer will read it and then again it will add some value and then write it back. So, in this manner, the final amount will be shown as 21,000, but that is possible if mutual exclusion that is the sequential access to your account is being provided from the cloud hence mutually exclusive access to executing the code that modifies or that accesses the bank account entry is going to be a crucial one.

(Refer Slide Time: 06:16)

Some other Mutual Exclusion use

- **Distributed File systems**
 - Locking of files and directories
- **Accessing objects** in a safe and consistent way
 - Ensure at most one server has access to object at any point of time
- **Server coordination**
 - Work partitioned across servers
 - Servers coordinate using locks
- **In industry**
 - Chubby is Google's locking service
 - Many cloud stacks use Apache Zookeeper for coordination among servers

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

And here comes the role of a mutual exclusion, there are other mutual exclusion example such as distributed file system here, before for supporting the concurrent access on a file and directories they have to be locked.

So, again the concept of mutual exclusion will be implemented here in this scenario of a distributed file system; similarly accessing the object in a safe and consistent manner. So, that for a concurrent access at most one server has access to the object at any point of time is to be ensured through the mutual exclusion similarly for the server coordination. So, if several servers are there, they have to be locked before some operations are to be performed or some work is to be done and that is done through the locks and that is also a way of ensuring the mutual exclusion.

Now, in the industry the system such as chubby is a Google's locking system that is implemented on the cloud, similarly many cloud stacks such as uses the notion of apache zookeeper for coordination among the servers and this also ensures the mutual exclusion implementation from the industry perspective.

(Refer Slide Time: 07:43)

Problem Statement for Mutual Exclusion

- **Critical Section** Problem: Piece of code (at all processes) for which we need to ensure there is at most one process executing it at any point of time.
- Each process can call three functions
 - `enter()` to enter the critical section (CS)
 - `AccessResource()` to run the critical section code
 - `exit()` to exit the critical section

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

All these things we are going to see in this further section of this lecture. So, for the mutual exclusion, we have a piece of code which is going to be very crucial to ensure that it has to be accessed at most by one process at a time and this becomes a critical section problem.

To solve this particular problem or to pose this problem there are 3 functions defined for a critical section problem or a enter to enter the critical section then within it, you can access the resources of a critical section using routine called access resource and finally, when the critical section use is over then there is a routine which will exit the critical section.

(Refer Slide Time: 08:37)

Bank Example

```
ATM1:           ATM2:  
enter(S),      enter(S),  
// AccessResource()    // AccessResource()  
obtain bank amount; obtain bank amount;  
add in deposit; add in deposit;  
update bank amount; update bank amount;  
// AccessResource() end // AccessResource() end  
exit(S); // exit      exit(S); // exit
```

Cloud Computing and Distributed Systems | Distributed Mutual Exclusion

Let us see in the bank example, now two ATMs which basically are accessing your bank account first has to run this enter with S; S is some function of which will ensure the mutual exclusion, then it will ensure the access of your bank account using access resource.

And finally, when the access of resource is finished, then it will end and exit that particular function call again both this ATMs will do this way. So, this access entry and access resource end exit, they will ensures the mutual exclusion one at a time.

(Refer Slide Time: 09:22)

Approaches to Solve Mutual Exclusion

- **Single OS:**
 - If all processes are running in one OS on a machine (or VM), then
 - Semaphores, mutexes, condition variables, monitors, etc.

Cloud Computing and Distributed Systems | Distributed Mutual Exclusion

This particular way of ensuring the critical section or a mutual exclusion through the critical section is normally done in a single operating system with the help of constructs like semaphore, mutexes, condition variables, monitors, etcetera.

(Refer Slide Time: 09:41)

Approaches to Solve Mutual Exclusion (2)

- Distributed system:
 - Processes communicating by passing messages

Need to guarantee 3 properties:

- **Safety** (essential): At most one process executes in CS (Critical Section) at any time
- **Liveness** (essential): Every request for a CS is granted eventually
- **Fairness** (desirable): Requests are granted in the order they were made

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

However in a distributed system such notions are not possible why because the distributed systems are primarily the message passing systems and so as the cloud systems.

So, but let us see in such systems, how we are going to ensure the mutual exclusion. So, such mutual exclusion requires to guarantee 3 different properties they are safety property this is very essential in the sense at most one process executes in the critical section at any time. Second property is called liveness which is also essential, this says that every request for a critical section is granted eventually and third one is called fairness that is also a desirable property which says that the requests are granted in the order in which the requests are made to the system.

(Refer Slide Time: 10:46)

Processes Sharing an OS: Semaphores

- Semaphore == an integer that can only be accessed via two special functions
- Semaphore S=1; // Max number of allowed accessors

1. **wait(S) (or P(S) or down(S)):**

```
while(1){ // each execution of the while loop is atomic
    enter()
    if(S > 0){
        S--;
        break;
    }
}
```

Each while loop execution and S++ are each **atomic** operations – supported via hardware instructions such as compare-and-swap, test-and-set, etc.

2. **signal(S) (or V(S) or up(s)):**

```
exit(); // exit
```

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Now, when there is a single operating system and the processes sharing the operating system using the notion of a semaphore. So, semaphore is a shared variable that in a single operating system scenario is possible. So, semaphore is a integer that can only be accessed via two special functions which are called wait and signal sometimes, they are also called as a P and V. Let us see that only one process is allowed to enter to access the critical section. So, the value of S is equal to 1. So, whenever there is a entry part of the code then wait or a P of S is executed that is nothing, but a while loop, but it is an atomic function which will reduce the value of S.

And this is then going to be atomic operation that is once it is started it has to finish in whole it cannot be interrupted in between, similarly, there is a signal or a V function that is when this is to be executed when the exit part of the code is executed so that the other process can enter into the critical section.

(Refer Slide Time: 12:07)

Bank Example Using Semaphores

```
Semaphore S=1; // shared
ATM1:
    wait(S);
    // AccessResource()
    obtain bank amount;
    add in deposit;
    update bank amount;
    // AccessResource() end
    signal(S); // exit
}
}
}

Semaphore S=1; // shared
ATM2:
    wait(S);
    // AccessResource()
    obtain bank amount;
    add in deposit;
    update bank amount;
    // AccessResource() end
    signal(S); // exit
}
}
}
```

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Using this semaphore, you can see the bank example when semaphore value variable as is initialized to 1. So, whenever there is a wait is executed, then it will basically decrement the value of S by 1 and enter into the critical section.

On the ATM 2, if at that point of time, it want to enter or it want to access the critical section then it will not be allowed why because the value of S is not 1 as per the conditions similarly, these particular signal will set the variable the value of S again to 1; so that the other process can execute into a critical section.

(Refer Slide Time: 13:01)

Next

- In a distributed system, cannot share variables like semaphores
- So how do we support mutual exclusion in a distributed system?

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So far so good that in a single operating system such variables are supported by the hardware, but in distributed systems shared variables are not a pass ability, why because the only way to communicate is via the message passing, there is no shared memory concept in the distributed system.

So, we will see how mutual exclusion is to be supported in a distributed system and that too in a cloud model.

(Refer Slide Time: 13:26)

System Model

- Before solving any problem, specify its System Model:
 - Each pair of processes is connected by reliable channels (such as TCP).
 - Messages are eventually delivered to recipient, and in FIFO (First In First Out) order.
 - Processes do not fail,
 - Fault-tolerant variants exist in literature.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, let us assume first the system model and let us see this particular way of implementing the mutual exclusion in this scenario that is the distributed systems. Now we assume that each pair of processes are connected by a reliable communication channel and the messages which are communicating they eventually are delivered to the recipients and also, they follow the principle, further we also assume that the processes do not fail.

So, fault tolerant variants also exist, but for the sake of simplicity, we will understand the concepts by assuming that the processes do not fail initially.

(Refer Slide Time: 14:12)

Central Solution

- o Elect a central master (or leader)
 - o Use one of our election algorithms!
- o Master keeps
 - o A **queue** of waiting requests from processes who wish to access the CS
 - o A special **token** which allows its holder to access CS
- o Actions of any process in group:
 - o **enter()**
 - ① Send a request to master ✓
 - ② Wait for token from master
 - o **exit()**
 - Send back token to master

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see first the central solution and then we will go on a distributed model of this particular solution, let us assume there is a single master or a leader which can be elected by any leader election algorithm and that master will keep a queue of all waiting requests from the concurrent accesses which the processes are making to access the critical section now there is ex there exist a special token which allows its holder to access the critical section. So, the actions of any process in a group are primarily, it will execute a enter by enter, it means that it will send a request to the master and then it waits for its response.

From the master, once it gets the token from the master token allows that particular process whosever is having the token can enter into a critical section and once it is use is over, then it will execute the exit which is nothing, but sending the token back to the master let us say this particular string central solutions solves the mutual exclusion problem.

(Refer Slide Time: 15:25)

Central Solution

- o Master Actions:
 - o On receiving a request from process P_i
 - if (master has token)
 - Send token to P_i
 - else
 - Add P_i to queue
 - o On receiving a token from process P_i
 - if (queue is not empty)
 - Dequeue head of queue (say P_j), send that process the token
 - else
 - Retain token

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Here in this particular scenario let us see the illustration of this central solution here we will see that there is a master which maintains a queue and also, it maintains a token. So, if any of the requesting process concurrent to and for the critical section execution, they have to make the request, they will send the request, this is the message number one and wait for the token. So, the token will be given back by another message is message number 2.

Now, after having the token, it can enter into a critical section. So, since there is only one token which is given by the masters other processes they can join in the queue why because the token is already given. Now having done its job the token will be returned back, this token will be returned back to the master and the master will also have the token with it with him, similarly once the token is there then it will grant the token to the next waiting process and so on. So, token will be granted in this order 1, 2, 3 and so on up to let us say N.

(Refer Slide Time: 17:43)

Analysis of Central Algorithm

- o Safety – at most one process in CS
 - o Exactly one token
- o Liveness – every request for CS granted eventually
 - o With N processes in system, queue has at most N processes
 - o If each process exits CS eventually and no failures, liveness guaranteed
- o FIFO Ordering is guaranteed, in order of requests received at master

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see the analysis of this central algorithm, it ensures safety why because there is exactly one token and which is controlled by the master.

So, at a time only one process is given in this particular token and the mutual exclusion that is the safety property they ensured liveness property; that means, other processes who are concurrently making the request to enter in a critical section will eventually be granted why because as soon as the exit call is executed the token will be returned back of the process which is in the critical section. Since there are no failures so, the token will be given to the next waiting process by the master. So, master ensures that eventually all the process will be given the token in some order and that order is the FIFO order.

So, the master maintains a queue of all the waiting processes and ensures that the token is to be distributed in that particular order hence all 3 properties safety liveness and fairness are ensured here in the central algorithm.

(Refer Slide Time: 19:02)

Performance Analysis

Efficient mutual exclusion algorithms use fewer messages, and make processes wait for shorter durations to access resources.

Three metrics:

- **Bandwidth:** the total number of messages sent in each enter and exit operation,
- **Client delay:** the delay incurred by a process at each enter and exit operation (when no other process is in, or waiting)
(We will prefer mostly the enter operation.)
- **Synchronization delay:** the time interval between one process exiting the critical section and the next process entering it (when there is only one process waiting)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us analyze the performance of the central algorithm although it basically ensures the mutual exclusion, but to analyze its performance whether it is efficient or not we will check whether it is going to use the fewer messages and whether the waiting of the for the critical section is also very shorter duration.

So, to ensure these two properties fewer messages and a short delay to access the common resource by the concurrent request, there are 3 different metrics which will ensure these two properties to be analyzed, the first one called the bandwidth that is total number of messages sent in each enter and exit operation. So, that becomes; so, the bandwidth says that if the more number of messages are required for entry and exit then bandwidth is going to be more and this will be against the fewer messages requirement of an efficient mutual exclusion algorithm.

The second metric is called the client delay that is the delay incurred by a process at each entry and exit operations when no other process is on or waiting for the critical section. So, that makes the analysis whether it requires a short delay to access the critical section resources or not. Third one is called synchronization delay that is the time interval between one process exits the critical section and next process enters it this is also important why because. So, after the exit there are some message communication after which the next process will enter. So, what is that particular delay is also going to be

important notion contributing the delays? So, synchronization delay is also going to be important analysis.

(Refer Slide Time: 21:20)

Analysis of Central Algorithm

- o **Bandwidth:** the total number of messages sent in each enter and exit operation.
 - o 2 messages for enter
 - o 1 message for exit
- o **Client delay:** the delay incurred by a process at each enter and exit operation (when no other process is in, or waiting)
 - o 2 message latencies (request + grant)
- o **Synchronization delay:** the time interval between one process exiting the critical section and the next process entering it (when there is only one process waiting)
 - o 2 message latencies (release + grant)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, let us analyze based on these 3 metrics the central algorithm as you know the bandwidth, the total number of messages sent in each enter and exit is nothing, but two messages to enter, two messages means first the request is sent to the master and the master has to send back the token. So, 2 messages are required to enter and as far as the exit is concerned the one which is the process which is in the critical section when it exits it has to return back the token back to the master. So, 1 message is required for the master and the bandwidth is 3 messages.

Similarly, the client delay will incur 2 messages this delay is incurred by a process at each enter and exit operation. So, exit operation. So, that will be required 2 message delay when there are no when there are no process is in the critical section or waiting because message request has to go and the master has to grant similarly as far as the synchronization delay is concerned it also requires the 2 message latency why because exit will send the token back to the master and master will then allow the next one to go.

(Refer Slide Time: 22:41)

But...

- The master is the performance bottleneck and SPoF (single point of failure)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Using the master, the mutual exclusion problem we have seen going to be solved, but there is a performance bottleneck and also the single point of failure, what happens if the master fails.

So, let us see some other distributed mutual exclusion algorithm which are going to solve this problem also and also the problem of a mutual exclusion.

(Refer Slide Time: 22:57)

Ring-based Mutual Exclusion

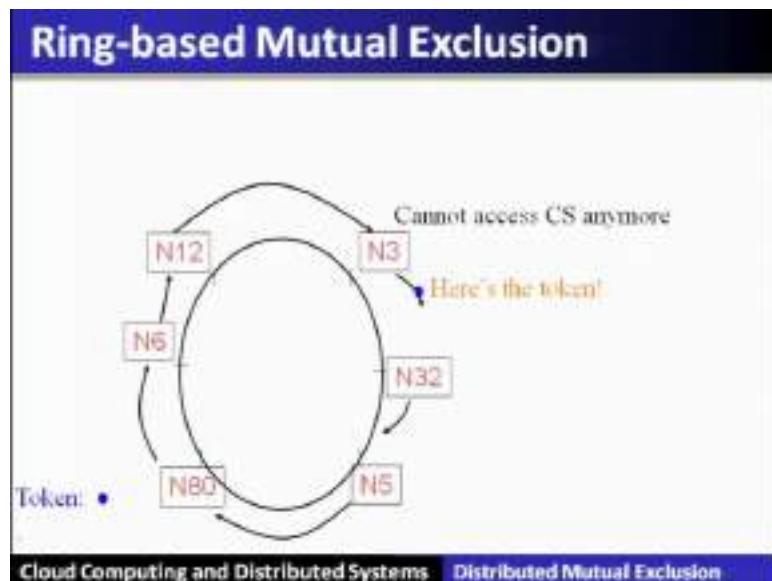
Token: •

Currently holds token.
can access CS.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

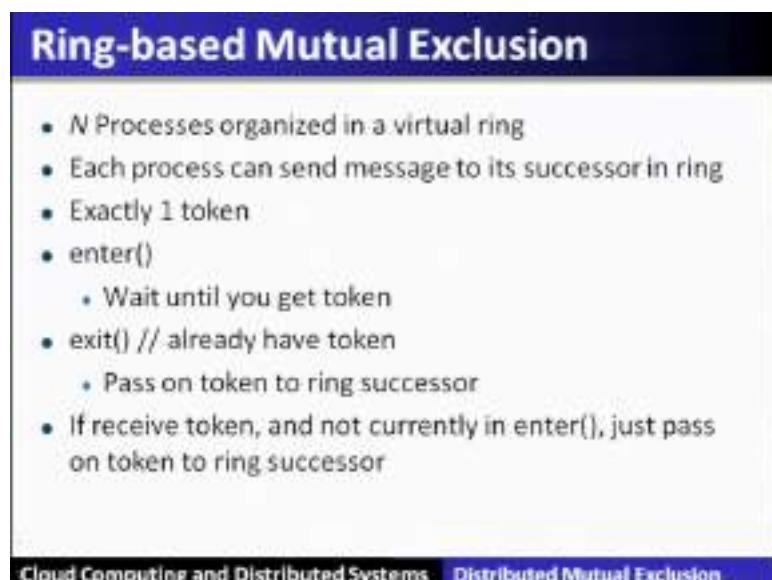
Let us consider a ring based mutual exclusion where logically all the nodes participating, they form a logical ring and there is one token which is circulating among those in a form of this overly ring structure.

(Refer Slide Time: 23:21)



Now, the nodes or the processes which are having the token they can enter into the critical section and the token will be circulated if the, if somebody requires it to enter into a critical section.

(Refer Slide Time: 23:39)



So, hence in this particular ring based mutual exclusion, let us assume that there are N processes which are organized in a virtual ring and each process can send messages to its successor in the ring exactly there is 1 token and enter function will wait until you get the token and exit is that if you already have a token, then I will pass on to the successor.

Now, if the receive token are not currently in the enter, then it will just pass on the token to the next successor.

(Refer Slide Time: 24:15)

Analysis of Ring-based Mutual Exclusion

- Safety
 - Exactly one token
- Liveness
 - Token eventually loops around ring and reaches requesting process (no failures)
- Bandwidth
 - Per enter(), 1 message by requesting process but up to N messages throughout system
 - 1 message sent per exit()

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

In this particular algorithm that is a ring based algorithm if let us see the analysis safety is ensured why because there is only one token liveness is also ensured why because the token eventually loops around the ring and reaches to the requesting process by assuming that there are no failures bandwidth, if we analyze per enter 1 message by requesting the process up to the up to N different messages throughout system and 1 message send per each exit.

(Refer Slide Time: 24:47)

Analysis of Ring-Based Mutual Exclusion (2)

- Client delay: 0 to N message transmissions after entering `enter()`
 - Best case: already have token
 - Worst case: just sent token to neighbor
- Synchronization delay between one process' `exit()` from the CS and the next process' `enter()`:
 - Between 1 and $(N-1)$ message transmissions.
 - Best case: process in `enter()` is successor of process in `exit()`
 - Worst case: process in `enter()` is predecessor of process in `exit()`

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Client delay is basically ranging from zero to N message transmission of entering after entering the `enter`. So, best case is already have the token then it will go into the critical section without any delay worst case means it has just send the token to a neighbor, it has to pass through again and minus one and come back to them that is N different message transmissions are required. Similarly synchronization delay if we analyze between the process which has exited from a critical section and next process who want to enter in a who is allowed to enter in a critical section is between from 1 to $N-1$ messages transmission

So, the best case when the process in the entry is successor of the process in the exit; so, that is the case only one message transmission is required the worst case basically is with the predecessor. So, it has to basically wait.

(Refer Slide Time: 25:45)

Next

- Client/Synchronization delay to access CS still $O(N)$ in Ring-Based approach.
- Can we make this faster?

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, here we see that the synchronization delay is quite substantial that is of the order N.
So, how we can make it a bit faster?

(Refer Slide Time: 25:55)

Lamport's Algorithm

- Requests for CS are executed in the increasing order of timestamps and time is determined by logical clocks.
- Every site S_i keeps a queue, **request queue**, which contains mutual exclusion requests ordered by their timestamps.
- This algorithm requires communication channels to deliver messages the **FIFO order**. Three types of messages are used- **Request, Reply and Release**. These messages with timestamps also **updates** logical clock.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Now, let us see the algorithm which is given by the Leslie Lamport for the distributed mutual exclusion here the request for the critical section are executed in increasing order of the timestamp and the time is being maintained by a help of the logical clock, which is also called a Lamport's clock.

Now, here every site S_i keeps a queue and this is called a request queue i which contains the mutual exclusion request which are ordered by their timestamps, this algorithm requires the communication channel to deliver the messages in FIFO order, there are 3 different type of messages which are used in this algorithm, they are request reply and release. These messages with timestamp also updates the logical clock.

(Refer Slide Time: 26:50)

The Algorithm

Requesting the critical section:

- When a site S_i wants to enter the CS, it broadcasts a $\text{REQUEST}(ts_{i,j}, i)$ message to all other sites and places the request on request_queue_i . ($ts_{i,j}$ denotes the timestamp of the request.)
- When S_j receives the $\text{REQUEST}(ts_{i,j}, i)$ message from site S_i , S_j places site S_i 's request on request_queue_j and it returns a timestamped REPLY message to S_i .

Executing the critical section: Site S_i enters the CS when the following two conditions hold:

- L1: S_i has received a message with timestamp larger than $(ts_{i,j}, i)$ from all other sites.
- L2: S_i 's request is at the top of request_queue_i .

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see the algorithm called Lamport's algorithm, the first phase first step is called requesting the critical section. So, when a site S_i want to enter in a critical section, it broadcasts a request message with a timestamp and the id of a process i .

This particular message is broadcasts to all other sites and also it places this request in its request queue when S_j another process receives this request from S_i , then it places it in its request queue and returns a reply with the timestamp to S_i back. The second step is the process can execute in a critical section if the following two conditions holds, the first one is called L1 condition which says that S_i has received the message with the timestamp larger than its own timestamp from all other sites; that means, his timestamp is the lowest that is it is the highest priority process which is on the top of the queue.

L2 property says that sis request is top of the request queue why because all other timestamps are higher than that so; obviously, it will be on the top of the queue.

(Refer Slide Time: 28:21)

The Algorithm

Releasing the critical section:

- Site S_i , upon exiting the CS, removes its request from the top of its request queue and broadcasts a timestamped RELEASE message to all other sites.
- When a site S_j receives a RELEASE message from site S_i , it removes S_i 's request from its request queue.

When a site removes a request from its request queue, its own request may come at the top of the queue, enabling it to enter the CS.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Third step is called releasing the critical section. So, a site S_i upon exiting the critical section removes its request from the top of its request queue and broadcasts a timestamp release message to all of the sites when a site S_j receives a release message from site S_i , it removes sis request from the from its request queue.

So, when a site removes a request from its request queue its own request may come at the top of the queue enabling it to enter into the critical section meaning in the sense that the release message will allow to change the q; that means, the one which has already exited will be removed from the request queue of all other sites including that site itself allowing the other waiting process the next waiting process can enter into critical section.

(Refer Slide Time: 29:24)

Correctness

Theorem: Lamport's algorithm achieves mutual exclusion.

Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites concurrently.
- This implies that at some instant in time, say t , both S_i and S_j have their own requests at the top of their request queues and condition L1 holds at them. Without loss of generality, assume that S_i 's request has smaller timestamp than the request of S_j .
- From condition L1 and FIFO property of the communication channels, it is clear that at instant t the request of S_i must be present in S_j 's request queue when S_i was executing its CS. This implies that S_j 's own request is at the top of its own request queue when a smaller timestamp request, S_i 's request, is present in the request queue — a contradiction!

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, correctness Lamport's algorithm achieves mutual exclusion let us see the proof of this theorem, proof is by contradiction suppose two sites S_i and S_j are executing into the critical section that is possible when the two conditions L1 and L2 must hold for both the sites at the same point of time this implies that at some instant of time t both S_i and S_j have their own request at their top of the request queues and the condition L1 holds at them without loss of generality assume that S_i 's request has a smaller timestamp than S_j .

Now, if S_i 's request has the smaller timestamp than S_j then from condition L1 and FIFO property of the channel it is clear that at instant t the request of S_i must be present in request queue of S_j when S_j was executing into its critical section. This implies that S_i 's own request queue is at the top of its own and S_j 's request is at the top of its own queue. So, which is not possible why because the timestamp of S_i is smaller than timestamp of S_j . So, it cannot be like this both the request S_i and S_j they are they are present in their request queues, but they have to be in the same order why because the timestamp of i is less than timestamp of j , hence both of them entering into the critical section is a contradiction.

(Refer Slide Time: 32:06)

Correctness

Theorem: Lamport's algorithm is fair.

Proof:

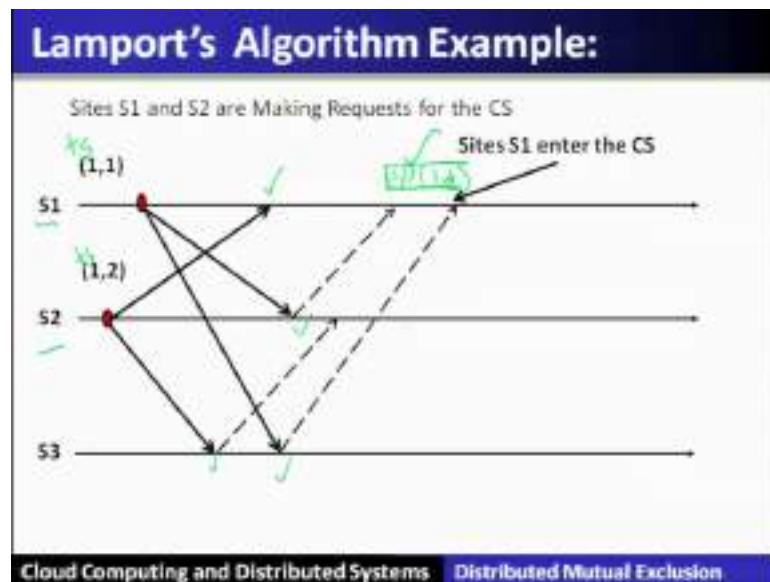
- The proof is by contradiction. Suppose a site S_i 's request has a smaller timestamp than the request of another site S_j and S_i is able to execute the CS before S_j .
$$S_i < S_j$$
- For S_i to execute the CS, it has to satisfy the conditions L1 and L2. This implies that at some instant in time say t , S_i has its own request at the top of its queue and it has also received a message with timestamp larger than the timestamp of its request from all other sites.
- But `request_queue` at a site is ordered by timestamp, and according to our assumption S_i has lower timestamp. So S_i 's request must be placed ahead of the S_j 's request in the `request_queue`. This is a contradiction!

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Now, another correctness criteria which says that the Lamport's algorithm achieves fairness proof of this is also given to be given by the contradiction suppose S_i 's request has a smaller timestamp then the request of another. Then the request another site S_j and S_i is able to execute the critical section before S_j for S_j to execute the critical section for S_j to execute the critical section it has to satisfy the condition L1 and L2 this implies that at some instant in time say t S_j has its own request at the top of its queue and it has also received a message with the timestamp larger than the timestamp of its request from all other sites

But request queue at site is ordered by the timestamp according to our assumption S_i has the low timestamp. So, S_i 's request must be placed ahead of S_j , S_j 's request in a request queue which is a contradiction hence, it achieves a fairness in the sense it follows the FIFO property or the properties that the one which is having the lower timestamp has higher priority.

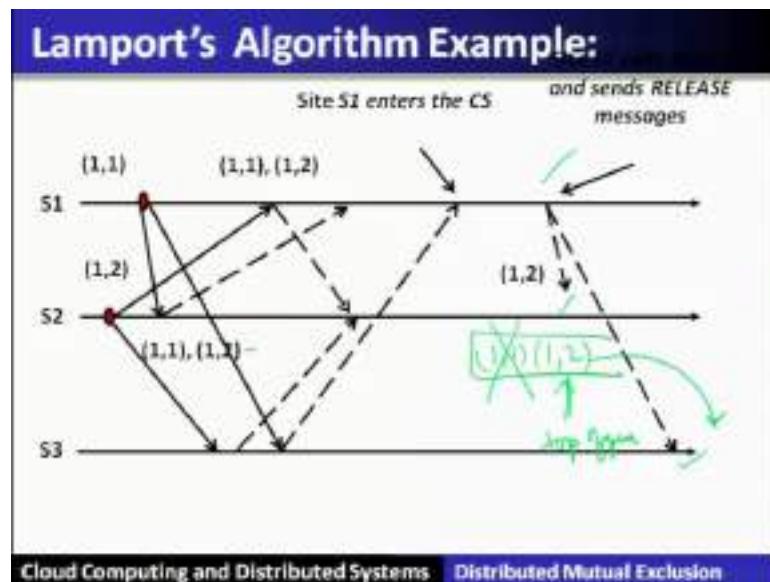
(Refer Slide Time: 33:47)



So, it can be; so, let us see the example which will illustrate this Lamport's algorithm, let us assume that simultaneously two sites S1 and S2, they want to enter in a critical section and so, they have requested and they have broadcast their request. So, S2's request will basically be received at S1 and S3 and S1's request will be received at S2 and S3. So, they also send their timestamp this is a timestamp and its number.

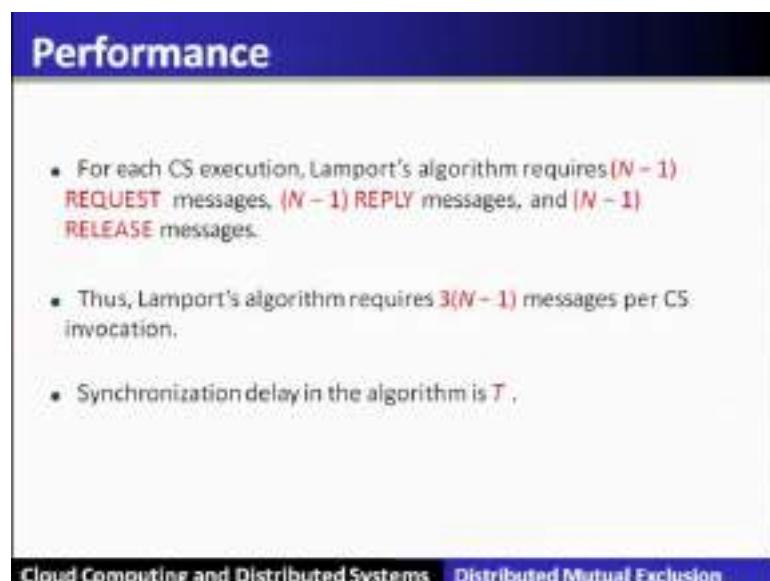
Now, as far as the second step is concerned which says that they have to reply back. So, S2 will reply to S1 and S3 will also reply and when S1 will receive the all the replies, then it will see that his request (1,1) is at the top of the queue. Similarly, S2 after receiving all the replies.

(Refer Slide Time: 35:03)



So, what will happen is now S1 will enter into a critical section. So, after it exists the critical section it will send a release message and this release message will reach to all of them including S2. So, once S2 will receive the release message it will remove it from its request queue so that his request will be at the top of the queue, now it will go in a critical section that will be shown in the next slide. So, now S2 will enter into a critical section.

(Refer Slide Time: 35:50)



Let us see the performance of this Lamport's algorithm. So, it requires $(N - 1)$ different messages in the form of request message $(N - 1)$ different messages it has to send in the form of reply and $(N - 1)$ in the form of release message total bandwidth of this algorithm is $3(N - 1)$ for per critical section in invocation.

Synchronization delay of this algorithm is quite obvious that is T why because after the release message the one which is at the top, it can go into critical section. So, it is only T as soon as the release message is received that is nothing, but a T delay.

(Refer Slide Time: 36:40)

An Optimization

- In Lamport's algorithm, **REPLY** messages can be omitted in certain situations. For example, if site S_j receives a **REQUEST** message from site S_i after it has sent its own **REQUEST** message with timestamp higher than the timestamp of site S_i 's request, then site S_j need not send a **REPLY** message to site S_i .
- This is because when site S_j receives site S_i 's request with timestamp higher than its own, it can conclude that site S_i does not have any smaller timestamp request which is still pending.
- With this optimization, Lamport's algorithm requires between $3(N - 1)$ and $2(N - 1)$ messages per CS execution.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

There is a possibility of optimization of this algorithm Lamport so; that means, optimization says that the reply can also play the role of the release as well. So, release message you can be suppressed. So, with this optimization Lamport's algorithm requires between $3(N - 1)$ and $2(N - 1)$ messages per critical section invocation.

(Refer Slide Time: 37:11)

Ricart-Agrawala's Algorithm

- Classical algorithm from 1981
- Invented by Glenn Ricart (NIH) and Ashok Agrawala (U. Maryland)
- No token
- Uses the notion of causality and multicast
- Has lower waiting time to enter CS than Ring-Based approach

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

The next algorithm is an improvement of the Lamport's algorithm which is called Ricart Agrawala's algorithm. So, this is a classical algorithm for distributed mutual exclusion given in 1981 by two famous persons one is Ricart, which was in NIH and the other one is Agrawala which was in Maryland university. Assumes no token it also assumes the notion of causality with the help of a Lamport's logical clock and also uses the multicast.

(Refer Slide Time: 37:59)

Key Idea: Ricart-Agrawala Algorithm

- enter() at process P_i
 - * **multicast** a request to all processes
 - * Request: $\langle T, P_i \rangle$; where $T =$ current Lamport timestamp at P_i
 - * Wait until **all** other processes have responded positively to request
 - * Requests are granted in order of causality
 - * $\langle T, P_i \rangle$ is used lexicographically; P_i in request $\langle T, P_i \rangle$ is used to break ties (since Lamport timestamps are not unique for concurrent events)

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Now, the key idea of Ricart Agrawala algorithm says that when it runs and it enters to a critical section; that means, it wants to go in a critical section let us say a process P_i it

will send a multicast to all other processes in the form of a request and that request is with the timestamp for causality and it will wait for the responses which are positive from all other sites. So, requests are granted in the order of causality so; that means, the one which is having the lowest timestamp is having the highest priority.

(Refer Slide Time: 38:49)

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are **FIFO**. The algorithm uses two types of messages: **REQUEST** and **REPLY**.
- A process sends a **REQUEST** message to all other processes to request their permission to enter the critical section. A process sends a **REPLY** message to a process to give its permission to that process.
- Processes use **Lamport-style logical clocks** to assign a timestamp to critical section requests, and timestamps are used to decide the priority of requests.
- Each process p_i maintains the **Request-Deferred array**, RD_i , the size of which is the same as the number of processes in the system.
- Initially, $\forall i, \forall j: RD_{ij}=0$. Whenever p_i defer the request sent by p_j , it sets $RD_{ij}=1$ and after it has sent a **REPLY** message to p_j , it sets $RD_{ij}=0$.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, Ricart Agrawala algorithm uses request and reply and also it uses the Lamport logical clock and also it uses an array which is called a request deferred array RD.

(Refer Slide Time: 39:09)

Description of the Algorithm

Requesting the critical section:

- When a site S_i wants to enter the CS, it broadcasts a timestamped **REQUEST** message to all other sites.
- When site S_i receives a **REQUEST** message from site S_j , it sends a **REPLY** message to site S_j if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i 's request's timestamp is smaller than site S_j 's own request's timestamp. Otherwise, the reply is deferred and S_i sets $RD_{ij}=1$.

Executing the critical section:

- Site S_i enters the CS after it has received a **REPLY** message from every site it sent a **REQUEST** message to.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see the algorithm it has again 3 different steps, the first step is requesting the critical section. So, when a site S_i wants to enter a critical section it broadcast timestamp request message to all the sites now, when a site S_j receives a request message from site S_i it sends a reply message to S_i if site S_j is neither requesting nor executing critical section or if the site S_j is requesting and its request timestamp is smaller than S_j 's own request timestamp. Then you send a reply, otherwise reply is deferred and S_j sets the index in its deferred request array setting the index equal to one why because it has not replied to i .

Executing critical section; that means, the site S_i enters critical section after it has received the reply from every site it has sent a request message to releasing the critical section when site S_i exists the critical section.

(Refer Slide Time: 40:20)

Contd...

Releasing the critical section:

(d) When site S_i exits the CS, it sends all the deferred REPLY messages: If $RD[i]=1$, then send a REPLY message to S_i and set $RD[i]=0$.

Notes:

- When a site receives a message, it updates its clock using the timestamp in the message.
- When a site takes up a request for the CS for processing, it updates its local clock and assigns a timestamp to the request.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

It sends all the deferred replies messages which are maintained here in request deferred array by sending the reply messages to S_i and set the request deferred to null. So, by this way after receiving the all the replies the other waiting processes can enter a critical section.

(Refer Slide Time: 40:53)

Correctness

Theorem: Ricart-Agrawala algorithm achieves mutual exclusion.

Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently and S_i 's request has higher priority than the request of S_j . Clearly, S_i received S_j 's request after it has made its own request.
- Thus, S_j can concurrently execute the CS with S_i only if S_i returns a REPLY to S_j (in response to S_j 's request) before S_j exits the CS.
- However, this is impossible because S_j 's request has lower priority.
- Therefore, Ricart-Agrawala algorithm achieves mutual exclusion.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

This Ricart Agrawala algorithm achieves mutual exclusion again, we can prove by contradiction which is quite obvious.

(Refer Slide Time: 41:07)

Ricart–Agrawala algorithm Example:

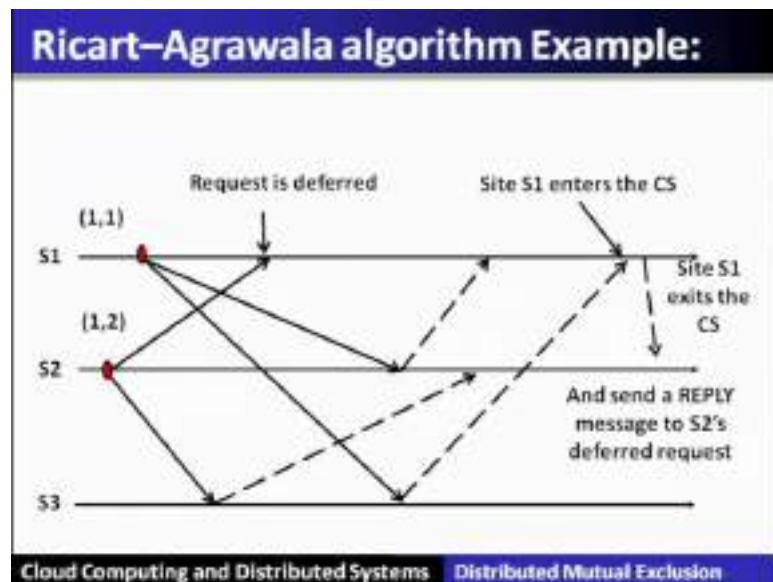
Sites S1 and S2 are Making Requests for the CS

(1,1)
S1
(1,2)
S2
S3

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

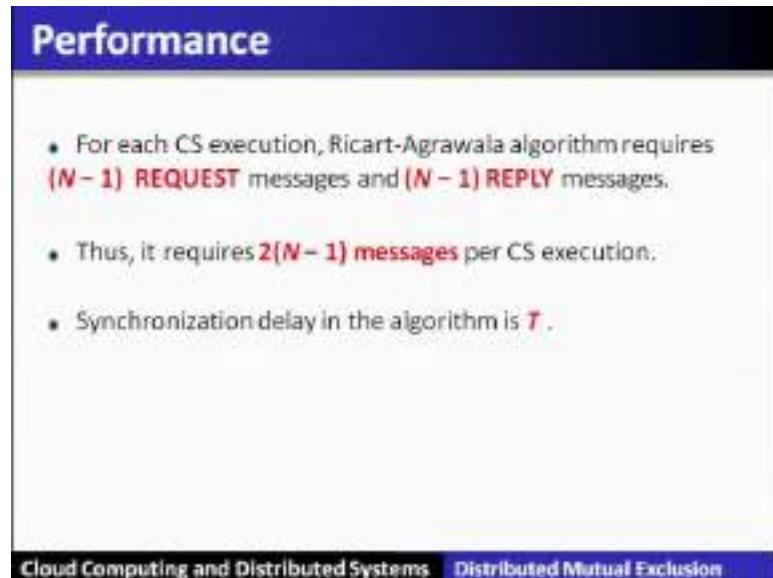
Let us see an example to understand the working of Ricart Agrawala algorithm let us see the 3 different sites out of them S1 and S2 concurrently making the request to enter in a critical section with their timestamps shown over here S1.

(Refer Slide Time: 41:20)



Now, as far as S2 when it sends a request to S1 timestamp, if you compare is lower. So, his request will be deferred. So, if S1 will not send the reply back to us to. So, S1 will enter into a critical section S1 will send a S1 will enter into a critical section and after exit, it will send the reply back to S2. So, the S2 can also enter into critical section.

(Refer Slide Time: 41:54)



Performance of Ricart Agrawala algorithm is $2(N - 1)$ messages per critical section invocation synchronization delay is same as Lamport algorithm that is T .

(Refer Slide Time: 42:12)

Comparison

- Compared to Ring-Based approach, in Ricart-Agrawala approach
 - Client/synchronization delay has now gone down to $O(1)$
 - But bandwidth has gone up to $O(N)$
- Can we get *both* down?

Compared to Ricart compared to ring based algorithm, Ricart Agrawala algorithm approach you can see that here the client server delay is you reduce that it is T means of the $O(1)$, but the bandwidth is of the $O(N)$. So, how the bandwidth can further be reduced quorum based approach.

(Refer Slide Time: 42:37)

Quorum-based approach

- In the '**quorum-based approach**', each site requests permission to execute the CS from a **subset of sites** (called a **quorum**).
- The **intersection property of quorums** make sure that only one request executes the CS at any time.

In the quorum based approach, each site requests permission to execute the critical section from a subset of sites which is called a quorum. This particular quorum or a subset of sites has to satisfy some properties for example, the property is called as

intersection properties of quorums. That make sure that only one request executes in the critical section at any point of time and not all sites are required to take permissions from Quorum based mutual exclusion algorithms differs from the previous previously discussed algorithms in two different ways.

(Refer Slide Time: 43:11)

Quorum-Based Mutual Exclusion Algorithms

Quorum-based mutual exclusion algorithms differ in **two** ways:

1. A site does not request permission from **all other sites**, but only from a **subset of the sites**.

The **request set** of sites are chosen such that
 $\forall i \forall j: 1 \leq i, j \leq N : R_i \cap R_j \neq \emptyset$.
Consequently, every pair of sites has a site which mediates conflicts between that pair.
2. A site can send out only **one REPLY** message at any time.

A site can send a **REPLY** message only after it has received a **RELEASE** message for the previous **REPLY** message.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

The first it differs that the site does not request permission from all other sites, but only from a subset of sites which is called a quorum. So, the request set of the sites are chosen to follow the intersection properties that is the request set of let us say a site i and j, if we take the intersection it will be a non null why because there must exist a site between these two request set which mediates the conflict between these two pairs.

The other way this particular quorum based algorithm differs from the previously discussed algorithm is that the site can send out only one reply message at any point of time; that means, a site can send a reply message only after it has received a release message from the previous reply messages. Unlike in the previous discussed algorithm where a particular site has to give has to reply to all other sites, here only one reply is required. So, this is an optimized algorithm compared to the other.

(Refer Slide Time: 44:33)

Contd...

Notion of 'Coteries' and 'Quorums':
A **coterie C** is defined as a set of sets, where each set $g \in C$ is called a **quorum**.

The following properties hold for quorums in a coterie:

- **Intersection property:** For every quorum $g, h \in C$, $g \cap h \neq \emptyset$.
For example, sets $\{1, 2, 3\}$, $\{2, 5, 7\}$ and $\{5, 7, 9\}$ cannot be quorums in a coterie, because first and third sets **do not have a common element**.
- **Minimality property:** There should be no quorums g, h in **coterie C** such that $g \supseteq h$ i.e. **g is superset of h**.
For example, sets $\{1, 2, 3\}$ and $\{1, 3\}$ cannot be quorums in a coterie because the first set is a **superset** of the second.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

In this algorithm, there is a notion of coteries and the quorums a coterie is defined as a set of sets where each set g which is an element which is in coterie C is called a quorum. The following properties hold for the quorums in a coterie the first one is called intersection property for every quorum let us say g and h which is there in coterie the intersection of these two quorums is a not null.

For example that sets $\{1, 2, 3, 2, 5, 7\}$ and $\{5, 7, 9\}$; 3 different sets shown here in this example cannot be the quorums of a coterie why because if you take the intersection of the first set that is $\{1, 2, 3\}$ and the intersection of the third one that is $\{5, 7, 9\}$, it does not have a not null; that means, it is null so; that means, it does not have a common member hence intersection properties not satisfied therefore, these 3 different sets shown in the example are not the coteries or they are not the quorums.

The second property which quorums in a coterie follows is called minimality property which says that there should be no quorum say for example, g and h in coterie C such that quorum g is a superset of quorum h in a coterie C that is g is a superset of h ; that means, it is violating the minimality property, if it is a superset of another quorum. So, for example, the set $\{1, 2, 3\}$ and $\{1, 3\}$ they cannot be quorum why because $\{1, 2, 3\}$ is a superset of $\{1, 3\}$ and therefore, minimality property is violated.

(Refer Slide Time: 46:38)

Maekawa's Algorithm

Maekawa's algorithm was **first quorum-based mutual exclusion algorithm**.

- The **request sets for sites** (i.e., **quorums**) in Maekawa's algorithm are constructed to satisfy the following conditions:
 - M1:** $(\forall i \forall j : i \neq j, 1 \leq i, j \leq N :: R_i \cap R_j = \emptyset)$
 - M2:** $(\forall i : 1 \leq i \leq N :: S_i \in R_i)$
 - M3:** $(\forall i : 1 \leq i \leq N :: |R_i| = K)$
 - M4:** Any site S_j is contained in K number of R_i s, $1 \leq i, j \leq N$.

Maekawa used the theory of projective planes and showed that $N = K(K - 1) + 1$. This relation gives $|R_i| = \sqrt{N}$.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Using the concept of quorums Maekawa has given the first algorithm which is called Maekawas algorithm for mutual exclusion that is the first quorum based mutual exclusion algorithm. In this algorithm the request sets for the sites which is also nothing, but the quorum are constructed to satisfy the following 4 properties, they are which is designated as M1 which says that critical section property; that means, for any two request sets if we take the intersection it is not null the another property M2 which says that any site is in the request set. So, that is all the sites are participating in some or the other request sets.

M3 property says that the cardinality of a request set is k and the value of k is computed as root of N, that is the request set that is the size of quorum is not n, but root N that is the subset of the sites M4 the fourth property says that any site S_j is contained in K number of R_i s so; that means, the fourth property says that any site is contained in K number of R_i s; that means, there is a property of a load balancing; that means, every site has to do equal amount of work here in Maekawa's algorithm.

(Refer Slide Time: 48:17)

Maekawa's Algorithm

- Conditions **M1** and **M2** are necessary for correctness; whereas conditions **M3** and **M4** provide other desirable features to the algorithm.
- Condition **M3** states that the size of the requests sets of all sites must be equal implying that all sites should have **to do an equal amount of work** to invoke mutual exclusion.
- Condition **M4** enforces that exactly the same number of sites should request permission from any site, which implies that all sites have "**equal responsibility**" in granting permission to other sites.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, the property M1 and M2 are the necessary for the correctness whereas, M3 and M4 they provide other desirable properties of the algorithm the property M3 states that the size of the request set of all the sites must be equal implying that all site should have to do an equal amount of work to invoke the mutual exclusion. Whereas, the condition enforces that exactly the same number of sites should basically request permission from any site which implies that all sites have equal responsibility in granting the permission to the other sites.

(Refer Slide Time: 48:51)

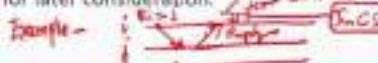
The Algorithm

A site S_j executes the following steps to execute the CS:

Requesting the critical section

(a) A site S_j requests access to the CS by sending **REQUEST(i)** messages to all sites in its request set R_j .

(b) When a site S_j receives the **REQUEST(i)** message, it sends a **REPLY(j)** message to S_i provided it hasn't sent a **REPLY** message to a site since its receipt of the last **RELEASE** message. Otherwise, it queues up the **REQUEST(i)** for later consideration.



Executing the critical section

(c) Site S_j executes the CS only after it has received a **REPLY** message from every site in R_j .

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see the algorithm which is given by the Maekawa for mutual exclusion which is called a quorum based mutual exclusion algorithm the site S_i executes the following step to execute the critical section the first one is to request for the critical section site S_i request to access the critical section by sending a request and request of i and this particular message is sent to all the sites in its request set R_i .

Now, when a site S_j receives the request i message it sends a reply message to S_i provided it has not sent a reply message to a site since its receipt of the last release message. Otherwise it queues up the request i message for later consideration third one is or second one is called executing the critical section that is site S_i executes the critical section only after it has received the reply message from every site in R_i .

Let us see through an example let us consider the 2 sites or 3 sites i, j and k . So, if i is requesting it has to send the request set, let us say the request set of i is having only j . So, it will send the message only to j not to k this is the first step in the second step when the site S_j receives the request message then it sends a reply to S_i provided it has not sends a reply to anyone since j has not sends a reply so, it will send a reply back. Now if it has send a reply already to some other site then it will not send then it will queue up this particular request needs queue this request will be there why because it has already given the reply.

The critical section; that means, when site j replies back and this is the only site in the request set; that means, i has got the replies from all in its request set, then it will be in critical section. So, at this point of time it will be in critical section, it will execute in a critical section.

(Refer Slide Time: 51:55)

The Algorithm

Releasing the critical section

(d) After the execution of the CS is over, site S_i sends a **RELEASE[i]** message to every site in R_i .

(e) When a site S_j receives a **RELEASE[i]** message from site S_i , it sends a **REPLY** message to the next site waiting in the queue and deletes that entry from the queue.

If the queue is empty, then the site updates its state to reflect that it has not sent out any **REPLY** message since the receipt of the last **RELEASE** message.

Cloud Computing and Distributed Systems | Distributed Mutual Exclusion

Now, let us see the third step, releasing the critical section after executing the critical section by a site i the site i sends the release message to every site in R_i now when site S_j receives message from S_i ; it sends a reply message back to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty then the site updates its state to reflect that it has not sent out any reply since the receipt of the last release message.

(Refer Slide Time: 52:27)

Correctness

Theorem: Moekawa's algorithm achieves mutual exclusion.
Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are concurrently executing the CS.
- This means site S_i received a **REPLY** message from all sites in R_i and concurrently site S_j was able to receive a **REPLY** message from all sites in R_j .
- If $R_i \cap R_j = \{S_k\}$, then site S_k must have sent **REPLY** messages to both S_i and S_j concurrently, which is a contradiction.

Cloud Computing and Distributed Systems | Distributed Mutual Exclusion

Maekawa's algorithm achieves mutual exclusion proof is by contradiction suppose two sites S_i and S_j concurrently executing in into the critical section this means that the S_i has received the reply from all the sites in its request set and S_j also has received the replies from its request sets R_j . Now we know that there is a intersection properties which says that R_i intersection R_j is not null therefore, there must exist a site without loss of generality.

Let us say that the site is k . So, k has; that means, this means that k has sent two replies at least, that is two R_i and R_j which is a contradiction why because the second property, that is the second property which says that it can send only one reply every site has to send only one reply which is the contradiction. Hence, the our assumption we say that the two sides are currently executing critical section hence it achieves the mutual exclusion.

(Refer Slide Time: 54:24)

Performance

$|R_i| = \sqrt{N}$

- Since the size of a request set is \sqrt{N} , an execution of the CS requires \sqrt{N} REQUEST, \sqrt{N} REPLY, and \sqrt{N} RELEASE messages, resulting in $3\sqrt{N}$ messages per CS execution.
 $\text{Request} = O(\sqrt{N})$
 $\text{Reply} =$
- Synchronization delay in this algorithm is $2T$. This is because after a site S_j exits the CS, it first releases all the sites in R_j and then one of those sites sends a REPLY message to the next site that executes the CS.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Next is the performance of this algorithm now we see that there are 3 different type of messages request set requesting the second one to get the replies and then finally, when it exist the critical section release, now you know that the size of request set is root N. So, the request messages will be of the order root N similarly reply also will be that much amount and the release also will be so; that means, there are 3 route N messages for critical section invocation is required here in this algorithm. Synchronization delay is $2T$ why because after S_i exit the critical section then it sends a release to all other sites and

then the site which has queued up the other requesting site it will again send the reply message.

So, release will trigger the reply message, hence this is one t this is another T that is total $2T$; that means, it will exit after it exist the critical section $2T$ is the synchronization delay to enter to a critical section by any requesting site, hence the synchronization delay is off to T .

(Refer Slide Time: 56:16)

Problem of Deadlocks

- Maekawa's algorithm can deadlock because a site is exclusively locked by other sites and requests are not prioritized by their timestamps.

Assume three sites S_i , S_j , and S_k simultaneously invoke mutual exclusion.

Suppose $R_i \cap R_j = \{S_{ij}\}$, $R_j \cap R_k = \{S_{jk}\}$, and $R_k \cap R_i = \{S_{ki}\}$.

Consider the following scenario:

1. S_j has been locked by S_i (forcing S_j to wait at S_{ij}). S_i is marked with a red circle and S_{ij} is marked with a red circle.
2. S_{jk} has been locked by S_j (forcing S_k to wait at S_{jk}). S_j is marked with a red circle and S_{jk} is marked with a red circle.
3. S_{ki} has been locked by S_k (forcing S_i to wait at S_{ki}). S_k is marked with a red circle and S_{ki} is marked with a red circle.

This state represents a deadlock involving sites S_i , S_j , and S_k .

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

The Maekawa's algorithm suffers from the deadlocks because a site is exclusively locked by the other sites and the requests are also not prioritize either timestamp here you have seen that there is no use of Lamport timestamps to see this plus considered the 3 different sites S_i , S_j and S_k simultaneously they want to execute the critical section. So, they invoke the mutual exclusion algorithm and we know that out of these sites S_i , S_j and S_k if you take the intersection of their request set let us assume that R_i and R_j have the common site that S_i , S_{ij} , S_j similarly S_j and S_k will have a common site as jk similarly for ki it will be S_{ki} .

Now, consider the scenario that S_{ij} has been locked by S_i ; that means, this will force S_j to wait at S_{ij} . So, for example, if this is if S_i forcing S_j to wait. So, here S_j will be waiting at S_{ij} because it has send a request, but it has not send the reply back reply is queued up. Why because it has already send the reply to some other to some other site similarly, S_{jk} has been blocked by S_j let us say here there is S_j and that is locked. Similarly S_k has

been locked S_k so; that means, these 3 different sites S_i , S_j and S_k , they are waiting for the replies from their common sites that is S_{ij} , S_{jk} and S_{ki} which has already given the replies and waiting for the replies from the other waiting process for this will form a circular wait and therefore, this may lead to a deadlock in this approach.

(Refer Slide Time: 59:05)

Handling Deadlocks

- Maekawa's algorithm handles deadlocks by requiring a site to yield a lock if the timestamp of its request is larger than the timestamp of some other request waiting for the same lock.
- A site suspects a deadlock (and initiates message exchanges to resolve it) whenever a higher priority request arrives and waits at a site because the site has sent a REPLY message to a lower priority request.

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

How to handle the deadlock Maekawa algorithm has also given the provision to handle the deadlock; so, Maekawa hand I will go algorithm handles deadlock by requiring site to yield the lock. If the timestamp of its request is larger than the timestamp of other requests waiting for the same lock [FL]. Handling deadlocks Maekawa's algorithm handles deadlocks by requiring a site to yield the lock if the timestamp of its request is larger than the timestamp of some other request waiting for some lock.

That means, it has given a provision of yielding a lock if higher priority process comes and finds that the replies has already been sent to a low priority process, then the lock will be yield and given back to the higher priority process therefore, this way that a lock will be broken up. Let us see now it does in Maekawas algorithm this concept of yielding.

(Refer Slide Time: 60:38)

Message types for Handling Deadlocks

Deadlock handling requires **three types of messages**:

FAILED: A FAILED message from site S_i to site S_j indicates that S_i can not grant S_j 's request because it has currently granted permission to a site with a higher priority request.

INQUIRE: An INQUIRE message from S_i to S_j indicates that S_i would like to find out from S_j if it has succeeded in locking all the sites in its request set.

YIELD: A YIELD message from site S_i to S_j indicates that S_i is returning the permission to S_j (to yield to a higher priority request at S_j).

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

To handle the deadlock in the Maekawa's algorithm there are 3 different type of messages the first one is call failed message failed message from a site i to j indicates that as i cannot grant S_j is this request because it has correctly granted the permission to a site with a higher priority request, take this example for example.

For example, this is let us say i and ji is requesting for a lock this will be given a failed message, if j has already given to a high priority process and this i is a low priority process. So, it will fail, the second message is called inquire if this is not the case N if i is a higher priority and j has already given to some low priority, then it will not fail, but it will inquire message an inquire message from S_i to S_j indicates that I would like to find out from S_j if it has succeeded and locking all it sites in its request set. So, here what j will do for example, j will send an inquire message to k to find out whether k is successful in getting all the locks in its request set if not then there will be a provision called yield message. So, yield message from site S_i to S_j will indicate that S_i is returning the permission to S_j to yield to a high priority request at S_j .

So, for example, after inquiring it has identified that it has not k has not succeeded in getting all the locks then what will happen this j will take the action it will yield the lock; that means, it will take the lock back from k and it will give it to higher priority process using a yield. So, in this way you see a low priority process is broken out from the circularly waiting set of sites using this 3 kind of messages failed inquire and yield.

(Refer Slide Time: 63:44)

Handling Deadlocks

Maekawa's algorithm handles deadlocks as follows:

- When a **REQUEST(i , j)** from site S_i blocks at site S_j because S_j has currently granted permission to site S_k , then S_j sends a **FAILED(j)** message to S_i if S_i 's request has lower priority. Otherwise, S_j sends an **INQUIRE(j)** message to site S_k .
- In response to an **INQUIRE(j)** message from site S_j , site S_k sends a **YIELD(k)** message to S_j provided S_k has received a **FAILED** message from a site in its request set and it sent a **YIELD** to any of these sites, but has not received a new **REPLY** from it.
- In response to a **YIELD(k)** message from site S_k , site S_j assumes as if it has been released by S_k , places the request of S_k at appropriate location in the request queue, and sends a **REPLY(j)** to the top request's site in the queue.

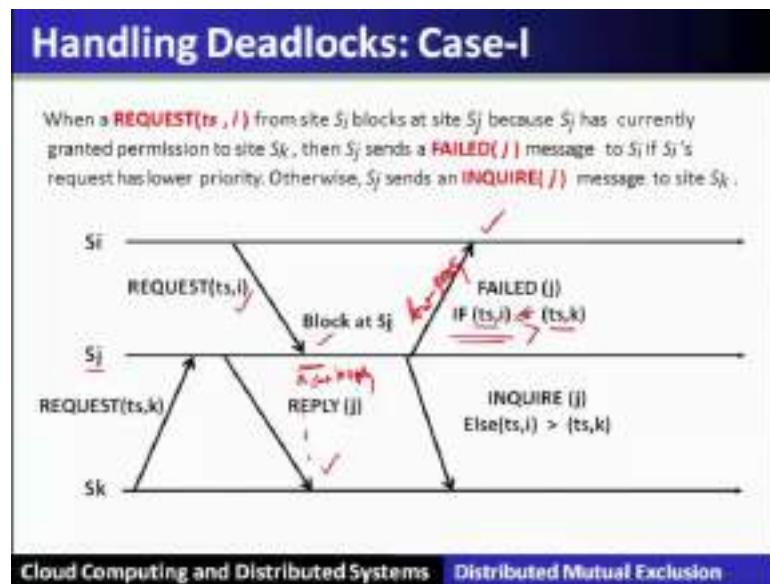
Maximum number of messages required per CS execution in this case is $5 \sqrt{N}$

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Let us see how this all happens inside the algorithm. So, now, it also uses the timestamp to determine the high priority processes. So, when a request with the timestamp t_i from a site S_i blocks at S_j because S_j has currently granted permission to site S_k then S_j will send a failed message to S_i if S_i 's request has a lower priority otherwise it will send an inquire message to site S_k .

Now, in response to inquire message from S_{jk} , S_j will send yield message to site S_k provided S_k has received message from a site in its request set; that means, S_k has not succeeded in getting all the locks acquired from its request set and you send yield to any of these sites. But has not received the new reply from it, in response to the yield message from the site S_k , S_j will assume that it has been released by S_k and places the request of S_k at the appropriate location in the request queue and send the reply to the top of the request set in the queue in this process the total number of messages required to enter in a critical section is $5 \sqrt{N}$.

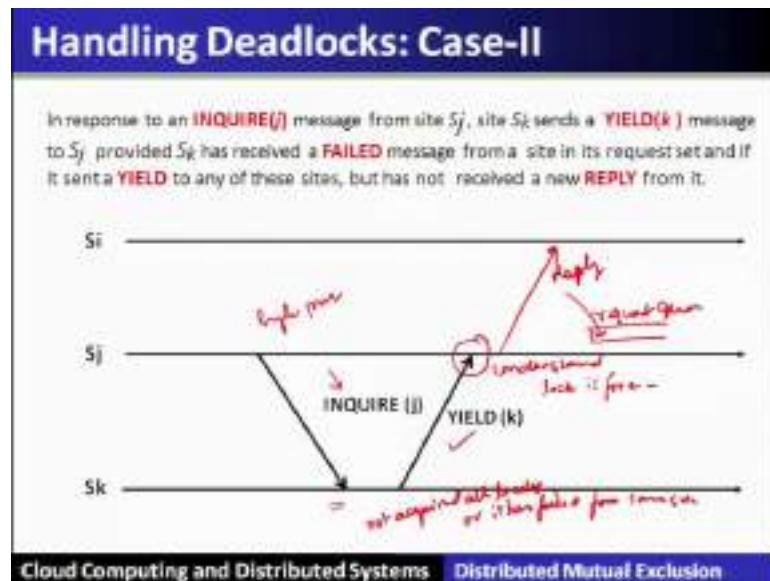
(Refer Slide Time: 65:26)



Let us see through this particular example we have already explained it, i is requesting and which gets block at S_j . Now S_j will have already sent the replies to k therefore, it gets a block due to this reply now then this particular message which is requesting will basically use the timestamps to compare their priorities.

Now, if the timestamp if the timestamp of i is basically less than the timestamp of k then it will do so; that means, if the timestamp of i is higher than the timestamp of k ; that means, it is if it is low priority, then it will send a failed message.

(Refer Slide Time: 64:42)



So, let us see through this example here in this case. So, if it is the lower if it is the high priority, then it will send a inquire message to find out the status of the lock and if it is found that S_k has not acquired all the locks or it has failed from some site then in that case it will yield the lock S_k will yield the lock.

S_j will understand that this lock is free it can be given to any other process. So, it will now check its request queue and whatever process is on the top of the queue, it will send the reply to the other higher waiting process. This way this deadlock is handled in this particular scenario this is what I have already explained.

(Refer Slide Time: 68:02)

The slide has a dark blue header bar with the title "Failures?" in white. The main content area is white with a black border. A single bullet point is listed: "• Other ways to handle failures: Use Paxos like approaches!". At the bottom, there is a dark blue footer bar with two items: "Cloud Computing and Distributed Systems" and "Distributed Mutual Exclusion".

Now, the failures we have seen the that the previous algorithms have assumed there is no failure. So, we will see some other method how to handle the failures and yet you can be solving the mutual exclusion problem. So, we will see that Paxos like schemes are being used widely in the industry solutions.

(Refer Slide Time: 68:25)

Industry Mutual Exclusion : Chubby

- Google's system for locking
- Used underneath Google's systems like BigTable, Megastore, etc.
- Chubby provides *Advisory* locks only
 - Doesn't guarantee mutual exclusion unless every client checks lock before accessing resource

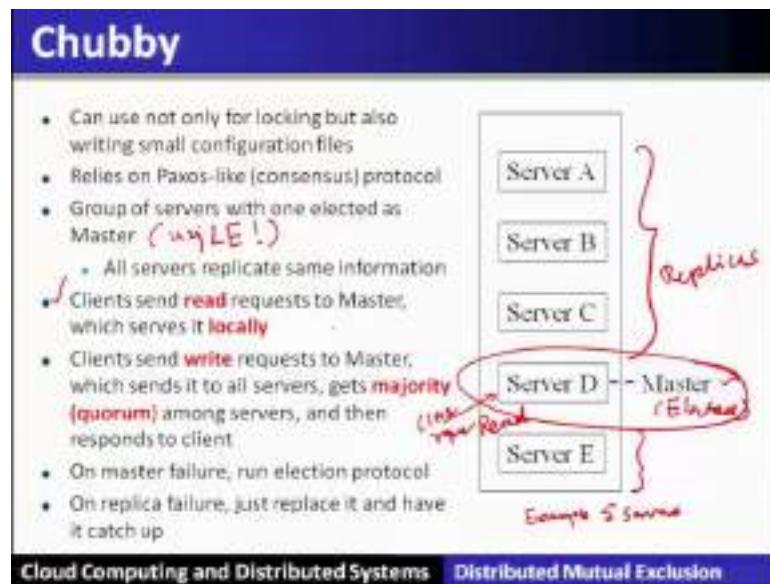
Reference: <http://research.google.com/archive/chubby.html>

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

So, we are going to discuss briefly about the Google's chubby system which is nothing, but a locking system which uses the Paxos like scheme for the failures handling failures and also to solve the problem of distributed mutual exclusion.

This Google's chubby system in its Google system you can see in the, its use in the BigTable and megastore like applications now the chubby uses advisory locks; advisory locks means that the client has to invoke this particular lock then only the mutual exclusion is guaranteed if the client does not do that then the mutual exclusion is not taken care of. So, that means, that it does not guarantees the mutual exclusion unless every client locks before accessing the critical resources.

(Refer Slide Time: 69:38)



Let us go and see the details of chubby system. So, chubby is a lock is a distributed lock service and it also basically writing into a small configuration files. So, locking and configuration files are two important services which this Google chubby system provides and which relies on the Paxos like consensus protocol to deal with the failures. In this particular system there is a group of servers out of them one is elected as a master using any leader election algorithm and other servers are the replicas.

In this example this is a group of 5 servers one which is called a master let us say that it is elected among other 5 servers and all other servers which are not elected they are called replica servers; in this particular protocol, the client will send the requests the read request will be sent to the master and master will serve locally. So, the client send the read request to the master which serves it locally directly.

(Refer Slide Time: 71:31)

Chubby

- Can use not only for locking but also writing small configuration files
- Relies on Paxos-like (consensus) protocol
- Group of servers with one elected as Master (any LE!)
- All servers replicate same information
- Clients send **read** requests to Master, which serves it **locally**
- Clients send **write** requests to Master, which sends it to all servers, gets **majority (quorum)** among servers, and then responds to client
- On master failure, run election protocol
- On replica failure, just replace it and have it catch up

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

However when a client sends a write request to the master then master will send it to all the servers it will broadcast and wait for the replies.

If the majority replies this becomes a majority. So, majority is also decided by the size of the quorum. So, if the majority gets the replies or response. So, the master correspondingly response back with the right operations, now you know that if the majority is taken care in the right operation and as far as any two quorums are concerned they are not null as far as the intersection properties are concerned so; that means, there must be some site which will like which will mediate. So, that all replicas are basically updated with this right operations eventually. As far as the failures are concerned, when the master fails when this particular master fails.

(Refer Slide Time: 73:01)

Chubby

- Can use not only for locking but also writing small configuration files
- Relies on Paxos-like (consensus) protocol
- Group of servers with one elected as Master (~~any LE!~~)
 - All servers replicate same information
- Clients send **read** requests to Master, which serves it **locally** ✓
- Clients send **write** requests to Master, which sends it to all servers, gets **majority (quorum)** among servers, and then responds to client ✓
- On master failure, run election protocol ✓
- On replica failure, just replace it and have it catch up

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Then again among the remaining the leader election will happen and let us say this will become a new master; now when a replica fails on the other hand when this replica fails.

(Refer Slide Time: 73:20)

Chubby

- Can use not only for locking but also writing small configuration files
- Relies on Paxos-like (consensus) protocol
- Group of servers with one elected as Master (~~any LE!~~)
 - All servers replicate same information
- Clients send **read** requests to Master, which serves it **locally** ✓
- Clients send **write** requests to Master, which sends it to all servers, gets **majority (quorum)** among servers, and then responds to client ✓
- On master failure, run election protocol ✓
- On replica failure, just replace it and have it catch up

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Then it will just replace it with some other new server and it will catch up with the other set of operations.

(Refer Slide Time: 73:50)

Conclusion

- Mutual exclusion important problem in cloud computing systems
- Classical algorithms
 - Central
 - Ring-based
 - Lamport's Algorithm
 - Ricart-Agrawala
 - Maekawa
- Industry systems
 - Chubby: a coordination service
 - Similarly, Apache Zookeeper for coordination

Cloud Computing and Distributed Systems Distributed Mutual Exclusion

Conclusion; mutual exclusion is an important problem in a cloud computing system and we have seen several classical mutual exclusion algorithms such as central algorithm ring based algorithm Lamport's algorithm, Ricart Agrawala algorithm and Maekawa's algorithm. We have seen the Maekawa's algorithm is more efficient as far as performance in compared to the Lamport's and Ricart Agrawala algorithms are concerned we have also seen that these algorithms are assuming that there are no failures as far as the industry systems are concerned they assume that a system fails and yet they are able to resolve the mutual exclusion.

Failures are handled in a Paxos like scheme that we have seen in a chubby system which is basically a coordination based lock system. Similarly we will see that apache zookeeper is also a coordination service that also uses the Paxos like schemes and also solve.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture - 13
Consensus in Cloud Computing and Paxos

Consensus in Cloud Computing and Paxos protocol.

(Refer Slide Time: 00:17)

The slide has a blue header bar with the word "Preface" in white. Below it is a white content area. At the top of the content area, there is a red header "Content of this Lecture:". Below this, there is a bulleted list of two items. At the bottom of the slide, there is a footer bar with two tabs: "Cloud Computing and Distributed Systems" and "Consensus in Cloud Computing & Paxos".

Content of this Lecture:

- In this lecture, we will discuss the '**consensus problem**' and its variants, its solvability under different failure models .
- We will also discuss the industry use of consensus using '**Paxos**'.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Preface content of this lecture. We will cover consensus problem and its variants. It is solvability under different failure models. We will discuss in this part of a lecture. We will also discuss the industry variant of consensus using a well-known protocol which is called Paxos.

(Refer Slide Time: 00:41)

Common issues in consensus

Consider a group of servers attempting together:

- Make sure that all of them receive the same updates in the same order as each other [Reliable Multicast]
- To keep their own local lists where they know about each other, and when anyone leaves or fails, everyone is updated simultaneously [Membership/Failure Detection]
- Elect a leader among them, and let everyone in the group know about it [Leader Election]
- To ensure mutually exclusive (one process at a time only) access to a critical resource like a file [Mutual Exclusion]

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Let us understand what a consensus is. So, before going ahead let us see the importance of this problem in cloud and distributed computing.

So, in a cloud computing different vendors they provide the reliability support for the services. The reliability services ensures that it is, let us say 5 number of factors such as 99.99999 percentage of reliability; that means, it is not possible to provide 100 percent reliable system in the face of failures. So, that means, if the underlying systems are the servers are failing how about the services reliability. So, none of the vendors or none of them can provide 100 percent reliability in their services; however, they support the reliabilities, and it is mentioned that 99 point how many number of 9's; that means, how reliable the system is, but it is not 100 percent reliable.

Now, the this particular reliability how they are ensuring that is achieved through the consensus protocols. So, that means, when a group of servers are attempting together to solve a particular problem in spite of failure they have to reach to a consensus. And therefore, they achieve this kind of solutions. But this is not so trivial it is very, very complex in different models that we will cover in this part of discussion.

So, let us understand where this consensus problems lies and what are their names in the distributed systems. So, let us consider a group of servers who are attempting together to make sure that all of them receives the same updates in the same order as they send. So,

this kind of problem is known as reliable multicast problem. And this requires the solution of consensus to be involved.

The second kind of problem is that to keep their own local list; where they know about each other and when they and when anyone leaves or fails. So, everyone updates simultaneously. So, this kind of problem is called membership problem or a failure detection problem. This also requires the consensus or this particular problem is also equivalent to the consensus, that we will see the later part.

Similarly, a group of servers who are attempting together to elect a leader among them, and this particular decision of electing the leader is known to everyone in that group is called a leader election problem. So, leader election under this particular failure requires the solution of consensus. So, if you can solve the consensus problem then you can solve the leader election, and that is why if they are all equivalent problems.

Similarly there is another problem which is considering that a group of servers attempting together to ensure the mutually exclusive access to the critical resources, such as files writing on a file and so on or an update on the replicated database. Which requires access to the competing access to one process among the competing processes at a particular instant of time. So, that is this problem is called as a mutual exclusion problem. In the failure this problem is also requires the consensus protocols for reliability aspect.

(Refer Slide Time: 06:21)

So what is common?

- All of these were groups of processes attempting to **coordinate** with each other and reach **agreement** on the value of something
 - The ordering of messages
 - The up/down status of a suspected failed process
 - Who the leader is
 - Who has access to the critical resource
- All of these are related to the **Consensus** problem

So, let us see that what is common in all these problems. So, here we have seen that all of these group of processes, they are attempting to coordinate with each other and reach an agreement on something in different problems that we have seen leader election mutual exclusion and so on. So, the agreement on that values depends upon different problems, let us understand what kind of agreement is required to solve this consensus problem.

So, it depends upon whether it is an ordering of messages at all the processes in the groups the up or down status of the suspected field process and who is a leader and who has the access to the critical resource. So, that depends upon different values to be agreed upon to solve this particular these kind of problems. All of these are related to the consensus problem.

(Refer Slide Time: 07:23)

What is a Consensus Problem?

Formal problem statement:

- N processes
- Each process p has
 - input variable x_p : initially either 0 or 1
 - output variable y_p : initially b (can be changed only once)
- **Consensus problem:** design a protocol so that at the end, either:
 1. All processes set their output variables to 0 (all-0's)
 2. Or All processes set their output variables to 1 (all-1's)

So, having known what a consensus is. So, let us formally define what a consensus problem is for our discussion in this lecture.

So, the formally statement says that, let us have a system of N processes where each process has the input variable x_p . Initially it can have the values either 0 or 1 in that variable x_p . Similarly, the output variable y_p initially which is assigned to b which can be changed only once and once output means, whatever value is assigned it will not be changed then after the end of consensus algorithm. So, consensus problem says to design a protocol. So, the at the end either all the processes in that particular group. They set

their output variables to 0s; that means, all 0's or the processes sets their output variables to 1, that is all 1's. Then only this particular statement is called the consensus problem.

(Refer Slide Time: 08:40)

What is Consensus? (2)

- Every process contributes a value
- **Goal is to have all processes decide same (some) value**
 - Decision once made can't be changed
- There might be other constraints:
 - **Validity:** if everyone proposes same value, then that's what's decided
 - **Integrity :** decided value must have been proposed by some process
 - **Non-triviality :** there is at least one initial system state that leads to each of the all-0's or all-1's outcomes

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Now, here in this particular consensus problem, every process will contribute toward value and the goal is to have all the processes decide on some value. So, once the decision is finalized it cannot be changed. There are some other constraints let us see, which will be required to be enforced to solve this particular consensus problem. The first constraint is called validity that everyone proposes same value, then that is what is being decided called validity.

Integrity means that decided value must have been proposed by some process. Non triviality says that there is at least one initial state that leads to each of the all 0's or all ones outcomes. So, 3 different properties or constraints they have to be satisfied. Then only we can say that the consensus problem is solved validity integrity non triviality.

(Refer Slide Time: 09:57)

Why is it Important?

- Many problems in distributed systems are **equivalent to** (**or harder than**) consensus!
 - Perfect Failure Detection
 - Leader election (select exactly one leader, and every alive process knows about it)
 - Agreement (harder than consensus)
- So consensus is a very important problem, and solving it would be really useful!
- **So, is there a solution to Consensus?**

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Why this consensus problem is important? So, here we can see in many problems in the distributed systems they are equivalent to the consensus problem or even harder than that; that means, if you can solve consensus problem those problems in the distributed systems are also been solved.

For example, failure detection. So, if you can detect a failure; that means, you can solve the consensus problem. Or consensus problem if it is, solvable then you can also detect the failures similarly leader election exactly one leader, and every alive process knows about it called leader election problem, if leader election problem is equivalent to the consensus problem. Similarly, the agreement problem; that means, they can reach all of them reach to an agreement to a common value is harder than consensus problem.

So, the consensus is very important problem and solving it would be really useful. So, is there any solution to a consensus problem we will see that it is very, very complex problem. It is not trivial to solve this particular problem in some of the models.

(Refer Slide Time: 11:08)

Two Different Models of Distributed Systems

- Different Models of Distributed Systems:
 - (i) Synchronous System Model and
 - (ii) Asynchronous System Model

(i) Synchronous Distributed System

- Each message is received within bounded time
- Drift of each process' local clock has a known bound
- Each step in a process takes $lb < \text{time} < ub$

E.g., A collection of processors connected by a communication bus, e.g., a Cray supercomputer or a multicore machine

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, there are 2 different models of distributed system which also is applied to the cloud computing systems. So, we will see under these models how the consensus problem can be solved.

So, the first of these 2 models is called synchronous distributed system. So, synchronous model says that each message is received within a bounded time. And the drift of each processes local clock is also bounded. And each step in a process is bounded in a time between the lower and upper bounds. So, meaning in the sense that in the synchronous distributed system, the assumptions are being made that each process is delivered within a bounded period of time and the processes local clocks also when they drift that also is bounded and each step in a process is also bounded then only it is called synchronous distributed systems.

(Refer Slide Time: 12:25)

Asynchronous System Model

(ii) Asynchronous Distributed System

- No bounds on process execution
- The drift rate of a clock is arbitrary
- No bounds on message transmission delays

E.g., The Internet is an asynchronous distributed system, so are ad-hoc and sensor networks.

- This is a more **general (and thus challenging)** model than the synchronous system model.
- A protocol for an asynchronous system will also work for a synchronous system (but not vice-versa)

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Asynchronous on the other hand, does not have any bounds specified on the process execution. Similarly, the clocks when they drift that particular bound is also arbitrary and the message transmission delays; that means, when the message is received, that particular delay also is not bounded then that model is called asynchronous distributed or asynchronous model. Internet is a perfect example of asynchronous distributed system model and there are many other networks which is an example of a asynchronous model.

So, there is a more general model than the synchronous system model; that means, if you can solve the problem under this model asynchronous system model. That means, that problem also can be solved in a synchronous model, but vice versa is not valid.

(Refer Slide Time: 13:35)

Possible or Not

- In the synchronous system model:
 - Consensus is solvable
- In the asynchronous system model:
 - Consensus is impossible to solve
 - Whatever protocol/algorithm you suggest, there is always a worst-case possible execution (with failures and message delays) that prevents the system from reaching consensus
 - Powerful result (FLP proof)
 - Subsequently, safe or probabilistic solutions have become quite popular to consensus or related problems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, let us see the consensus problem in this the 2 models scenarios. So, if the model is synchronous distributed system, then consensus is solvable that is being proved and we will see in this part of the lecture.

However, in asynchronous system model the consensus is impossible to solve, that is also being proved. So, that means, that whatever protocol algorithms, you have there is always a worst possible execution scenario with the failures and the message delays, that will prevent the system from reaching the consensus. That is why the consensus is impossible to solve. And this was proved in FLP a famous proof of this FLP we will see has proved that consensus in a asynchronous system is impossible.

So, using this particular results safe or probabilistic solutions to this particular problem have become quite popular to the consensus problem, because it has a lot of use.

(Refer Slide Time: 14:53)

FLP

- One of the most important results in distributed systems theory was published in April 1985 by Fischer, Lynch and Patterson (FLP)
- Their short paper 'Impossibility of Distributed Consensus with One Faulty Process', which won the Dijkstra award, placed an upper bound on what it is possible to achieve with distributed processes in an asynchronous environment.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

We will see all these. Let us see what FLP is, it is one of the most important results in a distributed system theory, published in April 1985 by Fisher Lynch and Patterson. That is called FLP problem. So, this particular problem, you can refer to that paper which is called as impossibility of distributed, consensus with one faulty process; which won the Dijkstra award, and has shown an upper bound on what it is possible to achieve with the distributed processes in asynchronous environment that we will discuss.

(Refer Slide Time: 15:40)

Let's Try to Solve Consensus!

System model (Assumptions):

- **Synchronous system:** bounds on
 - Message delays
 - Upper bound on clock drift rates
 - Max time for each process stepe.g., multiprocessor (common clock across processors)
- Processes can fail by stopping (crash-stop or crash failures)

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, with this particular background that, the system model plays a important role whether you can solve the consensus or not. So, we have to study the algorithms accordingly. So, we will first assume the system model to be the synchronous system model, and also we will assume that there exist a bound on the message delays. And they are exist an upper bound on clock drift rates. And there also exist the bound on each processes steps to complete in the model. And that is called synchronous system model that we will assume. We will also assume a failure model which is called a crash fault or a crash failure. This model assumes that the process when fails they will stop that is called crash fault model of failure.

So, with these 2 assumptions that we will assume a synchronous system model and a crash fault system model let us see how we are going to solve the consensus in this model.

(Refer Slide Time: 17:02)

Consensus in Synchronous Systems

- For a system with at most f processes crashing
 - All processes are synchronized and operate in "rounds" of time.
 - The algorithm proceeds in $f+1$ rounds (with timeout), using reliable communication to all members
 - Values^r_i : the set of proposed values known to p_i at the beginning of round r .

Cloud Computing and Distributed Systems
Consensus in Cloud Computing & Paxos

So, the consensus in this particular model, we are assuming that this system can have at most f processes crashing. So, there is a limit that is called f processes which can crash. All the processes are synchronized operates in the rounds of time. By round we mean that if the messages send it will be delivered when before the next round starts.

So, the algorithm proceeds in $f + 1$ rounds using a reliable communication to all the members. Now here in this round every round we will, every process will exchange the values with other processes which is denoted as values of process i at the round r . So, the

set of proposed values known to a process i at the beginning of round r is denoted by this particular variable and this variable will be communicated through the multicast among the group who are participating in this consensus problem.

(Refer Slide Time: 18:26)

Consensus in Synchronous System

Possible to achieve?

```

- For a system with at most  $f$  processes crashing
  - All processes are synchronized and operate in "rounds" of time
  - the algorithm proceeds in  $f+1$  rounds (with timeout), using reliable communication to all members.
  - 'Values $^r_i$ ', the set of proposed values known to  $p_i$  at the beginning of round  $r$ .
  /- Initially  $Values^0_i = \emptyset$ ;  $Values^1_i = \{v_i\}$ 
  /- for round = 1 to  $f+1$  do
    |   multicast ( $Values^{r-1}_i$  ←  $Values^r_i$ ) // iterate through processes, send each a message
    |    $Values^{r+1}_i \leftarrow Values^r_i$ 
    |   for each  $V_j$  received
    |      $Values^{r+1}_i = Values^{r+1}_i \cup V_j$ 
    |   end
    |   and
    |    $d_i = \min(Values^{f+1}_i)$  // consistent minimum based on say, id (not minimum value)
  
```

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, let us see the algorithm for this. So, again these particular values which are known to the process i at the beginning of the round r is to be communicate (Refer Time: 18:43). So, let us see that initialize this particular value at the 0th round at the beginning of the 0th round as empty. And for the round number one the process i will have the value V_i . So, let us start for round 1 to round $f + 1$. Because this particular since we have assumes that there are at most f different processes which can crash. So, algorithm has to take $f+1$ rounds to converge into the consensus problem.

So, let us understand the round number 1. So, round number 1 the process i will have it is value V_i and it will be multi casted. So, this particular values will be received and will be sent to all the processes and it will also receive the same values in the in that particular round 1 from the other processes let us say V_j . So, the values which is received out of that V_j will also be taken into that particular set of values, which will be collected at the end of round r that is here in this case.

So, when $f + 1$ rounds finishes, then these set of values which is received by a process i it will take the minimum of all the values. So, every process like V_i , they will take the

minimum, and this particular protocol guarantees that all the processes will have the same value that is d at the end of $f + 1$ rounds.

(Refer Slide Time: 20:54)

Why does the Algorithm work?

- After $f+1$ rounds, all non-faulty processes would have received the same set of values. Proof by contradiction.
- Assume that two non-faulty processes, say p_i and p_j , differ in their final set of values (i.e., after $f+1$ rounds).
- Assume that p_i possesses a value v that p_j does not possess. (L1 L2 L3)
 - p_i must have received v in the very last round. (f+1 round)
 - Else, p_i would have sent v to p_j in that last round.
 - So, in the last round: a third process, p_k , must have sent v to p_i , but then crashed before sending v to p_j .
 - Similarly, a fourth process sending v in the last-but-one round must have crashed; otherwise, both p_i and p_j should have received v .
 - Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.
 - This means a total of $f+2$ crashes, while we have assumed at most f crashes can occur => contradiction.

$p_i \neq p_j = \text{Same value is reached}$

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Now, let us see why this algorithm will work to solve the consensus problem. So, after $f + 1$ rounds all the non-faulty processor or a processes would have received the same set of values. And we will see that particular proof by the contradiction.

So, let us assume that there exists 2 non faulty processes. Let us say p_i and p_j and they differ in their final set of values. Let us say that after $f + 1$ rounds that they have different values. Now assume that process p_i has the value v without loss of generality. And that p_j does not possess this value. It may have a different value, we can change it that p_j will have this value, but p_i does not have that we have already assumed that without loss of generality.

Now, if p_i must have received the v in the very last round, that is $f + 1$ th round, else p_i would have sent v to p_j in that last round. So, in this last round a third process p_k must have sent v to p_i , but then crashed before sending v to p_j . Similarly, a 4th process sending v in the last, but one round must have crashed otherwise both p_k and p_j should have received v . Proceeding in this way we infer that at least one unique crash in each of the preceding round. This means that a total $f+1$ crashes, which will prevent the value v to be reached at p_j . While we have already assumed that there can be at most f crashes.

Therefore, this is a contradiction and this contradiction will say that p_i and p_j should have received the same value and they will reach the consensus.

(Refer Slide Time: 23:33)

Consensus in an Asynchronous System

- **Impossible to achieve!**
- Proved in a now-famous result by **Fischer, Lynch and Patterson, 1983 (FLP)**
 - Stopped many distributed system designers dead in their tracks
 - A lot of claims of “reliability” vanished overnight.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

However, this consensus if you change the system model, asynchronous model it will become impossible to achieve. And we have already discussed that FLP that is Fischer Lynch and Patterson have already proved this particular theorem, that is called FLP theorem.

(Refer Slide Time: 23:58)

Consensus Problem

- Consensus **impossible** to solve in asynchronous systems (**FLP Proof**)
 - **Key to the Proof:** It is impossible to distinguish a failed process from one that is just very very slow. Hence the rest of the alive processes may stay forever when it comes to deciding.
- But Consensus important since it maps to many important distributed computing problems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, consensus problem we have seen that this consensus impossible to be solved in asynchronous system model. And that is given by FLP proof. We will not going to the proof, but let us see what is the key issue which led to this particular proof. Now it is impossible to distinguish a failed process from that is just very, very slow. Hence, the rest of the alive process may stay forever when it comes to the deciding. So, that becomes a proof; that means, a failure or a system becoming very slow it is very difficult to detect it. Quickly it may take very long time to conclude it, and this will lead to that impossibility result that is called FLP.

Now, since consensus is are important problem. Since it maps to many important distributed computing problem. Therefore, we will see that in this asynchronous system model how we are going to solve the consensus.

(Refer Slide Time: 25:26)

Paxos

Paxos algorithm:

- Most popular "consensus-solving" algorithm
- Does not solve consensus problem (which would be impossible, because we already proved that)
- But provides safety and eventual liveness.
- A lot of systems use it
 - Zookeeper (Yahoo!), Google Chubby, and many other companies

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, once a solution is called Paxos algorithm. This Paxos algorithm is the most popular consensus solving algorithm. In the asynchronous system model; however, we have already seen that consensus in the asynchronous is unsolvable using FLP theorem, but as far as the Paxos is concerned, Paxos provides or solves to some extent the consensus problem in the asynchronous system model. But it provides the safety and eventual liveness, it is not the guaranteed liveness, but it is an eventual liveness.

So, the Paxos algorithm solves the consensus using these 2 constraints safety and eventual liveness. A lot of systems which are already deployed by the industry like

Yahoo and Google, they are using this algorithm Paxos algorithm in the form of Zookeeper. And also the Google stack that is called Chubby and many other companies also use the Paxos. So, that is why the Paxos is the most popular in solving the consensus.

(Refer Slide Time: 27:02)

Paxos

- Paxos is invented by **Leslie Lamport**
- Paxos provides **safety** and **eventual liveness**
 - **Safety:** Consensus is not violated – Non-faulty process reach agreement with same value.
Handwritten note: $\pi_i = \sigma_{i-1}$
 - **Eventual Liveness:** If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached. But there is no guarantee.
- FLP result still applies: Paxos is not **guaranteed** to reach Consensus (ever, or within any bounded time)

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Paxos is invented by the Leslie Lamport. He is a famous scientist working in Microsoft. And he has the winner of Turing award and several awards for his contribution in the distributed systems. So, he is basically the key to invent this Paxos system. So, Paxos will provide the safety and eventual liveness. So, safety says that consensus is not violated. Meaning to say that, if 2 processes which are non-faulty, they arrive at the decision with the same value.

That is called safety in the consensus; that means, that non faulty processes reach agreement with the same value; that means, if π_i and π_j . So, at the end they may decide on the same value 0 or let us say 1, but not the scenario whether π_i will receive 0 or π_j will receive 1. So, both of them either they decide on 0 or all of them decides on 1.

That means, the consensus is not violated and this ensures that Paxos supports or provides the safety or guarantees the safety properties the other property which is called eventual liveness. So, safety says that everything good happens. And eventual liveness says that if the things go well sometime in the future; that means, the messages failures

etcetera, then there is a good chance that consensus will be reached, but it is not guaranteed in this particular situation. That is sometimes a situation may arise that this liveness that eventually everything good happens is not guaranteed by the Paxos.

So, the FLP is still applied why because it is not the liveness, but it is an eventual liveness. So, Paxos is not guaranteed to reach the consensus maybe sometimes within at particular time bound

(Refer Slide Time: 30:36)

Paxos Algorithm

- Paxos has **rounds**; each round has a unique ballot id
- Rounds are asynchronous
 - Time synchronization not required
 - If you're in round j and hear a message from round $j+1$; abort everything and move over to round $j+1$
 - Use timeouts; may be pessimistic
- Each round itself broken into phases (which are also asynchronous)
 - Phase 1: A leader is elected (**Election**)
 - Phase 2: Leader proposes a value; processes ack
 - Phase 3: Leader multicasts final value

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Let us see the Paxos algorithm. Paxos has divided the algorithm in and this operates in the rounds. So, the rounds are asynchronous, that is time synchronization is not required, but if you are in the round j and hear the messages from round $j+1$ then you abort everything and move over to the next to the round $j+1$. But that does not require the time synchronization to do this. Timeouts can be used but maybe pessimistic so, each round itself broken into the phases which are also asynchronous.

So, let us see what are the phases this round of Paxos algorithm follows. So, the first phase is called the election. That is a leader is elected, second phase is called that the leader proposes a value. And also processes the acknowledgement ack. Phase 3 says that the leader multicast the final values.

(Refer Slide Time: 31:57)

Phase 1: Election

- Potential leader chooses a unique ballot id, higher than seen anything so far
- Sends to all processes
- Processes wait, respond once to highest ballot id
 - If potential leader sees a higher ballot id, it can't be a leader
 - Paxos tolerant to multiple leaders, but we'll only discuss 1 leader case
 - Processes also log received ballot ID on disk
- If a process has in a previous round decided on a value v' , it includes value v' in its response
- If **majority (i.e., quorum)** respond OK then you are the leader
 - If no one has majority, start new round
- (If things go right) A round cannot have two leaders (why?)



Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, the potential leader chooses a unique ballot id; that means, the higher than anything seen so far is being chosen as the ballot ID. It sends it to all the processes. Processes waits and respond once to the highest seen ballot ID.

Now, if the potential leader sees a higher ballot ID then it cannot be the leader. Now the Paxos in some cases is tolerant to the multiple leaders, but let us without loss of generality let us discuss only one leader over here. So, the processes also log the received ballots on the disk. If a process has in the previous round decided a value v' , it include that value v' in its response. If the majority that is the quorum responded then, that particular process will become the leader if no one has the majority then it will start again the new round. So, the things go right the round cannot have the 2 leaders.

(Refer Slide Time: 33:27)

Phase 2: Proposal

- Leader sends proposed value v to all
 - use $v=v'$ if some process already decided in a previous round and sent you its decided value v'
 - If multiple such v' received, use latest one
- Recipient logs on disk; responds OK



Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, this is the phase one which is an election. The phase 2 which is called a proposal that is the leader sends the proposed value v to all, and v' is the previous round value then the same value can be can be send. So, if multiple such v primes are received then in that case use the latest one.

Recipients logs all these values on the disk and responds to the OK messages here. So, they will send these values. And they will respond with the OK values.

(Refer Slide Time: 34:24)

Phase 3: Decision

- If leader hears a **majority** of OKs, it lets everyone know of the decision
- Recipients receive decision, log it on disk



Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Third phase is called decision if the leader hears a majority of ok. It let us everyone know of the decision. So, recipients receive the decision log on the disk.

(Refer Slide Time: 34:36)

Which is the point of No-Return?

- If/when a majority of processes hear proposed value and accept it (i.e., are about to/have respond(ed) with an OK!)
- Processes **may not know it yet**, but a decision has been made for the group
 - Even leader does not know it yet
- What if leader fails after that?
 - Keep having rounds until some round completes

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Now, at what stage the agreement is reached? That means, if the majority of the process hears the proposed value and accepted it; that means that is being checked with the help of this messages. Then it is known that the agreement is released and the processes may not know it yet, but the decision have been made in the rounds of this particular protocol Paxos.

But the decision has been made by the group even the leader does not know this particular decision at this point of time, before the algorithm completes. Now the question is if the leader fails, then what will happen? So, it will keep on iterating the rounds until some round completes it.

(Refer Slide Time: 35:54)

Safety

- If some round has a majority (i.e., quorum) hearing proposed value v' and accepting it, then subsequently at each round either: 1) the round chooses v' as decision or 2) the round fails
- Proof:**
 - Potential leader waits for majority of OKs in Phase 1
 - At least one will contain v' (because two majorities or quorums always intersect)
 - It will choose to send out v' in Phase 2
- Success requires a majority, and any two majority sets intersect

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, let us see that this particular protocol Paxos achieves or guarantees the safety. Now if some round has the majority hearing the proposed value v' and accepting it then subsequently in each round either the round chooses the same values v' as the decision or the round fails.

The proof says that the potential leader wait for the majority of ok's in the phase 1. And at least one will contain v' . And because if there are more than one majorities, but every message has to send only 1 ok. So, it will choose to send out v' primes in the phase 2. Success requires a majority and any 2 majority sets the intersect.

(Refer Slide Time: 36:54)

What could go Wrong?

- Process fails
 - Majority does not include it
 - When process restarts, it uses log to retrieve a past decision (if any) and past-seen ballot IDs. Tries to know of past decisions.
- Leader fails
 - Start another round
- Messages dropped
 - If too flaky, just start another round
- Note that anyone can start a round any time
- Protocol may never end
 - Impossibility result not violated
 - If things go well sometime in the future, consensus reached

Value v ok?

Please elect me!

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, it will not validate the safety property; that means, even if the majority sets when we take the intersection. Even then we will find the values v' to be reached in the agreement.

So, there while the rounds and the phases operates in the Paxos. There may be many type of crashes or a problems happens and let us see how tolerant it is in this particular scenarios. For example, a process fails. So, majority does not include it, and when the process restart it uses the log to retrieve the past decisions and past seen ballot ID's and tries to know the past decisions. Similarly, if the leader fails then it will start another round, messages are dropped and in that case you have to just start another round.

So, anyone can start around at any time and the protocol may never end; that is, impossibility results are not violated. So, if the things go well, sometime in the future consensus is reached, but that is not guaranteed.

(Refer Slide Time: 38:29)

What could go Wrong?

- A lot more!
- This is a highly simplified view of Paxos.
- **Lamport's original paper:**

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament discussed above is the legislative progeny of its part-time legislators. The legislature instituted a consistent quota of the parliamentary record, despite more frequent losses from the chamber and the forgetfulness of their members. The Paxos parliament is presented hereinafter a new way of implementing the state-of-the-art approach to the design of distributed systems.

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

So, here we will see here the original Lamports paper on Paxos; which is having a title that part time parliament can see and study conclusion.

(Refer Slide Time: 38:40)

Conclusion

- Consensus is a very important problem
 - Equivalent to many important distributed computing problems that have to do with reliability
- Consensus is possible to solve in a synchronous system where message delays and processing delays are bounded
- Consensus is impossible to solve in an asynchronous system where these delays are unbounded
- **Paxos protocol:** widely used implementation of a safe, eventually-live consensus protocol for asynchronous systems
 - Paxos (or variants) used in Apache Zookeeper, Google's Chubby system, Active Disk Paxos, and many other cloud computing systems

Cloud Computing and Distributed Systems Consensus in Cloud Computing & Paxos

Consensus is very important problem. Equivalent to many important problems in the distributed systems, that have to deal with the reliability. Consensus is possible to be solved in a synchronous system where the message delays and the processing delays are bounded. Consensus problem is impossible to be solved in a model that is asynchronous model where these delays that is the messages and the processing are unbounded.

So, the Paxos is widely used protocol which will implement the safe and eventual live consensus protocol for asynchronous system, and is heavily used in very many industries systems by Yahoo; such as Apache Zookeeper and Google, Google's internal stack such that Chubby system and so on.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture-14
Byzantine Agreement

(Refer Slide Time: 00:15)

Preface

Content of this Lecture:

- In this lecture, we will discuss about '**Agreement Algorithms for Byzantine processes!**'
- This lecture first covers different forms of the '**consensus problem**' then gives an overview of what forms of consensus are solvable under different failure models and different assumptions on the synchrony/asynchrony.
- Also covers agreement in the category of:
(i) **Synchronous message-passing systems with failures** and
(ii) **Asynchronous message-passing systems with failures**.

Cloud Computing and Distributed Systems Byzantine Agreement

Byzantine Agreement: Pre phase content of this lecture, we will discuss about agreement algorithms for byzantine processes. We will cover different forms of consensus problem. We will give the overview what forms of consensus are solvable under different failure model and under different assumptions on synchrony versus asynchrony. Also we will cover the range of agreement problems, such as in the category of synchronous message passing systems and asynchronous message passing systems with the failure models.

(Refer Slide Time: 01:02)

Introduction

- **Agreement among the processes** in a distributed system is a fundamental requirement for a wide range of applications.
- Many forms of coordination require the processes to exchange information to negotiate with one another and eventually reach a common understanding or agreement, before taking application-specific actions.
- A classical example is that of the commit decision in **database systems**, wherein the processes collectively decide whether to commit or abort a transaction that they participate in.
- In this lecture, we will study the feasibility of designing algorithms to reach agreement under various system models and failure models, and, where possible, examine some representative algorithms to reach agreement.

Cloud Computing and Distributed Systems Byzantine Agreement

Introduction, the agreement among the processes in a distributed system is one of the fundamental requirements for wide range of applications. Many forms of coordination requires the processes to exchange the information to negotiate with one another and eventually reach a common understanding or the agreement, before taking the application specific action.

The classical example is that of commit decision in any database system. There processes collectively decide whether to commit or abort a transaction which they are participating in. This lecture we will study the feasibility of designing the algorithm to reach to a consensus under various system models and fault models. And examine the representative algorithms for ensuring the agreement in the system.

(Refer Slide Time: 02:11)

Classification of Faults: Overview

- Based on components that failed
 - Program / process
 - Processor / machine
 - Link
 - Storage
- Based on behavior of faulty component
 - Crash – just halts
 - Fail stop – crash with additional conditions
 - Omission – fails to perform some steps
 - Byzantine – behaves arbitrarily
 - Timing – violates timing constraints

Cloud Computing and Distributed Systems Byzantine Agreement

Before that, let us see some of the basics in contrast in relation to the faults. Classification of the faults: Based on different components, which are failed such as program or a process can fail processor or machine can fail the link that is the network link can also fail or the storage can fail. There are different possibilities of failure. Now, based on these failure possibilities, there are different fault model which are now evolved in this particular study of subsystems. So, the first model is called crash word or it is also called as fail-stop. So, that particular model will when suffers from the failure that is called a crash or a fault, it will stop functioning.

This is one of the most simplest model fault model which can be assumed. We have seen the algorithms in the previous lessons. The, another type of model fault model is called omission fault. Here it fails to perform some steps, the next model which is very general kind of model that is called byzantine fault model. It behaves arbitrarily; that means, its behavior is not predictable, sometimes it may omit the messages. Otherwise, it is functioning ok, sometimes it is meant malfunctioning or maliciously functioning. This is all term in a very general fault model called byzantine fault model.

(Refer Slide Time: 04:21)

Classification of Tolerance: Overview

- **Types of tolerance:**

- **Masking** – System always behaves as per specifications even in presence of faults.
- **Non-masking** – System may violate specifications in presence of faults. Should at least behave in a well-defined manner.

- **Fault tolerant system should specify:**

- Class of faults tolerated
- What tolerance is given from each class

Cloud Computing and Distributed Systems

Byzantine Agreement

Timing which violates the timing constraint, is also a type of fault model. In this discussion we will consider the byzantine fault model, and we will see the agreement: what are the algorithms to deal with that. Now, there are terms which are called tolerance.

So, masking means system always behaves as per the specification even in the presence of faults. Non masking is system may violate the specification in the presence of fault, should at least behave in a well-defined manner. So, the fault tolerance system should specify what class of faults which are tolerated and what tolerance is given for each class.

(Refer Slide Time: 05:05)

Measuring Reliability and Performance

- **Distributed systems:**
 - Improve performance
 - Improve reliability
- Or do they? Need to measure to know.
- Need a vocabulary

Cloud Computing and Distributed Systems Byzantine Agreement

Furthermore, to measure the reliability and performance in a distributed system; that means, there are various measures for performance and reliability, which are to be ensured while giving the services. So, we will understand we will see how to measure them that is performance and the reliability.

(Refer Slide Time: 05:30)

SLIs, SLOs, SLAs, TLAs

- **SLI = Service Level Indicator**
⇒ What you are measuring
- **SLO = Service Level Objective**
⇒ How good should it be?
- **SLA = Service Level Agreement**
⇒ SLO + consequences
- **TLA = Three Letter Acronym**

Cloud Computing and Distributed Systems Byzantine Agreement

So, there are certain terms which are used by the industry when they provide the services in the form of a cloud or in a distributed system. Let us quickly review them. SLI, that is service level indicator, will indicate what are you measuring. Service level, objective

how good should it be, and service level agreement that is service level objective, plus what are the consequences. So, this is all specified in a 3 letter word that is called TLA.

(Refer Slide Time: 06:10)

Why study SLIs, SLOs, and SLAs?

- If you measure it, you can improve it
- Learn what matters
 - Don't waste time on things that don't matter!
- Reliability promises are part of business

Cloud Computing and Distributed Systems Byzantine Agreement

Now, why you want to measure it? Because, you are purchasing it as a service these guarantees and the service provider or the vendor also has to put various amount of resources. That means, he has to be at the cost to support this kind of guarantees in the hope in the terms of reliability. So, reliability promises are the part of the business. So, let us discuss it.

(Refer Slide Time: 06:38)

Reading an SLA

- **"I promise 99% uptime"**
- How often do you check if your system is up?
 - Sampling frequency
- What does it mean to be "up"?
 - Domain of responsibility
- Over what time interval do you promise 99% uptime?
 - Measurement interval

Cloud Computing and Distributed Systems Byzantine Agreement

So, if the service level agreements says that I promise 90, 99 % up time. What does this means? That means, to analyze this meaning is that you have to see that how often do you check if the system is up that is sampling frequency. What does it means to be up? That is domain of the responsibility over what time interval do you promise 99 % uptime that is measurement interval.

(Refer Slide Time: 07:14)

How many nines?		
Nines	Uptime	Downtime/month
1	90%	3 days
2	99%	7 hours
3	99.9%	43 minutes
4	99.99%	4 minutes
5	99.999%	25 seconds (5m/year)

Cloud Computing and Distributed Systems Byzantine Agreement

In all this can be measured or this can be specified how many 9's are there; that means, if there is only one 9. That is, if the uptime is 90 %; that means, in a month that is in a monthly billing cycle 90 % uptime comes out to be 3 days; that means, in a one month 3 days can be the down time. When it says that 2 9s that is 99 % in a month that computes it to be 7 hours in a month. The system can be down before it can be repaired and kept up. 99.9, that is 3 9's works out to be 43 minutes in a month; that means, within 43 minutes the system is to be repaired and made available to the services.

When it is 4 9's that is 99.99 that is works out to be 4 minutes; it is very quickly it has to be ensure it is maintenance. Now, when it comes out to be 5 9's that is 99.999 that is not in the month, that is in the year or in a month it comes out to be 25 seconds. That is 5 minutes per year that is a goal time. So, that kind of reliability if it is purchased by the service provider when service provider has to ensure; that means, it will provide this kind of uptime.

(Refer Slide Time: 08:55)

Cloud VM providers

- Consider Microsoft Azure, Amazon EC2, Google GCE (Google Compute Engine)
- Promise 99.95% uptime (22 minutes downtime/month)
 - Better than my net connection... right?
- 1-minute sampling frequency
 - GCE doesn't count <5 minute outages



Cloud Computing and Distributed Systems Byzantine Agreement

Cloud VM providers, blue chip cloud companies like, Microsoft Azure, Amazon, EC2, and Google compute engine, they all promise 99.9 5 that is 3 9's of up time that worked out to be 22 minutes' downtime per month. So, let us analyze it in terms of sampling frequency. They say that 1-minute sampling frequency; that means, in a 1 minute you have 60 seconds; that means, 59 seconds your system can be down. It can be up for one second and so on. So, there may be different tricks and place which need to be understood in this particular reliability game.

(Refer Slide Time: 09:53)

What does the SLA imply for provider?

- 99.95% (or 22 minutes/month downtime) means either:
 - They rarely expect their hardware or software to fail
 - When it fails they think they can fix it quickly



Cloud Computing and Distributed Systems Byzantine Agreement

But at the end of the day to support 99.95 % of time or; that means, within 22 minutes of downtime per month, the system is to be repaired and made available to the services. That is there rarely expect their hardware and software to fail, because if they fail then manually it is; that means, every person is sitting next to the computer, and these particular attending such system around the clock requires huge investment in terms of manpower and so on. That has to be fixed up quickly.

(Refer Slide Time: 10:43)

What does the SLA imply for you?

- **SLA requires you to have:**
 - Multiple VMs
 - Over multiple failure domains
 - Automatic failover
 - Monitoring
 - Tolerance of planned outages
 - Automatic machine provisioning (GCE)

Cloud Computing and Distributed Systems Byzantine Agreement

Now, as far as ensuring through service level agreement, the reliability means that the companies are required to support the services through multiple VMs; that means, whenever there is a failure in one site it can respond another VM at different site that the downtime can be reduced and the services continue to be service from the other new machine.

Then SLA also requires that over multiple failure domains, it ensures the reliability. And also ensures that automatic failover can be provisioned, continuous monitoring also is required and tolerance of the planned outages and automatic machine provisioning. So, all these things are required to be automated, then only that kind of service level agreement can be ensured.

(Refer Slide Time: 11:50)

Assumptions

1. Failure models
2. Synchronous/ Asynchronous communication
3. Network connectivity
4. Sender identification
5. Channel reliability
6. Authenticated vs. Non-authenticated messages
7. Agreement variable

Cloud Computing and Distributed Systems Byzantine Agreement

So, let us assume for our byzantine fault or agreement algorithms, we will see some of the models. We will assume a fault model or a failure model we will also assume the synchronous or asynchronous communication. Network connectivity we will also assume how the sender is to be identified, then channel reliability authentication versus unauthentication message and agreement variable.

(Refer Slide Time: 12:22)

1) Failure models

- A failure model specifies the manner in which the component(s) of the system may fail.
- There exists a rich class of **well-studied failure models**. The various process failure models are: (i) Fail-stop, (ii) Crash, (iii) Receive omission, (iv) Send omission, (v) General omission, and (vi) Byzantine or malicious failures
- Among the n processes in the system, at most f processes can be faulty. A faulty process can behave in any manner allowed by the failure model assumed.

Cloud Computing and Distributed Systems Byzantine Agreement

So, let us start one by one. So, failure model failure model specifies the manner in which the components of the system may fail. And the literature provides a well-defined failure

models which are fail-stop, crash, fault receive omission, send omission, general omission and byzantine or a malicious faults. So, in most of the study we assume that among n processes in the system at most f can be faulty. So, n f there are 2 parameters for such study in the system. So, a faulty process can behave in any manner allowed by the different fault models and such assumptions can be made.

(Refer Slide Time: 13:15)

Type of Process Failure Models

- i. **Fail-stop:** In this model, a properly functioning process may fail by stopping execution from some instant thenceforth. Additionally, other processes can learn that the process has failed.
- ii. **Crash:** In this model, a properly functioning process may fail by stopping to function from any instance thenceforth. Unlike the fail-stop model, other processes do not learn of this crash.
- iii. **Receive omission:** A properly functioning process may fail by intermittently receiving only some of the messages sent to it, or by crashing.
- iv. **Send omission:** A properly functioning process may fail by intermittently sending only some of the messages it is supposed to send, or by crashing.

Cloud Computing and Distributed Systems Byzantine Agreement

For example, the failure model or a fault model are of different types. The first one is called fail-stop in this model properly functioning process may fail by stopping the execution. From some instant henceforth additionally other processes can learn that the process has failed. Crash in this model a properly functioning process my fail by stopping to function from any instance henceforth.

Unlike, fail-stop other processes do not learn of this particular crash. Receive omission a properly functioning process may fail by intermittently receiving only some of the messages sent to it or by crashing. Send omission or properly functioning process my fail by intermittently sending only some of the messages it is supposed to send by crashing.

(Refer Slide Time: 14:13)

Contd...

- v. **General omission:** A properly functioning process may fail by exhibiting either or both of send omission and receive omission failures.
- vi. **Byzantine or malicious failure:** In this model, a process may exhibit any arbitrary behavior and no authentication techniques are applicable to verify any claims made.

Cloud Computing and Distributed Systems

Byzantine Agreement

General omission or properly functioning process may fail by exhibiting either or both of send omission and receive omission failures. Byzantine or malicious failure in this model a process may exhibit any arbitrary behavior, and no authentication techniques are applicable to verify any claims made. This is called byzantine or a malicious fault this is most general kind of fault which may exhibit any arbitrary behavior due to the malfunction or due to the malicious activities. Hence, it is called byzantine or a malicious failure.

(Refer Slide Time: 14:57)

2) Synchronous/Asynchronous Computation

-Synchronous Computation:

- i. Processes run in lock step manner [Process receives a message sent to it earlier, performs computation and sends a message to other process.]
- ii. Step of Synchronous computation is called '*round*'

-Asynchronous Computation:

- i. Computation does not proceed in lock step.
- ii. Process can send receive messages and perform computation at any time.

Cloud Computing and Distributed Systems

Byzantine Agreement

Computations are concerned there are two different models of computations which can be assumed in different agreement protocols. Synchronous computation, in this model the processes runs in a lock step manner; that is, the process receives a message performs the computation and send the message to the other process. So, the step of the synchronous computation is called a round. Whereas, asynchronous computation the computation does not proceed in a locked step the process can send receive message and perform the computation at any point of time.

(Refer Slide Time: 15:39)

3) Network connectivity
The system has **full logical connectivity**, i.e., each process can communicate with any other by direct message passing.

4) Sender identification
A process that receives a message always **knows the identity** of the sender process.

5) Channel reliability
The **channels are reliable**, and only the processes may fail (under one of various failure models). This is a simplifying assumption in our study.

Cloud Computing and Distributed Systems Byzantine Agreement

Third one is called network connectivity. The system has full logical connectivity that is assumed in that model. So, each process can communicate with any other by direct message passing. Sender identification a process that receives the message always knows the identity of the sender process. Channel reliability the channels are reliable. And only the processes may fail. This is to simplify and perform concentrated study of the subsystems.

(Refer Slide Time: 16:23)

6) Authenticated vs. Non-authenticated messages

- In this study, we will be dealing only with unauthenticated messages.
- With unauthenticated messages, when a faulty process relays a message to other processes, (i) it can forge the message and claim that it was received from another process, and (ii) it can also tamper with the contents of a received message before relaying it.
- Using authentication via techniques such as digital signatures, it is easier to solve the agreement problem because, if some process forges a message or tampers with the contents of a received message before relaying it, the recipient can detect the forgery or tampering.

Cloud Computing and Distributed Systems Byzantine Agreement

Authenticated versus non authenticated messages in this particular study we will be dealing only with unauthenticated messages. With unauthenticated messages when faulty process release a message to other process, it can forge the message and claim that it was received from another process.

It can also tamper with the contents of the received message before relaying it. Now, using authentication techniques such as digital signature it is easier to solve the agreement problem because if some process forges a message or tampers with the content of the received message before relaying it the receiver can detected.

(Refer Slide Time: 17:05)

7) Agreement variable

- The agreement variable **may be boolean or multivalued**, and need not be an integer.
- When studying some of the more complex algorithms, we will use a boolean variable.
- This simplifying assumption does not affect the results for other data types, but helps in the abstraction while presenting the algorithms.

Cloud Computing and Distributed Systems

Byzantine Agreement

The agreement variable is another parameter. So, agreement variable maybe a Boolean or multivalued and need not be an integer. So, when studying some more complex algorithms we will only consider the Boolean variable for simplicity. This simplifying assumption does not affect the result of other data types, but helps in the abstraction while presenting the algorithms.

(Refer Slide Time: 17:27)

Performance Aspects of Agreement Protocols

Few Performance Metrics are as follows:

(i) **Time:** No of rounds needed to reach an agreement

(ii) **Message Traffic:** Number of messages exchanged to reach an agreement.

(iii) **Storage Overhead:** Amount of information that needs to be stored at processors during execution of the protocol.

Cloud Computing and Distributed Systems

Byzantine Agreement

Performance aspects of the agreement algorithms; so, we will see some of the performance metrics; such as time which is measured in terms of rounds needed to reach

an agreement or terminate the agreement protocol. Second performance metrics is message traffic or that is the number of message exchanged in the algorithm to reach an agreement. Third one is the storage overhead that is the amount of storage required by the processor during the execution of the algorithm.

(Refer Slide Time: 18:15)

Problem Specifications

1. Byzantine Agreement Problem (single source has an initial value)

Agreement: All non-faulty processes must agree on the same value.

Validity: If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.

Termination: Each non-faulty process must eventually decide on a value.

2. Consensus Problem (all processes have an initial value)

Agreement: All non-faulty processes must agree on the same (single) value.

Validity: If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.

Termination: Each non-faulty process must eventually decide on a value.

Cloud Computing and Distributed Systems Byzantine Agreement

Let us see different variants of this agreement problem. The first problem is called Byzantine Agreement problem, which is having a single source with the initial value. So, it has 3 different conditions within it. Agreement says that all non-faulty processes must agree on the same value, that is the value of a source. Validity says that if a source process is non faulty, then the agreed upon value by all non-faulty processes must be the same as the initial value of the source. Termination says that each non faulty must eventually decide on the value.

Second barrier is called consensus problem. That is here all processes have an initial value, then it is called a consensus problem. Agreement says that all non-faulty processes must agree on the same single value; that means, they agree on only one value like byzantine. Validity that is if all the non-faulty processes have the same initial value, then the agreed upon value by all non-faulty processes, must be this must be that same value termination is non faulty must eventually decide on a value.

(Refer Slide Time: 19:47)

Contd...

3. Interactive Consistency Problem (all processes have an initial value)

Agreement: All non-faulty processes must agree on the same array of values $A[v_1 \dots v_n]$.

Validity: If process i is non-faulty and its initial value is v_i , then all non-faulty processes agree on v_i as the i th element of the array A . If process j is faulty, then the non-faulty processes can agree on any value for $A[j]$.

Termination: Each non-faulty process must eventually decide on the array A .

Third variant is called interactive consistency problem. Here all processes have an initial value. Agreement on non-faulty processes must agree on the same array of values. That means, if there are n different processes for there will be an array A , and there will be a particular value for each process.

So, this will be an array or a vector A of n different elements. And the vector or the array will be same in the agreement interactive consistency problem. Validity, that is if a process i is non faulty. And it is initial value is v_i in that vector A . Then all non-faulty processes agree on v_i as i th element of the array. If the process j is faulty, then the non-faulty processes can agree on any value for A of j . Termination that is each non faulty processes must eventually decide on an array A .

(Refer Slide Time: 21:01)

Equivalence of the Problems

- The three problems defined above are equivalent in the sense that a solution to any one of them can be used as a solution to the other two problems. This equivalence can be shown using a reduction of each problem to the other two problems.
- If problem A is reduced to problem B, then a solution to problem B can be used as a solution to problem A in conjunction with the reduction.
- Formally, the **difference between the agreement problem and the consensus problem** is that, in the agreement problem, a single process has the initial value, whereas in the consensus problem, all processes have an initial value.
- However, the two terms are used interchangeably in much of the literature and hence we shall also use the terms interchangeably.

Cloud Computing and Distributed Systems

Byzantine Agreement

Now, let us see there are three different variants there equivalence. So, the three problems defined above are equal in the sense that a solution to any one of them can be used as solution to the other two problem. This is this equivalence can be shown using a reduction. So, if a problem A is reduced to a problem B then the solution to a problem B can be used as a solution to a problem A in conjunction with the reduction.

So, let us see the difference between agreement and the consensus problem, is that in the agreement problem A single process has the initial value; where is an consensus problem all the processes have their own initial values; however, the two terms are used interchangeably in much literature and hence we shall use these terms interchangeably.

(Refer Slide Time: 21:58)

Overview of Results

- **Table 10.1** gives an overview of the results and lower bounds on solving the consensus problem under different assumptions.
- It is worth understanding the relation between the consensus problem and the problem of attaining common knowledge of the agreement value. For the “**no failure**” case, consensus is attainable.
- Further, in a synchronous system, common knowledge of the consensus value is also attainable, whereas in the asynchronous case, concurrent common knowledge of the consensus value is attainable.

Cloud Computing and Distributed Systems Byzantine Agreement

Table shows an overview of the result at the lower bound in solving the consensus problem under different assumptions. For that the synchronous system, the common knowledge of the consensus value is also attainable; where asynchronous system concurrent common knowledge of the consensus value is attainable.

(Refer Slide Time: 22:22)

Overview of Results

Failure mode	Synchronous system (message-passing and shared memory)	Asynchronous system (message-passing and shared memory)
No failure	agreement attainable; common knowledge also attainable	agreement attainable; concurrent common knowledge attainable
Crash failure	agreement attainable $f < n$ processes $\Omega(f + 1)$ rounds	agreement not attainable
Byzantine failure	agreement attainable $f \leq \lfloor (n - 1)/3 \rfloor$ Byzantine processes $\Omega(f + 1)$ rounds	agreement not attainable

Table 10.1: Overview of results on agreement. f denotes number of failure-prone processes. n is the total number of processes.

In a failure-free system, consensus can be attained in a straightforward manner

Cloud Computing and Distributed Systems Byzantine Agreement

So, this particular table summarizes, it we will discuss each and every part in more details.

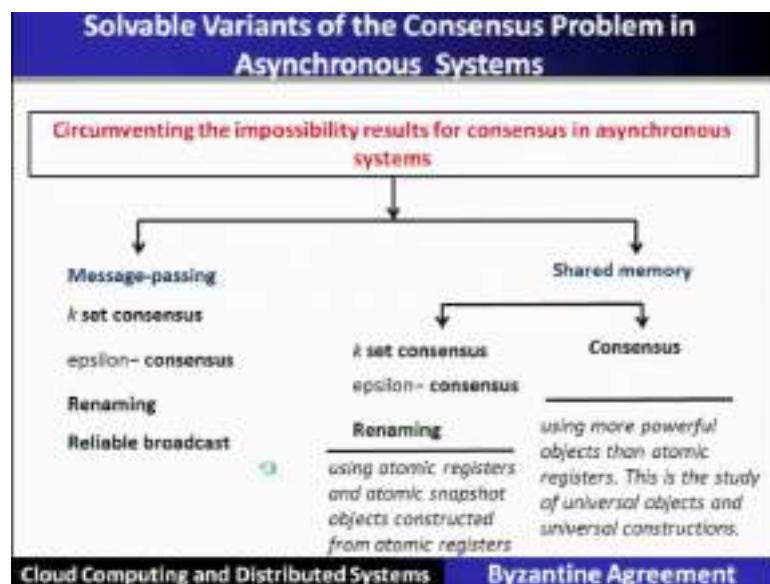
(Refer Slide Time: 22:30)

Contd...

- Consensus is not solvable in asynchronous systems even if one process can fail by crashing.
- Figure 10.1 shows further how asynchronous message-passing systems and shared memory systems deal with trying to solve consensus.

Cloud Computing and Distributed Systems Byzantine Agreement

(Refer Slide Time: 27:31)



Now, there are some solvable variants of the consensus problem. It is shown that the consensus problem in asynchronous system is impossible to be solved. Therefore, we will consider the synchronous system, hence circumventing the impossibility result for consensus asynchronous system, can be classified into different ways message passing and shared memory. Message passing is our point of discussion.

(Refer Slide Time: 23:07)

Weaker Consensus Problems in Asynchronous System	
Consensus Problem	Description
Terminating reliable broadcast	It states that a correct process always gets a message even if the sender crashes while sending. If the sender crashes while sending the message, the message may be a null message but it must be delivered to each correct process.
k-set consensus	It is solvable as long as the number of crash failures f is less than the parameter k . The parameter k indicates that the non-faulty processes agree on different values, as long as the size of the set of values agreed upon is bounded by k .
Approximate agreement	Like k-set consensus, approximate agreement also assumes the consensus value is from a multi-valued domain. However, rather than restricting the set of consensus values to a set of size k , ϵ -approximate agreement requires that the agreed upon values by the non-faulty processes be within ϵ of each other.
Renaming problem	It requires the processes to agree on necessarily distinct values.
Reliable broadcast	A weaker version of reliable terminating broadcast(RTB), namely reliable broadcast, in which the termination condition is dropped, is solvable under crash failures.

Cloud Computing and Distributed Systems

Byzantine Agreement

There are some weaker consensus problems in asynchronous system, that we will discuss in the end.

(Refer Slide Time: 23:13)

Contd...

- To circumvent the impossibility result, weaker variants of the consensus problem are defined in **Table 10.2**.
- The overheads given in this table are for the algorithms described.

Cloud Computing and Distributed Systems

Byzantine Agreement

(Refer Slide Time: 23:14)

Solvable Variants	Failure model and overhead	Definition
Reliable broadcast	crash failures, $n > f$ (MP)	Validity, Agreement, Integrity conditions
k-set consensus	crash failures, $f < k < n$ (MP and SM)	size of the set of values agreed upon must be less than k
ϵ -agreement	crash failures $n \geq 3f + 1$ (MP)	values agreed upon are within ϵ of each other
Renaming	up to f fail-stop processes, $n \geq 2f + 1$ (MP) Crash failures $f \leq n - 1$ (SM)	select a unique name from a set of names

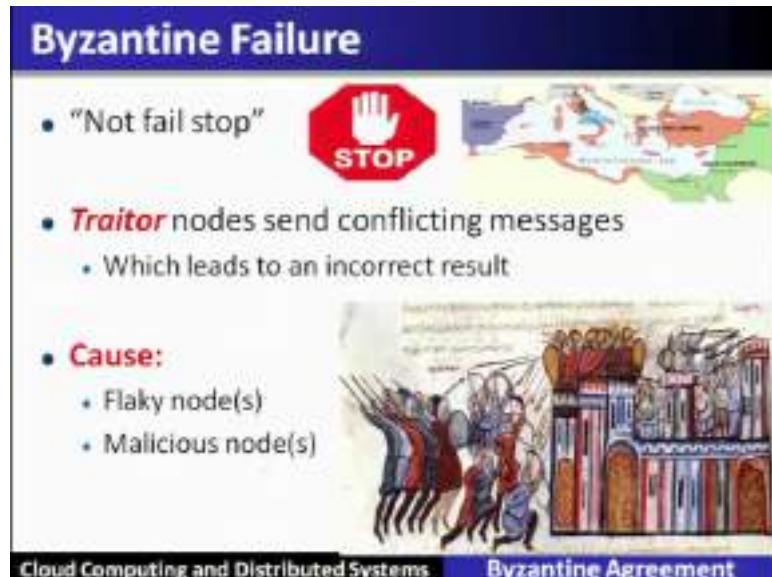
Table 10.2: Some solvable variants of the agreement problem in asynchronous system. The overhead bounds are for the given algorithms, and not necessarily tight bounds for the problem.
Here MP- Message Passing, SM- Shared Memory

Cloud Computing and Distributed Systems

Byzantine Agreement

So, this is also summarized some solvable problem of consensus in asynchronous system. Now, agreement in a synchronous message passing system.

(Refer Slide Time: 23:24)



Let us see: what is the byzantine failure. It is not a fail-stop failure. Why because it is having an arbitrary fault model or a malicious faults can also be categorized as byzantine failure. The name byzantine fault model is derived out of a particular story in which there is a different army located around a particular mountain.

They cannot see each other, but they can communicate by sending the messenger physically from 1 to 2 another group and their generals can now decide whether to attack on another army. So, it may be possible that these generals some are creators, they may send the confusing messages. Hence, maliciously they are sending messages and this model is modeled as the byzantine failure. And hence the name is basically given up from that location which is called byzantine. The causes of these failures are in our distributed system or in a cloud computing system is due to the flaky nodes or the malicious nodes.

(Refer Slide Time: 25:08)

Why study Byzantine failure?

- **Extreme fault tolerance:**
 - Bitcoin
 - Boeing 777 & 787 flight controls
- **Solving this problem is fun!**
 - This reason has really driven a lot of research, since at least the 1980's



Cloud Computing and Distributed Systems Byzantine Agreement

Now, the question is why we are considering this kind of fault model. Byzantine failures are most general kind of failure. So, the systems has to be build let us see the application, that is an extreme fault tolerance which is required in the case of bitcoin where the faults or this kind of failure or the behavior which is called byzantine failure is suspected from every agent in the bitcoin. Therefore, a byzantine fault model can be assumed while designing such system called bitcoin.

Another extreme fault tolerance which is required in the aircraft design that is in the Boeing planes at their flight control. So, that for any kind of failure the flight control systems are quite fault tolerant. So, this byzantine fault will give the extreme fault tolerance and these two examples which we have shown where such kind of fault model can be used to design the systems. Another reason of studying this byzantine fault is to

understand the research involved to deal with this kind of fault model and yet how they will be good get be designed.

(Refer Slide Time: 26:42)

What assumptions are you making?

- Can all nodes see all message? Some? None?
- Do nodes fail? How about the network?
- Finite computation?
- Static or dynamic adversary?
- Bounded communication time?
- Fully connected network?
- Randomized algorithms?
- Quantum or binary computers?

Cloud Computing and Distributed Systems Byzantine Agreement

There are different assumptions while discussing this kind of fault model, which are necessary to be understood let us understand this problem in more details.

(Refer Slide Time: 27:02)

Consensus: The Two Generals Problem



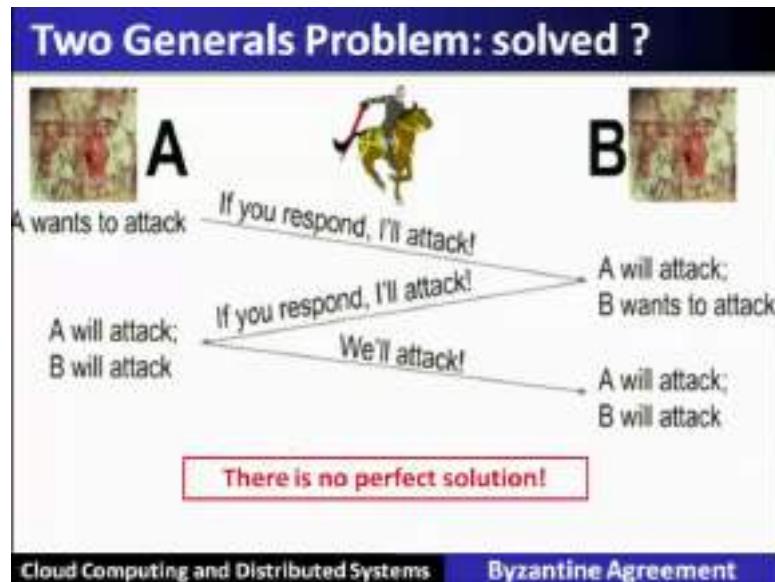
Two armies, A and B in separate valleys. C
Want to attack third army, C, in valley between them.
Must decide: attack tomorrow or not?
If they both attack: victory!
If neither attack: survival!
If just one attacks: defeat!
All messages sent by horse -- "through enemy territory".
Each messenger may or may not make it through.

Cloud Computing and Distributed Systems Byzantine Agreement

Now, agreement between two generals; let us say A and B which cannot directly communicate they can communicate through a messenger, which is shown here, who can go from who can take the message of A, and give it to the message B. Let us say that A

is communicating to B to decide if both of them can attack together, because alone they cannot defeat the enemy that is C. So, they have to exchange the messages in this regard. So, A is sending a message that if they attack if they both attack then it will be a victory. And the other one is replying back with some message and again wait for the response of A.

(Refer Slide Time: 28:19)



So, that means, a will send if you respond I will attack. This message will go from A to B and wait for the response of B. Now B will also response that if you respond I will attack and then again wait for a response. This message keeps on moving between A and B, and finally, they have to wait for the other sites perfect answer and never reaches to a consensus hence there is no perfect solution that is there is no consensus in this manner.

(Refer Slide Time: 29:00)

Byzantine Generals

- **The Byzantine Generals Problem**, Leslie Lamport, Robert Shostack and Marshall Peace. ACM TOPLAS 4.3, 1982.

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, AND MARSHALL PEASE
SRI International

Answers:

- How many byzantine node failures can a system survive?
- How might you build such a system?

Doesn't answer:

- Is it worth doing at all?

Cloud Computing and Distributed Systems Byzantine Agreement

The byzantine general problem is studied by Leslie Lamport, Robert Shostack and Marshall Peace in their paper, which is called the byzantine general problem which is published in 1982. This particular paper has answered, the questions like how many byzantine nodes failure can the system be survive and how many how might you build such a system. But it does not answer is it worth doing worth solving this particular problem. Let us see in more detail about the solution.

(Refer Slide Time: 29:41)



Cloud Computing and Distributed Systems Byzantine Agreement

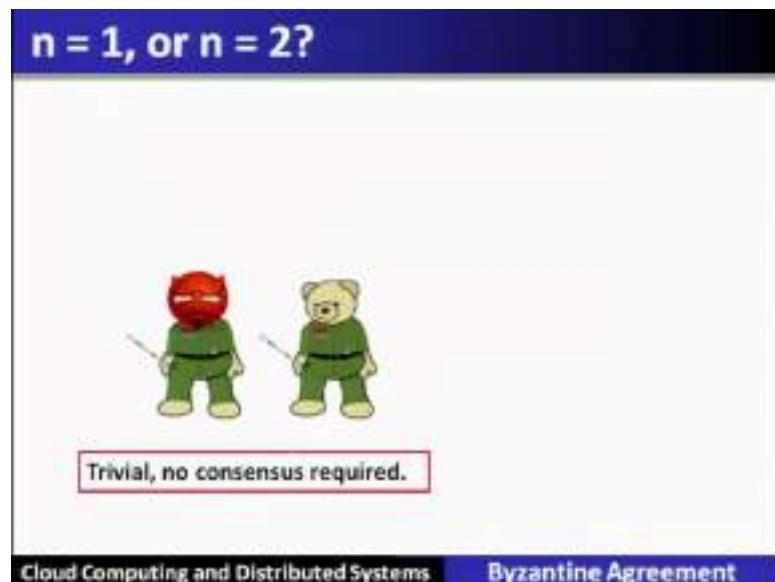
So, the problem is that different army commanders they are placed with their army around this particular mountain, which is called a byzantine. And they have to decide whether to attack or to retreat, and they have to evolve in a consensus what to do. Among them, some of them are traitors; that means, they may be confused by sending a wrong message. But if there is a majority that we will see that if majority is correct or perfect; that means, they are loyal then they may reach to an agreement what is that particular how many traitors can be there yet you can get the proper decision.

(Refer Slide Time: 30:36)



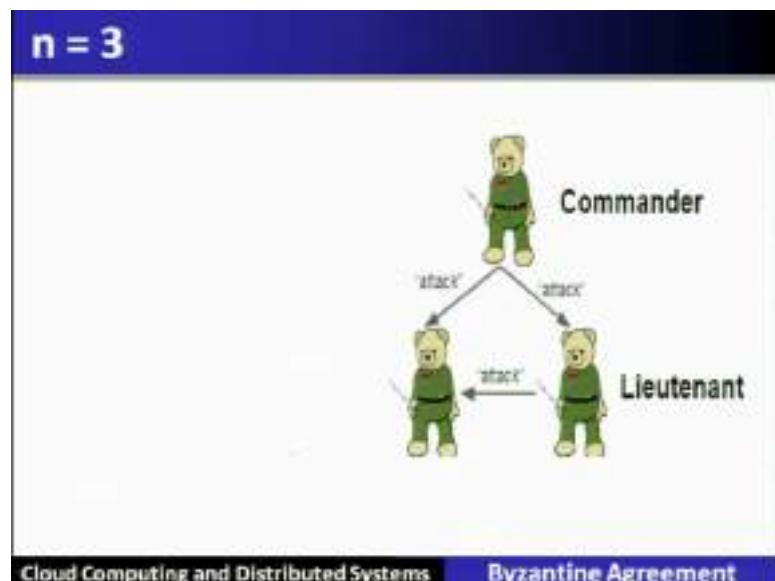
So, the question is how many traitors can you have still to solve the byzantine general problem. So, here we will assume that they communicate using oral messages and no crypto solutions are involved.

(Refer Slide Time: 30:56)



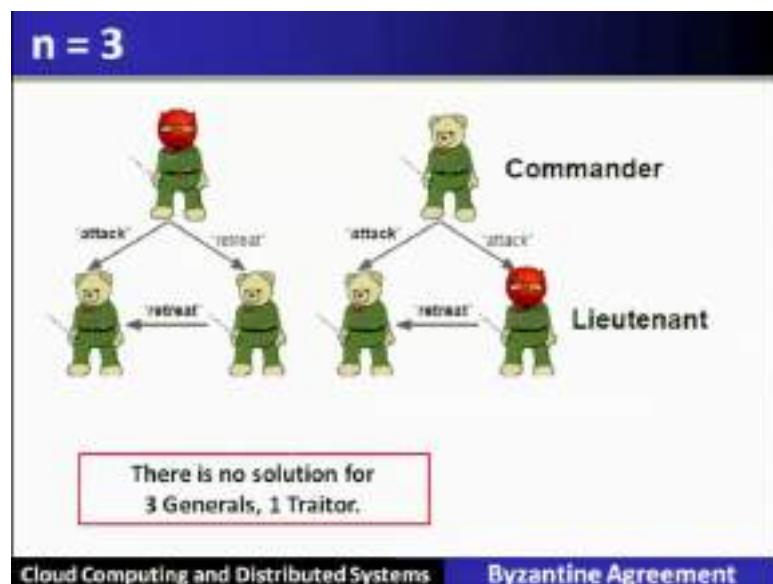
So, when $n = 1$ or $n = 2$ means if 2 generals are one general is there, then it is trivial and no consensus is required here in this case.

(Refer Slide Time: 31:12)



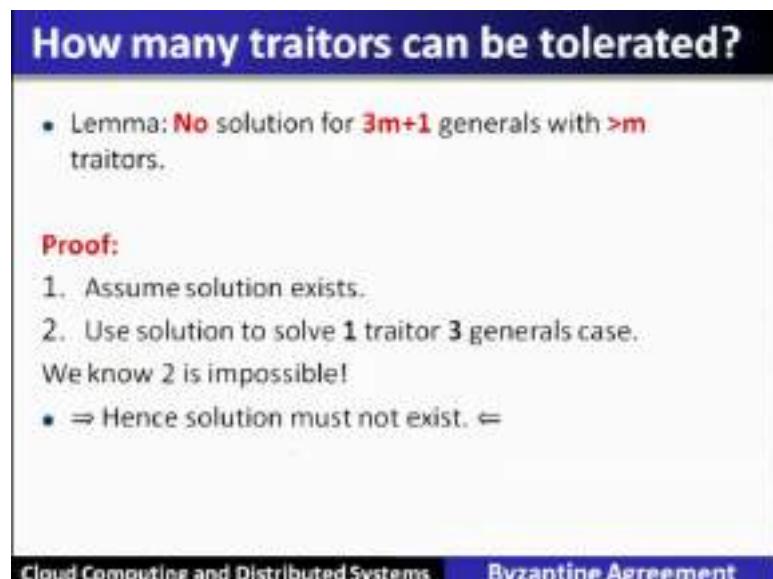
Let us see the condition when there are 3 different commanders. So, if in 3 different commanders if they say attack and this message will be exchanged.

(Refer Slide Time: 31:26)



But if one of them is traitor; the traitor will in first case we will give a confusing message to one of the commander he will say attack the other one he will send, retreat and on receiving this particular message they will exchange among themselves. And therefore, there is no majority, and this particular problem will not be able to be solved if there are 3 generals and among them one is traitor.

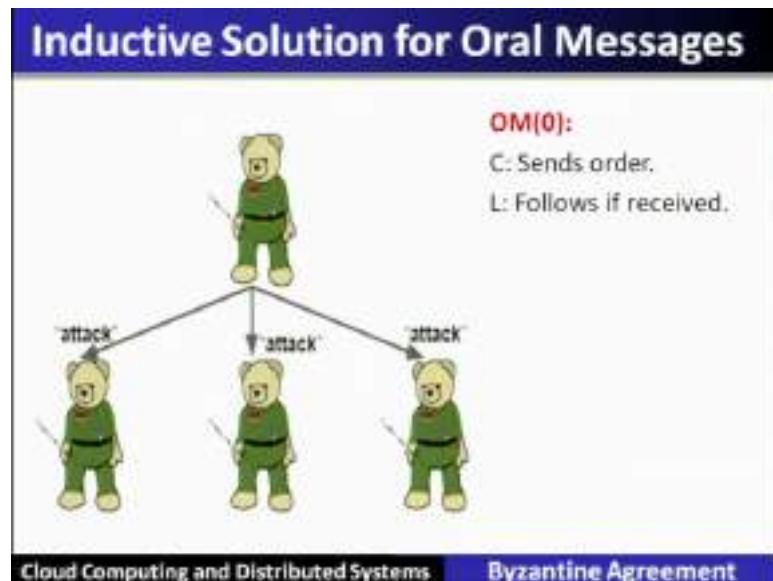
(Refer Slide Time: 32:02)



Therefore, how many such traitors can be there in a system. So, that it can be solved. So, there is a lemma that no solution for $3m+1$ generals with m is greater than traitors. So, it

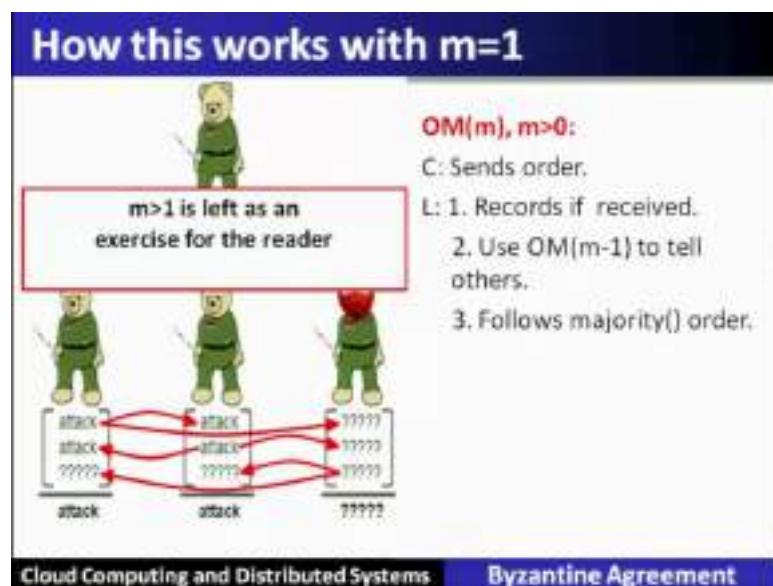
says that if the number of generals are less than $3m + 1$. Where m is the number of traitors there is no solution. So, we will see all these issues.

(Refer Slide Time: 32:37)



So, let us see that working of this algorithm.

(Refer Slide Time: 32:43)



In this particular case, both of them will send the attack one commander will send the attack to all 3. And they will exchange their messages and finally, they will take the majority. One of them is traitor, then he will send a confusing message, and even if their

messages are exchanged among themselves at the lower end and if the majority if the if the majority is taken, they may reach to a consensus.

(Refer Slide Time: 33:17)

Running Time	
Expensive	m Message Sent
	0 $O(n)$
	1 $O(n^2)$
	2 $O(n^3)$
	3 $O(n^4)$

So:

- Don't solve BGP;
- Use someone else's solution; or
- Keep n & m small

Cloud Computing and Distributed Systems Byzantine Agreement

So, that we will see how this particular algorithm will work, but at the end of this particular study of this byzantine general problem, we will see that number of messages, which are exchanged is exponential and is called quite huge.

(Refer Slide Time: 33:32)

2. Consensus Algorithm for Byzantine Failure (Message Passing, Synchronous Systems)	
<u>Model :</u>	
<ul style="list-style-type: none">– Total of n processes, at most f of which can be faulty– Reliable communication medium– Fully connected– Receiver always knows the identity of the sender of a message– Byzantine faults– Synchronous system: In each round, a process receives messages, performs computation, and sends messages.	

Cloud Computing and Distributed Systems Byzantine Agreement

So, let us see the algorithm, which is called a Byzantine Agreement problem in the message passing system, and also the synchronous system. So, the model we will assume

is let us assume that there are n processes out of that m can be. Faulty reliable communication medium we will assume fully connected receiver always knows the identity of the server. And we will assume the fault model as byzantine fault and the system is the synchronous that is in each round process receives. The messages send and performs a computation and send back another round of messages.

(Refer Slide Time: 34:12)

Solution for Byzantine Agreement Problem

- The solution of Byzantine Agreement Problem is first defined and solved by *Lamport*.
- *Pease showed that in a fully connected network, it is impossible to reach an agreement if number of faulty processes 'f' exceeds $(n-1)/3$ where n is number of processes*

Cloud Computing and Distributed Systems Byzantine Agreement

Solution to the Byzantine Agreement problem is first defined by Leslie Lamport. And Pease showed that in a fully connected network it is impossible to reach an agreement if the number of faulty processes.

(Refer Slide Time: 34:32)

Byzantine agreement can not be reached among three processes if one process is faulty

P_0 is Non-Faulty

```
graph TD; P0((P0)) <-->|1| P1((P1)); P0((P0)) <-->|1| P2((P2))
```

P_0 is Faulty

```
graph TD; P0((P0)) <-->|1| P1((P1)); P0((P0)) <-->|0| P2((P2)); P1((P1)) <-->|0| P2((P2))
```

Note: Here P_0 is a source process

Cloud Computing and Distributed Systems Byzantine Agreement

f exceeds $(n - 1)/3$ where n is the number of processes. So, let us see example here, when this particular problem cannot be reached to an agreement. Here there are 3 different processes out of which one can be faulty, if one is faulty then agreement is not reachable, because here the f exceeds $(n - 1)/3$.

(Refer Slide Time: 34:56)

Upper bound on Byzantine processes

In a system of n processes, the **Byzantine agreement problem** (as also the other variants of the agreement problem) can be solved in a synchronous only if the number of Byzantine processes f is such that $f \leq \lfloor (n - 1)/3 \rfloor$

$$f \leq \lfloor (n - 1)/3 \rfloor$$

$\rightarrow f = 1$ ✓ $n = 4$ solved

$\rightarrow f = 2$ $n = 7$
 $\rightarrow f = 3$ $n = 10$

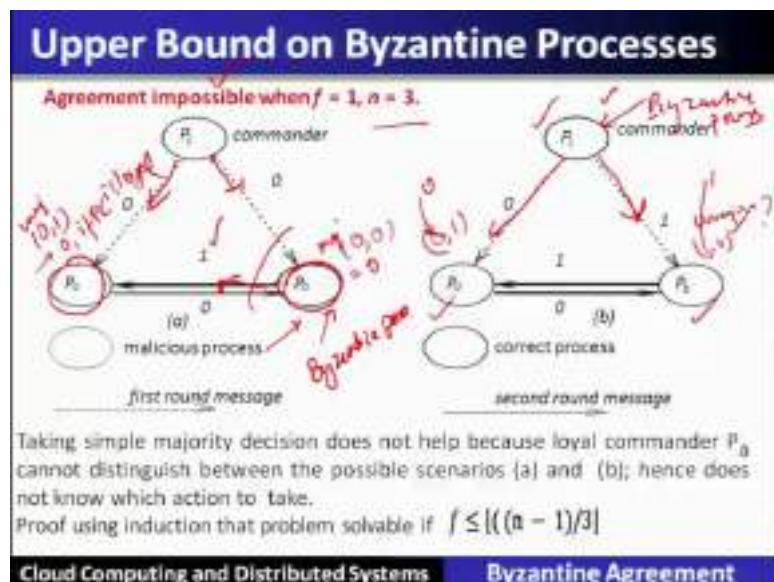
Cloud Computing and Distributed Systems Byzantine Agreement

There is an upper bound which says that in a system of n processes, Byzantine Agreement problem can be solved in a synchronous model; if the number of byzantine processes f is bounded upper by $(n - 1)/3$. So, if $f \leq (n - 1)/3$, then it can be solved. So,

take the example, if $f = 1$, then it comes out to be $n = 4$ then it can be solved. Similarly, when $f = 2$, $n = 7$ and when $f = 3$, then $n = 10$ and so on.

So, this particular formula will give minimum number of nodes having knowing how many processes can fail, what will be the size of minimum number of nodes should be present to tolerate with the byzantine faults.

(Refer Slide Time: 36:29)



So, here the byzantine problem is impossible; when n is equal to when $f = 1, n = 3$, why? Because it requires minimum as per the bound $n = 4$, but it is less than that. So, hence the Byzantine Agreement is impossible in this particular model. Let us see that if in the first case why it is impossible. We will assume that P_b is malicious or a byzantine process.

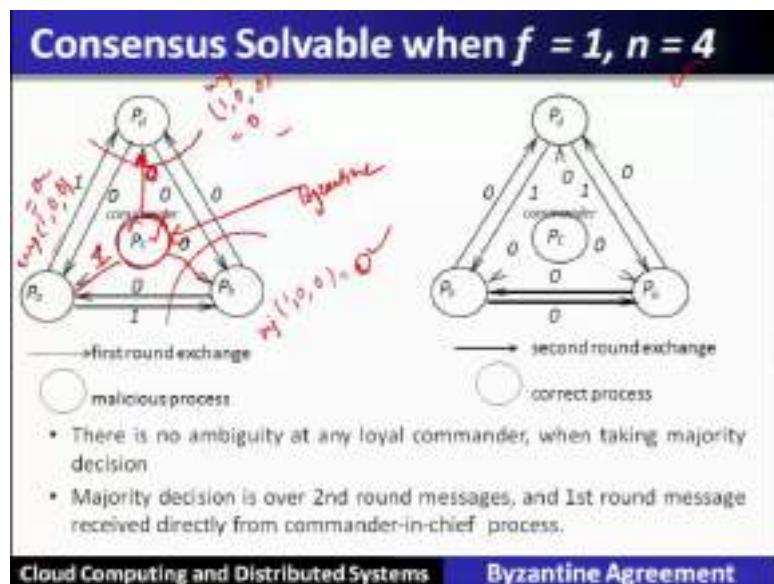
So, the commander who is loyal, he will send both the values 0, both the ends. P_b is the byzantine process, he will send the malicious way it will send after getting 0 he will send 1, because it is a malicious process. Whereas, P_a is concerned that is also loyal. So, if it gets 0 and we will exchange it so, this P_b will receive two different messages 0 and another 0. So, the majority of 2 0's will become 0. P_a is concerned it will receive 0 and it will receive 1. So, majority cannot be calculated and if we assume that P_c is loyal its value is retained, then it will be 0 if P_c is loyal.

But, if we see in another situation, when commander itself is malicious or a byzantine process; then it will send two different messages or a confusing messages to A and B. A

on receiving the same message it will send, why because this is loyal. B on receiving the message it will send the same message let us see it will receive Pb will receive 1 and 0. So, what is the majority? That is not known Pa will receive 0 and 1, what is the majority that is not known. So, if we go by the first principle that the source is retained, then here it will be taken up as 0 and this will be taken up as one.

So, there is no consensus.

(Refer Slide Time: 39:28)



Now let us see that when n is equal to 4, $f = 1$, then we will see here that consensus will be arrived. So, here let us see that the commander is the is byzantine. So, it will send different messages, it will send 0. But it will send 1 to Pa and it will send 0. These messages will be further exchange. So, Pa will send one to Pd and Pb. And one we take the total in values which is received by Pb it is 1,0,0 and, if you take majority that comes out to be 0. Similarly, Pd also will receive 1,0,0, and majority will be 0, and here also majority of 1,0,0 that becomes 0.

So, just see that here irrespective of the source being byzantine or a faulty the system is now reaching an agreement or a consensus of the same values at all other non-faulty processes. Sorry, hence through the example we have seen that if the condition that is the upper bound is satisfied, then this particular agreement problem can be solved let us see the algorithm for that which is called a Lamport Shostak Pease algorithm.

(Refer Slide Time: 41:03)

Lamport-Shostak-Pease Algorithm

This algorithm also known as **Oral Message Algorithm OM(f)**
where f is the number of faulty processes
 n = Number of processes and $n \geq 3f + 1$

Algorithm is Recursively defined as follows:

Algorithm OM(0)

1. Source process sends its values to each other process.
2. Each process uses the value it receives from the source.
[If no value is received default value 0 is used]

Cloud Computing and Distributed Systems Byzantine Agreement

This algorithm also well known as oral message algorithm; where the input to this particular algorithm is f , f is the number of faulty processor. Here $n \geq 3f+1$. This particular boundary we have already seen. This particular algorithm is recursive algorithm. So, the base case of the recursion is when, when it is 0 then the source process will send its value to each other process. And each process uses the value it receives from the source.

(Refer Slide Time: 41:51)

Contd...

Algorithm OM(f), $f > 0$

1. The source process sends its value to each other process.
2. For each i , let v_i be the value process i receives from source.
[Default value 0 if no value received] *→ 5 values model*
3. Process i acts as the new source and initiates **Algorithm OM($f-1$)** where it sends the value v_i to each of the $n-2$ other processes.
4. For each i and j (not i), let v_{ij} be the value process i received from process j in STEP 3. Process i uses the value **majority** (v_1, v_2, \dots, v_{n-1}).

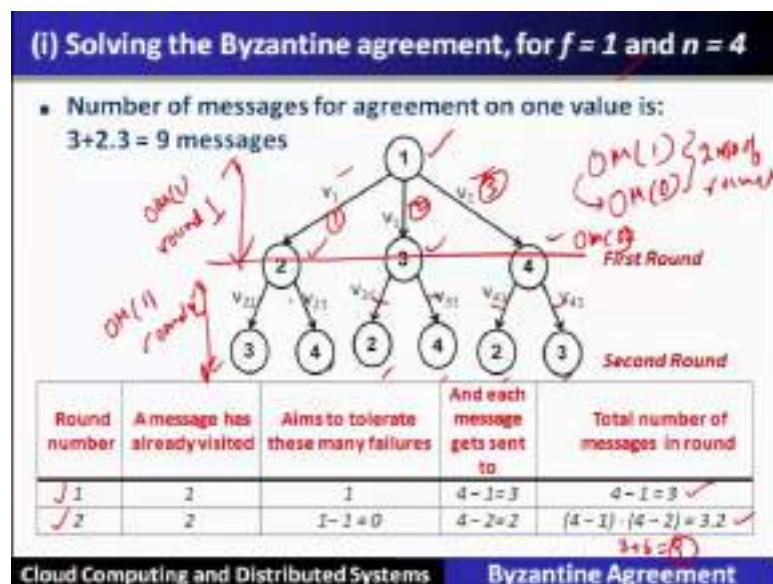
"The function **majority**(v_1, v_2, \dots, v_{n-1}) computes the majority value if exists otherwise it uses default value 0."

Cloud Computing and Distributed Systems Byzantine Agreement

Now, it will be the next step will be the recursion, recurse, it will recurse on OM of f, when $f > 0$. Source process sends its values to each other, that we have seen for each i, let v_i be the value the process i received from the source if there is no value is received then it will be 0, why because it is a synchronous algorithm.

We can assume such a value. So, process i will act as a new source and then initiates OM f minus 1. So, finally, what you will do? For each i and j, let be j be the values received by a process i in a step number 3; process i will compute the majority. So, the function majority computes the majority value if the if it exist. Otherwise it uses a default value of 0, here in this algorithm.

(Refer Slide Time: 43:02)



So, we have already seen the Byzantine Agreement, example also. Let us do the analysis of this particular algorithm. So, when $f = 1$ this algorithm will be called OM of 1. Which in turn will call OM(0) so; that means, there will be a 2 number of rounds. So, here 2 rounds are shown. In the first round, it will send the values, one of them will send the values the commander will send the values to the remaining 3 different process, let us say v_1 , which in turn every process will become source and will invoke the round 2 with OM(0)s.

So, there will be 3 invocation of OM 0, 1, 2 and 3. And once the messages are reached at all the ends they will take the majority. So, let us count how many messages are exchanged. So, in the first round 3 messages are exchanged 1 2 and 3. In the second

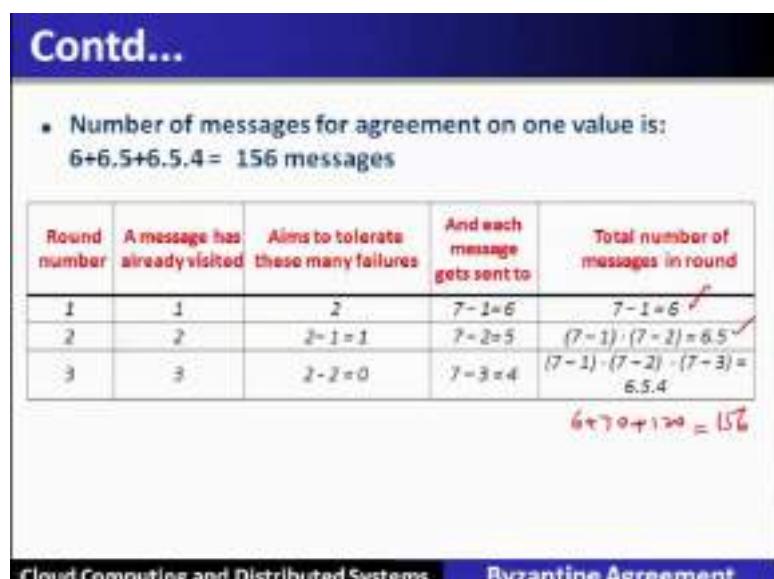
round so, in the second round there will be 6 messages 1, 2, 3, 4, 5, 6. So, total number of messages will be $3 + 6$; that is 9 different messages will be exchanged here in this particular case.

(Refer Slide Time: 45:00)



Now, when $f = 2$, and $n = 7$, there will be 3 different rounds.

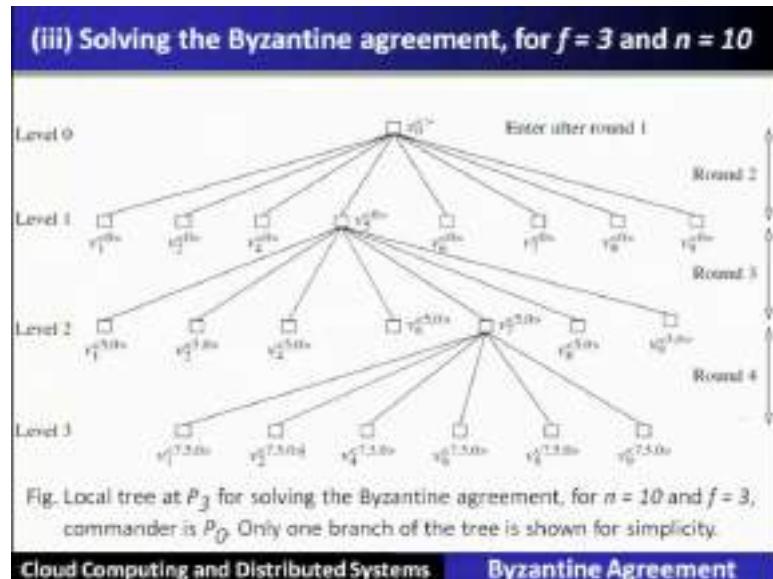
(Refer Slide Time: 45:10)



And if you see here how many messages will be exchanged? As you know that it will be $n - 1$, that is first round will have 6 messages; second round will have 6 times 5 and third

round will require 120 messages. So, as so that becomes 156 different messages are required when $f = 2$.

(Refer Slide Time: 45:47)



When $f = 3$, you just see that entire worked out example is given in the slide is quite complicated.

(Refer Slide Time: 46:00)

Contd...

- Number of messages for agreement on one value is:
 $9+9.8+9.8.7+9.8.7.6 = \underline{\underline{3609 \text{ messages}}}$

Round number	A message has already visited	Aims to tolerate these many failures	And each message gets sent to	Total number of messages in round
1	1	$3 - 1 = 2$	$10 - 1 = 9$	$10 - 1 = 9$
2	2	$3 - 1 = 2$	$10 - 2 = 8$	$(10 - 1) \cdot (10 - 2) = 9.8$
3	3	$3 - 2 = 1$	$10 - 3 = 7$	$(10 - 1) \cdot (10 - 2) \cdot (10 - 3) = 9.8.7$
4	4	$3 - 3 = 0$	$10 - 4 = 6$	$(10 - 1) \cdot (10 - 2) \cdot (10 - 3) \cdot (10 - 4) = 9.8.7.6$

Cloud Computing and Distributed Systems Byzantine Agreement

But we can summarize in the table how many messages are required. That is, 3609 too many number of messages are required 3 to tolerate 3 different faults. It becomes very,

very costly in the system to support such kind of failure tolerance. But from this point onwards several studies are conducted and different solutions are there.

(Refer Slide Time: 46:28)

Relationship between # Messages and Rounds				
Round number	A message has already visited	Aims to tolerate these many failures	And each message gets sent to	Total number of messages in round
1	1	f	$n-1$	$n-1$
2	2	$f-1$	$n-2$	$(n-1) \cdot (n-2)$
...
x	x	$(f+1)-x$	$n-x$	$(n-1)(n-2)\dots(n-x)$
$x+1$	$x+1$	$(f+2)-x-1$	$n-x-1$	$(n-1)(n-2)\dots(n-x-1)$
$f+1$	$f+1$	0	$n-f-1$	$(n-1)(n-2)\dots(n-f-1)$

Table: Relationships between messages and rounds in the Oral Messages algorithm for Byzantine agreement.

Complexity: $f + 1$ rounds, exponential amount of space, and $(n - 1) + (n - 1)(n - 2) + \dots + (n - 1)(n - 2)\dots(n - f - 1)$ messages

Cloud Computing and Distributed Systems Byzantine Agreement

So, in summary we can see that the complexity, we can see that as per the time complexity it will be having $f + 1$ number of rounds. But as whereas, the number of message complexity is concerned that also goes to an exponential number.

(Refer Slide Time: 46:54)

Impossibility Result for the Consensus Problem	
Fischer-Lynch-Paterson (FLP) Impossibility Result (By M. Fischer, N. Lynch, and M. Paterson, April 1985)	
<ul style="list-style-type: none"> Fischer et al. showed a fundamental result on the impossibility of reaching agreement in an asynchronous (message-passing) system. 	
<ul style="list-style-type: none"> It states that it is "Impossible to reach consensus in an asynchronous message passing system even if a single process has a crash failure" 	
<ul style="list-style-type: none"> This result, popularly known as the FLP impossibility result, has a significant impact on the field of designing distributed algorithms in a failure-susceptible system. 	
Cloud Computing and Distributed Systems	Byzantine Agreement

Now, agreement in asynchronous message passing systems with the failure; we have impossibility result for consensus problem in the asynchronous situation, given by

Fischer Lynch and Paterson, FLP problem it is called. It is impossible to reach the consensus in an asynchronous message passing system, even if a single process has the crash fault.

(Refer Slide Time: 47:19)

Weaker Versions of Consensus Problem	
Consensus Problem	Description
Terminating reliable broadcast	It states that a correct process always gets a message even if the sender crashes while sending. If the sender crashes while sending the message, the message may be a null message but it must be delivered to each correct process.
k-set consensus	It is solvable as long as the number of crash failures f is less than the parameter k . The parameter k indicates that the non-faulty processes agree on different values, as long as the size of the set of values agreed upon is bounded by k .
Approximate agreement	Like k-set consensus, approximate agreement also assumes the consensus value is from a multi-valued domain. However, rather than restricting the set of consensus values to a set of size k , approximate agreement requires that the agreed upon values by the non-faulty processes be within ϵ of each other.
Renaming problem	It requires the processes to agree on necessarily distinct values.
Reliable broadcast	A weaker version of reliable terminating broadcast(RTB), namely reliable broadcast, in which the termination condition is dropped, is solvable under crash failures.

Cloud Computing and Distributed Systems Byzantine Agreement

Now, this has led to the other development of a weaker version of a consensus problem, among them which is called a terminating reliable broadcast problem.

(Refer Slide Time: 47:32)

(i) Terminating Reliable Broadcast (TRB) Problem	
<ul style="list-style-type: none"> A correct process always gets a message, even if sender crashes while sending (in which case the process gets a null message). Validity: If the sender of a broadcast message m is non-faulty, then all correct processes eventually deliver m. Agreement: If a correct process delivers a message m, then all correct processes deliver m. Integrity: Each correct process delivers at most one message. Further, if it delivers a message different from the null message, then the sender must have broadcast m. Termination: Every correct process eventually delivers some message. 	

Cloud Computing and Distributed Systems Byzantine Agreement

So, the correct process always gets a message even if the sender crashes, while sending there are 3 condition validity if the sender of a broadcast message m is non faulty. Then

all correct processes eventually deliver if a correct process delivers a message then all correct processes delivers it, that integrity and termination. Termination says that every process eventually deliver some message.

(Refer Slide Time: 48:01)

Contd...

- The reduction from consensus to terminating reliable broadcast is as follows:
- A commander process broadcasts its input value using the terminating reliable broadcast. A process decides on a "0" or "1" depending on whether it receives "0" or "1" in the message from this process.
- However, if it receives the null message, it decides on a default value. As the broadcast is done using the terminating reliable broadcast, it can be seen that the conditions of the consensus problem are satisfied.
- **But as consensus is not solvable, an algorithm to implement terminating reliable broadcast cannot exist.**

Cloud Computing and Distributed Systems Byzantine Agreement

But here the consensus is not solvable the algorithm to implement terminating reliable broadcast cannot exist in this model.

(Refer Slide Time: 48:15)

(ii) Reliable Broadcast Problem

- **Reliable Broadcast (RB)** is RTB without terminating condition.
- RTB requires eventual delivery of messages, even if sender fails before sending. In this case, a null message needs to get sent. In RB, this condition is not there.
- RTB requires recognition of a failure, even if no msg is sent.
- **Crux:** RTB is required to distinguish between a failed process and a slow process.
- RB is solvable under crash failures; $O(n^2)$ messages

Algorithm: Protocol for reliable broadcast
[1] Process P_i initiates Reliable Broadcast;
[1a] broadcast message M to all processes;
[2] A process P_j, 1 ≤ j ≤ n, receives message M;
[2a] If M was not received earlier then
[2b] broadcast M to all processes;
[2c] deliver M to the application.

Cloud Computing and Distributed Systems Byzantine Agreement

Therefore, a variation alteration of this where termination condition is not there, then it is called a reliable broadcast problem, that is the compromised problem it is. So, reliable

broadcast problem is without terminating condition. So, therefore, it is preserving that impossibility result, but modified the problem. So, reliable broadcast problem requires eventual delivery of the messages, even if the sender fails before sending. In this case null message needs to be get send. This condition is not there in the reliable broadcast problem which was assumed in the previous algorithm RTB.

So, RTB is different this reliable broadcast RB is different than RTB. Why because, here the termination condition is not there, and this null message also is not there. This RB that is reliable broadcast problem is solvable under the crash fault with the complexity order n^2 .

(Refer Slide Time: 49:35)

Applications of Agreement Algorithms

1) Fault-Tolerant Clock Synchronization

- Distributed Systems require physical clocks to synchronized
- Physical clocks have drift problem
- Agreement Protocols may help to reach a common clock value.

2) Atomic Commit in Distributed Database System (DDBS)

- DDBS sites must agree whether to commit or abort the transaction
- Agreement protocols may help to reach a consensus,

[Cloud Computing and Distributed Systems](#) [Byzantine Agreement](#)

The applications of agreement algorithm are fault tolerant clock synchronization, atomic commit in distributed database systems.

(Refer Slide Time: 49:44)

Conclusion

- Consensus problems are fundamental aspects of distributed computing because they require inherently distributed processes to reach agreement.
- This lecture first covers:
 - Different forms of the consensus problem,
 - Then gives an overview of what forms of consensus are solvable under different failure models and different assumptions on the synchrony/asynchrony.

Cloud Computing and Distributed Systems Byzantine Agreement

Conclusion; so, consensus problems are fundamental in the study of a distributed system and the cloud computing system, where they require inherently distributed processes to reach to a common agreement. This lecture has covered different forms of consensus problem under the byzantine fault model. And we have then given the different form of consensus which are solvable under different fault models and in synchronous and asynchronous systems.

(Refer Slide Time: 50:27)

Contd...

- Then we have covered agreement in the following categories:
 - (i) Synchronous message-passing systems with failures:
 - Used Fault Models: fail-stop model and the Byzantine model.
 - (ii) Asynchronous message-passing systems with failures:
 - Impossible to reach consensus in this model.
 - Hence, several weaker versions of the consensus problem, i.e. terminating reliable broadcast, reliable broadcast are considered.

Cloud Computing and Distributed Systems Byzantine Agreement

Synchronous message passing systems with failures can also support both fault fail-stop as well as byzantine fail model. In asynchronous message passing system it is impossible to reach consensus in this model hence the weaker versions of consensus problem we have seen; that is reliable broadcast problem. We have seen the terminating reliable broadcast TRB is impossible.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 15
Failures and Recovery Approaches in Distributed Systems

(Refer Slide Time: 00:19)

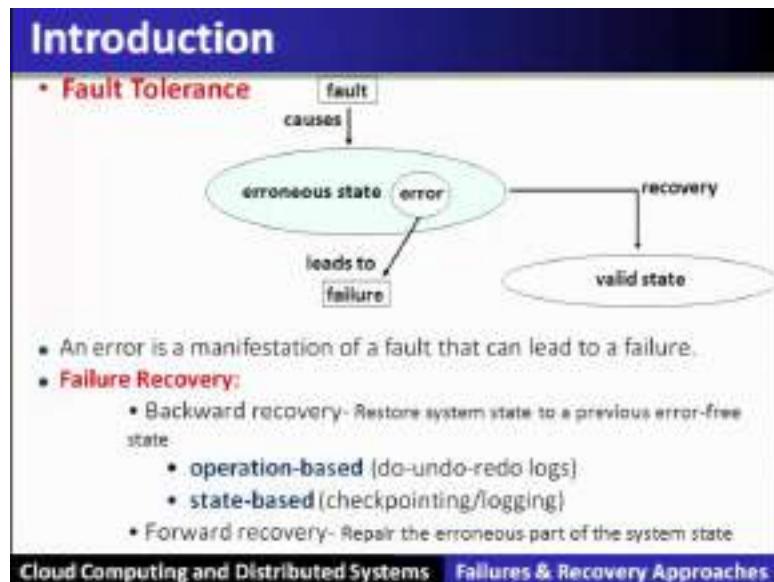
The slide has a dark blue header bar with the word "Preface" in white. Below it is a white content area. At the bottom, there is a dark blue footer bar with two tabs: "Cloud Computing and Distributed Systems" and "Failures & Recovery Approaches".

Content of this Lecture:

- In this lecture, we will discuss about basic fundamentals and underlying concepts of '**Failure and Rollback Recovery**' and
- Also discuss **Checkpointing & Rollback Recovery Schemes** in distributed systems i.e.
 - (i) Checkpoint based and
 - (ii) Log based

Failure and Recovery Approaches in Distributed Systems. Preface content of this lecture, we will discuss about basic fundamentals and underlying concepts of Failure and Rollback Recovery in distributed systems. Also we will discuss the Checkpointing and Rollback Recovery Schemes of distribute systems that is, checkpointing based schemes and log based schemes.

(Refer Slide Time: 00:44)



Introduction; so, error is a manifestation of a fault, that will lead to a failure. Here we will see that, in that particular state which is called erroneous state. The recovery is required to reach to a valid state which is an error free state.

So, the fault tolerance is achieved using recovery of a failure using two methods. The first one is called forward recovery method where the repair of the erroneous part of a system state is known and it will be carried out to reach or to recover the system from failed state to the valid state that is called forward recovery method. There is another failure recovery method which is called, a backward recovery method. It will restore the system state to a previous error free state; it does not require the knowledge of the erroneous part of the system state.

The backward recovery is of two types, that is operation based backward recovery method in which all the course grained operation like do, undo, redo. They are logged in a file and they are being operated on, that is why it is called operation based. The second type of backward recovery is called a state based. Here the states are stored or a logged, they are also called checkpointing. So, when a failure happens using this particular checkpointing or a log based logging method, it will try to recover to a restore the previous valid state.

(Refer Slide Time: 02:56)

Introduction

- Rollback recovery protocols
 - Restore the system back to a **consistent state** after a failure ✓
 - Achieve **fault tolerance** by periodically saving the state of a process during the failure-free execution ✓
 - Treats a distributed system application as a collection of processes that communicate over a network
- Checkpoints
 - The **saved states** of a process
- Why is rollback recovery of distributed systems complicated?
 - Messages induce inter-process dependencies during failure-free operation
- Rollback propagation
 - The dependencies may force some of the processes that did not fail to roll back
 - This phenomenon is called "**domino effect**" ✓

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, the rollback recovery protocols will restore the system from the previous consistent state after the failure and it achieves the fault tolerance by periodically saving the state of a process during the failure free execution. That means, to restore the system from the previous error free state that is called consistent state. After the failure, it requires to maintain while the system is failure free that is periodically saving the states which are called consistent state. So, that during the failure it can restart from that point onwards. This is going to be an important issue, how to save the state, so that the recovery or the system can recover whenever there is a fault happens. Now, here we treat the distributed system application as a collection of processes that they communicate over the network. In that model how the recording of the states are done, when the while in the failure free executions.

So, one such method is called a checkpoint; that means, checkpoint is a saved state of a process. So, why is the rollback recovery, in a distributed system is complicated. So, messages induced by inter process dependencies during the failure free operation and capturing this, particular dependencies which will be useful for the recovery is a complicated. One: Rollback propagation, these dependencies may force some of the processes which do not fail to rollback and this phenomenon is called domino effect.

(Refer Slide Time: 05:20)

Contd...

- If each process takes its checkpoints independently, then the system **can not avoid the domino effect**
- this scheme is called **independent or uncoordinated checkpointing**
- Techniques **that avoid domino effect**
 - **Coordinated checkpointing rollback recovery**
 - *processes coordinate their checkpoints to form a system-wide consistent state
 - **Communication-induced checkpointing rollback recovery**
 - ***forces each process to take checkpoints** based on information piggybacked on the application
 - **Log-based rollback recovery**
 - *combines checkpointing with logging of non-deterministic events relies on piecewise deterministic (PWD) assumption

Cloud Computing and Distributed Systems Failures & Recovery Approaches

If each process takes its checkpoint independently, then the system cannot avoid domino effect. This scheme is called independent or uncoordinated checkpointing. The advantage of uncoordinated checkpointing is that, checkpointing can be done easily without much overhead, but the drawback is that it suffers from domino effect. So, the techniques that can avoid domino effect are coordinated checkpointing and rollback recovery method; that means the processes. They coordinate to record their checkpoints to form the system wide consistent state. Therefore, with the coordination or with the coordinated checkpointing, this state that is called domino effect will not happen and all the state recorded is a consistent state.

Communication induced checkpointing rollback recovery method forces each process to take checkpoints based on the information which is piggybacked on the application, also useful to avoid the domino effect while taking the checkpoints. Third scheme is called log based rollback recovery scheme. This combines the checkpointing with the logging of non deterministic events which relies on a piecewise deterministic assumptions. So that means, log based rollback is nothing, but the combined checkpointing with the logging of non deterministic events; that means, that dependencies are properly exploited wire while taking the checkpoints. So, that the consistent state is being recorded. So, in all three cases we see that, we can avoid the checkpointing, but it requires some effort at the time of checkpointing.

(Refer Slide Time: 07:21)

A local checkpoint

- All processes save their **local states** at certain instants of time
- A **local check point** is a **snapshot of the state** of the process at a given instance

Assumptions:

- A process stores all local checkpoints on the **stable storage**
- A process is able to roll back to any of its existing local checkpoints

- $C_{i,k}$
 - The kth local checkpoint at process P_i
- $C_{i,0}$
 - A process P_j takes a checkpoint $C_{j,0}$ before it starts execution

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Let us see the preliminaries. A local checkpoint that is, all the processes save their local state at a certain instant of time is, so a local checkpoint is a snapshot of the state of a process at a given instance. So, let us assume some of the notations. A process stores all the local checkpoints on the stable storage. So, stable storage is that kind of storage which will survive even after the serious faults.

So, a process is able to rollback to any of the previous existing local checkpoints. Let $C_{i,k}$ denote the kth local checkpoint at a process i and $C_{i,0}$ denotes the process i, which takes the checkpoints $C_{i,0}$, before it starts the execution.

(Refer Slide Time: 08:23)

Consistent states

- A global state of a distributed system
 - a collection of the individual states of all participating processes and the states of the communication channels
- Consistent global state
 - a global state that may occur during a failure-free execution of distribution of distributed computation
 - If a process's state reflects a message receipt, then the state of the corresponding sender must reflect the sending of the message
- A global checkpoint
 - a set of local checkpoints, one from each process
- A consistent global checkpoint
 - a global checkpoint such that no message is sent by a process after taking its local point that is received by another process before taking its checkpoint
 - receipt of message is recorded in a checkpoint
→ send of message is also recorded in local checkpoint*

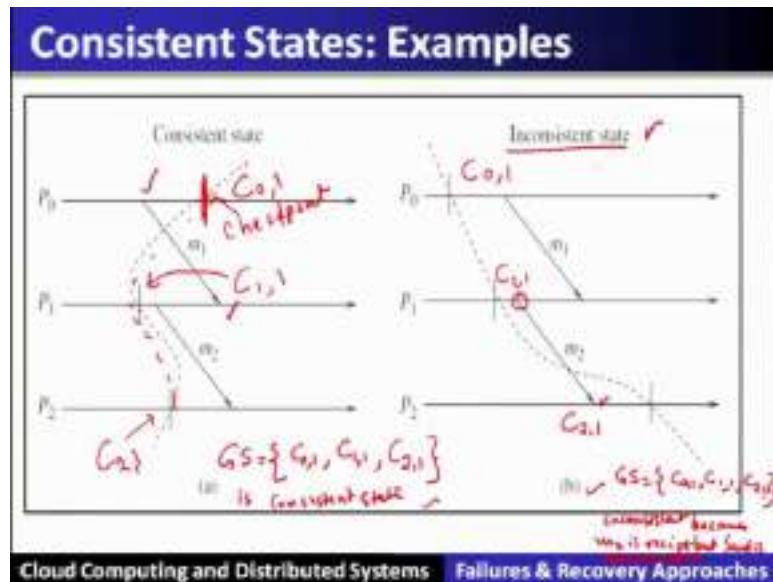
Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, let us see some of the definitions. A global state of a distributed system is a collection of individual states of all participating processes and the state of a communication channel that, particular global state which is called a consistent global state may occur during the failure free execution of a distributed computation.

Now, if a process state reflects a message receipt, then the state of a corresponding sender of that particular message must also reflect the sending of a message. If, this condition is captured in the consistent global state or if this condition is captured in a recording of a global state, then that global state is called a consistent global state. What is a global checkpoint? A set of local checkpoint, from each process is called a global checkpoint that, global checkpoint which holds the following condition is called a global checkpoint.

So, that conditions says, that a global checkpoint such that, no message is sent by a process after taking its local checkpoint that is received by another process, before taking its checkpoint; meaning to say that in a consistent global checkpoint, if the receipt of a message is recorded in any checkpoint in any local checkpoint; then this implies that the send of the message is also recorded in the local checkpoint. Hence this is called a global checkpoint, we will see these details.

(Refer Slide Time: 10:41)



Here, we can see we have shown the consistent state with a dotted line. So, that means, this vertical bar shows the checkpoint, let us say this is C0,1 and this is C1,1 and here this is called C2,1. This particular state that, is the global state which is defined by the checkpoints; C0,1, C1,1, C2,1. This particular global state is a consistent state. Because, let us see this particular message, the send is recorded but receive is not recorded that is possible in the set of global state or in the set of checkpoints.

Therefore, this is a consistent state however; on the other hand this example the other example, which is shown in the figure b is inconsistent state. Let us see the global state, this is C0,1, this is C1,1, this is C2,1, so that collection C0,1, C1,1, C2,1. This particular global state is inconsistent because, there is a receipt of a message m2 but its sent is not recorded. Hence this is going to be an inconsistent global state.

(Refer Slide Time: 13:22)

Interactions with outside world

- A distributed system often interacts with the outside world to receive input data or deliver the outcome of a computation
- **Outside World Process (OWP)**
 - a special process that interacts with the rest of the system through message passing
- **A common approach**
 - save each input message on the stable storage before allowing the application program to process it
- **Symbol "||"**
 - An interaction with the outside world to deliver the outcome of a computation

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Interaction with the outside world, the distributed system interacts with the outside world to receive the input data or deliver the outcome of the computation, which is denoted as outside world process. This is a special process that interacts with the rest of the world through the message passing. A common approach is to, save each input message on the storage before allowing the application program to process it. So, the interaction with the outside world, to deliver the outcome of a computation is denoted by the symbol double bar.

(Refer Slide Time: 14:05)

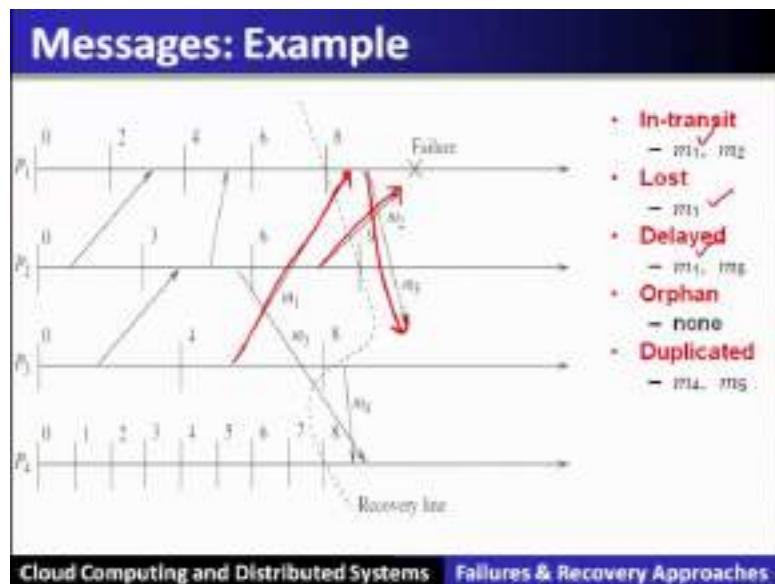
Messages

- In-transit message**
 - messages that have been sent but not yet received
- Lost messages**
 - messages whose 'send' is done but 'receive' is undone due to rollback
- Delayed messages**
 - messages whose 'receive' is not recorded because the receiving process was either down or the message arrived after rollback
- Orphan messages**
 - messages with 'receive' recorded but message 'send' not recorded
 - do not arise if processes roll back to a consistent global state
- Duplicate messages**
 - arise due to message logging and replaying during process recovery

Cloud Computing and Distributed Systems Failures & Recovery Approaches

We will see different type of messages, which are used in our system for recording, as well as to deal with the failure recovery. So, let us understand them and then, we will see how to deal with them, which are serious and which are not that serious, to be for the handling during the recovery method. So, the messages is which are called in transit messages, they have been sent but not yet received, that is called in transit message.

(Refer Slide Time: 14:48)



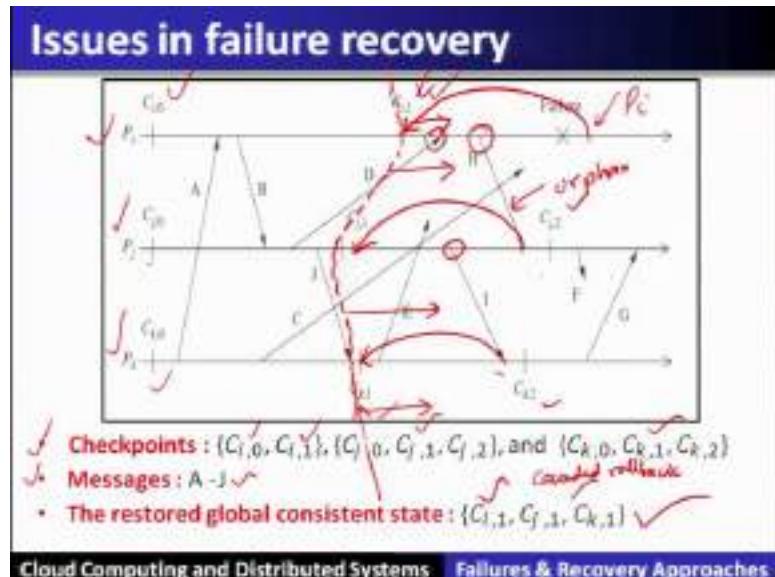
So, here in this particular figure the message m_1 and m_2 , so, m_2 is this particular message this is sent, but not being delivered.

Similarly, the message m_1 is also in the transit why because, it is sent but the receiving process that is p_1 is either not available due to the failure. Hence this message will be in the transit, will be considered in the transit. Lost messages, that is messages who send is done, but receive is undone due to the rollback. Here the last message is m_1 why because; the failure will not allow the p_1 to receive the message. Hence this particular message is also transit and also in the lost message.

Delayed messages, that is messages whose receive is not recorded because the receiving process was either down or the message arrived late after the rollback. Here the again the delayed message m_1 is also denoted as the delayed message and another message m_5 is also, this m_5 is also delayed. Orphan message is, so messages with receive recorded, but its send of that particular messages in the checkpointing this is called an orphan message. So, let us see, here it is not shown, we will discuss this kind of message later on.

Duplicate messages arise due to the message logging and replying during the process recovery.

(Refer Slide Time: 16:41)



Now, here let us understand the figure and then we will see these issues. So, the messages which are shown here, from A to J they are being labeled. Also the checkpoints the process P1, Pi has taken the 2 checkpoints they are Ci,0 and Ci,1 similarly process Pj has taken 1, 2 and 3 checkpoints, which are mentioned over here. Similarly Pk has taken 3 checkpoints 1, see, 1, 2 and 3. Now if there is a failure at this point of time in shown over here there is a failure then, the recovery requires the system to restart from the global consistent state that is, the most recent global consistent state. Let us assume that, P1 will restart from this checkpoint that is Ci,1 then, this particular receive of this particular message which if, it is already happened then it will not be received.

Similarly, the send of a message H, is also not being considered now in this scenario. So, if send is not recorded, but if this checkpoint Cj,2 is being considered, it has recorded the received, but send due to the failure is not recorded. So this becomes an orphan message which, in turn will trigger the rollback to the previous consistent state. So, this will trigger the rollback over here, this will trigger the roll back to the previous consistent state. Now this particular rollback will say that this particular message send, it is send is not recorded now, therefore, this particular message which is received it is send is not recorded. Therefore, the system this will trigger to rollback from this position. Therefore,

this will be the consistent checkpoint and the system will restart from this particular state onwards.

Now, the question is that, how manually we have seen that during the recovery of P1 this is also forcing Pj to recover all though Pj was not faulty. Similarly Pk also was triggered to be rolled back, so it is a cascaded rollback because, rollback of one process will create the orphan messages which in turn, will trigger the rollback of other non faulty processes. So, we are just seen through this particular example that, failure recovery is not a trivial task, it requires dependencies are to be captured and also it requires how the global state is to be recorded. So, that with the minimal loss the system can be restored after the failure

(Refer Slide Time: 20:58)

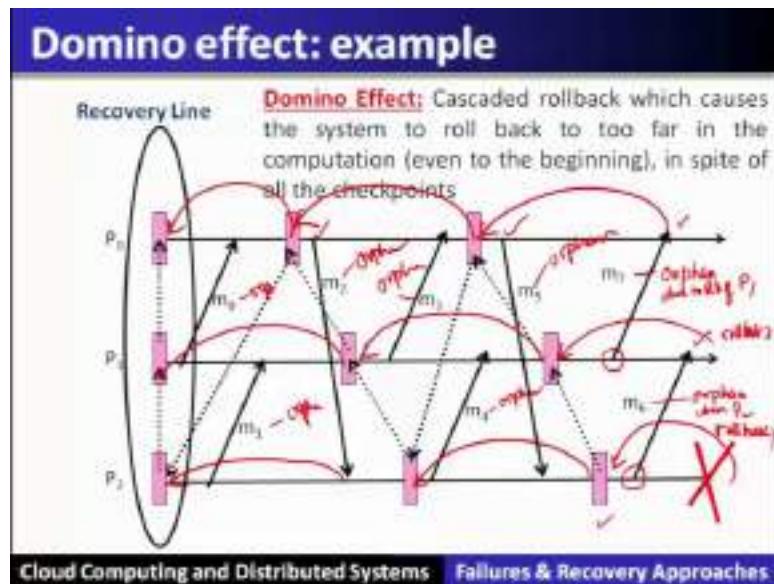
Issues in failure recovery

- The rollback of process P_i to checkpoint $C_{i,1}$ created an orphan message H
- Orphan message I is created due to the roll back of process P_j to checkpoint $C_{j,1}$.
- Messages C, D, E, and F are potentially problematic
 - Message C: a delayed message
 - Message D: a lost message since the send event for D is recorded in the restored state for P_j , but the receive event has been undone at process P_j
 - Lost messages can be handled by having processes keep a message log of all the sent messages
 - Messages E, F: delayed orphan messages. After resuming execution from their checkpoints, processes will generate both of these messages

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Here issues in failure recovery. The rollback of a process P_i to a checkpoint $C_{i,1}$, created an orphan message H. Orphan message is created due to the rollback of a process P_j to a previous to a checkpoint that is $C_{j,1}$. So, the messages here we are seeing in the example C, D, E, F they are basically the problematic in that sense, that they have become the orphan message and it requires a rollback to undo that effect.

(Refer Slide Time: 21:44)



Closely the domino effect, that is called cascaded rollback which causes the system to rollback. So, let us that process P2 is failed at this particular position. So, after the failure, it will recover from the previous checkpoint and this particular checkpoint will trigger the non faulty process P1 also to rollback because, it has received this message m6 whose send will be undone. So, that means, m6 will become orphan message, when it will be when process P2 will be rolled back

So, this particular orphan message will trigger to roll back to the previous consistent state. Now if it is rollback, then what will happen is m7 which is received at P0, but its send will be not there. So, this also will become orphan message due to the rollback of P1. So, therefore, this will trigger P1 to rollback from to the previous state. Now if P0 is rollback, then what will happen to this particular message which is m5, so m5 will become orphan in this condition and we will trigger P2 to further rollback to its previous state. Having roll back to the previous state this m4 will become orphan therefore, it will also trigger P12 further rollback.

Now, if P1 is rollback then m3 will become orphan and will trigger the P0 also to further rollback. Having seen that P0 is rollback then, this message m2 will become orphan and this will trigger the further rollback. If this is rollback then, these will also rollback why because, m1 will become orphan and if therefore, when this is rollback then m0 will

become orphan and finally, it will roll back to their previous state. So, this is an extreme condition of the domino effect.

(Refer Slide Time: 24:25)

Problem of Livelock

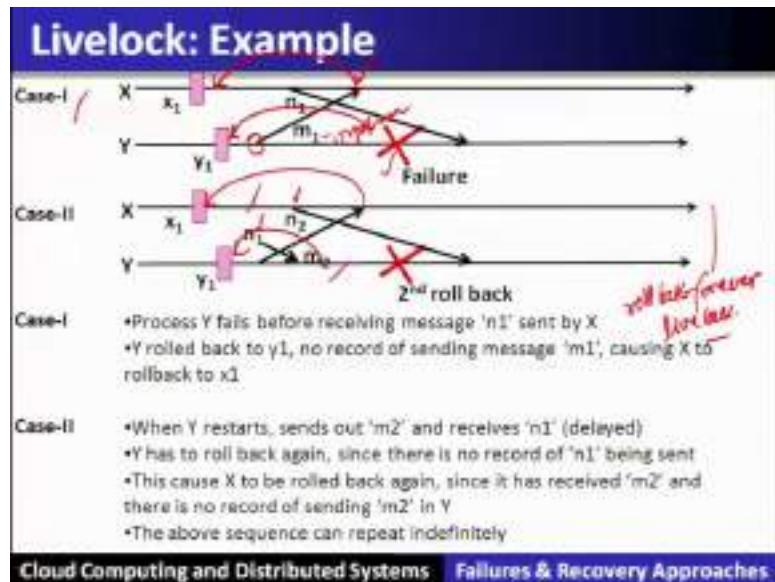
- **Livelock:** case where a single failure can cause an infinite number of rollbacks.
- **The Livelock problem** may arise when a process rolls back to its checkpoint after a failure and requests all the other affected processes also to roll back.
- In such a situation if the roll back mechanism has no synchronization, it may lead to the livelock problem.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, we will see another problem. We have seen earlier the problem of domino effect due to the orphan messages. Now we will see another problem that is called livelock problem of livelock.

The livelock case, where the single failure can cause an infinite number of rollback is called a livelock. The livelock problem may arise when a process roll back to its checkpoint after a failure and request all other affected processes also to roll back. In such a situation if rollback mechanism has no synchronization, which will lead to a infinite number of rollback this will lead to the livelock problem.

(Refer Slide Time: 25:16)

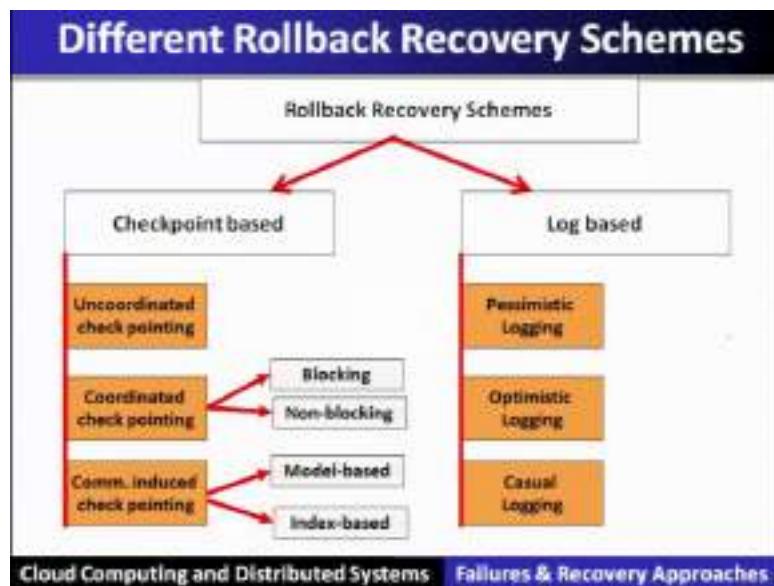


Let us see through this example. Here the case I says that, process Y fails because, the receiving message n_1 is sent by X. Y will be roll back to Y_1 , no record of sending m_1 causing X to roll back to X_1 . Let us understand this again.

In case I, you are seeing that the process Y is failed at this point of time, when it will recover then it will try to restart from this checkpoint. If it is restarting from this checkpoint or it is failing then, the send of this particular message m_1 will be lost, why because it is restarting at Y_1 . Hence m_1 will become orphan message. So, if m_1 becomes orphan message and it is being received at a process X, this will trigger X to roll back at its checkpoint. Now we see that process X after it restarts, it sends another message n_2 meanwhile; the previous message n_1 also is received, that is called m_2 .

Now, this particular message m_2 , which is received has no sender, that is the sender of n_1 that is the send of a n_1 is not there. So, this particular message is orphan which will trigger the second rollback. And once it is rolled back then the received of this particular message, which is now becoming orphan this also will rollback. So, this particular process will continue to rollback forever, both of them will rollback forever and this is called a livelock situation.

(Refer Slide Time: 27:22)



Different rollback recovery schemes, the other one is called log based rollback recovery scheme. Log based rollback recovery schemes are of three types: Pessimistic logging, optimistic logging and causal logging. We will see them in more details. Similarly checkpoint based rollback recovery scheme are also three types: Uncoordinated checkpointing then, coordinated checkpointing; coordinated checkpointing is also of two types blocking and non blocking and communication induced checkpointing is also there are two types; model based and index based checkpointing method.

(Refer Slide Time: 27:58)



So, checkpointing based recovery schemes.

(Refer Slide Time: 28:01)

Checkpoint Based Recovery: Overview

- 1. Uncoordinated Checkpointing:** Each process takes its checkpoints independently
- 2. Coordinated Checkpointing:** Process coordinate their checkpoints in order to save a system-wide consistent state. This consistent set of checkpoints can be used to bound the rollback
- 3. Communication-induced Checkpointing:** It forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, as we have seen that they have the three different types, uncoordinated, coordinated and communication induced checkpointing. Uncoordinated checkpointing here the each process takes its checkpoints independently initial. This is quite simple, but the implications are more severe at the time of recovery. Coordinated checkpointing that is a processes, together they coordinate to take the checkpoints in order to capture the consistent global checkpoints. This, consistent global checkpoints can be useful for the rollback recovery without much effort

Third type of checkpointing is called communication induced pointing. It forces each process to take checkpoint based on the information which is piggybacked on the message application it will receive from the other processes. This kind of checkpoint we have already seen in a global state recording algorithm given by the lamport.

(Refer Slide Time: 29:08)

1. Uncoordinated Checkpointing

- Each process has autonomy in deciding when to take checkpoints
- Advantages
 - The **lower runtime overhead** during normal execution
- Disadvantages
 - **Domino effect** during a recovery
 - **Recovery from a failure is slow** because processes need to iterate to find a consistent set of checkpoints
 - Each process maintains **multiple checkpoints** and periodically invoke a garbage collection algorithm
 - Not suitable for application with frequent output commits
- The processes record the dependencies among their checkpoints caused by message exchange during failure-free operation

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, let us see the uncoordinated checkpointing. Here each process has autonomy in deciding when to take checkpoints. Advantage is that, it has lower runtime overhead during the normal course of execution, but the disadvantage is that this kind of uncoordinated checkpointing will suffer from domino effect during the recovery time.

So, recovery from failure is also slow why because, at the time of recovery they have to check and find out they have to search for a consistent set of checkpoints, because all local check points are uncoordinated, so now, at the time of recovery they have to do this. We will maintain a multiple checkpoints and periodically invoke a garbage collection algorithm. This kind of method is not suitable for application with frequent output commits.

(Refer Slide Time: 29:59)

Example: Direct dependency tracking technique

- Assume each process P_i starts its execution with an initial checkpoint $C_{i,0}$
- $J_{i,x}$: checkpoint interval, interval between $C_{i,x-1}$ and $C_{i,x}$
- When P_j receives a message m during $J_{j,y}$, it records the dependency from $J_{i,x}$ to $J_{j,y}$, which is later saved onto stable storage when P_j takes $C_{j,y}$

Cloud Computing and Distributed Systems Failures & Recovery Approaches

The dependency, direct dependency tracking technique, which is shown here in this particular example, says that, let P_i be the process, which starts its execution by recording its initial checkpoint. Now i_x is an interval between $C_{i,x-1}$ and $C_{i,x}$ the checkpoints between these two checkpoints this interval is called i_x .

So, when P_j process receives a message m during that the interval that is, j in j process the interval is y . It records the dependency from the interval of a process i th interval to the interval of j th interval and which is later stored into the stable storage when P_j takes its checkpoint $C_{j,y}$.

(Refer Slide Time: 31:10)

2. Coordinated Checkpointing

- **Blocking Checkpointing**

After a process takes a local checkpoint, to prevent orphan messages, it remains blocked until the entire checkpointing activity is complete

Disadvantages

- The computation is blocked during the checkpointing

- **Non-blocking Checkpointing**

-The processes need not stop their execution while taking checkpoints

-A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Blocking checkpointing: So, after a process takes a local checkpoint to prevent the orphan messages, it remains blocked until the entire checkpointing activity is complete. The disadvantage of blocking checkpointing is that the computation is blocked during the checkpointing process or a checkpointing duration.

Nonblocking checkpointing: The process need not stop their execution, while taking the checkpoints. The fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent.

(Refer Slide Time: 31:50)

Example

Example (a) : checkpoint inconsistency

- message m is sent by P_0 after receiving a checkpoint request from the checkpoint coordinator
- Assume m reaches P_1 before the checkpoint request
- This situation results in an inconsistent checkpoint since checkpoint
- $C_{1,x}$ shows the receipt of message m from P_0 , while checkpoint $C_{0,x}$ does not show m being sent from P_0

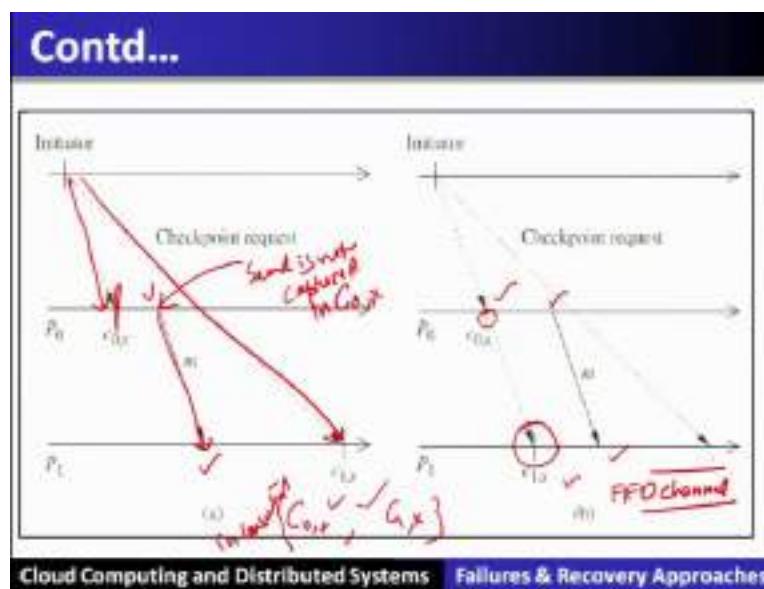
Example (b) : a solution with FIFO channels

- If channels are FIFO, this problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, forcing each process to take a checkpoint before receiving the first post-checkpoint message.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, example, here the checkpoint inconsistency, that we have already seen that the message m is send by P_0 , after receiving the checkpoint request from the checkpoint coordinator.

(Refer Slide Time: 32:04)



Here in this example. So, the checkpoint coordinator will send the message to take a checkpoint that will be received. So, assume m reached P_1 before the checkpoint request.

So, this particular m reaches here to P1, before this particular before this request. This situation results in an inconsistent checkpoint, since the checkpoints c1,x shows the receipt of a message m from p0, while the checkpoint c0,x does not show that m being sent from. So, here we can see this particular situation, in this checkpoint the send of m is not recorded, not captured in the checkpoint c0,x; whereas, this receive of m is captured here in c1,x. Hence this collection 0,x and this is an inconsistent checkpoint.

Now, another example shows that if, a FIFO channel is considered, if the channels are FIFO this problem can be avoided by preceding the first post checkpoint message on each channel by the checkpoint request, forcing each process to take a checkpoint before receiving the first post checkpoint message. So, here in this particular situation, so, after taking a checkpoint, it assumes a FIFO (Refer Time: 33:56) This particular checkpoint message will reach at this instant. So, therefore, the message which is sent after taking checkpoint at P 0, will also be received at a later point of time. So, this is possible if we assume a FIFO channel, in this particular situation.

(Refer Slide Time: 34:27)

3. Communication-induced Checkpointing

- **Two types of checkpoints**
-autonomous and forced checkpoints

- Communication-induced checkpointing piggybacks protocol-related information on each application message
- The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line
- The forced checkpoint must be taken before the application may process the contents of the message
- In contrast with coordinated checkpointing, no special coordination messages are exchanged

Cloud Computing and Distributed Systems Failures & Recovery Approaches

The third type of checkpointing is called communication induced checkpointing. This is also of two types of, autonomous and forced checkpoint. So, communication induced checkpointing, the piggybacks protocol, that is the related information on the application messages will be sent the receiver of each message uses the piggyback information to determine, if it has to take a forced checkpoint or to advance the global recovery line.

The fourth checkpoint must be taken before the application may process the content of the message. In contrast with the coordinated checkpointing, no special coordination messages are exchanged here in this particular situation.

(Refer Slide Time: 35:10)

Contd...

- Two types of communication-induced checkpointing
 - model-based checkpointing and
 - index-based checkpointing.

In '*model-based checkpointing*', the system **maintains checkpoints and communication structures** that prevent the domino effect or achieve some even stronger properties.

In '*index-based checkpointing*', the system **uses an indexing scheme** for the local and forced checkpoints, such that the checkpoints of the same index at all processes form a consistent state.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

There are two types of communication: Induced checkpointing, model based checkpointing and indexed based checkpointing. In model based checkpointing, the system maintains the checkpoints and communication structures that prevents the domino effect or to achieve some even stronger properties. In index based checkpointing, the system uses an indexing scheme for local and forced checkpoints such that, the checkpoints of the same index at all the processes forms a consistent state.

(Refer Slide Time: 35:39)

The slide has a blue header containing the title 'Log-based Rollback Recovery Schemes'. At the bottom, there is a blue footer bar with the text 'Cloud Computing and Distributed Systems' and 'Failures & Recovery Approaches'.

Let us see the log based rollback recovery scheme.

(Refer Slide Time: 35:43)

The slide has a blue header containing the title 'Log-based Rollback Recovery: Overview'. Below the title is a list of bullet points describing the scheme:

- It combines checkpointing with logging of nondeterministic events.
- It relies on the **piecewise deterministic (PWD) assumption**, which postulates that all nondeterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's determinant (all info. necessary to replay the event).
- By logging and replaying the nondeterministic events in their exact original order, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed.
- Log-based rollback recovery is in general attractive for applications that frequently interact with the outside world which consists of input and output logged to stable storage.

At the bottom, there is a blue footer bar with the text 'Cloud Computing and Distributed Systems' and 'Failures & Recovery Approaches'.

So, it combines the checkpointing with the logging of nondeterministic events. It relies on piecewise deterministic assumptions, which postulate that all nondeterministic events, that a process executes can be identified and that the information necessary to replay each event during the recovery can be logged in the events determinants. By logging and replaying the nondeterministic event in their exact original order, the process can deterministically recreate its pre failure state even if, by state has not been checkpointed.

So, log based rollback recovery is in general attractive for the application, that frequently interact with the outside world, which consists of input and output logged to the stable storage.

(Refer Slide Time: 36:35)

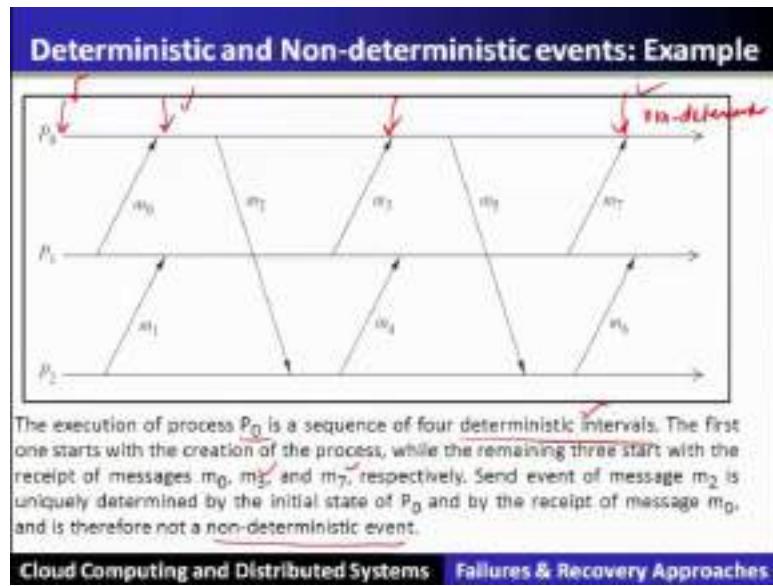
Contd...

- A log-based rollback recovery makes **use of deterministic and nondeterministic events in a computation**.
- **Deterministic and Non-deterministic events**
 - Non-deterministic events can be the receipt of a message from another process or an event internal to the process
 - a message send event is **not** a non-deterministic event,
 - the execution of process P_0 is a sequence of four deterministic intervals
 - Log-based rollback recovery assumes that all non-deterministic events can be identified and their corresponding determinants can be logged into the stable storage.
 - During failure-free operation, each process logs the determinants of all non-deterministic events that it observes onto the stable storage

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, log based rollback recovery makes use of, deterministic and nondeterministic events in the computation. So, deterministic and nondeterministic events, so nondeterministic event can be the receipt of a message from another process or the event, which is internal to a to a process. A message send is not a nondeterministic event. The execution of a process P0 is a sequence of four deterministic events in the following figure

(Refer Slide Time: 37:05)



So, here this is the creation of this event is a nondeterministic event. Then receive of the message m_0 is also a nondeterministic event. The receipt of a message m_3 is also nondeterministic event and the receipt of a message m_7 is also nondeterministic event. So, here it is shown that the execution of process P_0 is a sequence of four deterministic event. The first one starts the creation of a process. These are the four deterministic, so these are the nondeterministic events and in this example it is shown as four different intervals of a deterministic event.

The first one starts at the creation of a process that I have already told, while the remaining three start with the receipt of a message m_0 , m_3 and m_7 respectively. Send event of a message m_2 , this is not uniquely determined by the initial state of P_0 and by the receipt of a message m_0 . Therefore, it is not a nondeterministic event. So, these are all only four are nondeterministic events, which will divide into four deterministic intervals.

(Refer Slide Time: 38:28)

No-orphans consistency condition

Let e be a non-deterministic event that occurs at process p

Depend(e)
-the set of processes that are affected by a non-deterministic event e . This set consists of p , and any process whose state depends on the event e according to Lamport's happened before relation

Log(e)
-the set of processes that have logged a copy of e 's determinant in their volatile memory

Stable(e)
-a predicate that is true if e 's determinant is logged on the stable storage

always-no-orphans condition
 $\forall e : \neg \text{Stable}(e) \Rightarrow \text{Depend}(e) \subseteq \text{Log}(e)$

Cloud Computing and Distributed Systems Failures & Recovery Approaches

No orphan consistency conditions, let e be a nondeterministic event, that occurs at a process, p depend e a set of processes that are affected by a nondeterministic event, is denoted by this. This set consist of p and any process whose state depends on the event e according to the lamports happened before relation; log e , the set of processes that have logged a copy of e 's determinants, in their volatile memory.

Stable e a predicate that is true, if e is determinant is logged on the stable storage. Always no orphan conditions says that for all e , not orph $\text{stable } e$ implies that it is $\text{depend } e$ is a subset of $\text{log } e$; that means, it is there in the $\text{log } e$.

(Refer Slide Time: 39:24)

Log-based recovery schemes

- Schemes differ in the way the determinants are logged into the stable storage.

1. **Pessimistic Logging:** The application has to block waiting for the determinant of each nondeterministic event to be stored on stable storage before the effects of that event can be seen by other processes or the outside world. It simplifies recovery but hurts the failure-free performance.
2. **Optimistic Logging:** The application does not block, and the determinants are spooled to stable storage asynchronously. It reduces failure free overhead, but complicates recovery.
3. **Casual Logging:** Low failure free overhead and simpler recovery are combined by striking a balance between optimistic and pessimistic logging.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, log based recovery schemes differ in the way determinants are logged onto the stable storage. Therefore, it is called as pessimistic logging optimistic and casual logging. The application has to block waiting for the determinant of each nondeterministic event to be stored on the stable storage before the effects of that event can be seen by the other processes or the outside world. It simplifies the recovery, but hurts the failure free performance.

Optimistic logging the application does not block and determines and the determinants are spooled to the stable storage asynchronously. It reduces the failure free overhead, but complicates the recovery method. Casual logging is, low failure free overhead simpler recovery is combined by striking a balance between optimistic and pessimistic.

(Refer Slide Time: 40:18)

1. Pessimistic Logging

- Pessimistic logging protocols assume that a failure can occur after any non-deterministic event in the computation
- However, in reality failures are rare

Synchronous logging:

- $\forall e: \neg \text{Stable}(e) \Rightarrow |\text{Depend}(e)| = 0$
- If an event has not been logged on the stable storage, then no process can depend on it.
- stronger than the always-no-orphans condition

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, let us see the pessimistic; pessimistic logging protocol, assume a failure can occur after any nondeterministic event in the computation. However, in reality these failures are rare. So, this is shown in the example

(Refer Slide Time: 40:35)

2. Optimistic Logging

- Processes log determinants asynchronously to the stable storage
- Optimistically assume that logging will be complete before a failure occurs
- Do not implement the *always-no-orphans* condition
- To perform rollbacks correctly, optimistic logging protocols track causal dependencies during failure free execution
- Optimistic logging protocols require a non-trivial garbage collection scheme
- Pessimistic protocols need only keep the most recent checkpoint of each process, whereas optimistic protocols may need to keep multiple checkpoints for each process

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, another scheme is called optimistic logging, here the processes log determinants asynchronously to the stable storage. So, optimistically assumes that, logging will be complete before the failure occurs. This is also shown in the example.

(Refer Slide Time: 40:52)

3. Causal Logging

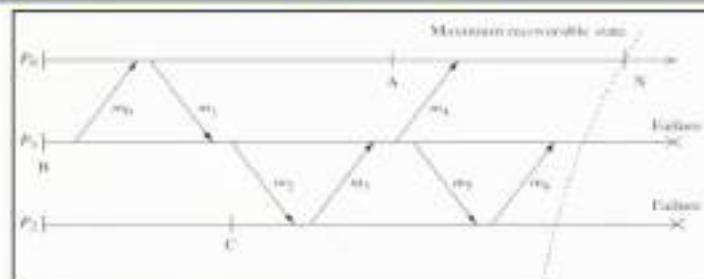
- Combines the advantages of both pessimistic and optimistic logging at the expense of a more complex recovery protocol
- Like optimistic logging, it does not require synchronous access to the stable storage except during output commit
- Like pessimistic logging, it allows each process to commit output independently and never creates orphans, thus isolating processes from the effects of failures at other processes
- Make sure that the always-no-orphans property holds
- Each process maintains information about all the events that have causally affected its state

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Third one is called causal logging; it combines the advantage of both pessimistic and optimistic locking, at the expense of more complicated recovery protocols.

(Refer Slide Time: 41:03)

Causal Logging: Example



Messages m_5 and m_6 are likely to be lost on the failures of P_1 and P_2 at the indicated instants. Process P_0 at state X will have logged the determinants of the nondeterministic events that causally precede its state according to Lamport's happened-before relation. These events consist of the delivery of messages m_0 , m_1 , m_2 , m_3 , and m_4 . The determinant of each of these non-deterministic events is either logged on the stable storage or is available in the volatile log of process P_0 .

Cloud Computing and Distributed Systems Failures & Recovery Approaches

This also is shown in this running example.

(Refer Slide Time: 41:06)

The slide has a white background with a blue rounded rectangle at the top containing the title. At the bottom, there is a blue footer bar with white text.

Checkpointing and Recovery Algorithms

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Checkpointing and recovery algorithms.

(Refer Slide Time: 41:08)

The slide has a white background with a blue rounded rectangle at the top containing the title. At the bottom, there is a blue footer bar with white text.

Koo-Toueg Coordinated Checkpointing Algorithm

- Koo and Toueg (1987) proposed a coordinated checkpointing and recovery technique that takes a consistent set of checkpoints and avoids 'domino effect' and 'livelock problems' during the recovery
- It includes 2 parts:
 - (i) The checkpointing algorithm and
 - (ii) The recovery algorithm

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Koo and Toueg coordinated checkpointing algorithm, this algorithm is given in 1987. This is a proposed for coordinated checkpointing and rollback recovery techniques, that takes constant set of checkpoints checkpointing and avoid the domino effect and the livelock problem, during the recovery in the algorithm. So, this algorithm has two parts; the first part is called checkpointing algorithm, the second part is called recovery algorithm.

(Refer Slide Time: 41:39)

Contd...

(i) The Checkpointing Algorithm

-**Assumptions:** FIFO channel, end-to-end protocols, communication failures do not partition the network, single process initiation, no process fails during the execution of the algorithm

-Two kinds of checkpoints: permanent and tentative

- Permanent checkpoint:** local checkpoint, part of a consistent global checkpoint

- Tentative checkpoint:** temporary checkpoint, become permanent checkpoint when the algorithm terminates successfully

The checkpointing algorithm assumes the FIFO channel and the end to end protocol communication failures do not partition the network. A single process will initiate this algorithm and no process fails during the execution of the algorithm.

Now, this algorithm checkpointing takes two kinds of checkpoints that is, the first one is called the tentative checkpoint and then it will be made permanent checkpoint. Tentative checkpoint is a temporary checkpoint, will become permanent checkpoint when the algorithm terminates successfully. A permanent checkpoint is a local checkpoint, part of the consistent global checkpoint.

(Refer Slide Time: 42:20)

Contd...

Checkpointing Algorithm:
2 phases

1. The initiating process takes a tentative checkpoint and requests all other processes to take tentative checkpoints. Every process can not send messages after taking tentative checkpoint. All processes will finally have the single same decision: do or discard
2. All processes will receive the final decision from initiating process and act accordingly

Correctness: for 2 reasons

- Either all or none of the processes take permanent checkpoint
- No process sends message after taking permanent checkpoint

Optimization: maybe not all of the processes need to take checkpoints (if no change since the last checkpoint)

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Let us see the checkpointing algorithm. It comprises of two phases; the first phase says that the initiating process takes a tentative checkpoint and requests all other process to take the tentative checkpoints.

Every process cannot send message, after taking the tentative checkpoint. All processes will finally, have a single same decision do or discard. The second phase says that, all processes will receive the final decision from initiating process and act accordingly. So, this algorithm is correct for a two reason, either all or none of the processes will take the permanent checkpoint and no process sends the message after taking a permanent checkpoint. Hence this algorithm is correct.

There are some optimizations, maybe not all process need to take the checkpoints. So, if there is no change in the state since, the last checkpoint, so therefore, there is a possibility that some of the process may not required to take a checkpoint. Hence that optimization if, it is included then it will also reduce the overhead of checkpointing algorithm.

(Refer Slide Time: 43:30)

Contd...

(ii) The Rollback Recovery Algorithm:

- Restore the system state to a consistent state after a failure with assumptions: single initiator, checkpoint and rollback recovery algorithms are not invoked concurrently
- **2 phases**
 1. The initiating process send a message to all other processes and ask for the preferences – restarting to the previous checkpoints. All need to agree about either do or not.
 2. The initiating process send the final decision to all processes, all the processes act accordingly after receiving the final decision.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

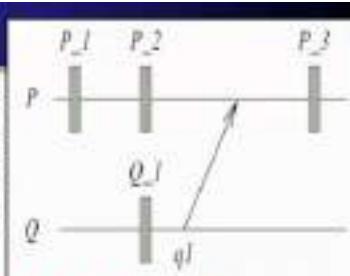
The second part of this algorithm is called, rollback recovery algorithm. It restores the system state to a consistent state after failure with the assumption that; it is a single initiator checkpoint and rollback recovery algorithm are not in invoked concurrently.

Rollback recovery algorithm also of two phases: The first phase says that, the initiating process will send a message to all other process and ask for the preferences, that is restarting to a previous checkpoints all need to agree about either do or not. So, the initiating process will send the final decision to all the processes.

(Refer Slide Time: 44:11)

Example

Suppose **P** wants to establish a checkpoint at **P_3**. This will record that **q1** was received from **Q** - to prevent **q1** from being orphaned, **Q** must checkpoint as well



- Thus, establishing a checkpoint at **P_3** by **P** forces **Q** to take a checkpoint to record that **q1** was sent
- An algorithm for such coordinated checkpointing has two types of checkpoints - tentative and permanent
- **P** first records its current state in a tentative checkpoint, then sends a message to all other processes from whom it has received a message since taking its last checkpoint
- Call the set of such processes **T**

Cloud Computing and Distributed Systems Failures & Recovery Approaches

All the processes act accordingly, after receiving the final decision. Here this is an example. Here suppose P wants to establish a checkpoint at P3, this will record that q1 was received from Q, to prevent from being orphaned q must checkpoint as well. Thus establishing the checkpoints at P3 by P, forces Q to take a checkpoint to record that Q was sent. An algorithm for such coordinative checkpointing has two types of checkpoint: Tentative and permanent. So, here then, it will send a message to all the process from whom, it has received the message since, taking its last checkpoint, call that such a process.

(Refer Slide Time: 44:54)

Contd...

The message tells each process in Π (e.g., Q), the last message, m_{pq} , that P has received from it before the tentative checkpoint was taken. If m_{pq} was not recorded in a checkpoint by Q, to prevent m_{pq} from being orphaned, Q is asked to take a tentative checkpoint to record sending m_{pq} .

- If all processes in Π , that need to, confirm taking a checkpoint as requested, then all tentative checkpoints can be converted to permanent.
- If some members of Π , are unable to checkpoint as requested, P and all members of Π abandon the tentative checkpoints, and none are made permanent.
- This may set off a chain reaction of checkpoints.
- Each member of Π can potentially spawn a set of checkpoints among processes in its corresponding set.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

So, the message tells each process, the last message m_{pq} that p has received from its tentative checkpoint and recorded the checkpoint by Q to prevent m_{pq} that message Q_1 from being orphaned and q is asked to take a tentative checkpoint, to record sending m_1 . If all the processes in the checkpoint need to confirm as requested, then this particular tentative checkpoint can be converted to the permanent checkpoint.

(Refer Slide Time: 45:23)

Contd...

- **Correctness:** resume from a consistent state
- **Optimization:** may not recover all, since some of the processes did not change anything

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Now, as far as the correctness is concerned, we can see here that will be resume from the consistent state. There is some optimizations why because, it is not that all processes are required to be recovered for example; since the last checkpoint there is no activity. Hence this particular process should not be rolled back. So, these are the few optimizations which are possible in this algorithm.

(Refer Slide Time: 45:48)

Other Algorithms for Checkpointing and Recovery	
Algorithm	Basic Idea
Juang–Venkatesan (1991) algorithm for asynchronous checkpointing and recovery	Since the algorithm is based on asynchronous checkpointing, the main issue in the recovery is to find a consistent set of checkpoints to which the system can be restored. The recovery algorithm achieves this by making each processor keep track of both the number of messages it has sent to other processors as well as the number of messages it has received from other processors.
Manivannan–Singhal (1996) quasi-synchronous checkpointing algorithm	The Manivannan–Singhal quasi-synchronous checkpointing algorithm improves the performance by eliminating useless checkpoints. The algorithm is based on communication-induced checkpointing, where each process takes basic checkpoints asynchronously and independently, and in addition, to prevent useless checkpoints, processes take forced checkpoints upon the reception of messages with a control variable.
Peterson–Kearns (1993) algorithm based on vector time	The Peterson–Kearns checkpointing and recovery protocol is based on the optimistic rollback. Vector time is used to capture causality to identify events and messages that become orphans when a failed process rolls back.
Helary–Mostafacou–Netzer–Raynal (2000, 1997) communication-induced protocol	The Helary–Mostafacou–Netzer–Raynal communication-induced checkpointing protocol prevents useless checkpoints and does it efficiently. To prevent useless checkpoints, some coordination is required in taking local checkpoints.

There are other algorithm for checkpointing and rollback recovery, which are shown here, Juang and Venkatesan, Manivannan and Single, Peterson and Kearns, Helary and

Mostefaoui. All these algorithms deals with the checkpointing and rollback recovery with some heuristics involved.

(Refer Slide Time: 46:12)

Conclusion

- **Rollback recovery achieves fault tolerance** by periodically saving the state of a process during the failure-free execution, and restarting from a saved state on a failure to reduce the amount of lost computation.
- There are three basic approaches for checkpointing and failure recovery: (i) **uncoordinated**, (ii) **coordinated**, and (iii) **communication induced checkpointing** and for log based: (i) Pessimistic, (ii) Optimistic and (iii) Casual Logging
- Over the last two decades, **checkpointing and failure recovery** has been a very active area of research and several checkpointing and failure recovery algorithms have been proposed. In this lecture, we described '**Koo-Toueg Coordinated Checkpointing Algorithm**' and given an overview of other algorithms.

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Conclusion rollback recovery, achieves the fault tolerance. There are three different approaches: Uncoordinated, coordinated and communication induced and for log based pessimistic, optimistic and casual. Over the two decades checkpointing and failure recovery has been an active research and several algorithms for rollback have proposed. We have discussed one prominent algorithm in this direction Koo-Toueg coordinated checkpointing algorithm

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 16
Design of Key-Value Stores

(Refer Slide Time: 00:16)

The slide has a blue header bar with the word "Preface" in white. Below it is a white content area with a red header "Content of this Lecture:". A bulleted list follows:

- In this lecture, we will discuss the design and insight of **Key-value/NoSQL stores** for today's cloud storage systems.
- We will also discuss one of the most popular cloud storage system i.e. **Apache Cassandra** and different consistency solutions.

At the bottom, there is a dark blue footer bar with two items: "Cloud Computing and Distributed Systems" and "Design of Key-Value Stores".

Design of Key Value Stores; preface content of this lecture, we will discuss the design and under the (Refer Time: 00:22) of new generation storage that is called key value store or it is also called as a NoSQL store, which is being provided as a services by the cloud storage system. We will also discuss one of the popular storage system provided by the cloud providers which uses underlying the key value store that is called Cassandra; we will also cover different consistency solutions.

(Refer Slide Time: 01:07)

The Key-value Abstraction

- (Business) Key → Value
- (flipkart.com) item number → information about it
- (easemytrip.com) Flight number → information about flight, e.g., availability
- (twitter.com) tweet id → information about tweet
- (mybank.com) Account number → information about it

Cloud Computing and Distributed Systems Design of Key-Value Stores

So, as I told you that the new generation storage system which are being provided by most of the cloud systems are now providing the abstraction of a key value store. So, let us understand what do you mean by key value store. So, key value store can be understood by different services which are running on the internet. For example, the Flipkart used to sell the items; every item is having a particular unique number that is called item number. And this item number has various information such as the product cost, name of the product, manufacturer, who purchased rate and how much quantity is available.

Item number becomes the key and all the information related to that item becomes the value. Similarly, if let us say we want to book a flight ticket through some service then the flight number will become the key and related to that flight number that information about the flight and the availability of seat and other information becomes the value.

Similarly, the Twitter; tweet every tweet is having an id call tweet id and tweet id is having the information about the tweet all the details about that particular tweet who send it and so on. Similarly, online banking provides the information in the form of the account number. So, account number becomes a key and who owns that account, how much money is there in the account and other details are the value. So, every business here deals with this a key and the value kind of notion; hence we will see that key value

abstraction is very much required in different businesses. And we will see how this abstraction is provided in a new generation storage system to using the cloud.

(Refer Slide Time: 03:43)

The Key-value Abstraction (2)

- It's a dictionary datastructure.
 - Insert, lookup, and delete by key
 - Example: hash table, binary tree
- But distributed.
- Seems familiar? Remember Distributed Hash tables (DHT) in P2P systems?
- Key-value stores reuse many techniques from DHTs.

Cloud Computing and Distributed Systems Design of Key-Value Stores

Key value abstraction though looks very simple can be implemented using the traditional approaches like dictionary data structures which supports insert, lookup, delete by the key; for example, hash table and binary tree. But the drawback or the problem with that is scheme is that it can run on single server machine as a process.

So, for a small data set this all was workable, but when the data becomes huge then the single server will not work; hence the distributed system is required to manage the key value store that is and also large number of clients who are accessing this huge amount of information for that the distributed system is required. So, instead of hash table the concept of similar to the hash table is being supported in distributed hash table in peer to peer systems that we have already seen can be used in the designing of such key value store.

(Refer Slide Time: 05:00)

Is it a kind of database ?

- Yes, kind of
- Relational Database Management Systems (RDBMSs) have been around for ages
- MySQL is the most popular among them
- Data stored in tables
- Schema-based, i.e., structured tables
- Each row (data item) in a table has a primary key that is unique within that table
- Queried using SQL (Structured Query Language)
- Supports joins

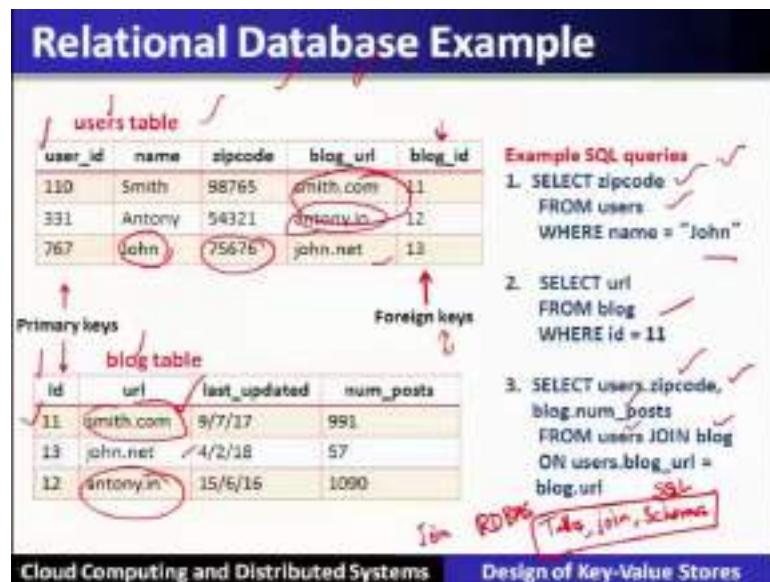
Cloud Computing and Distributed Systems

Design of Key-Value Stores

As far as key value store let us see whether is it a kind of database. Yes it is a kind of data base let us understand what is let us recall; what is the RDBMS: Relational Data Base Management System provided the language to access the data through the keys called MySQL and in RDBMS data is stored in the form of tables; which follows the schema that is the these tables are structured.

And every row is called a data item has a primary key that will uniquely identify a row or a tuple in that particular table. This kind of database supports the operations called join across multiple tables to fetch the data from it.

(Refer Slide Time: 06:12)



Let us see this particular example here; we have shown 2 different tables one is the users table the other is the blog table in the RDBMS format. The user table has the primary key as user id and the blog table has the primary key as the id in the blog table. This primary key a blog table is also being provided as blog id in the user table hence this is called a foreign key in the terms of RDBMS. Now you can generate the SQL queries for example, the query comprising of one table is shown over here. So, the name field which is having the John in this table user will print the zip code which is being queried by a customer. Similarly, from the blog table the id whose value is 11 his url will be printed.

Now comes a query which requires 2 different tables user and a blog to be joined where in the user blog url is equal to the blog url. So, here we can see this particular entry is common this is also common and so also this common. So, they will join using Cartesian product and then it will find out the users zip code and the blog num posts.

So, this is called a join operation; if 2 different tables are required for a Cartesian product and then it can basically satisfy the query and supports it. So, in this relational database we have seen the tables, we have seen the join operation, we have seen the schema which is being provided by RDBMS. And on top of it the SQL language which will basically allow us to do these operations on the database.

(Refer Slide Time: 09:10)

Mismatch with today's workloads

- **Data: Large and unstructured:** Difficult to come out with schemas where the data can fit
- **Lots of random reads and writes:** Coming from millions of clients.
- **Sometimes write-heavy:** Lot more writes compare to read
- **Foreign keys rarely needed**
- **Joins infrequent**

Cloud Computing and Distributed Systems Design of Key-Value Stores

However, in the earlier times that was all fine to involve the database and serve a particular organization, but now today's workload is across the globe. Hence the workload is quite heavy to be tackled by RDBMS; there are different requirements of in today's work load. For example, data is very large and cannot be structured that is called unstructured; that means, it cannot follow any schemas.

The second mismatch in today's workload is that lot of random reads and writes are required. So, it is a heavy reads and heavy write operations are being supported compared to the read. So, heavy write operations are supported sometimes and foreign key is rarely required and join is very infrequent in today's workload.

(Refer Slide Time: 10:24)

Needs of Today's Workloads

- Speed
- Avoid Single point of Failure (SPoF)
- Low TCO (Total cost of operation and Total cost of ownership)
- Fewer system administrators
- Incremental Scalability
- Scale out, not scale up

Cloud Computing and Distributed Systems

Design of Key-Value Stores

So, to deal with this workload these are the following characteristics which are required to be handled in the modern storage system; that is the speed single point of failure to be avoided total cost of operation. And a total cost of ownership should be low fewer system administration and incremental scalability and scale out not the scaling up.

(Refer Slide Time: 10:56)

Scale out, not Scale up

- **Scale up = grow your cluster capacity by replacing with more powerful machines**
 - Traditional approach
 - Not cost-effective, as you're buying above the sweet spot on the price curve
 - And you need to replace machines often
- **Scale out = incrementally grow your cluster capacity by adding more COTS machines (Components Off the Shelf)**
 - Cheaper
 - Over a long duration, phase in a few newer (faster) machines as you phase out a few older machines
 - Used by most companies who run datacenters and clouds today

Cloud Computing and Distributed Systems

Design of Key-Value Stores

So, scale out means that we can incrementally grow your data center or your cluster by adding more machines; that becomes the cheaper way of managing the data center. On

the other hand, earlier times the entire cluster capacity was replaced by with more machines that is called scale of that is not required these days in the data center scenario.

(Refer Slide Time: 11:30)

Key-value/NoSQL Data Model

- NoSQL = "Not Only SQL"
- Necessary API operations: **get(key) and put(key, value)**
 - And some extended operations, e.g., "CQL" in Cassandra key-value store
- **Tables**
 - "Column families" in Cassandra, "Table" in HBase, "Collection" in MongoDB
 - Like RDBMS tables; but ...
 - May be unstructured: May not have schemas
 - Some columns may be missing from some rows
 - Don't always support joins or have foreign keys
 - Can have index tables, just like RDBMSs

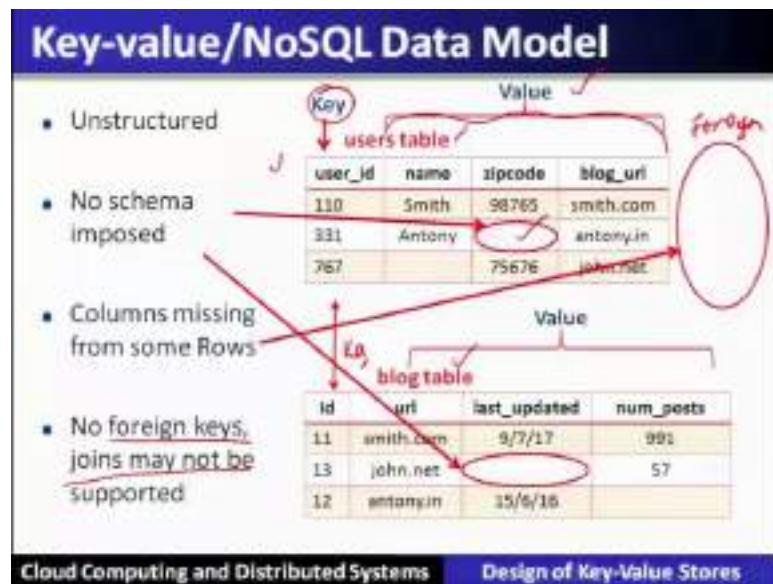
Cloud Computing and Distributed Systems Design of Key-Value Stores

So, let us see that the solution here is the key value abstraction; key value is store which is also called as a no SQL data model. So, no SQL is not only SQL or not only SQL which provides the operations in the form of APIs that is the get by a key and a put the key and a value.

So; that means, get with the key; that means, it will fetch the value of the key which is maintained by the data store. Put the key value means that it is going to be updated with the new values using the put operation which is supported by an API. So, as far as the new generation data is store such as Cassandra provides a similar kind of language which is similar to SQL, but not exactly SQL it is called Cassandra query language which will allow to use the key value store as per as tables are concerned here the tables are called column families in Cassandra. But it is called table in Hbase it is called collection in MongoDB that is an NoSQL data storage.

These terms are now redefined now these particular tables are unstructured; that means, they will not follow any schema; that means, that is all the columns may be missing from some of the rows and so on. And, also this kind of NoSQL database do not supports join operations or do not have the foreign key concept they can have the index tables just like RDBMS.

(Refer Slide Time: 13:47)



So, let us see these differences with RDBMS its highly unstructured database for example, here although these data stores are called tables. For example, the same user table if we consider user id will be the key and the remaining fields are called value. Similarly in another table that is a blog table id is called a key and rest are all called values.

This is highly unstructured in the sense it is not following the schema some of the column values are missing from some other row that is fine. And there is no foreign key which is being supported here, since there is no foreign key has the joins are also not supported.

(Refer Slide Time: 14:52)

Column-Oriented Storage

NoSQL systems often use column-oriented storage

- RDBMSs store an entire row together (on disk or at a server)
- NoSQL systems typically store a column together (or a group of columns).
 - Entries within a column are indexed and easy to locate, given a key (and vice-versa)
- **Why useful?**
 - Range searches within a column are fast since you don't need to fetch the entire database
 - E.g., Get me all the blog_ids from the blog table that were updated within the past month
 - Search in the last_updated column, fetch corresponding blog_id column
 - Don't need to fetch the other columns

Cloud Computing and Distributed Systems Design of Key-Value Stores

This particular new generation storage is also called as a column oriented storage unlike in RDBMS which stores the entire rows together on a disc or a server NoSQL; NoSQL systems typically store the column together. So, the entries within the particular column are indexed and easy to locate given a key and vice versa. This is useful to support the range queries within a particular column, since they are fast and do not need to face the entire database for the retrieval of data for that query.

Let us go and see the design of one such system that is called apache Cassandra, which supports the new type of storage that is called key value storage or it is also called is a NoSQL. So, Cassandra is a key value store which is supported by the distributed system.

(Refer Slide Time: 16:04)

Cassandra

- A distributed key-value store
- Intended to run in a datacenter (and also across DCs)
- Originally designed at Facebook
- Open-sourced later, today an Apache project
- Some of the companies that use Cassandra in their production clusters:
 - Blue chip companies: IBM, Adobe, HP, eBay, Ericsson
 - Newer companies: Twitter
 - Nonprofit companies: PBS Kids
 - Netflix: uses Cassandra to keep track of positions in the video,

Cloud Computing and Distributed Systems Design of Apache Cassandra

That is called distributed key value store; obviously, it runs on the data center not on a single server Cassandra was originally designed at the Facebook and was later very popular and open source did by the Apache project. And many companies' uses Cassandra in their production clusters such as IBM, HP, Ericsson, Twitter, PBS kids, Netflix and so on.

(Refer Slide Time: 16:39)

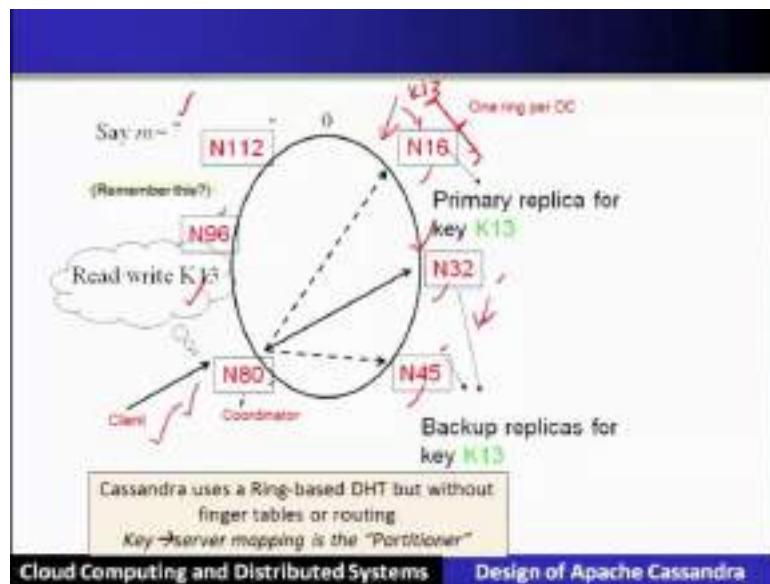
Inside Cassandra: Key -> Server Mapping

- How do you decide which server(s) a key-value resides on?

Cloud Computing and Distributed Systems Design of Apache Cassandra

Let us go and see inside the Cassandra. So, the first task we will look into that how the keys are to be mapped on the server, which will store those key and their corresponding value that is key value store how that is being stored in which server that mapping.

(Refer Slide Time: 17:09)



Cassandra uses the concept of a ring which is being provided by the chord distributed hash table. So, it uses chord like ring based distributed hash table to map the servers and the keys for key value storage, but with a difference that it does not use finger tables or it does not use routing and the key to the server mapping is done by the partitioner.

So, here for example, this is the ring which we have already seen defined for m is equal to 7; these are the different nodes which are mapped on the ring. As far as the client is concerned it want to read and write a key let us say 13. So, it will be done through the partitioner maybe for per customer there is a partitioner here. So, this particular key K 13 will be hashed that is using shravan function and it will try to map here. So, as per as this kind of storage is concerned; so, this particular will be map to the next nearest server or the node.

So, N 16 will be storing it, but it will also store the replications of it. So, the second replication will be stored on the next server; that is N 32 and the next server will be that and 40 5 will be storing up all are called replicas that is K 13 will be now maintained at 3 different servers consecutive services in Cassandra.

So, that was the key to the server mapping and this not only stores a particular key, but also maintains the replicas also here in this particular way.

(Refer Slide Time: 19:53)

Data Placement Strategies

- Replication Strategy:
 1. SimpleStrategy
 2. NetworkTopologyStrategy
- 1. **SimpleStrategy:** uses the Partitioner, of which there are two kinds
 1. RandomPartitioner: Chord-like hash partitioning
 2. ByteOrderedPartitioner: Assigns ranges of keys to servers
 - Easier for **range queries** (e.g., Get me all twitter users starting with [a-b])
- 2. **NetworkTopologyStrategy:** for multi-DC deployments
 - Two replicas per DC
 - Three replicas per DC
 - Per DC
 - First replica placed according to Partitioner
 - Then go clockwise around ring until you hit a different rack

Cloud Computing and Distributed Systems Design of Apache Cassandra

Now this particular data placement follows some strategies let us see what are the strategies which are supported in Cassandra for data placement. There are two strategies which are supported for data placement in Cassandra one is called simple strategy, the other one is called network topology strategy. Simple strategy uses the partitioner as we have seen earlier in the slide. So, simple strategy using the partitioner of which there are two different kind of partitioner here which is being applied.

So, the first kind of partitioner is called random partitioner that is nothing, but it is a chord like hash partitioning we have seen in the previous slide. This particular way that is called a random partitioner why random? Because, distributed hash table hashing using hashing it basically stores the data. The second kind of strategy of the partitioner or second type of partitioner is called byte ordered partitioner it assigns the range of keys to the server.

So, range of keys to the servers means that these range of keys are stored at one place and will become easier for the range queries; so that was the simple strategy for that replication. The second replication strategy for the data placement is called network topology strategy; that is meant for multi data center deployments. So, it says that two replicas per data center or three replica sometimes per data center.

When it calls per data center that is first replica will be placed according to the partitioner that we have seen in the simple strategy. And then it will go a clock wise around the ring until you find another server on a different rack. So, rack wise replication is done to support the rack failure that is called as network topology strategy is nowadays very much useful in the cloud scenario. So, both strategies are supporting the replication. So, there are 2 kinds of application we have covered one is called simple strategy; the other one is called network topology strategy.

(Refer Slide Time: 22:34)

The slide has a dark blue header with the title 'Snitches'. The main content area is white with black text. It lists two main categories: 'Maps' and 'Some options'. 'Maps' is described as IPs to racks and DCs, configured in `cassandra.yaml` config file. 'Some options' includes:

- **SimpleSnitch**: Unaware of Topology (Rack-unaware)
- **RackInferring**: Assumes topology of network by octet of server's IP address
 - $101.102.103.104 = x.<\text{DC octet}>.<\text{rack octet}>.<\text{node octet}>$
- **PropertyFileSnitch**: uses a config file
- **EC2Snitch**: uses EC2
 - EC2 Region = DC
 - Availability zone = rack
- Other snitch options available

At the bottom of the slide, there is a footer bar with two items: 'Cloud Computing and Distributed Systems' and 'Design of Apache Cassandra'.

Now, there are snitches; that means, when IPs are mapped to the racks and data centers. These particular configurations are stored in `Cassandra.yaml` configuration files. So, there are some options for mapping first is called simply snitch. So, this particular strategy is unaware of the topology; that means, IPs when their map we will not know from that mapping which of the racks or a data center the IP is mapped. The second one is called rack inferring will assumes the topology of the network by updates of servers IP addresses.

So; that means, by looking up using rack infer we can know that IP is mapping to which of the racks on which of the data centers and so on. Third property is called property files snitch uses a configuration file and easy to snitch uses the easy to that is easy to regions which data center and also the availability zone which rack and so on. Let us see the write operations on this kind of store.

(Refer Slide Time: 24:09)

Writes

- Need to be lock-free and fast (no reads or disk seeks)
- Client sends write to one coordinator node in Cassandra cluster
 - Coordinator may be per-key, or per-client, or per-query
 - Per-key Coordinator ensures writes for the key are serialized
- Coordinator uses Partitioner to send query to all replica nodes responsible for key
- When X replicas respond, coordinator returns an acknowledgement to the client
 - X? *why not all replicas?*

Cloud Computing and Distributed Systems Design of Apache Cassandra

So, it need to be log free and fast for write operations. So, the client sends the write request to one of the coordinator in the Cassandra cluster, the coordinator maybe per key basis or per client or per query basis. So, per key coordinator ensures the writes for the key are serialized for maintaining the consistency. The coordinator uses partitioner to send the query to all the replica nodes responsible for the key. Now an X replicas respond the coordinate returns and acknowledgement to the client; now what do you mean by X replicas, why not all replicas? That we will see; now that strategy which is applied in Cassandra.

(Refer Slide Time: 25:15)

Writes (2)

- Always writable: Hinted Handoff mechanism
 - If any replica is down, the coordinator writes to all other replicas, and keeps the write locally until down replica comes back up.
 - When all replicas are down, the Coordinator (front end) buffers writes (for up to a few hours).
- One ring per datacenter
 - Per-DC coordinator elected to coordinate with other DCs
 - Election done via Zookeeper, which runs a Paxos (consensus) variant

Cloud Computing and Distributed Systems Design of Apache Cassandra

So, the first strategy which says that always writable we say that if any replica is down, the coordinator writes to all other replica and keep the write locally until the down replica comes up and then it will do the backup. So, it is the responsibility of the coordinator to take care of the replica which is down. When all the replicas are down then coordinator will be the at the front end and it will buffer all the writes, but it is not forever, but it will be for a few hours and once these replicas are up they will be backed up. So, this is called a hinted handoff mechanism why because here the coordinator will manage to how these replicas updated it.

Now, another strategy is called one ring for data center that we know that if it is a multi data center; multiple data centers are involved then for each data center there is one ring; so, for every data center there is there is a ring. So, per data center one of these particular nodes will be elected as the coordinator. This particular coordinator job is to co ordinate with the other data centers and this election is done through the zookeeper which uses paxos like consensus for leader election.

(Refer Slide Time: 27:36)

Writes at a replica node

On receiving a write

1. Log it in disk commit log (for failure recovery) – ~~at replica node~~
2. Make changes to appropriate memtables
 - ✓ **Memtable** = In-memory representation of multiple key-value pairs
 - Typically append-only datastructure (fast) ✓
 - Cache that can be searched by key
 - Write-back as opposed to write-through ✓

Later, when memtable is full or old, flush to disk ✓

- ✓ Data File: An **SSTable** (Sorted String Table) – list of key-value pairs, sorted by key ✓
 - SSTables are immutable (once created, they don't change) ✓
 - ✓ Index file: An SSTable of (key, position in data sstable) pairs ✓
 - ✓ And a Bloom filter (for efficient search) ✓

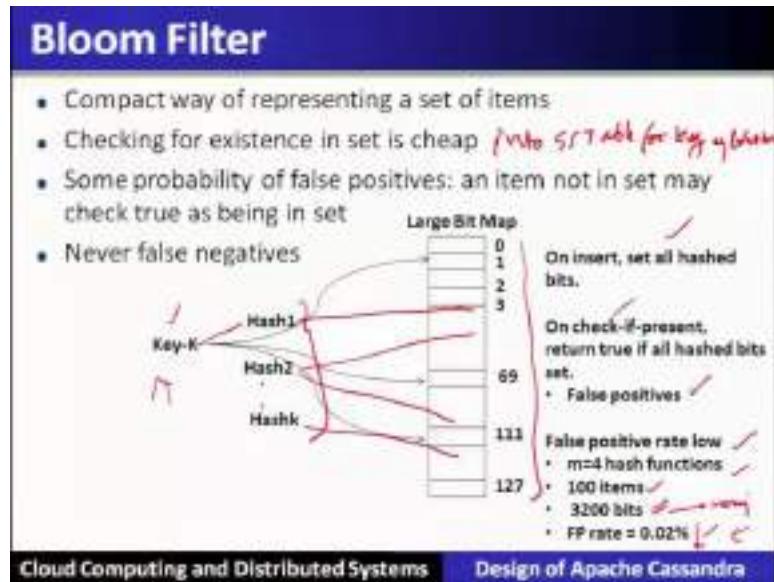
Cloud Computing and Distributed Systems Design of Apache Cassandra

On receiving a write request at the replica node; it will log in its disc for ensuring the failure recovery. We will see the algorithms which use the log file to recover from the failure so, that it will be a minimal loss or the write operations which are incomplete and if it is filled in between they can be recovered.

Make changes to the appropriate memtables; memtable is an in memory representation of multiple key value pairs. So, typically only the append only data structure is there since its in memory; so, it becomes a fast operations. So, this is maintained in the cash that can be searched by the key; it is in contrast if it is these particular changes are directly done, then it is called a write back then it is called the write through as opposed to the write back. Now when the memtable because this is in memory when it is full or it becomes old, then it will be flush to the disc; that means, it will be made permanent.

So, for that there will be a file which is called SStable is stored string table; this is a list of key value pair which are sorted by the key. These SStable are immutable that is once created they do not change, now they are indexed using a file called index file. So, that they can be checked up for the key its position of the data in the SStable and so on; for that efficient search a bloom filter is being applied on SStable.

(Refer Slide Time: 30:00)



Let us understand what a bloom filter is; a bloom filter is a randomized data structure is a compact way or presenting beside of items checking for the existence in the set becomes cheap. Because here we are checking into SStable for a particular key using the bloom filter, to support the search operation in a fast manner; this is key bloom filter has some probability of false positives; that means, if a item is not present in the set may sometimes turn to be true. But if a item is present then it will never give a false information hence it is called negative.

Never have the false negative it may have the false positive, but that is not a false positive can be handled easily; that means, it has to look up again and search for that particular item in the set. So, let us see how the bloom filter works for a particular key; there are K different hash functions which are applied and every has function will hash into the values from 0 to 127. So, for a particular key and a particular hash function one of these bits will be mapped and it will be turn to 1; similarly this also and so on.

Since this particular SSTable stores many different keys; so, all of them will be mapping the values of 1 is off if it is already there then is no change if it is 0, then return 1s. So, on insert therefore, the set all has bits will be turn on. So, on check if they are present all 1s, then it will turn true if one of them is 0 then it will be false; that means, it is not present.

There are some false positives; so false positive analysis if we see and here in the scenario when we have 4 different hash functions. And there are hundred different items stored for searching and if these number of bits are 3200; then it is analyze that the false positive rate is 0.02 percent. This can further be reduced by increasing these bits that is the bitmap; if you can increase then this can further go down. So, the bloom filter is used here to search the SSTables of particular keys; hence as a index file, bloom filter is applied for an efficient search.

(Refer Slide Time: 33:32)

Compaction

Data updates accumulate over time and SSTables and logs need to be compacted

- The process of compaction merges SSTables, i.e., by merging updates for a key
- Run periodically and locally at each server

Cloud Computing and Distributed Systems Design of Apache Cassandra

Now, another operation which is supported to deal with the multiple SSTables; which will be accumulated over the period of time and the log tables is called compaction. So, the process of compaction is to merge different SSTables that is merging and updates for a particular key and run periodically, locally at each server this compaction process.

(Refer Slide Time: 34:06)

Deletes

Delete: don't delete item right away

- Add a **tombstone** to the log
- Eventually, when compaction encounters tombstone it will delete item

Cloud Computing and Distributed Systems Design of Apache Cassandra

Let us see how the delete operation is being supported in Cassandra. Delete is supported that do not delete the item right away; instead it will add in a tombstone to the log and eventually when the compaction and counters the tombstone then it will delete the item.

(Refer Slide Time: 34:36)

Reads

Read: Similar to writes, except

- Coordinator can contact X replicas (e.g., in same rack)
 - Coordinator sends read to replicas that have responded quickest in past
 - When X replicas respond, coordinator returns the latest-timestamped value from among those X
 - (X? We will check it later.)
- Coordinator also fetches value from other replicas
 - Checks consistency in the background, initiating a **read repair** if any two values are different
 - This mechanism seeks to eventually bring all replicas up to date
- At a replica
 - A row may be split across multiple SSTables => reads need to touch multiple SSTables => reads slower than writes (but still fast)

Cloud Computing and Distributed Systems Design of Apache Cassandra

Now, let us see how the reads operations are supported. Read is similar to the write except the coordinator can contact X replicas not all; in the same rack. Coordinator sends the read request to the replicas that have responded the quickest in the past, when X replica responded the coordinator returns the latest time stamped value from among those X replicas. So, the values of X; that means, not all we will see now the coordinator also fetches the values from other replicas.

So, checks the consistency in the background and if multiple replicas have different values; that means, they are all not same. Then there is a then it will initiate the read repair if the two values are different; this mechanism seeks to eventually bring up all the replicas up to date eventually. So, at a replica a row maybe is split across multiple SSTables. So, reads need to touch multiple SSTables; that means, read becomes slower in that case than the writes, but it still it is the faster.

(Refer Slide Time: 36:05)

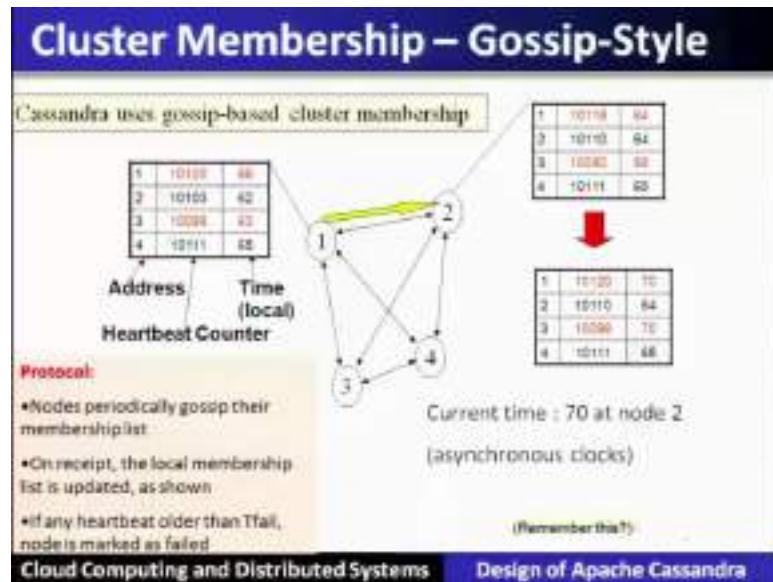
Membership

- Any server in cluster could be the coordinator
- So every server needs to maintain a list of all the other servers that are currently in the server
- List needs to be updated automatically as servers join, leave, and fail

Cloud Computing and Distributed Systems Design of Apache Cassandra

So, membership any server in the cluster could be the coordinator. So, every server need to maintain a list of all other servers; that are currently working or interacting as a servers. So, these list need to be updated automatically as new server join fail or leaves.

(Refer Slide Time: 36:29)



This membership is done using the gossip style failure detection. So, in this method or a protocol the nodes periodically gossip there membership information. When we say gossip; that means, randomly it will exchange its information to some randomly selected neighbors; not all neighbors hence it is a gossip. On the receipt of this local membership the list is updated and if any heartbeat is older than; than Tfail node then it is marked as the failed.

So, it is kind of membership; that means, it will try to figure out the nodes which are failed or which are active; hence this particular gossip style membership is maintained in Cassandra.

(Refer Slide Time: 37:38)

Suspicion Mechanisms in Cassandra

- Suspicion mechanisms to adaptively set the timeout based on underlying network and failure behavior
- **Accrual detector:** Failure Detector outputs a value (PHI) representing suspicion
- Applications set an appropriate threshold
- **PHI calculation for a member**
 - Inter-arrival times for gossip messages
 - $\text{PHI}(t) = -\log(\text{CDF or Probability}(t_{\text{now}} - t_{\text{last}})) / \log 10$
 - PHI basically determines the detection timeout, but takes into account historical inter-arrival time variations for gossiped heartbeats
- In practice, $\text{PHI} = 5 \Rightarrow 10\text{-}15 \text{ sec detection time}$

Cloud Computing and Distributed Systems Design of Apache Cassandra

Now, there is a suspension mechanism in Cassandra to detect the failures. Accrual detector is being used here the failure detector output value which is called phi representing the suspicion level. So, applications will set an appropriate threshold to deal with that; phi calculations for a member is based on inter arrival times for the gossip messages. So, phi of a particular time is equal to the log of cumulative distribution function or a probability of t now minus t last divided by log 10.

So, phi basically determines the detection time out, but takes into account the historical inter arrival time variations for the gossip heartbeats. In practice phi is equal to 5, this will imply that 10 to 15 seconds is basically the detection time in case of the failures.

(Refer Slide Time: 38:53)

Cassandra Vs. RDBMS

- MySQL is one of the most popular (and has been for a while)
- On > 50 GB data
- MySQL**
 - Writes 300 ms avg
 - Reads 350 ms avg
- Cassandra**
 - Writes 0.12 ms avg
 - Reads 15 ms avg
- Orders of magnitude faster
- What's the catch? What did we lose?

Cloud Computing and Distributed Systems Design of Apache Cassandra

Therefore, if we compare the Cassandra with RDBMS now we have seen that Cassandra is supporting the NoSQL data store in the form of key value store. So, here if we compare for the data which is more than 50 GB; MySQL has the write latency the write operation is 300 millisecond on an average speed is 350 milliseconds whereas, Cassandra will take 0.12 milliseconds and read will take 15 milliseconds. So, this is in the order of magnitudes much faster CAP theorem. So, CAP theorem was proposed by Eric Brewer of Berkeley.

(Refer Slide Time: 39:55)

CAP Theorem

- Proposed by Eric Brewer (Berkeley)**
- Subsequently proved by Gilbert and Lynch (NUS and MIT)
- In a distributed system you can satisfy atmost 2 out of the 3 guarantees:
 - 1. Consistency:** all nodes see same data at any time, or reads return latest written value by any client
 - 2. Availability:** the system allows operations all the time, and operations return quickly
 - 3. Partition-tolerance:** the system continues to work in spite of network partitions

Cloud Computing and Distributed Systems

CAP Theorem

And which was subsequently proved by Gilbert and Lynch of NUS and MIT. In the distributed system, as per as their observation is that you can satisfy at most 2 out of 3 different guarantees. What are the 3 guarantees? The first one is called Consistency will form C in the CAP, second one is called Availability that is A in the CAP, third is the Partition tolerance that is P in the CAP. So, what you mean by consistency? That consistency says that all the nodes see the same data at any time or the read returns latest value by any client called consistency.

This is ensured in the RDBMS, second aspect is called availability that is the system allows the operation at all points of time and the operations return quickly that is called availability that also is supported in most of the RDBMS. Third is called partition tolerance which says that system continues to work in spite of network partition that is failures which lead to the network partition that is called partition tolerance. So, let us see what CAP theorems says; CAP theorems says in a distributed systems or in a cloud system you can satisfy at most 2 out of 3 guarantees; that means, either consistency or availability you can support in the cloud system; that means, not partition tolerance and so on.

So, we will see that partition tolerance is important why? Because now data in the cloud is stored over multiple data centers; which are geographically distributed hence it has to be partition tolerance. If partition tolerance is very much required, then out of consistency and availability one of these 2 things has to be also taken up because at most 2 can be satisfied. Availability is very much important, why?

Because these services which are serving customer will otherwise lose the revenue; so, availability is also one of the important where we can compromise is called consistency. So, a compromise consistency is very much to be create and developed in the modern scenarios in the cloud computing system; so, in that perspective we will see the CAP theorem.

(Refer Slide Time: 42:46)

Why is Availability Important?

- **Availability** = Reads/writes complete reliably and quickly.
- Measurements have shown that a 500 ms increase in latency for operations at Amazon.com or at Google.com can cause a 20% drop in revenue.
- At Amazon, each added millisecond of latency implies a \$6M yearly loss.
- **User cognitive drift:** If more than a second elapses between clicking and material appearing, the user's mind is already somewhere else
- SLAs (Service Level Agreements) written by providers predominantly deal with latencies faced by clients.

Cloud Computing and Distributed Systems

CAP Theorem

So, availability that is reads and writes will complete reliably and quickly measurements have shown that 5 millisecond increase in the latency; for the operations at Amazon dot com or a Google can cost 20 % drop in the revenue in the sense many customer will move away and will buy from other commerce site.

So, at Amazon each added milliseconds of latency will imply that 6 million revenue loss per year will happen. So, latency is very important that is availability is one of the important factor or a guarantee in the cloud computing system. These guarantees are being supported by the; by the service providers to their clients for example, this Cassandra is being used by Netflix. And the Netflix requires the availability, this guarantees by for this particular service.

(Refer Slide Time: 44:12)

Why is Consistency Important?

- **Consistency** = all nodes see same data at any time; or reads return latest written value by any client.
- When you access your bank or investment account via multiple clients (laptop, workstation, phone, tablet), you want the updates done from one client to be visible to other clients.
- When thousands of customers are looking to book a flight, all updates from any client (e.g., book a flight) should be accessible by other clients.

Cloud Computing and Distributed Systems **CAP Theorem**

So, Service Level Agreements written by the providers predominately deal with the latency which are faced by the client. The second component in CAP is called consistency let us see why the consistency is important to deal and what are the guarantees in what are the scenarios. So, consistency means all the nodes see the same data at anytime and the read returns the latest written value by any client. So, when you access your bank account; where multiple clients you want to update from one client to be visible to the other clients.

That is done in the RDBMS, but in our scenario when thousands of customers are booking a flight ticket all the updates from the client should be accessible by the other clients hence it is called the consistency.

(Refer Slide Time: 45:14)

Why is Partition-Tolerance Important?

- Partitions can happen across datacenters when the Internet gets disconnected
 - Internet router outages
 - Under-sea cables cut
 - DNS not working
- Partitions can also occur within a datacenter, e.g., a rack switch outage
- Still desire system to continue functioning normally under this scenario

Cloud Computing and Distributed Systems

CAP Theorem

Third one is called partition tolerance why the partition tolerance is an important aspect in what condition that we will see. So, partitions can happen across the data centers when internet gets disconnected due to the internet router outages or the trans oceanic sea cables are cut or DNS is not working. So, partitions can occur within the data centers also for example, if the top of the racks switch is not functioning. So, all the servers within the data center will be in the partition. When in spite of these issues we want that the services should continue as normal in the scenario, hence partition tolerance is one of the important factors.

(Refer Slide Time: 46:09)

CAP Theorem Fallout

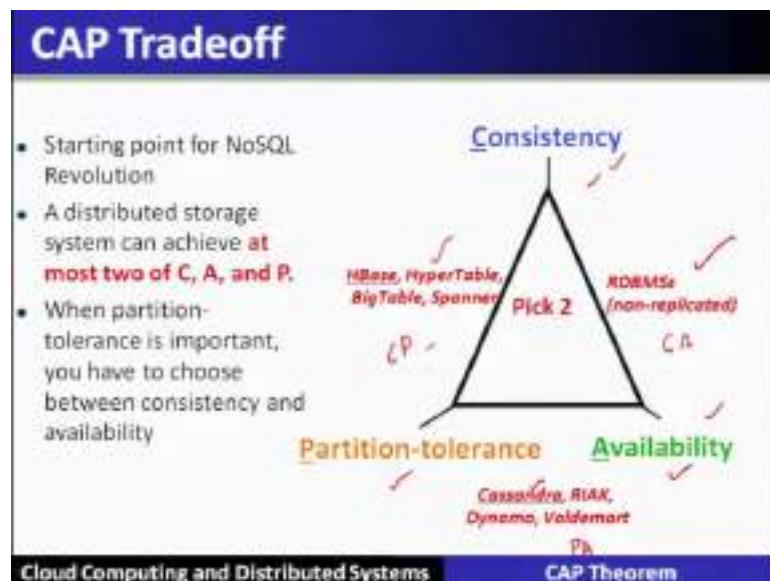
- Since partition-tolerance is essential in today's cloud computing systems, CAP theorem implies that a system has to choose between consistency and availability
- **Cassandra**
 - Eventual (weak) consistency, Availability, Partition-tolerance
- **Traditional RDBMSs**
 - Strong consistency over availability under a partition

Cloud Computing and Distributed Systems

CAP Theorem

But the CAP theorem guarantees that only 2 out of 3 important parameters can be ensured in any design of a cloud computing system. So, as far as the Cassandra is concerned Cassandra chooses to give the guarantees of availability and partition tolerance whereas, consistency it will be compromising it and that consistency is called eventual consistency that we will see now. In traditional database that is in traditional RDBMS supports a strong consistency; that means, it supports a strong consistency over availability under the partition; so, it is not partition tolerant.

(Refer Slide Time: 47:01)



Let us see through this particular figure that CAP tradeoff. So, here this is an RDBMS; the RDBMS normally runs on a particular server it is not replicated. Hence, partition tolerance is not that important, but it supports consistency and availability. Similarly the NoSQL or a key value store like Cassandra, RIAK or Dynamo and Voldemort guarantees the partition tolerance and also guarantees the availability.

Whereas, consistency is compromised; similarly another type of key value store that is HBase, HyperTable, BigTable and Spanner guarantees consistency and partition whereas, availability is concerned it can be compromised. So, we will see different products are available around the CAP trade off.

(Refer Slide Time: 48:17)

Eventual Consistency

- If all writes stop (to a key), then all its values (replicas) will converge eventually.
- If writes continue, then system always tries to keep converging.
 - Moving "wave" of updated values lagging behind the latest values sent by clients, but always trying to catch up.
- May still return stale values to clients (e.g., if many back-to-back writes).
- But works well when there are a few periods of low writes – system converges quickly.

Cloud Computing and Distributed Systems

CAP Theorem

So, let us start with the different form of consistencies which are being supported in different products. So, eventual consistency which is supported in Cassandra let us go and see the details of it. So, if all the writes stopped to a particular key then all its values that is replica will converge eventually. And if the right continues then system always tries to keep converging so; that means, since the clients may return with the still values; if there are many back to back writes happening and they are all not converged. But it works well when there are a few periods of low writes where the system converges quickly.

(Refer Slide Time: 49:09)

RDBMS vs. Key-value stores

- While RDBMS provide **ACID**
 - Atomicity
 - Consistency
 - Isolation
 - Durability
- Key-value stores like Cassandra provide **BASE**
 - Basically Available Soft-state Eventual Consistency
 - Prefers Availability over Consistency

Cloud Computing and Distributed Systems

CAP Theorem

Now, if you compare the RDBMS verses the key value stores; RDBMS supports an acid properties in the transaction whereas, the key values store like Cassandra supports just opposite to it that is called BASE; Basically Available Soft-state Eventual consistency which will prefer availability over the consistency as eventual consistency is mentioned here in the base condition.

(Refer Slide Time: 49:40)

Consistency in Cassandra

- Cassandra has **consistency levels**
- Client is allowed to choose a consistency level for each operation (read/write)
 - **ANY:** any server (may not be replica)
 - Fastest: coordinator caches write and replies quickly to client
 - **ALL:** all replicas
 - Ensures strong consistency, but slowest
 - **ONE:** at least one replica
 - Faster than ALL, but cannot tolerate a failure
 - **QUORUM:** quorum across all replicas in all datacenters (DCs)
 - What?

Cloud Computing and Distributed Systems CAP Theorem

Let us see what are the different consistency levels supported in the Cassandra and how it is supported that we will see. So, client is allowed to chose the consistency level for each operation that is the read and write. If the consistency level is any; that means, any server is basically is good enough to provide the value, it is the fastest in terms of that coordinator caches the writes and replicas quickly response to the client request.

Now in the consistency level is all; that means, all the replicas are required to be consulted and the latest writes values will be given back to the client. It ensures the strong consistency no doubt, but it will be a slower operation; if the consistency level is 1 then at least one replica responds and that response will be given back. It is faster, but it cannot tolerate the failures; now if the consistency level is the quorum and quorum across all replicas in the data center is required to get their values. So, that it may be returned; quorums we have already seen in the previous discussions.

(Refer Slide Time: 51:18)

Quorums for Consistency

In a nutshell:

- Quorum = majority
 - > 50%
- Any two quorums intersect
 - Client 1 does a write in red quorum
 - Then client 2 does read in blue quorum
- At least one server in blue quorum returns latest write
- Quorums faster than ALL, but still ensure strong consistency

Five replicas of a key-value pair

Cloud Computing and Distributed Systems CAP Theorem

Let us see about the quorum how the quorum will support the consistency. So, quorum is the majority for example, here in the figure 5 different replicas are there. So, out of 5 the majority is 3 so; that means, out of 3; 2 different quorums can be found here in the example.

If these 2 quorums intersects; so, there exist at least one server which is common in both the quorums. Now whenever the client writes chooses through the quorum; so, the client 1 writes in the red quorum. So, when it writes in the red quorum then the client 2 want to read from the bloom quorum, then in the bloom quorum all 3 different replicas will give the values; this will be having the updated value. So, that way the client will be served with the latest value here in this case; quorums are faster than all, but still it ensure the strong consistency.

(Refer Slide Time: 52:34)

Quorums in Detail

- Several key-value/NoSQL stores (e.g., Riak and Cassandra) use quorums.
- **Reads**
 - Client specifies value of R ($\leq N$ = total number of replicas of that key).
 - R = read consistency level.
 - Coordinator waits for R replicas to respond before sending result to client.
 - In background, coordinator checks for consistency of remaining $(N-R)$ replicas, and initiates read repair if needed.

Cloud Computing and Distributed Systems **CAP Theorem**

To several key value store and no SQL store such as Riak and Cassandra uses the concept of a quorum to support the strong consistency. Here the reads the client is specifies the value of $R < \text{the total number of replicas of that particular key} \leq N$ that is bounded by N . So, R is the read consistency level.

So, the coordinator waits for all replicas to respond before sending results to the client. On the background the coordinator checks for the consistency of the remaining N minus R replicas and initiates the read repair if needed. So, the reads client specifies the value of R which is bounded by the total number of replicas of that key; R will specify the read consistency level. So, the coordinator wait for R replicas to respond before sending results back to the client. In the background the coordinator checks for the consistency of remaining N minus R replicas and initiate the read repair if any is required.

(Refer Slide Time: 53:57)

Quorums in Detail (Contd..)

- Writes come in two flavors
 - Client specifies $W \leq N$
 - $W =$ write consistency level.
 - Client writes new value to W replicas and returns. Two flavors:
 - Coordinator blocks until quorum is reached.
 - Asynchronous: Just write and return.

Cloud Computing and Distributed Systems

CAP Theorem

So, that is quite simple; as far as write is concerned write will come into different flavors. So, client specifies the write consistency which is bounded by N . So, client writes the new value to W replicas and returns. That is why it is having 2 different flavors that is the coordinator blogs until the quorum is reached or it will be an asynchronous in the sense it was done write and return back. Now let us see the quorums in more details.

(Refer Slide Time: 54:40)

Quorums in Detail (Contd.)

- $R =$ read replica count, $W =$ write replica count
- Two necessary conditions:
 1. $W+R > N$
 2. $W > N/2$
- Select values based on application
 - **($W=1, R=1$):** very few writes and reads
 - **($W=N, R=1$):** great for read-heavy workloads
 - **($W=N/2+1, R=N/2+1$):** great for write-heavy workloads
 - **($W=1, R=N$):** great for write-heavy workloads with mostly one client writing per key

Cloud Computing and Distributed Systems

CAP Theorem

When R is a read replica count and W is the write replica count there are two necessary conditions have to be followed; write plus read replica count should be greater than N and $\text{write}(W) > N / 2$. So, these are the necessary conditions in the quorum. So, select the values based on the applications; if there are very few writes and reads, then $W = 1$ and $R = 1$ when there are read heavy workloads then $W = N$ and $\text{read} = 1$.

When there is a write heavy workload, then it is $W = N / 2 + 1$ and R is also $N / 2 + 1$. Now N write heavy workloads with mostly one client writing per key; then $W = 1$ and $R = N$.

(Refer Slide Time: 55:49)

Cassandra Consistency Levels (Contd.)

- Client is allowed to choose a consistency level for each operation {read/write}
 - ANY: any server (may not be replica)
 - Fastest: coordinator may cache write and reply quickly to client
 - ALL: all replicas
 - Slowest, but ensures strong consistency
 - ONE: at least one replica
 - Faster than ALL, and ensures durability without failures
 - **QUORUM**: quorum across all replicas in all datacenters (DCs)
 - Global consistency, but still fast
 - **LOCAL_QUORUM**: quorum in coordinator's DC
 - Faster: only waits for quorum in first DC client contacts
 - **EACH_QUORUM**: quorum in every DC
 - Lets each DC do its own quorum: supports hierarchical replies

Cloud Computing and Distributed Systems **CAP Theorem**

Now, let us see the Cassandra consistency level which is being supported there. So, client is allowed to choose the consistency level for each operations; that is read or write. So, any means any server may not be the replica; it is the fastest that is the coordinator may cache write and reply quickly to the client; all means all replicas it is slowest. One is means that at least one replica quorum is that quorum across all replicas in all the data centers. Local quorum means quorum in the coordinators data center each quorum means quorum in every data center.

(Refer Slide Time: 56:42)

Types of Consistency

- Cassandra offers **Eventual Consistency**
- Are there other types of weak consistency models?

Cloud Computing and Distributed Systems CAP Theorem

There are different types of consistencies which Cassandra offers. So, Cassandra offers eventual consistency what are the other type of consistency that is the weaker form of consistency model that we will see.

(Refer Slide Time: 56:55)

Eventual Consistency

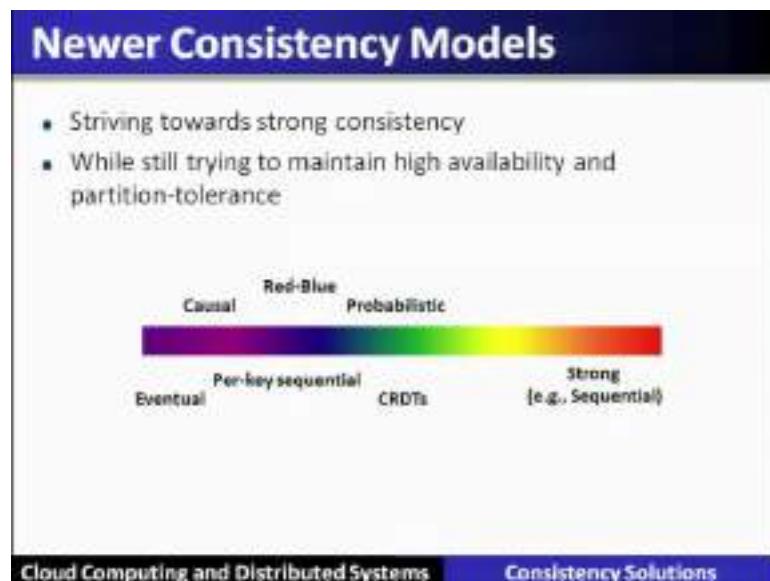
- Cassandra offers Eventual Consistency
 - If writes to a key stop, all replicas of key will converge
 - Originally from Amazon's Dynamo and LinkedIn's Voldemort systems

Cloud Computing and Distributed Systems Consistency Solutions

So, we will see different range of consistency solutions which are available starting from eventual that is the weakest, moving towards the strong consistency. Now if a strong consistency is supported then; obviously, the response will be slower and if we want a response to be faster, then we have to move towards eventual consistency.

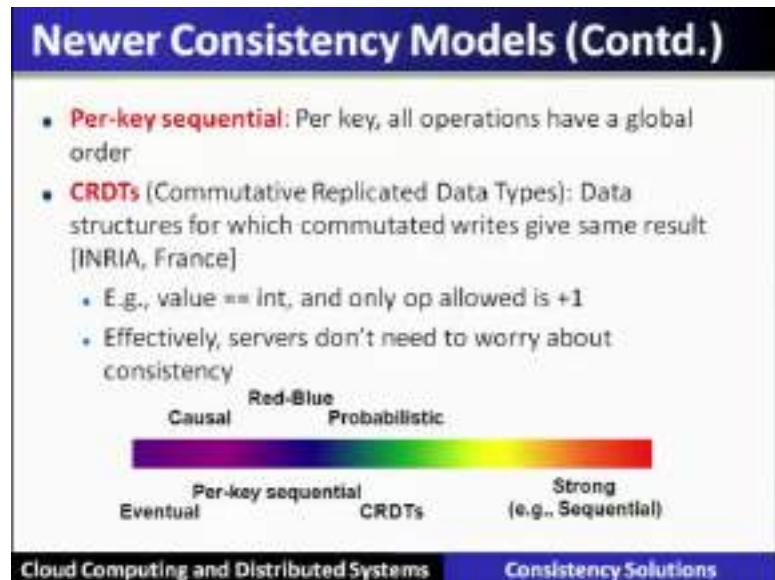
So, Cassandra supports eventual consistency. So, if write to a key will you stop; that means, all the replica of the key will converge. So, that was originally from Amazons dynamo and LinkedIn's Voldemort system this idea was taken up and used in the Cassandra.

(Refer Slide Time: 57:48)



Now, there are other newer consistency models; some are striving towards strong consistency, while still trying to achieve the high availability and partition tolerance. So, these are some of the different schemes which are shown over here causal, red blue, probabilistic, per keys sequential, CRDTs and so on.

(Refer Slide Time: 58:12)



So, let us see these newer consistency models which are available in the literature and will also be applied in most of the products. So, per key sequential means for per key all the operations have to be ordered globally. So; that means, every client has to go through a particular coordinator and coordinator will ensure the per key sequential operations.

That means a global ordering is being followed, but the problem is of scale; scalability. The another one is called another consistency model is called CRDTs; it is a newer consistency model that is commutative replicated data types. Here the data structures for the commutated writes will give the same results. For example, if the operation plus 1 is there plus 1 means increment of a particular value. So, whose ever if let us say 2 processes uses this particular operation.

So, it is immaterial if we can exchange or we can interchange the order in which these operations are being allowed; hence it is called commutative operations. So, besides plus operator there are other such operations which are being supported through CRDTs. So, they are effectively servers do not need to worry about the consistency, but they have to deal with the operations and this is supported in CRDTs.

(Refer Slide Time: 59:52)

Newer Consistency Models (Contd.)

- **Red-blue Consistency:** Rewrite client transactions to separate operations into red operations vs. blue operations [MPI-SWS Germany]
 - Blue operations can be executed (commutated) in any order across DCs
 - Red operations need to be executed in the same order at each DC

The diagram shows a horizontal color bar representing the consistency spectrum. On the left is 'Causal' (purple), followed by 'Red-Blue' (yellow-green). On the right is 'Probabilistic' (red). Below the bar, from left to right, are 'Eventual' (purple), 'Per-key sequential' (blue), 'CRDTs' (green), and 'Strong (e.g., Sequential)' (red).

Cloud Computing and Distributed Systems Consistency Solutions

Now, the next level is called next level of consistency another newer model is called blue red blue consistency model. In red blue consistency model, the client transactions are separated into two different operations. Red operation that is commutative type and blue operations; so, the blue operations can be executed that is using the commutative type in any order across the data centers. Whereas, red need to be executed in the same order; that means, they cannot be commuted this is called red blue consistency.

(Refer Slide Time: 60:33)

Newer Consistency Models (Contd.)

Causal Consistency: Reads must respect partial order based on information flow [Princeton, CMU]

The diagram shows three clients (Client A, Client B, Client C) interacting with two keys (K1, K2) over time. Client A performs a write W(K1, 22). Client B performs a write W(K2, 55). Client C receives a read R(K1) and returns 22. Client C receives a read R(K2) and returns 55. Handwritten notes explain the causal dependencies: 'R(K1) may return 22 or 33' and 'R(K2) must return 55'. A red arrow points to the note 'R(K1) must return 22' with the handwritten note 'R(K1) - causal pre-condition'.

Red-Blue

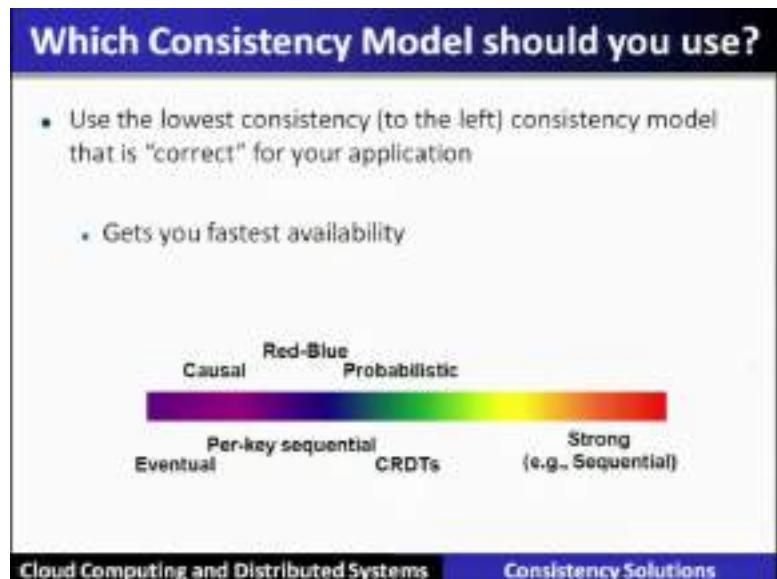
The diagram shows a horizontal color bar representing the consistency spectrum. On the left is 'Red-Blue' (yellow-green), followed by 'Causal' (purple). On the right is 'Probabilistic' (red). Below the bar, from left to right, are 'Eventual' (purple), 'Per-key sequential' (blue), 'CRDTs' (green), and 'Strong (e.g., Sequential)' (red).

Cloud Computing and Distributed Systems Consistency Solutions

Another newer kind of consistency is called causal consistency; here the reads must respect the partial order based on the information flow; that is given by the Princeton University. For example, there are 3 different clients A B and C; the client A will write down in key K 1 value 33 and after that client B will read the value of key K 1, which will be return 33. So, this particular these 2 operations write and read they are causally related; thereafter in client B will write down key K 2 with the value 55 and thereafter the client C will read the value of K 2 which also will return 55.

So, the client Bs K 2 and client Cs K 2 there and read of K 2 these 2 operations are also causally related. Finally, the client C when will read the key K 1; it must return 33 y because there is a causal path; although it is not through the message exchange, but using read and writes will follow the causal path and that is the relation which will bind this is called causal consistency model.

(Refer Slide Time: 62:15)



Now, here out of this range of consistency model which one will be useful for a particular applications is depends upon the availability; that means, which is supported by the applications.

(Refer Slide Time: 62:31)

Strong Consistency Models

- **Linearizability:** Each operation by a client is visible (or available) instantaneously to all other clients
 - » Instantaneously in real time
- **Sequential Consistency [Lamport]:**
 - » "..., the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."
 - » After the fact, find a "reasonable" ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.
- Transaction ACID properties, example: newer key-value/NoSQL stores (sometimes called "**NewSQL**")
 - » Hyperdex [Cornell]
 - » Spanner [Google]
 - » Transaction chains [Microsoft Research]

Cloud Computing and Distributed Systems Consistency Solutions

Let us see some of the strong consistency model the first one is called linearizability. So, here each operation by the client is visible or is available instantaneously to all other clients. So, this is just like RDBMS which supports this kind of consistency that is called linearizability. Now, in the systems like key value store, there is another consistency which is also categorized as a strong consistency is called sequential consistency which is given by the Lamport here the result of any execution is the same as if the operations of all the processor for executive some sequential order and operations of each individual processor appear in this sequence in the order is specified by the program.

So, here the order is not that order which is being provided in linearizability, but eventually all the processor will see the same order and that is fine in some of the applications. Now comes the acid properties there are newer NoSQL database called NewSQL which supports the transactions acid properties such as hyperdex, spanner and transaction chains by different vendors.

(Refer Slide Time: 64:10)

Conclusion

- Traditional Databases (RDBMSs) work with strong consistency, and offer ACID
- Modern workloads don't need such strong guarantees, but do need fast response times (availability)
- Unfortunately, CAP theorem
- **Key-value/NoSQL systems offer BASE**
(Basically Available Soft-state Eventual Consistency)
 - Eventual consistency, and a variety of other consistency models striving towards strong consistency
- We have also discussed the design of Cassandra and different consistency solutions.

Cloud Computing and Distributed Systems

Consistency Solutions

Conclusion traditional databases RDBMS work with the strong consistency and offers the acid properties. Whereas, the modern workloads do not need such a strong guarantees, but do need fast response time that is the availability.

Unfortunately the CAP theorem says that out of 3 any 2 can be guaranteed. We have seen the key value store NoSQL; which is not offering CAP which is not offering acid, but it is offers base that is basically available soft state eventual consistency. Eventual consistency and a variety of other consistency model striving strives towards strong consistency, we have discussed the design of Cassandra and also different consistency solutions.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 17
Design of HBase

(Refer Slide Time: 00:16)

The slide has a blue header bar with the word "Preface" in white. Below it is a white content area with a red header "Content of this Lecture:". A bulleted list follows, starting with "In this lecture, we will discuss:" and including items like "What is HBase?", "HBase Architecture", etc. At the bottom of the slide, there is a navigation bar with two tabs: "Cloud Computing and Distributed Systems" and "Design of HBase".

Preface

Content of this Lecture:

- In this lecture, we will discuss:
 - What is HBase?
 - HBase Architecture
 - HBase Components
 - Data model
 - HBase Storage Hierarchy
 - Cross-Datacenter Replication
 - Auto Sharding and Distribution
 - Bloom Filter and Fold, Store, and Shift

Cloud Computing and Distributed Systems Design of HBase

Design of a HBase: Preface; content of this lecture. In this lecture we will cover about HBase, its architecture, the components. The data model which is used in HBase and HBase storage hierarchy, Cross-Datacentre Replication, Auto Sharding and Distribution, Bloom Filter and it is use in Fold, Store and Shift.

(Refer Slide Time: 00:41)

HBase is:

- An open source NOSQL database.
- A distributed column-oriented data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) for scalability, fault tolerance, and high availability.
- Hbase is actually an implementation of the BigTable storage architecture, which is a distributed storage system developed by Google.
- Works with structured, unstructured and semi-structured data.

Cloud Computing and Distributed Systems Design of HBase

Let us see what HBase is. HBase is an open source NOSQL database. This particular HBase is distributed column oriented data store, that can scale horizontally to 1000s or 2000s of commodity servers and petabytes of indexed storage. Let us understand one term what do you mean by the horizontal scaling. By meaning of horizontal scaling is that, if let us say this is the database, and this is the stored on a particular server. As it grows, in size there are 2 methods, one is called scale up that is scale vertically, the other is called horizontally. Here in the horizontally scaling, we keep on adding the servers as we grow in the size. So, this is called or it is also called scale out of the servers.

So, we keep on adding many servers and so on. So, this particular analogy we can take from a chariot, which is driven by the horse. So, we have to keep on adding the number of horses. That is called horizontally scaling. So, this is a new kind of technology; that means, in today's world we have to scale horizontally so that as we require we can add more number of commodity servers. Hence, the scaling is possible easily with the cost effective manner. So, this particular HBase uses the scale horizontally or horizontally scaling method so that 1000s of commodity servers and more we can add to it so that it can basically stored. The column oriented data store which is distributed across all these servers. We will be looking up how HBase will do this; that means, it will maintain a distributed databases on the cloud.

So, HBase is designed to operate on top of Hadoop distributed file system. So, it assumes that Hadoop distributed file system is available, and it uses HDFS for its all storage purposes and that HDFS will allow its advantages of scalability fault tolerance and high availability to the HBase. So, HBase is actually an implementation of big table, which is an storage architecture developed by the Google. And based on this HBase is designed and made open source using Apache. So, it works with the structured unstructured and semi structured data.

(Refer Slide Time: 04:23)

HBase

- **Google's BigTable** was first “blob-based” storage system
- **Yahoo!** Open-sourced it → HBase
- Major Apache project today
- **Facebook** uses HBase internally
- **API functions**
 - Get/Put(row)
 - Scan(row range, filter) – range queries
 - MultiPut
- Unlike Cassandra, HBase prefers consistency
(over availability)

ACID
BASE + eventual consistency

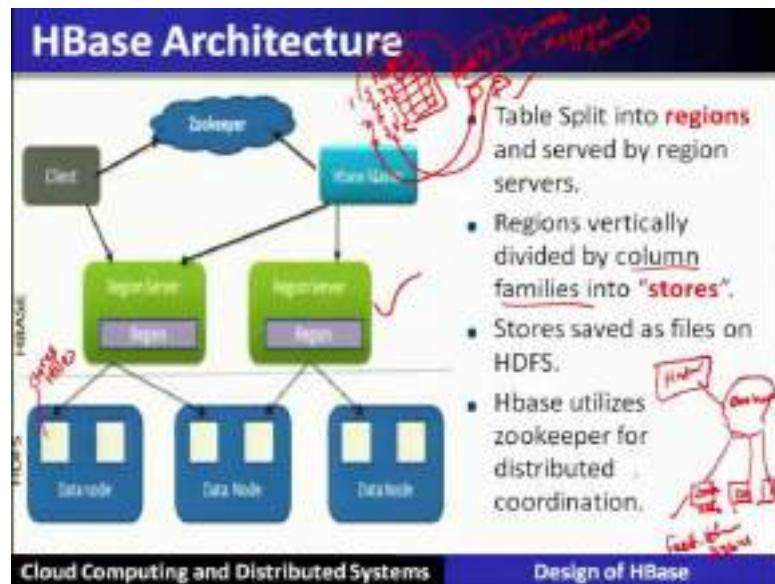
Cloud Computing and Distributed Systems Design of HBase

So, HBase is based on Google's big table, that was the first block based storage system. Yahoo open sourced it and call it as HBase. Now, major apache project today uses HBase. For example, Facebook users HBase internally in one of its stack. The API's which are supported in HBase are quite easy and simple to be used. For example, API's such as get and put, the row wise in a table of a database is being supported by get and put.

Similarly, to support the range queries that is range of the rows, let us say the ranges from a to d will also be supported using a can. So, it will fetch all rows which falls in that range to support the range queries in HBase. Multi put option is also available through API's. Here we will see that we have also already seen a Cassandra in a NOSQL store. So, unlike Cassandra HBase prefers consistency over availability.

So, both of them are NOSQL database and they follow the base model in contrast to the acid properties of the transaction. So, this is an eventual consistency. We will see in more detail how the HBase uses to perform the consistency over the availability against (Refer Time: 06:13)

(Refer Slide Time: 06:15)



So, let us go and see the architecture of HBase. HBase you can see here comprises of several components such as client, HBase master then region server. And also another important component which is called is Zookeeper which we have already started for coordination service. So, here Zookeeper will provide the coordination across providing the fault tolerant master and also different data nodes and so on. So, this architecture will be a fault tolerant distributed system, architecture which we have seen in the previous slides this Zookeeper will continue for this database or data store operations.

Here the data base which is structured in the form of a table. This particular table will be split into the regions and being served by the region s servers. So, for example, let us say we have a table which is having the rows in this manner, row 1, row 2 and so on up to row k. So, by splitting the table means we can split in the row wise. So, these splits are being served this is let us a split1 2 and 3 these splits are served by the servers. And we call them as region servers. So, also the second split and the third split. So, different region servers are there here to support them in the HBase.

Now, regions vertically divided by the column families in to the stores. We will see what do you mean by the column families. So, for the time being we just have to understand that what is basically the basic unit of a storage that will be handled through the HBase and stored on the disk. So, that is called column families. So, column families are nothing but the column names; which is having more than one column attributes that is called column families and that will be stored.

So, this is stores and these stores are saved as the files on HDFS. So, these are nothing but the stores or it is also called HFiles which are store on HDFS. We will see in more detail about this part. HBase utilizes the Zookeeper for distributed coordination; that means, the Zookeeper will ensure that all these data nodes are which are basically called as H which are called as h stores are alive and also the master is also available at all points of time.

(Refer Slide Time: 10:16)

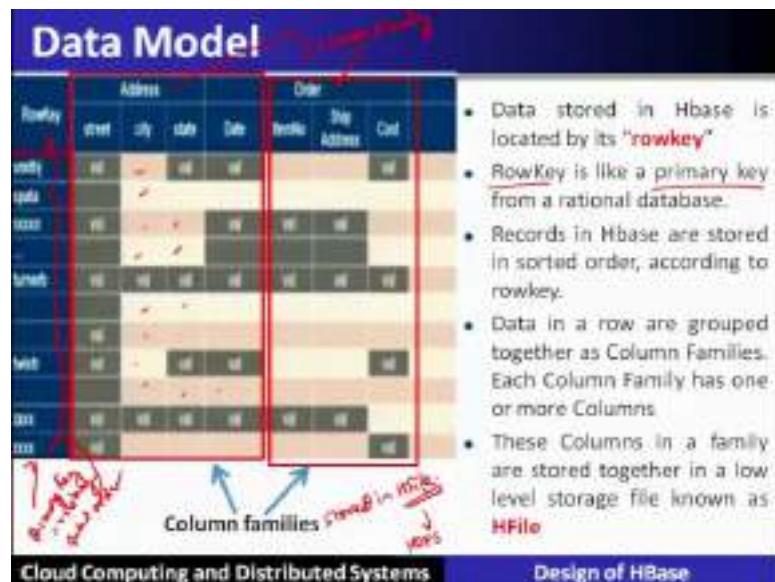
HBase Components

- **Client:**
Finds RegionServers that are serving particular row range of interest
- **Hmaster:**
Monitoring all RegionServer instances in the cluster
- **Regions:**
Basic element of availability and distribution for tables
- **RegionServer:**
Serving and managing regions
In a distributed cluster, a RegionServer runs on a DataNode

Cloud Computing and Distributed Systems Design of HBase

So, HBase components, the client first find the region servers, that are serving particularly row range of interest, then HBase master which is called h master will monitor all the regions server instances in the cluster. So, master is responsible to manage the entire cluster which is used in HBase. Then comes the region servers, this basic element of eligibility and distribution of the table. So, these region servers are serving and managing these regions in a distributed cluster a region server runs on a data node which is maintained by HDFS.

(Refer Slide Time: 11:07)

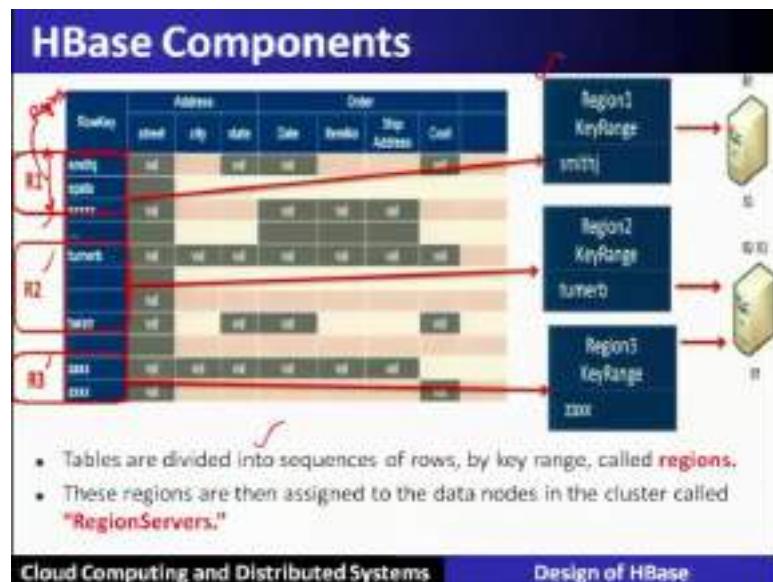


Let us see the data model in more details to understand all these parts, which are managed by HBase. So, data is stored in HBase is located by it is rowkey so, this is the rowkeys. Rowkeys is a primary key; this term is taken from the rational database. So, row key is just like a primary key now records in HBase are stored in the sorted order according to the rowkeys. So, all these rowkeys are in the sorted order.

The data in a row are grouped together as the column families. These columns in the family are stored together in a low level storage file known as HFiles. Here the example of column families, are the address and the order, which are shown here in this particular box as Column Family. So, address has various attributes such as street city state date. Similarly order Column Family has various attributes such as item number ship address cost etcetera.

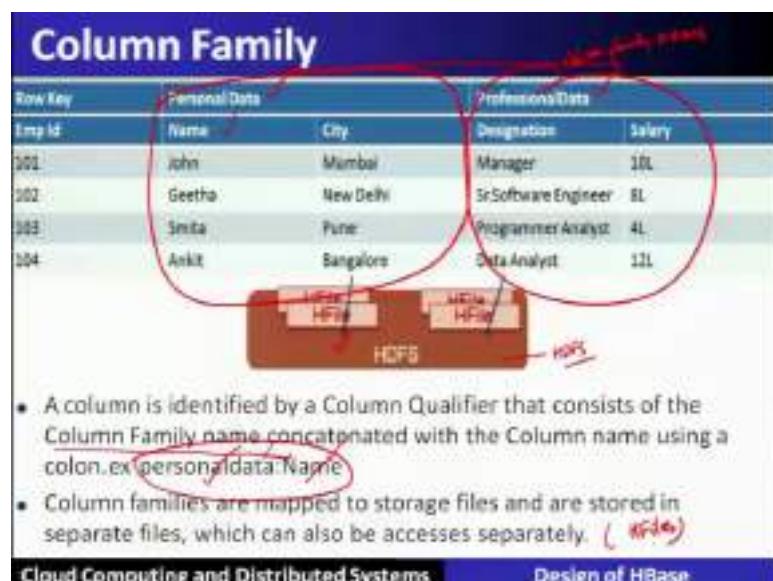
These particular attributes do not have to be essential having the values. They may be null also. So, these columns in a family together they are stored together in a low level storage file which is called HFile and which is maintained by or which is stored by HDFS.

(Refer Slide Time: 13:30)



Here same thing is also shown in this particular figure. That tables are divided into the sequence of rows by the key range and they are called the regions. So, this key range here is called the region. So, this is region 1, 2 and 3. So, each region is stored in the server which is called the region server. These regions are then assign to the data nodes in a cluster which is called the region servers.

(Refer Slide Time: 14:06)



Now as for as column families are concerned, a column is identified by a column qualifier, that consists of the Column Family name for example, here the personal data is

the Column Family name. And professional data is also a Column Family names. So, this Column Family name is concatenated with the column name using colon. And will identify the column qualifier.

For example, if you want to check the name; which is the column qualifier, then it has to be starting with the personal data followed by the name that is personal data. Colon name is an example over here. That is how the column is identified by the column qualifier. So, column qualifier name if you want to check, then it will be starting with the Column Family name and that qualifier that is the name. So, column families are map to the storage files and are stored in the separate files which can be accessed separately sample.

So, for example, here this particular Column Family personal data will be stored in the form of HFile. Similarly, is the professional data is another Column Family this also will be stored in the HFiles which is in the HDFS. So, the column families are the HFiles are map to the storage files. And are stored in a separate files which are access separately that is called HFiles.

(Refer Slide Time: 16:19)

Cell in HBase Table				
Row Key	Column Family	Column Qualifier	Timestamp	Value
John	PersonalData	City	123456790123	Mumbai
				KEY → VALUE <i>Data</i>
<ul style="list-style-type: none"> • Data is stored in HBASE tables Cells. • Cell is a combination of row, column family, column qualifier and contains a value and a timestamp • The key consists of the row key, column name, and timestamp. • The entire cell, with the added structural information, is called Key Value. 				
Cloud Computing and Distributed Systems		Design of HBase		

Cell in a HBase table. So, data is stored in HBase tables cells. Cell is a combination of row, Column Family column qualifier, and it contains a value and a timestamp. For sample let us say start with the row key, then comes the Column Family, then comes the column qualifier. And this column qualifier will have a timestamp and it is value. The key in HBase is comprised off row key Column Family column qualifier and timestamp

and this is called a key. And this column qualifier will have its own value. For example, this particular zone with their personal data. It is column qualifier city and that city value is Mumbai. So, this is the data which is being stored in the form of a cell.

So, the key consists of the row key, column name and time stamp that we have already seen here in this example. The entire cell with the added structure information is called the key value. So, this is the key and this qualifier will get the value from here in the cell. And this is how the data is stored.

(Refer Slide Time: 17:55)

HBase Data Model

- **Table:** Hbase organizes data into tables. Table names are Strings and composed of characters that are safe for use in a file system path.
- **Row:** Within a table, data is stored according to its row. Rows are identified uniquely by their row key. Row keys do not have a data type and are always treated as a byte[] (byte array).
- **Column Family:** Data within a row is grouped by column family. Every row in a table has the same column families, although a row need not store data in all its families. Column families are Strings and composed of characters that are safe for use in a file system path.

Cloud Computing and Distributed Systems Design of HBase

So, HBase data model comprises of the table, HBase organizes data into the table names are the strings, and composed of characters that are safe for use in the file system path. Rows within the table; the data is stored according to its rows, rows identified uniquely by the rowkey. Row keys do not have the data type and are always treated as the byte array. Column family's data within a row is grouped by the Column Family. Every row in the table has some column families although a row need not store the data in all its column families. In that case it will be null. So, column families are the strings and composed of characters that are safe for use in file system path.

(Refer Slide Time: 18:49)

HBase Data Model

- **Column Qualifier:** Data within a column family is addressed via its column qualifier, or simply , column. Column qualifiers need not be specified in advance. Column qualifiers need not be consistent between rows. Like row keys, column qualifiers do not have a data type and are always treated as a byte[].
- **Cell:** A combination of row key, column family, and column qualifier uniquely identifies a cell. The data stored in a cell is referred to as that cell's value.

Cloud Computing and Distributed Systems Design of HBase

Qualifier; this is the unit where the data is stored on it is also called a value. So, data within the Column Family is addressed via the qualifier or simply the column or a column qualified need not be specified in advance, column qualifier need not be consistent between the rows. Like row keys, column qualifier do not have the data type and are treated as by array. Cell, is a combination of row key Column Family and column qualifier uniquely identify the cell. Data is stored in a cell is referred to the cells value.

(Refer Slide Time: 19:29)

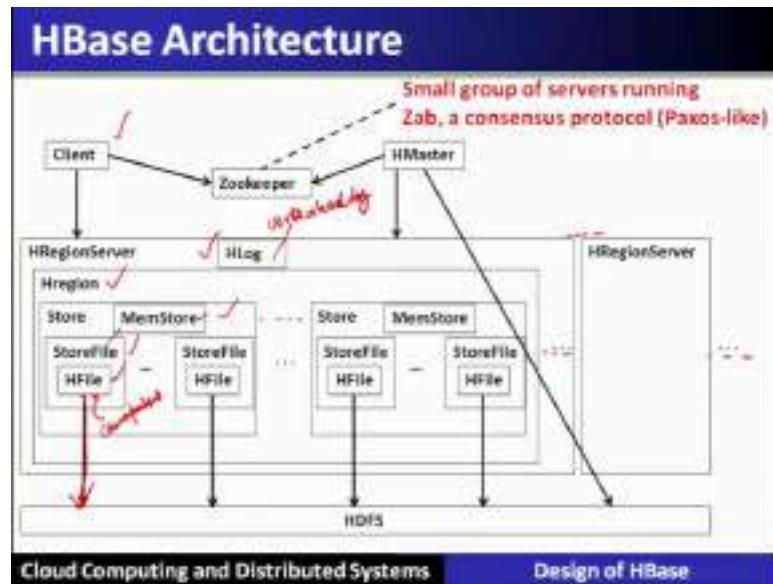
HBase Data Model

- **Timestamp:** Values within a cell are versioned. Versions are identified by their version number, which by default is the timestamp is used.
- If the timestamp is not specified for a read, the latest one is returned. The number of cell value versions retained by Hbase is configured for each column family. The default number of cell versions is three.

Cloud Computing and Distributed Systems Design of HBase

Timestamp this is also very important notion in supported in HBase. So, the values within a cell are versioned. Is versions are identified by their version number which by default is the timestamp is used. If the timestamp is not specified for the read, the latest one is returned the number of cell values versions retained by HBase is configured for each Column Family. The default number of cell versions is 3.

(Refer Slide Time: 20:01)

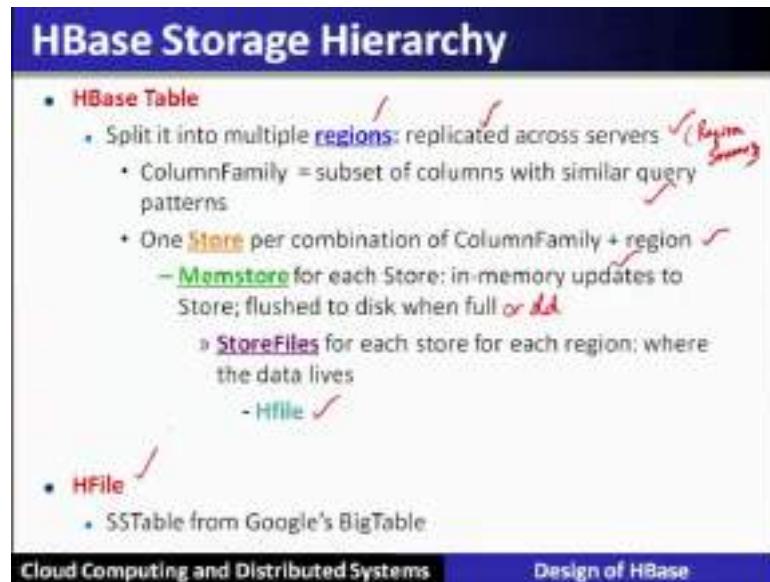


Let us see the HBase architecture in further details. So, HBase has an entity which is called a client. Now client will access the HBase through HRegionServers, which are shown here in this block. There may be more than one HRegionServers HRegionServers has within it called HLog. So, HLog is a write ahead log; that means, before the ride starts that operation is to be logged in HLog, that is called write ahead log. So, HRegionServer has HLog.

So, every HRegionServer maintains several HRegions now each HRegion will in turn maintain different stores. Each store in turn has a mem table and stores the StoreFile. StoreFile in turn has the HFile and HFile is stored in HDFS. We have already seen what is the Hfile so, it is nothing but it will store the column families.

. So, several column families together will comprise the StoreFiles. And these StoreFiles when they are accessed from the desk they will be stored in memory, that is called MemStore. So, this MemStore means as the data is accessed, it will be remain in memory and hence the accesses will be fast thereafter.

(Refer Slide Time: 22:07)



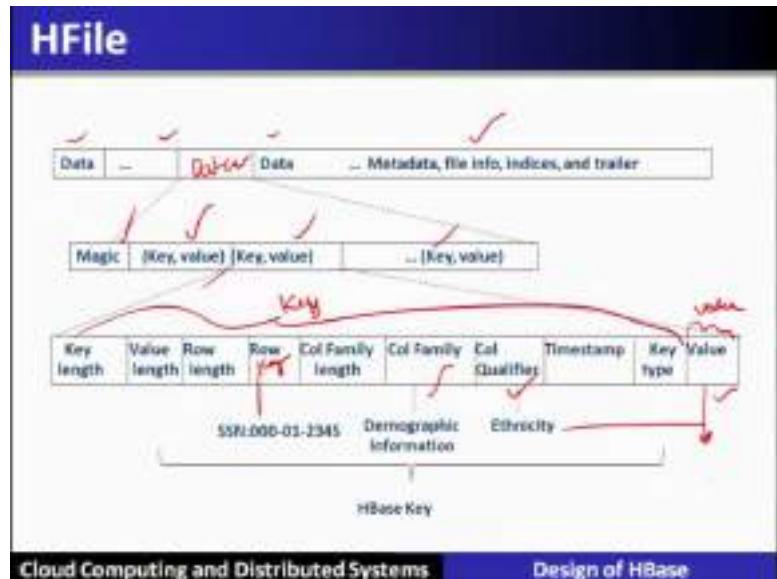
Now let us see this HBase storage hierarchy, HBase table we is split into the multiple regions, that we have already seen these regions are nothing but the range of rows, they are called regions. This is splitted rows which are called regions are now replicated across different servers and they are called as. So, the table is stored after the splitting into the rain server. Why this way the table is stored? If the table is very big which cannot be stored in one server multiple servers are used to store the tables so that not only big table, but it can be so efficiently if it is stored in multiple region servers.

This also regions are replicated across the servers. So, replication if we see as far as HDFS is concerned by default the replication number is 3. So, these are replicated so that it will be accessible very efficiently. So, the column families are nothing but a subset of the columns with similar query pattern. So, column families will support the query pattern and they are together stored. So, one store per combination Column Family and the region is there in HBase.

So, there is also a concept we have seen about the MemStore for each store. So, MemStore for each store is nothing but an in memory updates to the store. So, that will be retained, and when the disk is full or this become old then only it will be flush to the disk. This ensures the number of iOS are to be reduced in this particular manner by using the MemStore, and also the accesses are going to be fast. Now comes the StoreFile. StoreFiles for each store for each region where the data lives is maintained or is stored in

the form of Hfile. Now H it is file you know that it is stored into HDFS. So, as file we have seen the SSTable, use this SSTable is also there in Google big table design.

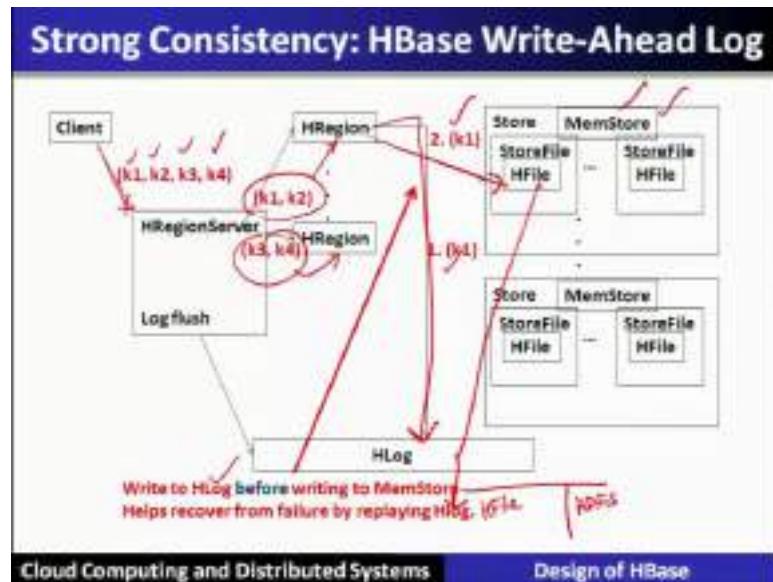
(Refer Slide Time: 25:22)



Let us go and see the details of HFile which is stored into HDFS and which is the unit of the data which is stored in sbase. HFile comprises of data, data and large amount of data followed by the meta data, file information indices and trailers. So, let us see what is there inside a data which is stored in HFile. So, data file is starting with the magic number which will uniquely identify a particular data followed by the key value pairs, several key value pairs. Let us see: what is there inside key value pair which we have already seen, but for the sake of completeness let us understand it.

. So, key comprises of; so, it has 2 parts key part and the value part. So, key comprises of key length, value length, row and ColumnFamily length, ColumnFamily, column qualifier, timestamp and the key type, then followed by it is value. The example which we have seen earlier here also we can see that the row key is nothing but a primary key over here. And then followed by a ColumnFamily name and then column qualifier: that means, the ethnicity of a particular person here in this example and this value will be about that particular name. So, this is called HBase key. And it is corresponding value which is being stored into the data part of HFile.

(Refer Slide Time: 27:31)



HBase supports strong consistency. For that it uses HLog which is called a write ahead log. Let us understand this aspect of HBase. Now client contacts to the HRegionServer for the keys k_1, k_2, k_3 and k_4 . Now, this HRegionServer will divide this particular keys as per their availability in different servers. So, k_1, k_2 is maintained here in a HRegion, and k_3 and k_4 is maintained another HRegion. So, as far as h is concerned for the keys k_1 what it will do? It will first go and write into HLog.

This is the first operation it will do so; that means, write to HLog before writing to the MemStore helps to recover from the failure by replaying an HLog. So, it will be first log and then it will send for writing that is the second step. Second step is to write to the MemStore. So, this values are written for the write operation into the MemStore. And once the MemStore is full, the disk is full or it is old then only that corresponding file will be placed into HFile which is there in HDFS.

(Refer Slide Time: 29:32)

Log Replay

- After recovery from failure, or upon bootup (HRegionServer/HMaster)
 - Replay any stale logs (use timestamps to find out where the database is with respect to the logs)
 - Replay: add edits to the MemStore

Cloud Computing and Distributed Systems Design of HBase

So, log replay, after the recovery from the failure and upon boot up the HRegionServer with the master they will replay any stale logs. According to the timestamp to find out where the database is with respect to the logs. So, using replay it will be constant after the failure. That particular entire operations which were not complete which were not persistent. So, replace will add edits to the MemStore also.

(Refer Slide Time: 30:06)

Cross-Datacenter Replication

- Single "Master" cluster
- Other "Slave" clusters replicate the same tables
- Master cluster synchronously sends HLogs over to slave clusters
- Coordination among clusters is via Zookeeper
- Zookeeper can be used like a file system to store control information
 1. /hbase/replication/state
 2. /hbase/replication/peers/<peer cluster number>
 3. /hbase/replication/rs/<chlog>

Cloud Computing and Distributed Systems Design of HBase

Now we will see the cross data centre application. Now here we have a single master, which manages the cluster and there are other slave clusters which replicates the tables.

Master cluster synchronously sends HLogs over to the slave cluster so that they can store all that logs which are created for the write operations. Co-ordination among the clusters is carried out via Zookeeper. So, Zookeeper can be used like a file system to store control information also, which are listed over here, state information than pure cluster number HLog all this particular information is stored control information is stored in the form of a file system.

(Refer Slide Time: 31:16)

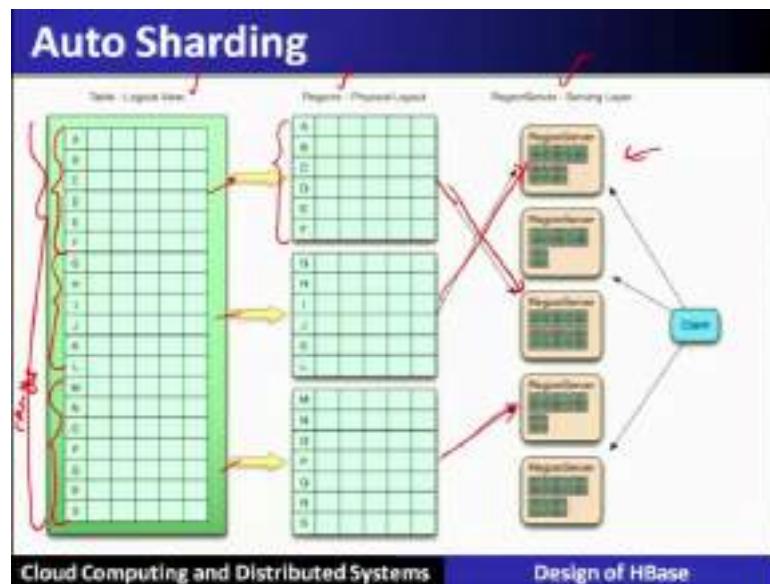
Auto Sharding and Distribution

- Unit of scalability in HBase is the Region
- Sorted, contiguous range of rows
- Spread "randomly" across RegionServer
- Moved around for load balancing and failover
- Split automatically or manually to scale with growing data
- Capacity is solely a factor of cluster nodes vs. Regions per node

Cloud Computing and Distributed Systems Design of HBase

Now there is also a process which is called auto sharding and distribution. So, that means, as we told that it supports horizontally scaling and that is provided using auto sharding. So, unit of scalability in HBase is the region, region is nothing but range of row keys, they are sorted continuous range of rows spread randomly across RegionServer moved around for load balancing and failover. So, splitted automatically or manually to scale growing the data with the growing data; so, capacity is solely the factor of cluster nodes versus the region per node.

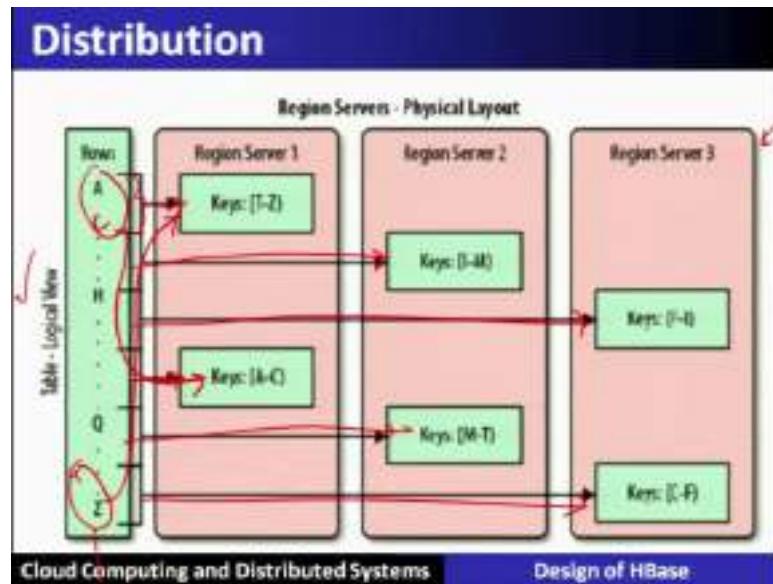
(Refer Slide Time: 32:09)



Let us see how this horizontally scaling is supported using auto sharding. So, this is the logical view of a particular table, which are sorted order in the form of rowkeys. This particular row keys are the splitted into the regions. That is in the first split up to A to F will be put in the first region. From G to L will be split into the second region. And from M to S it will go into the third split. These splits are stored these splits are called region. And these regions are stored in the region server; so, the client with the help of HRegionServer.

So, client using the master or the Zookeeper, it will directly go and access this HRegionServer and can access this particular table data. So, Zookeeper is basically the coordination service, and HBase for the client will give this information to contact the HRegionServers and directly the client accesses the data through HRegions.

(Refer Slide Time: 33:46)



Let us see the distribution again in more detail. The rows are now distributed; that means, sharding is that is the split, and how these particular splits are distributed across the region server is shown in this particular figure. So, if this is the rows according to their row keys are sorted order.

So, these different splits are maintained in the random order to different RegionServer, that we have seen here. So, for example, the first A to C key is now stored in the region server 1. Similarly, from T to Z range of keys they are also stored in the in the region server 1. So, that way these splits are organized in the region servers. So, as this particular row increases, this the number of servers are also increasing if it is, then the new particular regions are being supported in a new servers or they may be balancing with each other. And this information is maintained by the master.

(Refer Slide Time: 35:30)

Bloom Filter

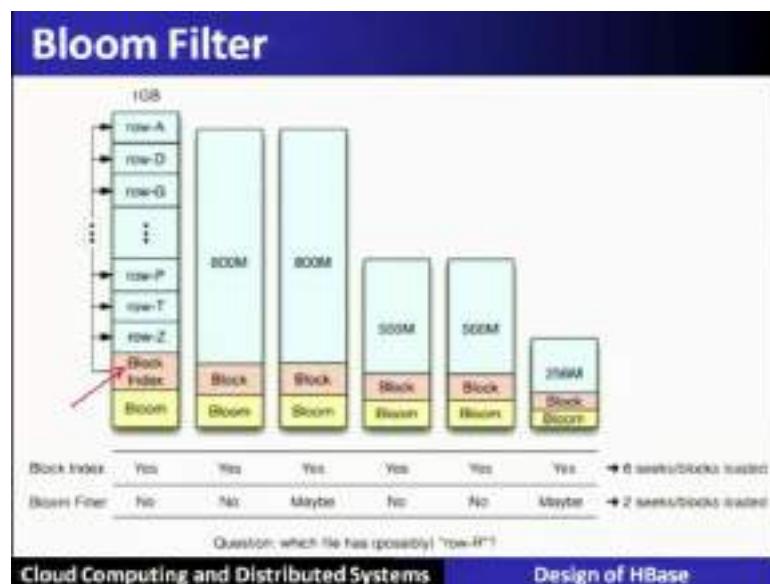
- Bloom Filters are generated when HFile is persisted
 - Stored at the end of each HFile
 - Loaded into memory
- Allows check on row or row + column level
- Can filter entire store files from reads
 - Useful when data is grouped
- Also useful when many misses are expected during reads (non existing keys)

Cloud Computing and Distributed Systems Design of HBase

Now, here there is the use of bloom filter, let us see what is the use of bloom filter here in HBase. So, bloom filters are generated when the HFile is persisted; that means, as long as the file is in Mem table. This is not done and as soon as the file is written on HDFS, the bloom filter index is generated. So, these are stored at the end of each HFile and also loaded into the memory so that whenever that file is preferred or accessed by the client; the bloom filter can hash it. And can get that information about the rows and the column families and it will access in a fast manner.

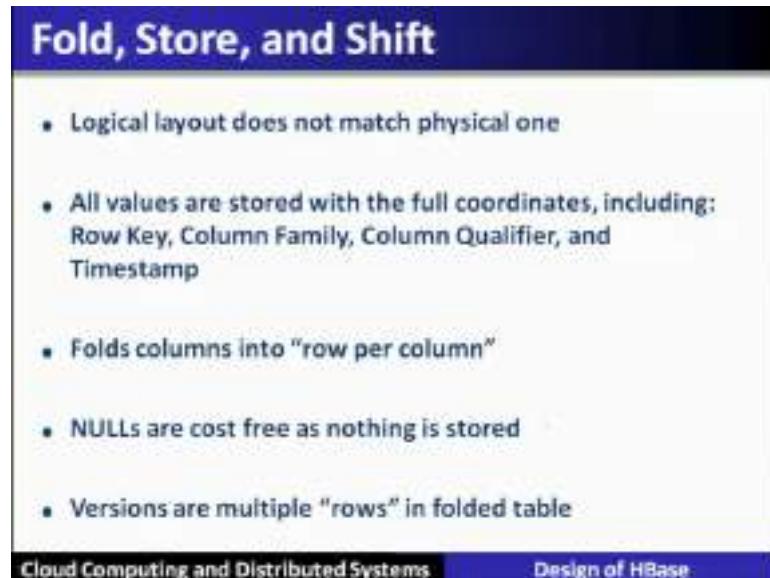
Now, bloom filter will allow to check on the rows basis. So, that indexes on the rows are done in the form of bloom filters. It can also be possible that rows and column level both together are indexed in the form of bloom filter, then an extra overhead will be added up in case rows and columns also are being indexed. So, either in the form of rows then bloom filter will create index or if let us say that rows and column level if it is together done then other indexes are created. Now, this bloom filter can filter the entire StoreFiles from the reads it is useful, when the data is grouped also useful when many misses are expected during the reads.

(Refer Slide Time: 37:19)



So, here there is a example that at the end of this table this blocks indexes are created, and being maintained to access this particular files. So, these are to be maintained at the region servers. So, that whenever there is a access whenever an access to a HFile then basically through the bloom filter indexes it can be made efficient access.

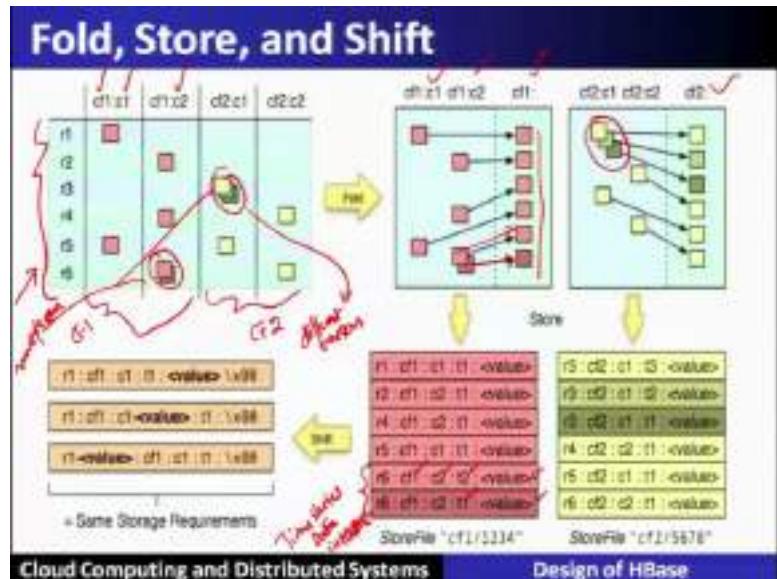
(Refer Slide Time: 37:48)



So, fold store and shift logical layout does not match the physical one all the values are stored with full coordinate's row key Column Family, Column Qualifier and Timestamp

that we have seen. And it will fold columns into the row per C column nulls are cost free, as there are nothing is stored and versions are the multiple rows in the folder table.

(Refer Slide Time: 38:12)



Let us see this example that how these information or a data is stored in HFile. Now here this is the typical table which is nothing but sorted order in the form of a row keys, r 1 to r 6 it is shown. And it is having Column Family cf 1 with the column reference. So, that is Column Family one with having a particular attribute c 1 ColumnFamily 1 will have another attributes cf 2. So, this is one Column Family which is shown as the red.

Similarly, there is another ColumnFamily cf 2, which is shown as the yellow color. Now this particular cell or this particular attribute that is called c 2 is having the multiple entries so, also here. This and this means that it has different versions, which is stored here this is folded and stored into the HFile.

Let us see how this is all being carried out. So, when this particular table is folded, let us say that this particular table that is called cf 1 is folded. So, all this particular data is stored as far as ColumnFamily one is concerned. So, multiple versions of that particular column attribute is also stored. Similarly, here also you can see that this is having multiple versions, and all are restored in the column families when it is folded up.

Here we can see that this particular entry which is r 6, having multiple entries that is r 6 with the ColumnFamily 1 C 2 is basically the ColumnFamily well attribute. It is having 2

timestamps; that means, at 2 timestamp 2 versions, these 2 different values are there. So, that means, we can also store the time series data in HBase. So, HBase supports this unstructured data, and it also supports this concept so that time series and multiple versions can also be stored with their timestamps.

(Refer Slide Time: 41:13)

Conclusion

- Traditional Databases (RDBMSs) work with strong consistency, and offer ACID
- Modern workloads don't need such strong guarantees, but do need fast response times (availability)
- Unfortunately, CAP theorem
- **Key-value/NoSQL systems offer BASE**
 - Eventual consistency, and a variety of other consistency models striving towards strong consistency
- **In this lecture, we have discussed:**
 - HBase Architecture, HBase Components, Data model, HBase Storage Hierarchy, Cross-Datacenter Replication, Auto Sharding and Distribution, Bloom Filter and Fold, Store, and Shift

Cloud Computing and Distributed Systems Design of HBase

Conclusion; traditional database is like RDBMS work with strong consistency and offer acid properties; whereas the modern workloads do not need such a strong guarantees, but do need fast response times that is in terms of high availability. Unfortunately, there is a cap theorem, which says that any 2 out of 3 we have to choose.

So, as far as Cassandra is concerned we have seen the Casandra has chosen availability with partitioning, and has compromised with the consistency that is called eventual consistency. This particular database which is HBase prefers consistency over availability with partition. So, this key values database that is NOSQL database offers n base property instead of acid property. This means basically available storage eventual consistency in this lecture. We have covered HBase architecture HBase components data model. HBase storage cross data centre replication, auto sharding and the use of bloom filter, and fold store and shift for HFiles.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 18
P2P Systems in Cloud Computing

Peer to Peer Systems in Cloud Computing.

(Refer Slide Time: 00:18)

The slide has a blue header bar with the word "Preface" in white. Below it is a white content area with a thin black border. At the top of the content area, the text "Content of this Lecture:" is written in red. Below this, there are two bullet points in black text:

- In this lecture, we will discuss the Peer to Peer (P2P) techniques in cloud computing systems.
- We will study some of the widely-deployed P2P systems such as: **Napster, Gnutella, Fasttrack and BitTorrent** and P2P Systems with provable properties such as: **Chord, Pastry and Kelips**.

At the bottom of the slide, there is a dark blue footer bar with the text "Cloud Computing and Distributed Systems" and "P2P Systems in Cloud Computing" in white.

Preface content of this lecture we will discuss peer to peer systems and we will see the internals or under the hood of peer to peer systems as a techniques, which will be applied in various cloud computing systems. We will study some of the widely deployed peer to peer systems such as Napster, Gnutella, Fasttrack, BitTorrent and peer to peer systems with the provable properties that is from academia they are Chord, Pastry and Kelips.

(Refer Slide Time: 00:58)

Need of Peer to Peer Systems

- First distributed systems that seriously focused on scalability with respect to number of nodes
- P2P techniques be abundant in cloud computing systems:
 - Key-value stores (e.g., Cassandra, Riak, Voldemort) use Chord p2p hashing

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, let us see the need of peer to peer systems. So, distributed systems that focuses on scalability with the number of nodes is one of the basic issues which is required to support large number of clients. So, to support large number of clients or highly scalable large scale distributed system is required. And therefore, we will see one such system which is widely being accepted is called peer to peer system. The techniques of peer to peer system we will discuss which we will see that is currently being applied in various cloud computing systems such as key value stores that is Cassandra, Riak, Voldemort uses the chord peer to peer systems like consistent hashing and virtual ring.

(Refer Slide Time: 02:00)

P2P Systems

Widely-deployed P2P Systems:

1. Napster
2. Gnutella
3. Fasttrack
4. BitTorrent

P2P Systems with Provable Properties:

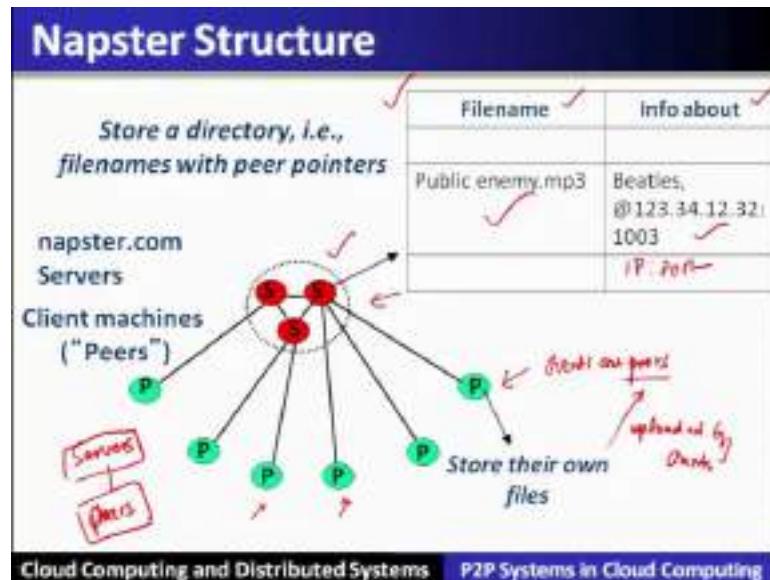
1. Chord
2. Pastry
3. Kelips

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, widely deployed peer to peer systems are such as Napster, Gnutella, Fasttrack, BitTorrent; we will discuss these particular widely deployed peer to peer systems their internals details that is under the (Refer Time: 02:18), the techniques which will focus on how these systems are evolving. Then we will

discuss peer to peer systems which are basically resultant from academia such as chord, pastry and kelips.

(Refer Slide Time: 02:38)



Let us see the Napster peer to peer system design. So, Napster is designed in the form of the peers; that means, the clients are the peers which are shown here as P. So, the clients are peers in this particular system which are shown as this particular Ps. Now these clients or these peers used to store the files which are being uploaded by the clients. Besides this there is a set of Napster servers which will store the dictionary or at the directory of filenames with peer pointers of where they are stored.

For example, the structure of the directory which the napster dot com servers used to store will be of this format that is filename and the peer pointers about the file name. For example, public enemy.mp3 is the name of the file and this information about this particular file on which machine that is IP address this particular file and the port address where this file is stored. So, this particular structure is maintained by the Napster servers. So, it has the servers and the clients they are called peers.

(Refer Slide Time: 04:43)

Napster Structure

Client

- Connect to a Napster server:
 - Upload list of music files that you want to share
 - Server maintains list of <filename, ip_address, portnum> tuples. **Server stores no files.**

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, the client used to connect to the Napster server it can upload the list of files that you want to share. The server does not stores any file, but it maintains the list of file names its IP address and the port number this particular tuple in their directory.

(Refer Slide Time: 05:05)

Napster Operations

Client (contd.)

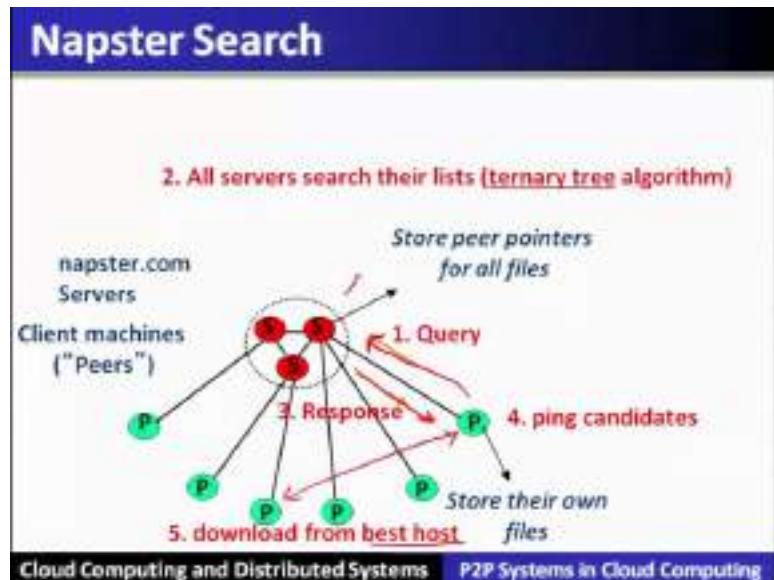
- **Search**
 - Send server keywords to search with
 - (Server searches its list with the keywords)
 - Server returns a list of hosts - <ip_address, portnum> tuples - to client
 - Client pings each host in the list to find transfer rates
 - Client fetches file from best host
- **All communication uses TCP (Transmission Control Protocol)**
 - Reliable and ordered networking protocol

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Search operation when a client want to search for a particular file with the keyword, then it will send this keyword to the server. Server searches its directory with the keyword and forms a list of successful results and returns this list of hosts that is nothing, but IP address and port number where this files are available and these information that is the list will be sent to the client. Client pings directly to these hosts which are provided in the form of list to find out the transfer rate which is being provided by the different host; transfer rate by mean that the bandwidth which is available to download the file.

So, the client fetches the file from the best possible host which basically will give the fast download; here there will, here the communications are in the form of TCP that is the reliable communication.

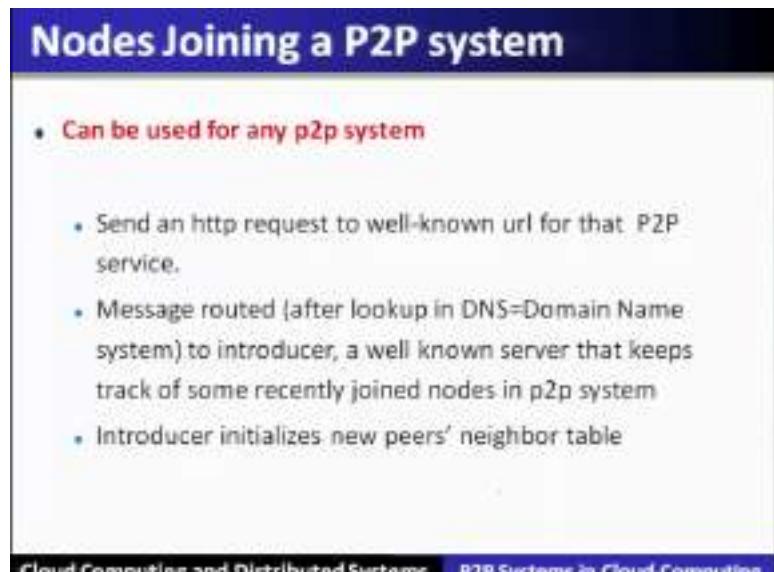
(Refer Slide Time: 06:23)



Now, let us see the servers, the Napster dot com servers are basically maintained in the form of a ternary tree algorithm; that means, each node of a tree is having 3 different child, in contrast to the binary tree having 2 childs.

Now, this particular servers will store the peer pointers for all the files and whenever the client will query this particular file with the servers, this will be checked here in its directory and send back the response that is a list of IP addresses and the port number, which has those files. Then this particular peer we directly contact to those peers which is having the file and find out the best one; best host to download from.

(Refer Slide Time: 07:37)



Now, coming to the nodes joining to this peer to peer system. So, it sends a http request to a well known url for the peer to peer service. For example, napster.com; so, the message routed after the lookup from the DNS system to that introducer which is nothing, but a well known server that keeps track of recently joined nodes in the peer to peer system. So, introducer initializes the new peers neighbour table.

(Refer Slide Time: 08:15)

Issues with Napster

- Centralized server a source of congestion
- Centralized server single point of failure
- No security: plaintext messages and passwords
- napster.com declared to be responsible for users' copyright violation
 - "Indirect infringement"
 - Next P2P system: Gnutella

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, let us see the issues which are basically affecting the Napster. So, the servers are primarily the centralized servers and hence it will be a source of congestion; also due to the centralized server there is a possibility of single point of failure. And here also there is no provision of the security and the napster dot com declared to be responsible for the copyright violation that is called indirect infringement.

(Refer Slide Time: 09:00)

Gnutella

- Eliminate the servers ✓
- Client machines search and retrieve amongst themselves
- Clients act as servers too, called servents
- Gnutella (possibly by analogy with the GNU Project) is a large peer-to-peer network. It was the first decentralized peer-to-peer network of its kind.
- [Mar 2000] release by AOL, immediately withdrawn, but 88K users by [Mar 2003]
- Original design underwent several modifications

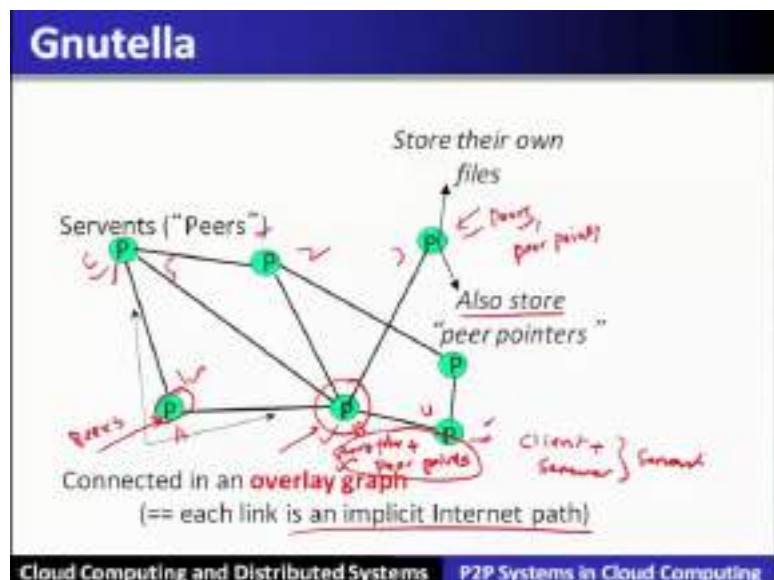
servents

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, we will see the next peer to peer system which will eliminate these problems; to eliminate these problems Gnutella peer to peer system has in its design eliminated the use of servers.

Therefore the client machine search and retrieve among themselves therefore, the client will act as a server too and therefore, they are called as the servants. So, Gnutella is a large peer to peer network and it was first decentralized peer to peer network of its kind.

(Refer Slide Time: 09:48)



So, let us see the design of Gnutella. So, Gnutella design is organized in the form of the overlay graph. So, overlays are constructed over the underlying internet path that is the servants which are shown or which are the peers in Gnutella. They are forming an overlay graph with comprising of other peers who are connected through a path that is through internet path.

So; that means, if this particular peer is having the links or the connections with the neighbouring peers; that means, this particular peer A has a internet path with B and C: similarly all other peers will have the path and form and overlay a graph. Now this particular peers will store their own files and also store the peer pointers of the neighbouring peers that is the neighbouring peers which are storing the files.

For example, this particular peer has 1,2,3,4 and 5; 5 different peers. So, this particular peer will store his own files plus all the peer pointers that is the information about the files which are stored in these neighbours in the 5 neighbours are maintained over here and that is why it is the client plus the server that is called servant.

So, Gnutella has given the concept of an overlay graph.

(Refer Slide Time: 12:09)

How do I search for a particular file?

- Gnutella **routes** different messages within the overlay graph
- Gnutella protocol has 5 main message types
 1. **Query** (search)
 2. **QueryHit** (response to query)
 3. **Ping** (to probe network for other peers)
 4. **Pong** (reply to ping, contains address of another peer)
 5. **Push** (used to initiate file transfer)
- Into the message structure and protocol
 - All fields except IP address are in little-endian format
 - 0x12345678 stored as 0x78 in lowest address byte, then 0x56 in next higher address, and so on.

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

And then provide then provide the routes for different messages to flow on the overlay graph for searching purpose and other operations. Therefore, Gnutella protocol has supported 5 different messages types the first one is the query; that means, whenever a particular client want to search a particular file that is done through the to the message type called query.

So, in the query the keyword which keyword of that is the name of a file will be given for searching. The response to that query will be collected through the QueryHit, then as per as to maintain the overlay graph ping and pong there are 2 different message types; they will probe the network for the other peers in the overlay graph and pong will be the reply to the ping. Finally, the fifth message type called push will be used to initiate the file transfer even if the responders are behind the firewall. The information which are encapsulated or packed into the message that is the message structure and the protocol follows the little endian format.

(Refer Slide Time: 13:41)

How do I search for a particular file?

Descriptor Header

Descriptor ID	15	16	TTL	Hops	Payload length
0			17	18	22

Paylod

Descriptor ID: ID of this search transaction
Payload descriptor: Type of payload
TTL: Decrement at each hop, Message dropped when ttl=0
Hops: Incremented at each hop
Payload length: Number of bytes of message following this header

Gnutella Message Header Format

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, this is the structure of the message header it starts with the descriptor ID that is for each search transaction it is having a unique ID, then follows is the type of the message that is whether it is ping pong push query and QueryHit that we have already explained.

Then comes next field which is called TTL Time To Live; now this Time To Live will control the flooding of the messages on the overlay graph that is it will be decremented at each host and the message will be dropped when TTL becomes 0. Then comes the number of hops it will be incremented at each hop and then it will be having the payload.

(Refer Slide Time: 14:39)

How do I search for a particular file?

Query (0x80)

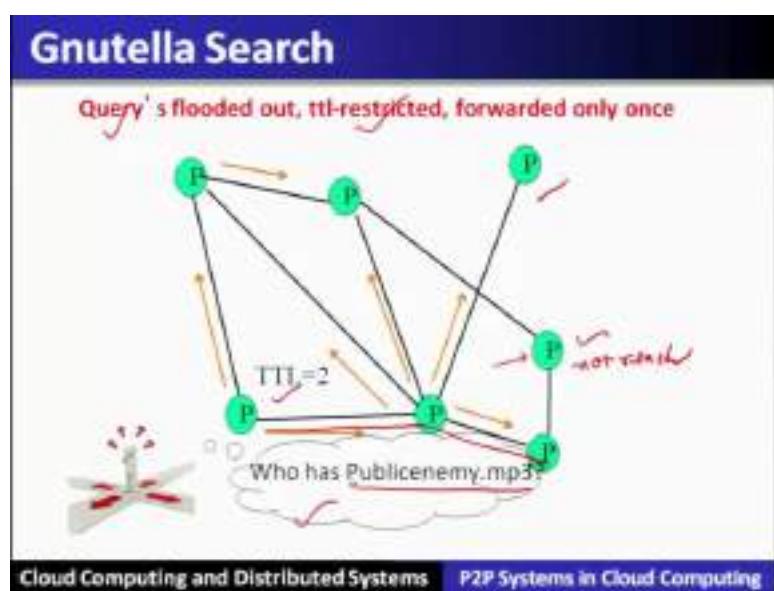
Minimum Speed	Search criteria (keywords)	
0	1	...

Payload Format in Gnutella **Query** Message

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

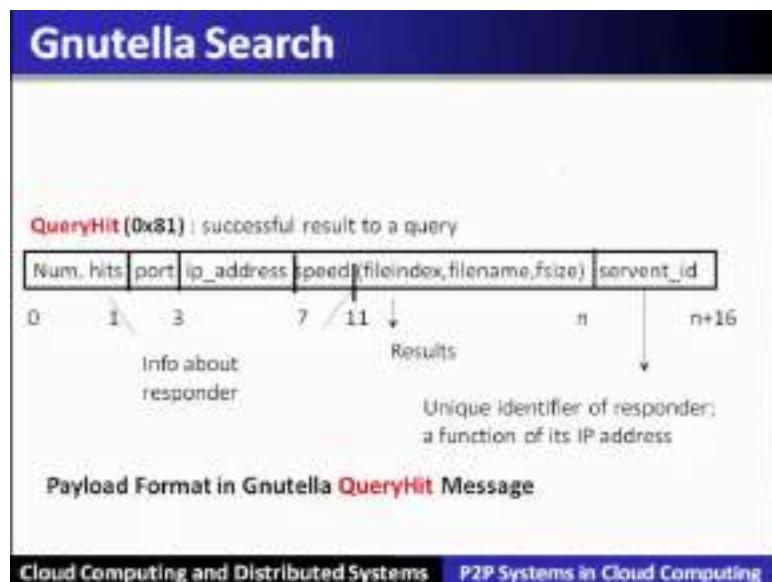
So, this is the structure of the Gnutella query message.

(Refer Slide Time: 14:47)



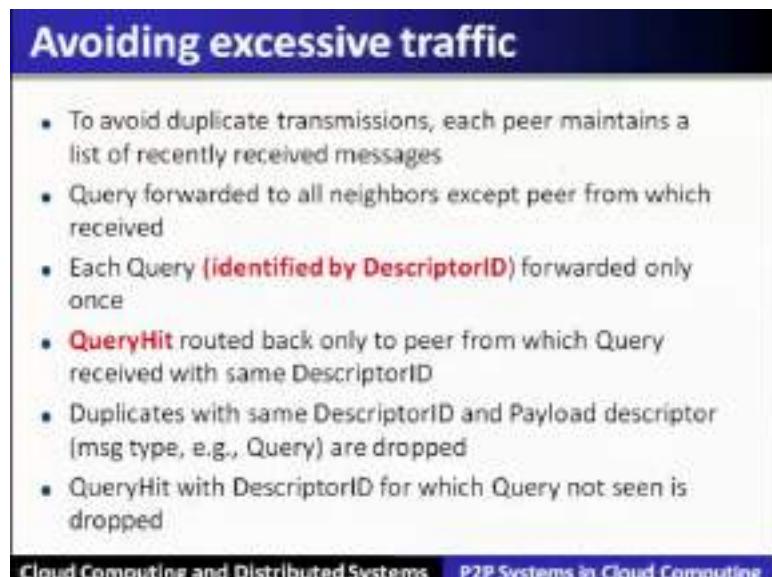
Now, let us see how the search operation is supported in the Gnutella. So, the query message will be containing the search keyword and let us say that it want to search this particular file that is publicenemy dot mp 3. Now this particular message will also have the TTL that is 2; that means, this message will be flooded, but it will be TTL restricted that is up to 2 hop within that 2 hop; it will search. So; that means, this particular message this is one hop and this is another hop since this particular peer is having at the third hop therefore, it will not reach to this point; so, also for this particular peer.

(Refer Slide Time: 15:47)



So, these messages will be forwarded only once. So, the QueryHit means the peers when they receive the query message they will search and give the response in the form of QueryHit and it will this particular result in the form of a QueryHit will be routed on the reverse path.

(Refer Slide Time: 16:09)



To avoid the excessive flooding the Gnutella has made various provisions; to avoid the duplicate transmissions and the query is forwarded to all the neighbors except the peer from which it is received. And it will be forwarded only once QueryHit will be routed back only to the peer from which the query is received with the same descriptor ID duplicates with the same descriptor ID are dropped. QueryHit with descriptor ID for which query not seen is also dropped.

(Refer Slide Time: 16:46)

After receiving QueryHit messages

- Requestor chooses “best” QueryHit responder
 - Initiates HTTP request directly to responder’s ip+port

```
GET /get/<File Index>/<File Name>/HTTP/1.0\r\nConnection: Keep-Alive\r\nRange: bytes=0-\r\nUser-Agent: Gnutella\r\n\r\n
```
- Responder then replies with file packets after this message:

```
HTTP 200 OK\r\nServer: Gnutella\r\nContent-type:application/binary\r\nContent-length: 1024 \r\n\r\n
```

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, after receiving the QueryHit message that is a response the requester chooses the best response and then the responder will reply the file packets after this particular message.

(Refer Slide Time: 17:07)

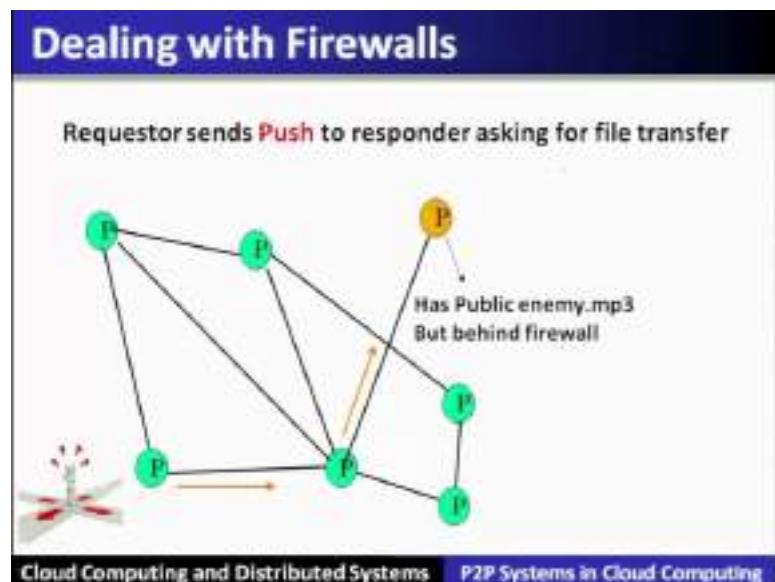
After receiving QueryHit messages (2)

- HTTP is the file transfer protocol. Why?
 - Because it's standard, well-debugged, and widely used.
- Why the “range” field in the GET request?
 - To support partial file transfers.
- What if responder is behind firewall that disallows incoming connections?

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

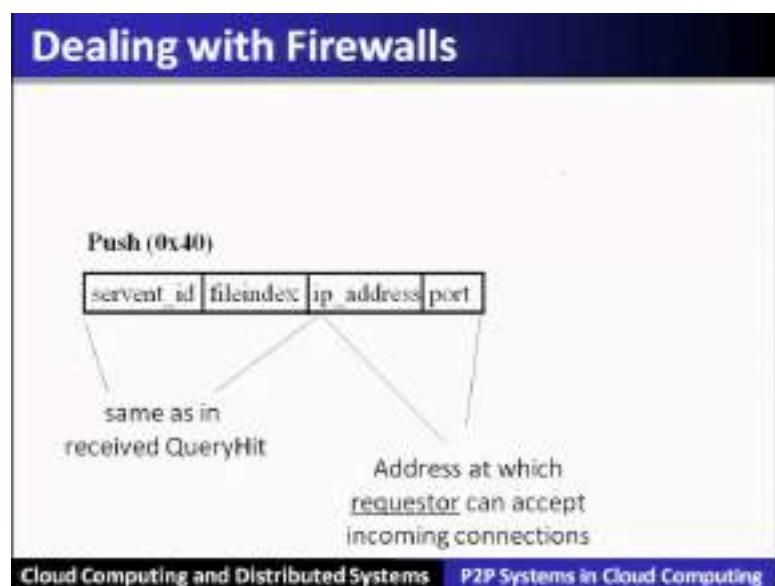
This Gnutella uses HTTP as the file transfer protocol; now if the responder is behind the firewall that disallows the incoming connections.

(Refer Slide Time: 17:19)



Then a push message is being used. So, the requester send the push to the responder asking for the file transfer.

(Refer Slide Time: 17:31)



And it will allow this particular access in this particular scenario.

(Refer Slide Time: 17:35)

Dealing with Firewalls

- Responder establishes a TCP connection at ip_address, port specified. Sends
GIV <File Index>:<Servent Identifier>/<File Name>\n\n
- Requestor then sends GET to responder (as before) and file is transferred as explained earlier
- What if requestor is behind firewall too?
 - Gnutella gives up
 - Can you think of an alternative solution?

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

If the requester is also behind firewall then the Gnutella gives up.

(Refer Slide Time: 17:45)

Ping-Pong

Ping (0x00)
no payload

Pong (0x01)

Port	ip_address	Num. files shared	Num. KB shared
------	------------	-------------------	----------------

- Peers initiate Ping's periodically
- Pings flooded out like Queries, Pongs routed along reverse path like QueryHits
- Pong replies used to update set of neighboring peers
 - to keep neighbor lists fresh in spite of peers joining, leaving and failing

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, ping and pong there are 2 different type of messages used in Gnutella for maintenance of the overlay graph.

(Refer Slide Time: 17:54)

Summary: Gnutella

- No servers
- Peers/servents maintain “neighbors”, this forms an overlay graph
- Peers store their own files
- Queries flooded out, ttl restricted
- QueryHit (replies) reverse path routed
- Supports file transfer through firewalls
- Periodic Ping-pong to continuously refresh neighbor lists
 - List size specified by user at peer : heterogeneity means some peers may have more neighbors
- Gnutella found to follow **power law** distribution:
$$P(\#links = L) \sim L^{-k}$$
 (k is a constant)

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, to summarize Gnutella has no servers; that is the client also does the operation of server hence they are called servants. This avoids the single point failure of that particular server and also ensures the scalability; large scale peer to peer system possible using Gnutella to support large number of clients. This peers of the servants are maintained in the form of a overlay graph and peers store their own files the queries are flooded out, but they are TTL restricted.

Here the periodic ping pong will refresh the neighbor list; that means, it will ensure the maintenance of the overlay graph. Now as far as Gnutella is concerned it follows the power law distribution that is the probability of number of links of a particular peer has will be of the order that is reciprocal of L raised to power minus k which is a constant.

(Refer Slide Time: 18:59)

Problems

- Ping/Pong constituted 50% traffic
 - **Solution:** Multiplex, cache and reduce frequency of pings/pongs
- Repeated searches with same keywords
 - **Solution:** Cache Query, QueryHit messages
- Modem-connected hosts do not have enough bandwidth for passing Gnutella traffic
 - **Solution:** use a central server to act as proxy for such peers
 - **Another solution:**
→ FastTrack System

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Here the problems in the Gnutella is that ping pong constitutes a 50 percent of the traffic which can be solved with the help of multiplexing. Similarly, the repeated searches with the same keyword also can be reduced using a caching method and also there is a many host which do not have the enough bandwidth for passing the Gnutella traffic; so, we will see another solution which is called a FastTrack system.

(Refer Slide Time: 19:36)

Problems (Contd...)

- Large number of *freeloaders*
 - 70% of users in 2000 were freeloaders
 - Only download files, never upload own files
- Flooding causes excessive traffic
 - Is there some way of maintaining meta-information about peers that leads to more intelligent routing?
 - Structured Peer-to-peer systems

Example: Chord System

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

(Refer Slide Time: 19:41)

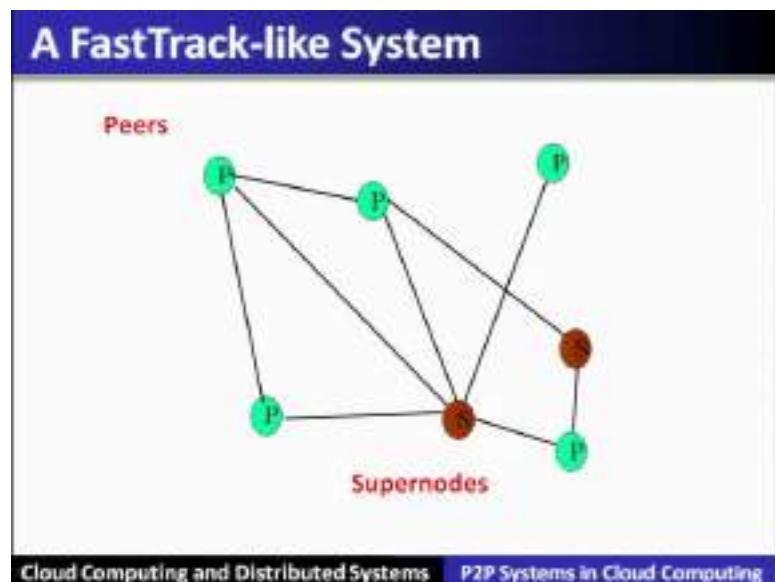
FastTrack

- Hybrid between Gnutella and Napster
- Takes advantage of “healthier” participants in the system
- Underlying technology in Kazaa, KazaaLite, Grokster
- Proprietary protocol, but some details available
- Like Gnutella, but with some peers designated as *supernodes*

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

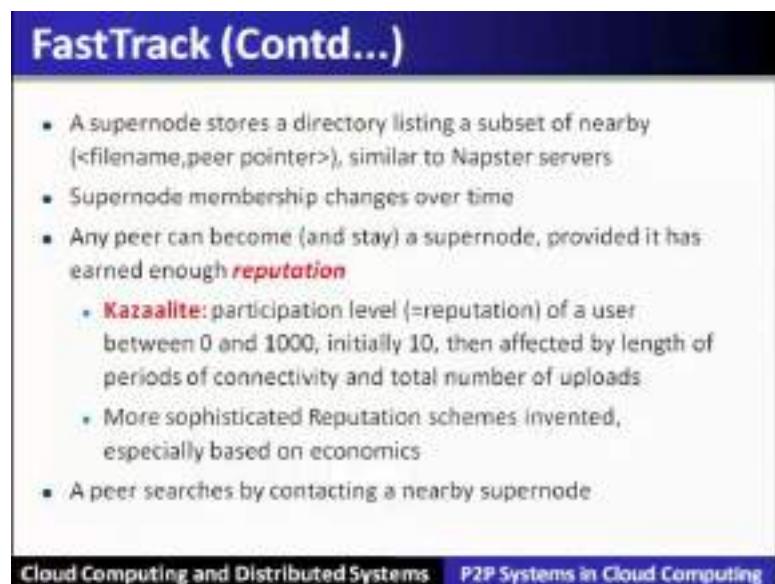
So, with the fasttrack system this is an hybrid of Gnutella and Napster to take the advantage of the servants which are basically the healthier in the participation; that means, whose reputations are high as far as uploading is concerned.

(Refer Slide Time: 20:04)



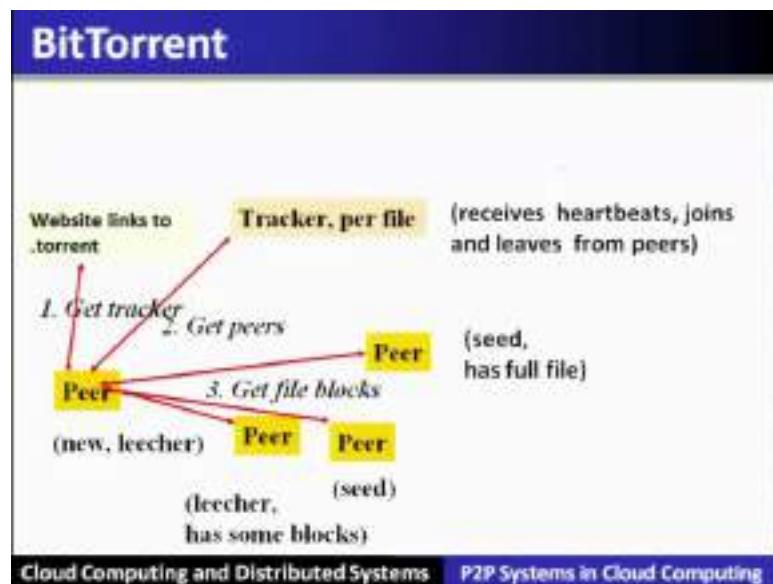
So, fasttrack is like Gnutella overlay structure, but here there will be a heterogeneous kind of nodes; that means, some nodes which are called supernodes will have more responsibility than the normal peers.

(Refer Slide Time: 20:17)



So, supernodes stores the directory listing a subset of the nearby host similar to the Napster and supernode membership changes over the time; it is not any peer can become a supernode, but the peer which is having an enough reputation can become the supernode. So, peer searches by contacting a nearby supernode here in this case.

(Refer Slide Time: 20:48)



Now, comes the next peer to peer system which is called a BitTorrent. So, BitTorrent has used various reputation mechanisms to ensure that more number of peer should involve in this particular system. Therefore, the peers are given different responsibilities that is the peer which is having the full file at that node stored is called a seed and the peer which is having some blocks of a file, but not full file is called leecher.

And when a new peer joins which does not have these particular store files; they are called again the leecher and also for every file there is a tracker where the files are stored.

(Refer Slide Time: 21:48)



BitTorrent is split the file into the blocks of size 32 kilobyte to 256 kilobytes and it applies the algorithm which is called the local rarest first block policy; that means, it will first prefer the early downloads of a blocks that are least replicated among the neighbors.

Then it applies another algorithm which is called tit for tat for bandwidth usage; that means, it provides the blocks to the neighbors that provided at the best download rates. So, whoever is providing the best downloads in this case, it will also be supported in that manner that is the incentives mechanism. Third algorithm is called which is used here in BitTorrent is called choking that is it will limit the number of neighbours to which the concurrent uploads are handling up to 5 to avoid the choking.

(Refer Slide Time: 22:59)

DHT (Distributed Hash Table)

- A **hash table** allows you to insert, lookup and delete objects with keys → point $O(1)$ - buckets
- A **distributed hash table** allows you to do the same in a distributed setting (objects=files) → nodes, hosts, cluster
- Performance Concerns:**
 - Load balancing ✓
 - Fault-tolerance ✓
 - Efficiency of lookups and inserts ✓
 - Locality ✓ → distance
- Napster, Gnutella, FastTrack are all DHTs (sort of)
- So is Chord, a structured peer to peer system

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, we will discuss the distributed hash table. So, we have seen the hash table which is maintained in particular process and which supports the 3 different operations called insert lookup. And it will delete the objects with the key values that we have seen is maintained by a particular process.

The idea of hash table is to support these operations very efficiently let us say with the $O(1)$ complexity. However, on the other hand the distributed hash table also allows the same set of operations insert lookup and delete, but in a distributed settings that is it is not confined in one process, but it is to be maintained over the distributed systems. Now, here in the hash table these keys are stored in the form of buckets, but here in the distributed hash table; these keys are stored on the nodes or you can also call them as a hosts or the cluster.

So, cluster can be very big; now as far as distributed hash table is concerned it has to deal with the performance concerned that is a perfect load balancing has to be ensured fault tolerance. In the sense if the nodes are failing and a new nodes are joining; then it should continue to give the services of insert, lookup and delete operations. Similarly, the efficiency of lookup and inserts are to be ensured and the property of locality; that means, whenever insert lookup and delete insert lookup operations are supported it has to be supported with the nearest cluster node in a close proximity that is with a smaller distance these insert and lookup they are being supported.

Now we have seen Napster Gnutella and fasttrack they are some sort of DHT, but not exactly DHT because here these performance were not a prime concern. Therefore, we will see the chord protocol peer to peer which is a structured peer to peer system which is a real sense is a distributed hash table why? Because it addresses all the performance concerns and on the other hand it supports an efficient insert, lookup and delete of these particular system.

(Refer Slide Time: 26:08)

Comparative Performance			
	Memory	Lookup Latency	#Messages for a lookup
Napster	$O(1)$ $(O(N) \text{ @server})$	$O(1)$	$O(1)$
Gnutella	$O(N)$	$O(N)$	$O(N)$
Chord	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, as far as comparative performance of Napster Gnutella and chord if we compare here the memory which is required here for the chord will be of the $O(\log(N))$ the lookup will be of the $O(\log(N))$ and the number of message for the lookup also is of the $O(\log(N))$ which is quite improvement compare to the Gnutella system which is of the $O(N)$.

(Refer Slide Time: 26:42)

Chord
<ul style="list-style-type: none"> Developers: I. Stoica, D. Karger, F. Kaashoek, H. Balakrishnan, R. Morris, Berkeley and MIT Intelligent choice of neighbors to reduce latency and message cost of routing (lookups/inserts) Uses Consistent Hashing on node's (peer's) address <ul style="list-style-type: none"> $\text{SHA-1}(\text{ip_address}, \text{port}) \rightarrow 160 \text{ bit string}$ Truncated to m bits <i>System defined</i> Called peer id (number between 0 and $2^m - 1$) Not unique but id conflicts very unlikely Can then map peers to one of 2^m logical points on a circle

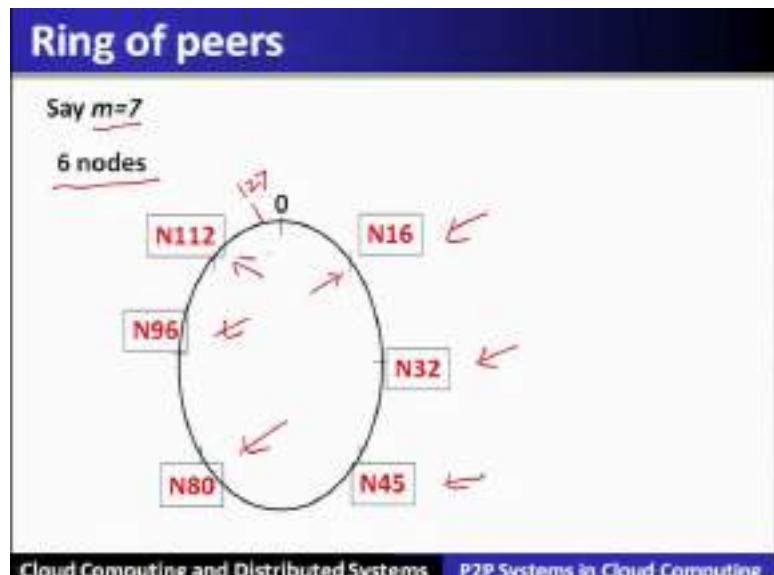
Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, let us discuss the chord system chord is developed by the Berkeley and MIT scientists; here the chord provides a intelligent choice of the neighbours to reduce latency and the message cost of routing or lookups for inserts.

The chord uses consistent hashing on the peer nodes address; chord uses consistent hashing on the nodes address. So, it does using the SHA 1 function which will take IP address and the port address and will give 160 bit string which will be truncated to m bits; m bits m is the system defined parameter. These particular m bits will generate the numbers ranging from 0 to $2^m - 1$ and these numbers are called peer id's.

These particular peer ids are not unique, but it will avoid the conflict if this particular numbers are large compared to the; to the peers. So, it can map the peers to one of these 2^m logical points in the form of a circle.

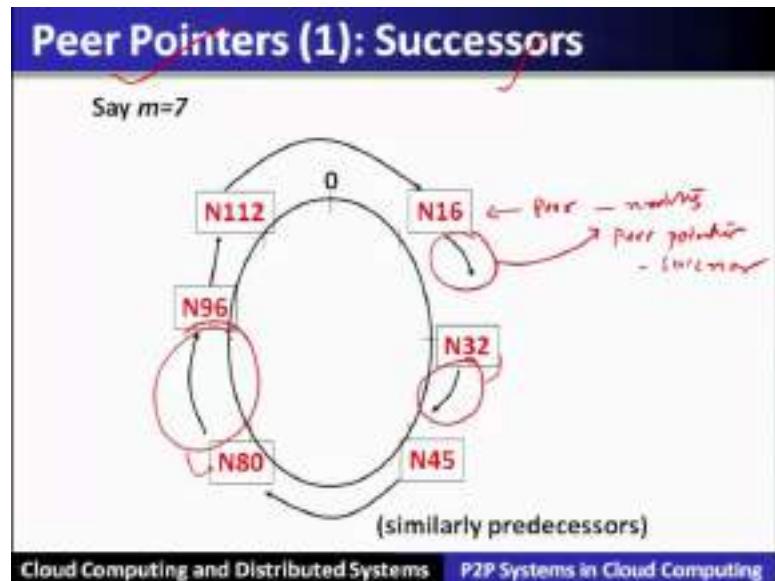
(Refer Slide Time: 28:42)



Let us take the example of this way of organizing the peers, let us say $m = 7$ and we have less then less number of nodes that is only 6 nodes are there. So, $m = 7$; that means, here comes 127 after 127; the 0, 1, 2 and so on they will start.

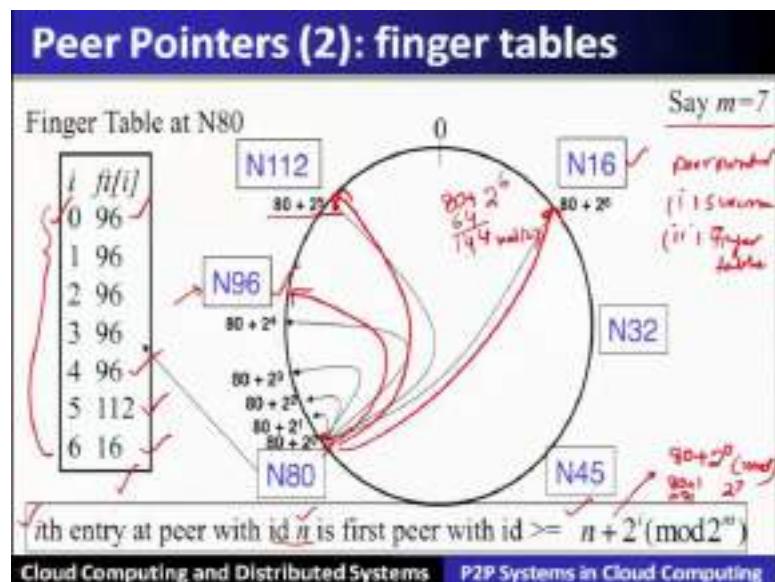
So, 16 will come and join at this particular location, node number 32 will join here at that location 45, 80, 96, 122 will be joining in the form of a ring.

(Refer Slide Time: 29:30)



Now, every peer will maintain a peer pointer which is nothing, but a successor; peer pointer is the successor. Similarly, the predecessors can also be there, but chord uses only the successor. So, every peer will maintain the pointer of its successor and this will constitute a logical ring structure.

(Refer Slide Time: 30:13)



So, besides successor; another form of peer pointer which is maintained is called a finger table, here the size of the finger table is limited to the size of m here let us say that $m = 7$. So, the size of finger table is starting from 0 to 6 that is total 7 entries are there. The finger table uses the rule which says that the i th entry at the table with id n is the first peer id which is greater than or equal to this equation that is $n + 2^i (\text{mod } 2^m)$.

For example for peer with ID 80 this particular table is constructed where in the 0th entry; $i = 0$. So, that becomes $n = 80 + 2^0 \bmod 2^7$ that is $80 + 1$ that is 81. So, 81 will be next to the N80; so as far as this

rule is concerned. So, it will be pointing to the first peer with the id greater than or equal to that id that is 81. So, greater than 81 the first particular peer will be 96; so 96 will be stored and this way up to 4 entries will basically be stored at 96.

When fifth entry will come; the value will be exceeded 96, but it will less than 112; so, 112 will be stored according to this particular rule. Now when $i = 6$ that is $80 + 2^6$ that is 64; 144 mod, 127 that comes out to be 16 and it will be stored in the 16; in this way other nodes can also form their own finger table using the same rules.

So, there are 2 different peer pointers which are stored here in the chord; the first one is called successor, the second one is called finger table. The finger table if you see is growing exponentially so, that the search becomes faster search in the sense the routing becomes faster.

(Refer Slide Time: 33:38)

What about the files?

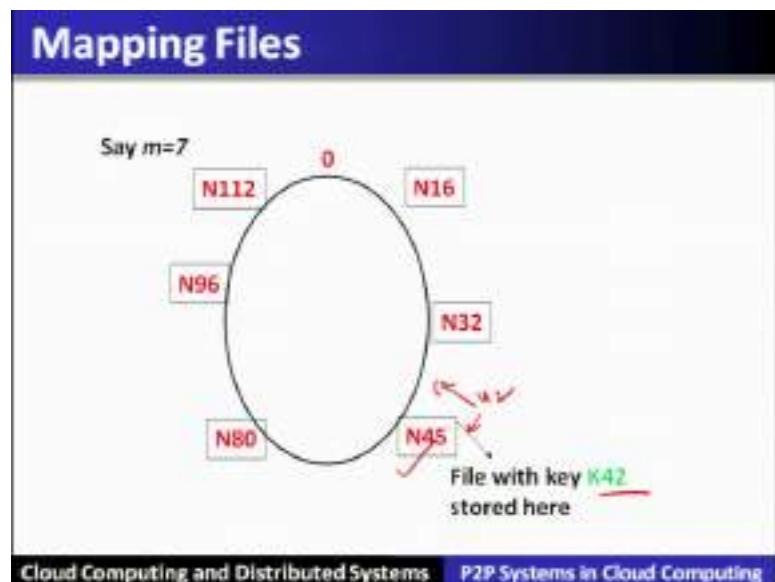
- Filenames also mapped using same consistent hash function
 - SHA-1(filename) → 160 bit string (key)
 - File is stored at **first peer with id greater than or equal to its key (mod 2^m)**
- File `cnn.com/index.html` that maps to key K42 is stored at first peer with id greater than 42
 - Note that we are considering a different file-sharing application here : *cooperative web caching*
 - The same discussion applies to any other file sharing application, including that of mp3 files.
- Consistent Hashing => with K keys and N peers, each peer stores $O(K/N)$ keys, (i.e., $< c \cdot K/N$, for some constant c)

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, the finger table is used here in the chord for routing of the search keys for various operations. So, that was the indexing or the mapping of the nodes to the ring structure; what about the files? Files are also mapped using the same consistent hash function. So, the file is stored at the first peer with the id greater than or equal to its key mod 2^m .

So, let us take an example that a particular file with the key K42 which is hashed and obtained K42 is stored at the first peer id, which is greater than 42.

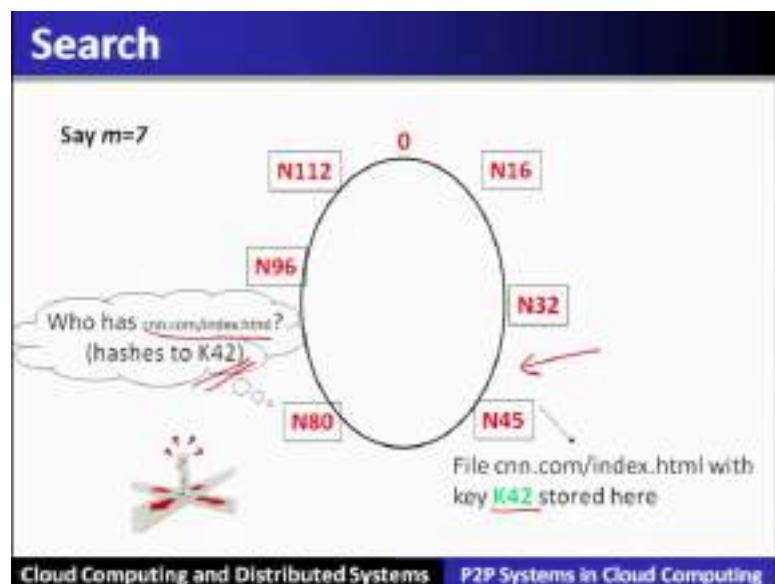
(Refer Slide Time: 34:29)



Here in this case let us say the 42 lies over here. So, the next node which is higher than 42 is 45. So, the file with the key 42 will be stored on this particular node. Now this particular method is not only used for the file sharing application, but for other applications such as cooperative web caching.

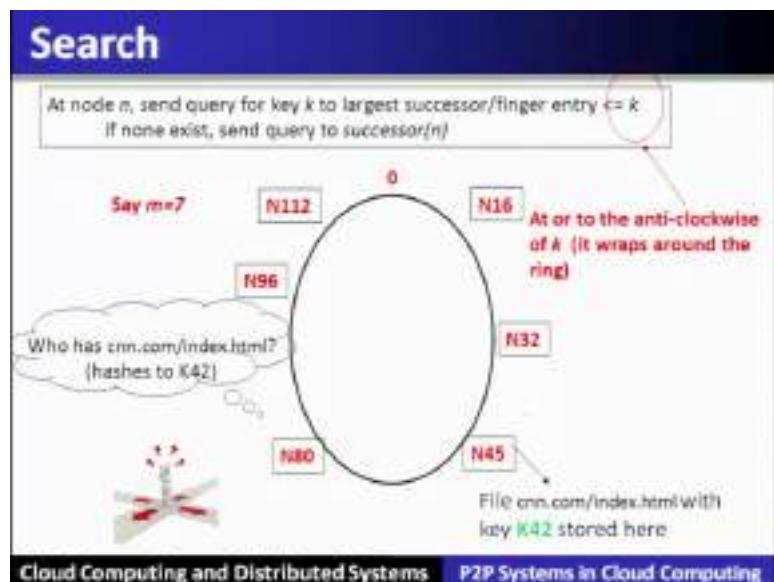
Now it uses the consistent hashing which ensures that the system with K keys and N peers will support $O(K/N)$ different keys per node; that means, that is bounded by some constant therefore, the load balancing is ensured here in the consistent hashing.

(Refer Slide Time: 35:27)



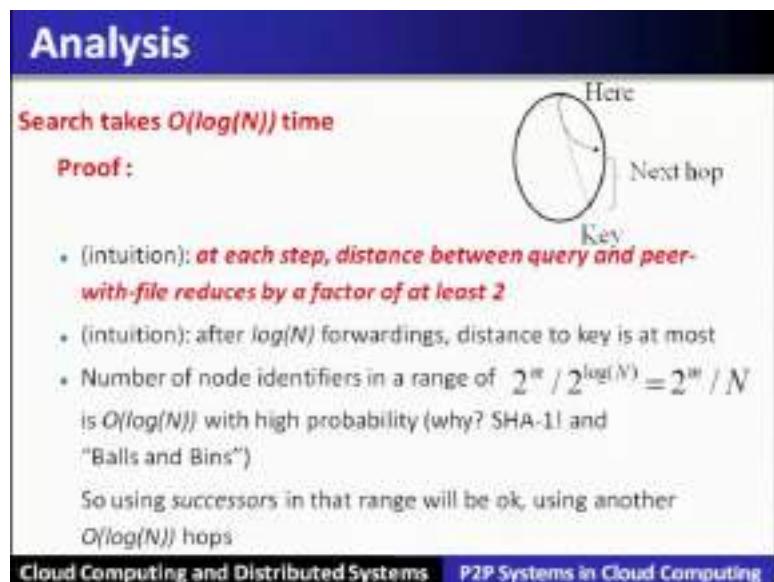
The search let us see how it happens. So, if the key if this is the file which want to be searched. So, applied to the consistent hashing let us say it generates a key 42. So, key 42 is basically the key of that file which want to access or you want to search. So, key 42 lies over here; so the next node which stores K 42 is N 45 and it will be solved in this particular manner.

(Refer Slide Time: 36:17)



So, at node n search query for the key to the larger successor or to the finger table entry which is less than k is being used and if none of these contains that particular file then search query is passed onto the successor and so on.

(Refer Slide Time: 36:39)



Let us see the analysis the search takes of the order $\log N$ time. So, here as far as the node is concerned every time in the search between the next hop and the key; the distance is reduced by a factor of at least 2. So, after $\log N$ forwarding the distance to the key is at most $2^m / 2^{\log(N)}$ that is nothing, but $2^m / N$, which is of the $O(\log(N))$ with the high probability. So, using successors in that range will be using another $O(\log(N))$ hops.

(Refer Slide Time: 37:33)

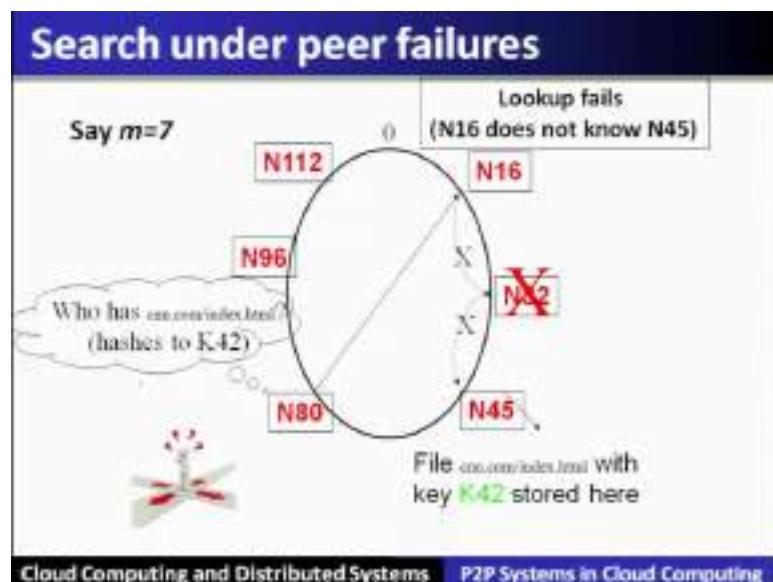
Analysis (Contd.)

- $O(\log(N))$ search time holds for file insertions too (in general for **routing** to any key)
 - “Routing” can thus be used as a **building block** for
 - All operations: insert, lookup, delete
- $O(\log(N))$ time true only if finger and successor entries correct
- When might these entries be wrong?
 - When you have failures

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Now, this particular search which takes of the $O(\log(N))$; it is also required for the file insertions, but this particular search time that is of the $O(\log(N))$ is valid if the finger and the successor entries are correct; that means, if what happens when these entries are not correct; that means, the nodes maintaining them are down; that means, the failures.

(Refer Slide Time: 38:06)



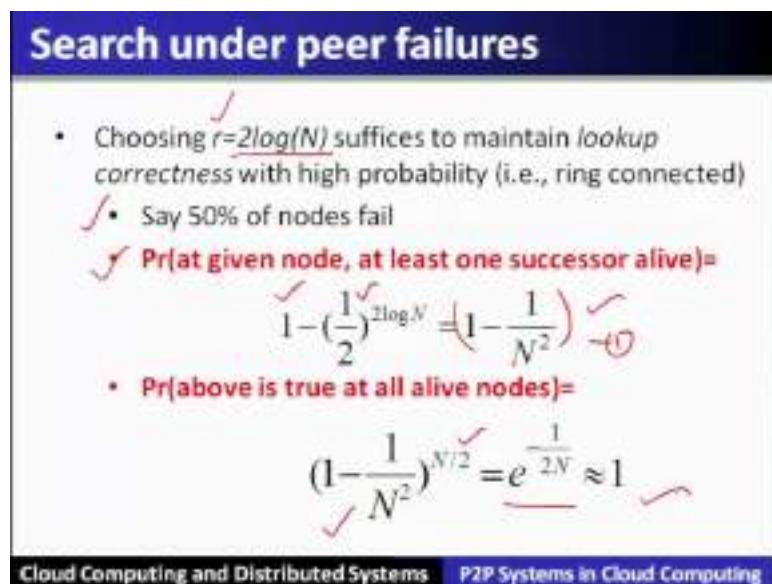
Let us see here in this example if the node 32 which contains all the information is down.

(Refer Slide Time: 38:18)



So, we have to deal with this particular solution by maintaining the successor entries by maintaining r different multiple successor entries instead of 1 to deal with the failures.

(Refer Slide Time: 38:36)



Now, the question is how many successor entries are required to deal with the peer failures? Here, the number of successor entries which are maintained to deal with the peer entries is 2 times log, which suffices to maintain the lookup correctness with the high probability.

Let us see this particular aspect now let us assume that 50 % of the nodes fail which is very high percentage normally not it will not reach to that 50 % not fail, but let us assume that. So, let us find out the probability that at a given node at least one successor is alive. So, the probability that the successor is failed is $(1/2)^{2\log N}$ and the probability that at least one successor survives is 1 minus that figure that comes out $(1-1/N^2)$.

Now, probability that this is true at all alive nodes; so we will use this particular value here $N^{N/2}$ that comes out to be $e^{(1/2N)}$ that comes out to be 1 that is if the number of nodes is too large then this becomes 1. So, these are the examples given for the understanding.

(Refer Slide Time: 40:24)

Need to deal with dynamic changes

- ✓ Peers fail
- New peers join
- Peers leave
 - P2P systems have a high rate of **churn** (node join, leave and failure)
 - 25% per hour in Overnet (eDonkey)
 - 100% per hour in Gnutella
 - Lower in managed clusters
 - Common feature in all distributed systems, including wide-area (e.g., PlanetLab), clusters (e.g., Emulab), clouds (e.g., AWS), etc.

So, all the time, need to:

- Need to update successors and fingers, and copy keys

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, here we have to deal with the dynamic changes; that means, when the peer fails or when a new peer joins or the peer leaves; the peer to peer system; that means, has if it is having high churn then it needs to update the successor and the finger table and also copy some of the nodes.

(Refer Slide Time: 40:44)

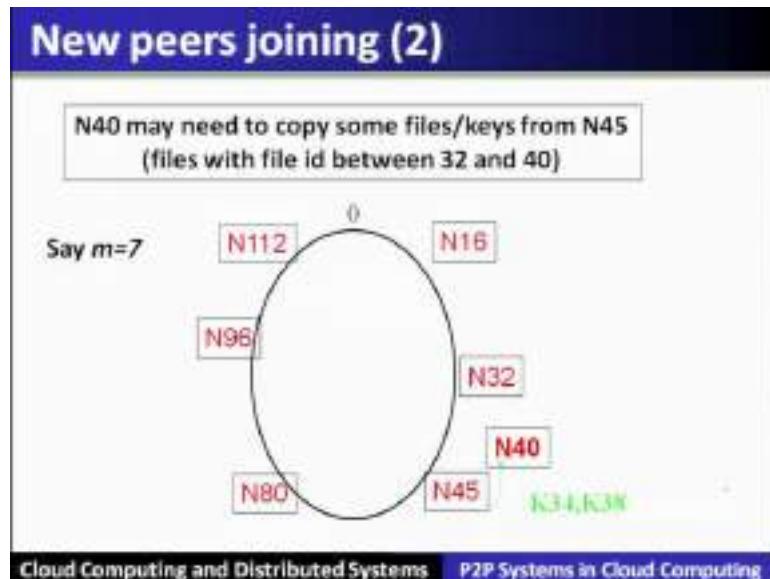


Let us see the example how the new peers join in chord system, here let us see the example that the new peer N 40 wants to join. So, it has to contact to the introducer and introducer will give the addresses of some of the peers. Here let us assume that the 2 addresses the introducer gives to N 40 they are N 45

and N 32. So, N 32 will update its successor as N 40 and then N 40 initializes the successor to N 45 and initialize its finger table from it.

N 40 periodically talks to the neighbours to update its finger table for that it uses a stabilization protocol and it will try to ensure the proper updation.

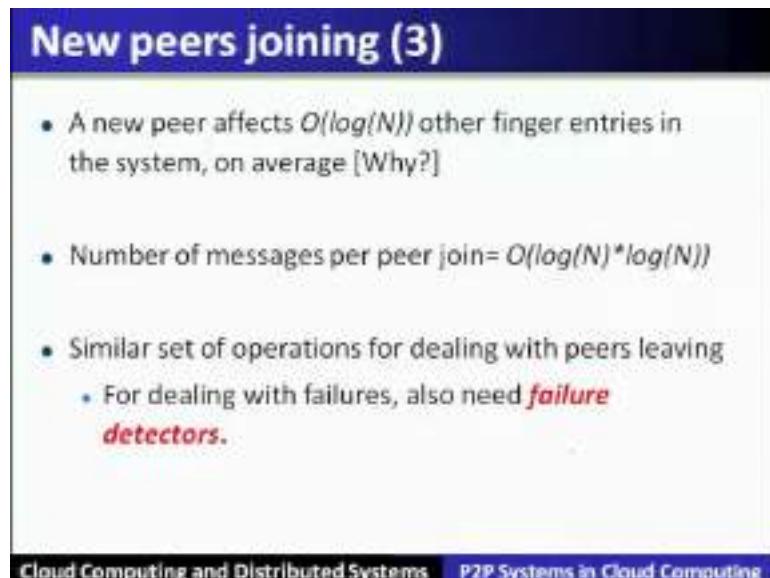
(Refer Slide Time: 42:05)



Besides this N 40 may also need to copy some of the files or the keys from N 45 that is the files with the ids between 30 and 40. So, between N 32 and 45 all the files or the keys are required to be divided here in N 40 for example, N 40 will be storing the keys between N 32 and N 40.

So, for example, the keys K 34 and K 38 will be copied here which was earlier maintained by N 45 and this will be copied here in N 40.

(Refer Slide Time: 42:57)



So, this way the new peers will join. So, that the new peer affects of the $O(\log(N))$ other finger table entries in the system on an average. So, the number of messages per peer join will be of the $O(\log(N) * \log N)$; so, that will be of the $O(\log^2(N))$. So, similar set of operations are required to deal with the peers leaving the systems or dealing with the failure detectors.

(Refer Slide Time: 43:28)

Stabilization Protocol

- Concurrent peer joins, leaves, failures might cause loopiness of pointers, and failure of lookups
- Chord peers periodically run a *stabilization* algorithm that checks and updates pointers and keys
- Ensures *non-loopiness* of fingers, eventual success of lookups and $O(\log(N))$ lookups with high probability
- Each stabilization round at a peer involves a constant number of messages
- Strong stability takes $O(N^2)$ stabilization rounds

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, stabilization protocol the concurrent peer joins leaves and failures might cause loopiness of the pointers and failures of the lookup, periodically runs this stabilization protocol and checks the updates the pointers and keys.

(Refer Slide Time: 43:55)

Churn

- When nodes are constantly joining, leaving, failing
- Significant effect to consider: traces from the Overnet system show hourly peer turnover rates (**churn**) could be 25-100% of total number of nodes in system
- Leads to excessive (unnecessary) key copying (remember that keys are replicated)
- Stabilization algorithm may need to consume more bandwidth to keep up
- Main issue is that files are replicated, while it might be sufficient to replicate only meta information about files
- Alternatives
 - Introduce a level of indirection, i.e., store only pointers to files (any p2p system)
 - Replicate metadata more, e.g., Kelios

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, strong stability takes of the $O(N^2)$ stabilization round. So, churn in the chord system sees that the nodes are constantly joining and leaving; here it leads to the excessive key copying and stabilization protocol also requires the bandwidth. So, the issues; is that the files are replicated. So, the alternative is

to provide another level of indirection store only the pointers instead of replicating the entire files. So, this particular issues will be handled in the next peer to peer system that is called Kelips.

(Refer Slide Time: 44:36)

Virtual Nodes

- Hash can get non-uniform → Bad load balancing
 - Treat each node as multiple virtual nodes behaving independently
 - Each joins the system
 - Reduces variance of load imbalance

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Here there is a concept of the virtual nodes for proper load balancing. So, if there are too many number of keys are stored at a particular node; then there is a concept of virtual nodes that they may be partitioned into the virtual nodes and hence it will introduce the load balancing.

(Refer Slide Time: 44:55)

Remarks

- Virtual Ring and Consistent Hashing used in Cassandra, Riak, Voldemort, DynamoDB, and other key-value stores
- **Current status of Chord project:**
 - File systems (CFS, Ivy) built on top of Chord
 - DNS lookup service built on top of Chord
 - Internet Indirection Infrastructure (I3) project at UCB
 - Spawns research on many interesting issues about p2p systems

<https://github.com/sit/dht/wiki>

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Therefore the concept of the virtual ring which was given here in the chord system and also the concept of consistent hashing which was used in the chord system; virtual ring and consistent hashing together these concepts are used in the modern key value store that is in the cloud computing systems like Cassandra, Riak, Voldemort and DynamoDB that we will see in the further lectures.

(Refer Slide Time: 45:26)

Pastry

- Designed by Anthony Rowstron (Microsoft Research) and Peter Druschel (Rice University)
- Assigns ids to nodes, just like Chord (using a virtual ring)
- Leaf Set - Each node knows its successor(s) and predecessor(s)

Now, we will see another peer to peer system that is called pastry, which is developed by the Rice University and Microsoft research. Pastry also assigns ids to the nodes just like the chord and uses the virtual ring, here there is a leaf set that is each node knows its successors and predecessors.

(Refer Slide Time: 45:53)

Pastry Neighbors

- Routing tables based on prefix matching
 - Think of a hypercube
- Routing is thus based on prefix matching, and is thus $\log(N)$
 - And hops are short (in the underlying network)

But there is a difference here the routing table in pastry is based on prefix matching which is nothing, but the based on the hypercube structure. So, routing is thus based on prefix matching and is $\log(N)$.

(Refer Slide Time: 46:16)

Pastry Routing

- Consider a peer with id 01110100101. It maintains a neighbor peer with an id matching each of the following prefixes (* = starting bit differing from this peer's corresponding bit):
 - *
 - 0*
 - 01*
 - 011*
 - ... 0111010010*
- When it needs to route to a peer, say 01110111001, it starts by forwarding to a neighbor with the largest matching prefix, i.e., 011101*

Let us consider how the routing in the pastry is done using prefix matching; consider a peer with the id shown over here and it maintains the neighbour peers with an id matching each of the following prefixes.

So; that means, star means that it is by starting bit differing from this peers corresponding bit for example, if it is starting with a star; that means, if the bit is 0 so; that means, it is starting from 1; that means, its neighbour. Similarly 0* means the neighbours first bit is matched with this peer 0, the second bit should be 0 instead of 1 in our peer.

So, this way the one which is having the largest prefix match will be will be the neighbour and therefore, in the routing it will be forwarded to that particular neighbor.

(Refer Slide Time: 47:23)

Pastry Locality

- For each prefix, say 011*, among all potential neighbors with the matching prefix, the neighbor with the shortest round-trip-time is selected
- Since shorter prefixes have many more candidates (spread out throughout the Internet), the neighbors for shorter prefixes are likely to be closer than the neighbors for longer prefixes
- Thus, in the prefix routing, early hops are short and later hops are longer
- Yet overall "stretch", compared to direct Internet path, stays short

So, using this prefix matching the neighbour with the shortest round trip time will be selected; using the properties of this structure that is nothing, but the hypercube. Since the shorter prefixes have more candidates the neighbour for the shorter prefixes are likely to be closer than the neighbours for the longer prefixes. Therefore, prefix routing provides and that is the early hops are shorter and the later hops are longer; overall stretch compared to the direct internet path stays short.

(Refer Slide Time: 48:04)

Summary: Chord and Pastry

- **Chord and Pastry protocols:**
 - More structured than Gnutella
 - Black box lookup algorithms
 - Churn handling can get complex
 - $O(\log(N))$ memory and lookup cost
 - $O(\log(N))$ lookup hops may be high
 - Can we reduce the number of hops?

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, if we compare the chord and pastry; both are more structure and churn handling can complex the memory requirement is of the $O(\log(N))$ and lookup host is of the $O(\log(N))$ can be reduced this number of hops.

(Refer Slide Time: 48:26)

Kelips : A 1 hop Lookup DHT

- k "affinity groups"
 - $k \sim \sqrt{N}$
- Each node hashed to a group (hash mod k) ✓
- Node's neighbors
 - ✓ • (Almost) all other nodes in its own affinity group
 - ✓ • One contact node per foreign affinity group

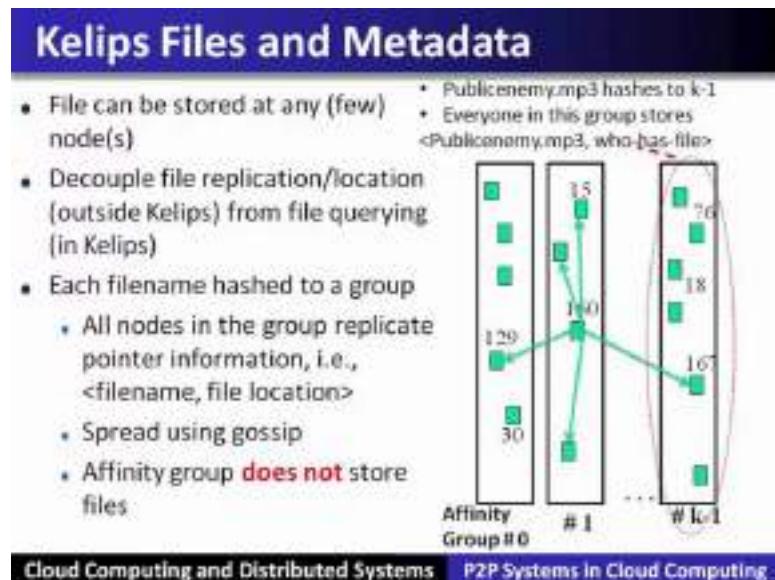
Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, the next peer to peer system is called Kelips is to provide one hop lookup DHT method; that means, instead of $\log N$ in the previous methods; it will provide of the $O(1)$ hop lookup that is Kelips. So,

Kelips supports affinity groups; so, it supports k affinity groups. So, k is of the order root N . So, here affinity group are number from 0 to $k - 1$ and the value of k is root N . So, each node is hashed to one of these groups that is hash mod k .

Now, nodes neighbours are almost all the nodes in that affinity group for example, 160 is having the neighbour with its other group members that is in this affinity group. So, they maybe $k - 1$ because this membership is root N . Beside this this particular node is also having one contact node for foreign affinity group. So, this is; so this is the affinity group its neighbour affinity groups; that means, with all other affinity group with one member it is also having in its neighbourhood lists.

(Refer Slide Time: 50:13)



So, therefore, let us see how the file is stored the file can be stored at any of these few nodes. Now let us decouple the file replication with the location from querying in the Kelips. So, each file name is hashed to a group and all the nodes in the group replicate the pointer information not the entire file; this information is spread using the gossip protocol.

(Refer Slide Time: 50:48)

Kelips Lookup

- Lookup**
 - Find file affinity group
 - Go to your contact for the file affinity group
 - Failing that try another of your neighbors to find a contact
- Lookup = 1 hop (or a few)**
 - Memory cost $O(\sqrt{N})$
 - 1.93 MB for 100K nodes, 10M files
 - Fits in RAM of most workstations/laptops today (COTS machines)

Publicenemy.mp3 hashes to k-1
Everyone in this group stores <Publicenemy.mp3, who-has-file>

Affinity Group #0 #1 ... #k-1

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, affinity group does not store these particular files; similarly as far as the lookup is concerned it will find the affinity group and go to the contact for the file affinity group failing, that try another neighbour to find out the contact. So, lookup will be of the order that is of the $O(1)$; so, 1 hop.

(Refer Slide Time: 51:18)

Kelips Soft State

- Membership lists**
 - Gossip-based membership
 - Within each affinity group
 - And also across affinity groups
 - $O(\log(N))$ dissemination time
- File metadata**
 - Needs to be periodically refreshed from source node
 - Times out

Publicenemy.mp3 hashes to k-1
Everyone in this group stores <Publicenemy.mp3, who-has-file>

Affinity Group #0 #1 ... #k-1

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

Memory cost is of the order \sqrt{N} ; membership lists are being maintained using gossip based membership protocol within each affinity group is maintained and also across the affinity group and; that means, it requires of the $O(\log(N))$ dissemination time; file metadata needs to be periodically refreshed and when time out.

(Refer Slide Time: 51:39)

Chord vs. Pastry vs. Kelips

- Range of tradeoffs available:
 - Memory vs. lookup cost vs. background bandwidth (to keep neighbors fresh)

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, as far as if we see the trade off with chord pastry and Kelips. So, it is the memory versus lookup cost versus background bandwidth to keep.

(Refer Slide Time: 51:53)

P2P Systems

Widely-deployed P2P Systems:

1. Napster
2. Gnutella
3. Fasttrack
4. BitTorrent

P2P Systems with Provable Properties:

1. Chord
2. Pastry
3. Kelips

Cloud Computing and Distributed Systems P2P Systems in Cloud Computing

So, in conclude we have seen that there are widely deployed peer to peer systems Napster, Gnutella, Fasttrack and BitTorrent; they differ and they improve further up to the BitTorrent. We have also seen the peer to peer systems came out from academia with all provable properties such Chord, Pastry and Kelips.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 19
Map Reduce

(Refer Slide Time: 00:15)

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**MapReduce paradigm**', its internal working and implementation overview.
- We will also see many examples and different applications of MapReduce being used, and look into how the '**scheduling and fault tolerance**' works inside MapReduce.

Cloud Computing and Distributed Systems **MapReduce**

MapReduce, preface content of this lecture we will discuss ‘MapReduce paradigm’ its internal working and its implementation overview. We will also see many examples, using MapReduce and different applications of MapReduce. And also we will look into the aspects how the scheduling and fault tolerance is done inside the MapReduce.

(Refer Slide Time: 00:42)

Introduction

- **MapReduce** is a programming model and an associated implementation for **processing and generating large data sets.** *in big size*
- Users specify a **map** function that processes a key/value pair to generate a set of intermediate key/value pairs, and a **reduce** function that merges all intermediate values associated with the same intermediate key.
- Many real world tasks are expressible in this model.

Cloud Computing and Distributed Systems MapReduce

Introduction: MapReduce is a programming model and is associated and implementation for processing and generating large data sets. So, the large data sets we mean that if the data cannot be stored and processed within one computer system that is in the range of petabytes size.

Therefore, we require a new paradigm that is called MapReduce, which we will enable to process such a large data sets, which are useful in today's different applications. So, here in this simple programming model users have to specify a function which is called a map that processes key value pair to generate a set of intermediate key value pairs, which in turn will be taken up by another function which is called a reduce that merges all the intermediate values associated with the same intermediate keys. And therefore, together map and reduce will work together to basically solve all the many applications in the real world. So, many real world tasks are expressible in this particular model.

(Refer Slide Time: 02:18)

Contd...

- Programs written in this functional style **are automatically parallelized and executed** on a large cluster of commodity machines.
- The **run-time system** takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication.
- This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.
- A **typical MapReduce computation processes** many terabytes of data on thousands of machines. Hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on **Google's clusters** every day.

Cloud Computing and Distributed Systems MapReduce

Now, programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run time system takes care of the details of partitioning the input data, scheduling the programs execution across the set of machines, handling machine failures, managing the required inter machine communication that is all will be made abstraction from the programmers. So, this allows the programmers without having any experience with the parallel and distributed system to easily utilize these resources for solving the problem which comprises of a large data set.

A typical MapReduce computation processes many terabytes of data on thousands of machines. So, that is the scale on which this MapReduce computation can work. Hundreds of MapReduce programs have been implemented and upwards of one thousand of MapReduce jobs are executed on Google's cluster every day.

(Refer Slide Time: 03:29)

Distributed File System

- Chunk Servers**
 - File is split into contiguous chunks
 - Typically each chunk is 16-64MB
 - Each chunk replicated (usually 2x or 3x)
 - Try to keep replicas in different racks
- Master node**
 - Also known as Name Nodes in HDFS
 - Stores metadata
 - Might be replicated
- Client library for file access**
 - Talks to master to find chunk servers
 - Connects directly to chunkservers to access data

Cloud Computing and Distributed Systems MapReduce

This MapReduce internally uses a distributed file system, which is known as HDFS and there are certain components which will be useful in the design. So, we will discuss this HDFS in a separate lecture, but for this point of discussion at this point of time we will understand that the distributed file system has the master node which is also known as the name node in HDFS. And this stores the metadata and these may be replicated as per different file system and here we are using the HDFS file system.

There are some client libraries for the file accesses. So, they will talk to the master to find out the available chunk servers, which connects directly to the chunk server to access the data. So, chunk servers are the servers who manage the splits into the contiguous chunks. And each chunk is of the size 16-64 MB and these chunks are replicated normally by default it is 3 times replicated in HDFS file system. This replication is also ensures that these replication also goes into different racks. So, that it will become a rack tolerant, rack fault, tolerant system it can build up in this process of replication.

(Refer Slide Time: 05:16)

Motivation for Map Reduce (Why ?)

- Large-Scale Data Processing
 - Want to use 1000s of CPUs
 - But don't want hassle of managing things ✓
- MapReduce Architecture provides
 - Automatic parallelization & distribution ✓
 - Fault tolerance
 - I/O scheduling
 - Monitoring & status updates

Cloud Computing and Distributed Systems MapReduce

So, let us see the motivation of using MapReduce for large scale data processing. So, to handle this scale of data processing we required 1000s of CPUs and also we do not want the hassle of managing such a large system. Therefore, the MapReduce is a programming model and its architecture will provide automatic parallelization. And the distribution, fault tolerance and I/O scheduling monitoring and status updates we will see their more details in further slides.

(Refer Slide Time: 06:04)

What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)
Sum of squares:
 - `{map square '(1 2 3 4)}`
 - Output: `(1 4 9 16)`
 - [processes each record sequentially and independently]
- `{reduce + '(1 4 9 16)}`
 - `(+ 16 (+ 9 (+ 4 1)))`
 - Output: 30 ✓✓
- [processes set of all records in batches]
- Let's consider a sample application: Wordcount
 - You are given a huge dataset (e.g., Wikipedia dump or all of Shakespeare's works) and asked to list the count for each of the words in each of the documents therein. ✓

Cloud Computing and Distributed Systems MapReduce

So, MapReduce paradigm let us understand from the scratch about this paradigm. These terms map and reduce their borrow from the functional language which is called a lisp. So, for example, the map function which is there in the list, if it is having the operation or a function called square and a Lisp is given. Then this particular function will be applied on each and every element of this particular list.

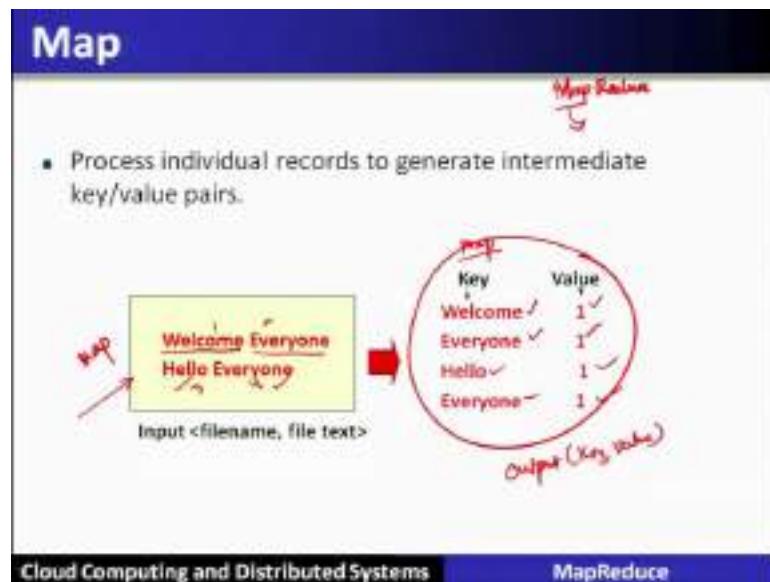
Now this particular function can be executed in parallel why because, this particular function if it is applied on these input values they can be able to generate the output that is the 1^2 is 1, 2^2 is 4, 3^2 is 9, 4^2 is 16. So, if four different machines are involved or four different threads are executing this is square at four different elements. So, in one go it will be able to generate this.

So, that is how the inherent parallelism is being exploited here in the functional languages and that is how the MapReduce will also do the same way without botheration of internally how this parallel programming is to be done by the programmer end. Similarly there will be a reduce function so, in the reduce function there is an operation that is the function which is defined.

So, function is a plus which involves all the elements, which are to be added and given the output. So, internally it will be represented in this particular equation form and two operators at a time because, plus is a binary operation two at a time this will form a complete expression and this will be evaluated to give the output 30. So, these map and reduce functions which are there in functional language like lisp, it is being inspired and it is not that map and reduce of lisp function, but it is a separate construct which is called a MapReduce which is now available to solve this kind of problem.

So, the problems of application for example, a large document large data set for example, it is a dump of a Wikipedia. And here you want to find out the word Shakespeare or some other keyword and you want to list the count for each of these words in a given document there in if that is the application. Then a MapReduce construct will be very easy and its program is called a word count. So, let us see how the word count program will work, where in we can submit a huge data set and using the word count program of MapReduce it can able to solve.

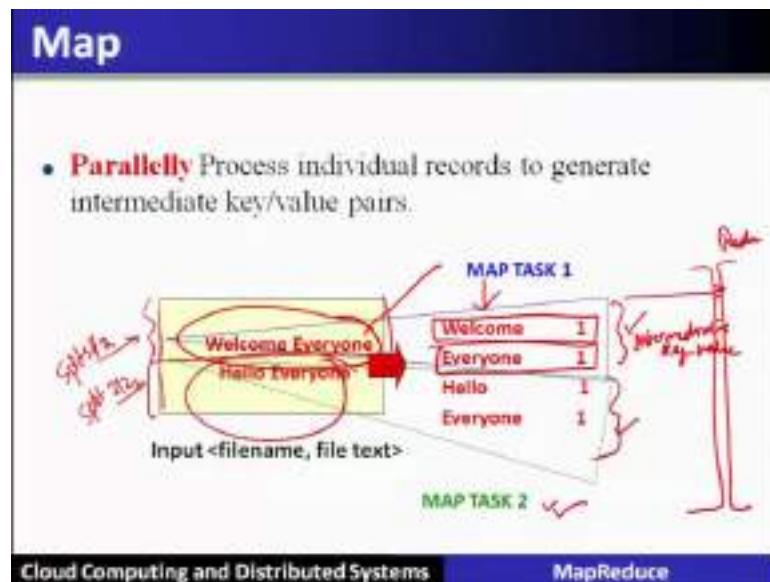
(Refer Slide Time: 09:35)



So, now in the MapReduce there are two functions map and reduce. So, out of them the map function let us discuss first. So, the process individual record to generate the intermediate key value pairs so, take this snapshot of the entire big document, let us say it has four different words in this particular sentence that is welcome everyone hello everyone. Now this particular document when it is given as the input. Then this map function should be able to generate the key value pairs. So, key means every keyword it will generate and it is count it will emit in the map that value of 1. So, for every word it will emit 1 count.

Similarly, hello also will emit 1 count and everyone is appearing again so, it will be generating that particular count so that is the function of map. So, map will produce this as an output. So, when map is given this particular filename, it will generate this particular output that is in the form of a key value pair.

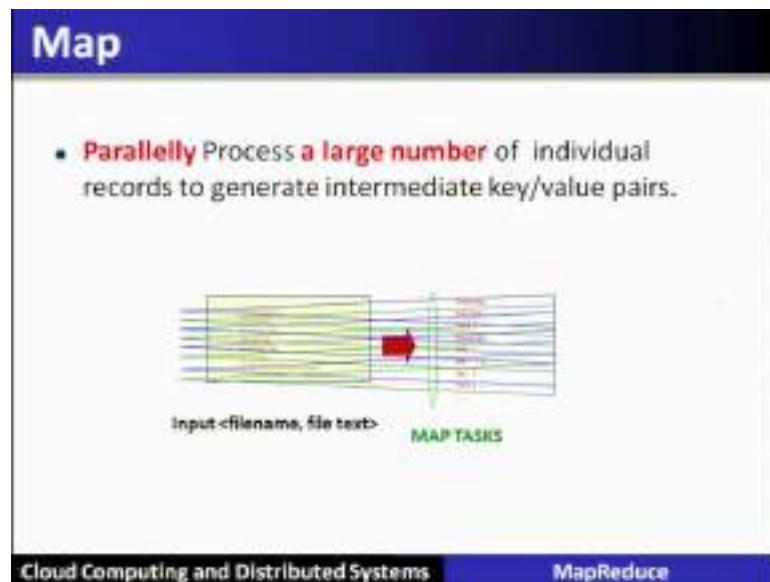
(Refer Slide Time: 11:23)



Now, this particular map function will process these individual records to generate the intermediate key value pairs that is what we have shown. Now, this particular task can be executed by two different map tasks or we can divide this file into two different, two different chunks or we can call them as splits. So, there are two splits and we intend to execute in parallel. So, that this particular document can be processed big size document can be processed.

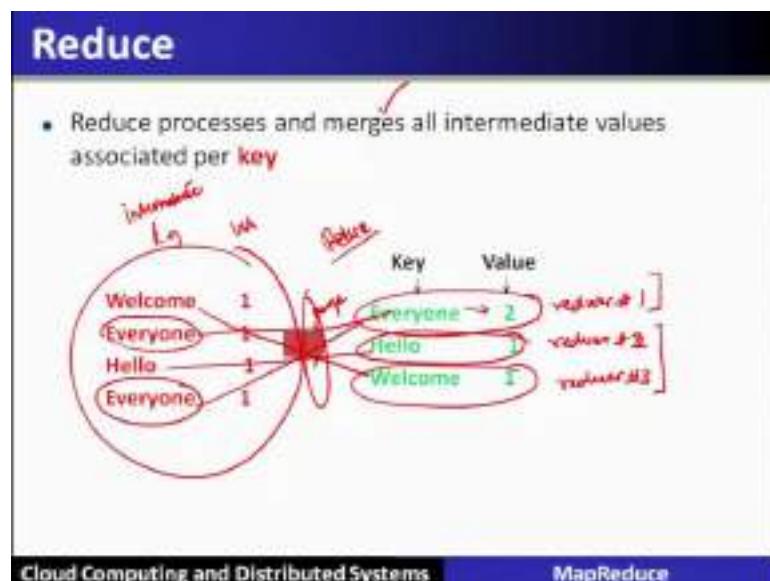
So, in that process these particular splits are given to different servers, they are called as a map task 1 and the other one is called map task 2. So, this particular input is given to the map task 1 and this map will generate the intermediate key value pairs they are called intermediate key value pairs. So, also the map task 2 also will do the same thing. These intermediate key value pairs will now be given as the input to the next phase which is called reduced phase. So, for reduced phase the output of the map phase will become the input.

(Refer Slide Time: 13:08)



So, the parallelly processing a large number of individual records to generate the intermediate key value pair is basically, inherent in this paradigm that is called map using the map tasks.

(Refer Slide Time: 13:26)

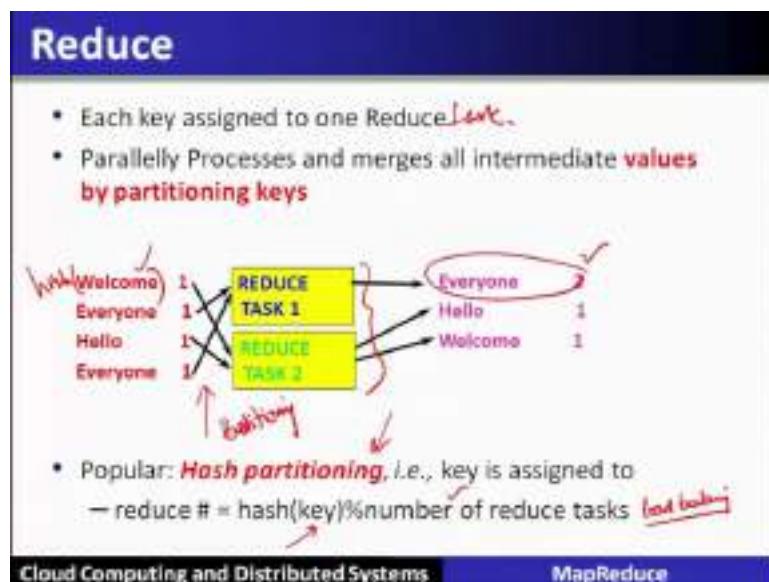


Then this particular output of map phase is given to the reducer. That is the key value pairs are given this is called intermediate key value pairs, key value pairs are given to the reduce function. So, reduce processes and merges all the intermediate values associated per key together in this phase. So, here you can see that everyone is appearing twice. So,

they should be merged in the reduce phase and therefore, it will give the final output associated with every key.

So, let us see when this happens using the reduce function what it does? It combines or it merges, these two keywords and it will generate the combined values. Similarly, hello is appearing only once and welcome is also appearing once. Now to merge these different keys which are having the same keys they have to be merged together in the reducer phase. So, they have to be given to the same reducer. And this hello keyword may be given to the separate reducer. Now there are requirements of three different reducers, but if you have only two machines available, which can be assigned to these reducer. So, a particular machine can handle two different or more than one different reducers also.

(Refer Slide Time: 15:40)



So, let us see the details of this reduce function. So, each key will be assigned to one reduce to one reducer or to one reduce task. So, parallelly they will be processed and it will merge all the intermediate values associated of the same key to one reducer by partitioning this particular keys. So, here in this case you can see the partitioning or these keys are basically able to map to reducers which you have here shown in this example.

So, for example, map 1 will go to reduce task 2, everyone will go to reduce task 1 hello will go to reduce task 2 and everyone will go to the reduce task 1. And reduced task 1 will then combine these 2 everyone into that combined function that is it will aggregate.

Now this kind of partitioning of the keys to the reduce task can be done using hash partitioning.

So, using hash partitioning we can hash a particular key. And this particular key when it is hashed then it will be taken the modulo of the number of reduce tasks. In this manner this particular keys will be partitioned equally across all the reduce task. So, there will be a proper load balancing of these keywords which are assigned to this reduce function.

(Refer Slide Time: 17:40)

Programming Model

- The computation takes a set of **input key/value pairs**, and produces a set of **output key/value pairs**.
- The user of the MapReduce library expresses the computation as two functions:
 - The Map**
 - The Reduce**

Cloud Computing and Distributed Systems MapReduce

So, let us see the programming model in more detail of this MapReduce. So, the computation takes a set of input key and values pairs and produces a set of output key value pairs. So, the users they have to use the MapReduce library to express their computations using these two functions.

(Refer Slide Time: 18:06)

(i) Map Abstraction

- Map, written by the user, takes an input pair and produces a set of **intermediate key/value pairs**.
- The MapReduce library groups together all intermediate values associated with the same **intermediate key 'I'** and passes them to the **Reduce function**.

Cloud Computing and Distributed Systems

MapReduce

The map abstraction the map written by the user takes the input pairs. Pair and produces a set of intermediate key value pairs. MapReduce library a groups together all intermediate key values associated with that intermediate key and passes them to the reduce function.

(Refer Slide Time: 18:23)

(ii) Reduce Abstraction

- The **Reduce function**, also written by the user, accepts an **intermediate key 'I'** and a set of values for that key.
- It merges together these values to form a **possibly smaller set of values**.
- Typically just **zero or one output value** is produced per Reduce invocation. The **intermediate values** are supplied to the user's reduce function via **an iterator**.
- This allows us to **handle lists of values** that are too large to fit in memory.

Cloud Computing and Distributed Systems

MapReduce

Reduce abstraction reduce function is also written by the user accepts the intermediate keys, which are generated from the previous phase that is from the map phase. And the set of values for that it merges together these values to form a possibly a smaller set of

values. Typically zero or one output value is produced per reduce invocation. The intermediate values are supported to the users reduce functions where it later. This allows us to handle the list of values that are too large to be fit in the memory.

(Refer Slide Time: 19:00)

Map-Reduce Functions for Word Count

```
map(key, value):
    // key: document name; value: text of document
    for each word w in value:
        emit(w, 1)

reduce(key, values):
    // key: a word; values: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```

Cloud Computing and Distributed Systems MapReduce

Let us see through an example of a word count. Here the map function will take the key values and key is nothing, but the name of the document and value is the text of that particular document. So, when given this particular document the name of a document and this is the text of a document or the entire document is given as a value then for each word which are separated by either the blanks are other separator. So, each word will be identified in the document or in the value of that is called the document. And it will emit that word w with a value 1 in the word count program.

Similarly, this output will be now taken up by the reducer, now here the key is a word and the values are the values which are being generated over here. So, values and iterator will count these values. So, let us begin this process with initialization of result is equal to 0. So, for each count v in these particular values, the result will be incremented by that value and it will emit for key this corresponding result and that will be output from this MapReduce function.

(Refer Slide Time: 20:35)

Map-Reduce Functions

- **Input:** a set of key/value pairs ✓
- User supplies two functions:
 $\text{map}(k, v) \rightarrow \text{list}(k_1, v_1)$ ✓
 $\text{reduce}(k_1, \text{list}(v_1)) \rightarrow v_2$ ✓
- (k_1, v_1) is an intermediate key/value pair
- **Output** is the set of (k_1, v_2) pairs
map ↗ ↘ *reduce*

Cloud Computing and Distributed Systems MapReduce

So, in general input will be a set of key value pairs, that we have already seen and the user has to specify it is own application through the map function; which will emit a list of intermediate key value pairs are now given as the input to the reduce function which will generate the output that is the values of all keys. So, output will be that keys and the final values which will be generated by the reduce function. And so, let us see some of the applications.

(Refer Slide Time: 21:24)

Applications

- Here are a few simple applications of interesting programs that can be easily expressed as **MapReduce computations**.
- **Distributed Grep:** The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency:** The map function processes logs of web page requests and outputs $(\text{URL}; 1)$. The reduce function adds together all values for the same URL and emits a $(\text{URL}; \text{total count})$ pair.
- **ReverseWeb-Link Graph:** The map function outputs $(\text{target}; \text{source})$ pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: $(\text{target}; \text{list}(\text{source}))$

Cloud Computing and Distributed Systems MapReduce

The applications of this MapReduce paradigms are many some of them we will here list them to understand the details of these applications. So, distributed graph application; that means, distributed graph can be applied on the log files or an error files wherein we can check or we can find out that pattern and find out the what kind of error it is incurring. So, since the log files are too big therefore, this distributed graph is an ideal application. Similarly, the count of URL access frequency is also one of the important application. And the list of URL's are being generated by any proxy server or in the cloud who is running the websites.

So, how many such accesses are being made across different websites accesses by the user. So, they are total count also becomes a MapReduce application. Similarly a reverse web link graph that means, a map that means, we want to find out the graph of those websites and who are accessing those websites. So, that is called link graph. So, we want to find out the reverse web link graph means to that website how many people they are accessing it. So, if we want to know that that is that application is called reverse web link graph.

(Refer Slide Time: 23:06)

Contd...

- **Term-Vector per Host:** A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word; frequency) pairs.
- The map function emits a (hostname; term vector) pair for each input document (where the hostname is extracted from the URL of the document).
- The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final (hostname; term vector) pair.

Cloud Computing and Distributed Systems MapReduce

Similarly, term vector per host, a term vector summarizes the important words that occur in a document or a set of documents as the list of words and their frequency which is called a term vector per host.

(Refer Slide Time: 23:27)

Contd...

- **Inverted Index:** The map function parses each document, and emits a sequence of (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (word; list(document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.
- **Distributed Sort:** The map function extracts the key from each record, and emits a (key; record) pair. The reduce function emits all pairs unchanged.

Cloud Computing and Distributed Systems

MapReduce

So, this particular application can also be programmed using MapReduce inverted index that is the map function parses each document and emit a sequence of words. So, the reduce function accepts all pair of the given words and sorts the corresponding document ID and emits the world. So, the set of all output pair forms a inverted index and it is easy to augment this computation to keep track of what position. Similarly, distributed sort that is the sorting algorithm in a large scale is called distributed sort.

(Refer Slide Time: 23:59)

Applications of MapReduce

(1) Distributed Grep:

- Input: large set of files
- Output: lines that match pattern
- Map – *Emits a line if it matches the supplied pattern*
- Reduce – *Copies the intermediate data to output*

Cloud Computing and Distributed Systems

MapReduce

So, let us see some of the use of MapReduce in these applications. So, distributed graph here the input is a large set of files and the output is the lines into the file that matches that particular given pattern. So, map will emit a line if that pattern is matches the supplied pattern and reduce will copy the intermediate data to the output.

(Refer Slide Time: 24:28)

Applications of MapReduce

(2) Reverse Web-Link Graph:

- **Input:** Web graph: tuples $\{a, b\}$ where $(\text{page } a \rightarrow \text{page } b)$
- **Output:** For each page, list of pages that link to it
- **Map –** *process web log and for each input <source, target>, it outputs <target, source>*
- **Reduce -** *emits <target, list(source)>*

Cloud Computing and Distributed Systems MapReduce

So, this can be written using a simple program like word count. Another application is called as reverse web link graph here the input is the web graph in the form of a tuples a, b where the page a will basically accessing or having a link to the page b. Now, given this particular input the output for each page is the list of pages that link to it. To do this, the map function what it will do? It will process this particular web log for each input and it outputs the reverse of that that is the target and the source.

Then reducer will collect the target and form the list of source. Let us take an example; let us see a graph, a b and c. So, here in this particular graph, there are three nodes a b c or let us say there are three pages a b c, a is now referring to b or accessing to b and b is accessing to a and a is accessing to c. So, the list of pages here it will be in the form of a b then, then b c, then c a and then b a. Now the task is to find out for example, in a how many links are being accessed? So, here it is showing there are two links. Let us see how using MapReduce we can do this.

So, this will be the input this value will be the input, this is the key and this is the value this will be input to the map function. So, the map function will be given this input. Now

the map will emit value, comma key so, that means it will emit this particular output. Let us say here it is given as source and target, this is called source and this is called target. So, this will be given as the values to this particular thing. Now this will emit the target, comma source. So, MapReduce output will emit b, comma a it will output c, comma b it will output a, comma c it will output a, comma b. Then this particular output will be now fed to the reducer, reducer will now generate based on the key as the target it will find out the list of source.

So, this particular key for example, a will be combined and the list will be generated for a c comma b this will be list for c there will be list called b and for b the list will be a this will be generated out of the reducer. So, here we can see the node a is having two links pointed one by b and the other by c. So, using MapReduce program we can easily write down the map and reduce function which is very small program. But here we have to know the tricks to solve these particular problems. And when a big programmer or a big file is given it will be able to generate these values.

(Refer Slide Time: 29:01)

Applications of MapReduce

(3) Count of URL access frequency:

- Input: Log of accessed URLs, e.g., from proxy server
- Output: For each URL, % of total accesses for that URL
 $\text{req}(URL) \rightarrow \text{out}(URL) \rightarrow \text{Reduce}$
- Map – **Process web log and outputs <URL, 1>**
- Multiple Reducers - **Emits <URL, URL_count>**
 (So far, like Wordcount. But still need %)
 $\xrightarrow{\text{MAP + 2 Reducers}}$ $\xrightarrow{\text{URL_count}}$ $\xrightarrow{\text{page}}$
- Chain another MapReduce job after above one
- Map – **Processes <URL, URL_count> and outputs <1, (<URL, URL_count>) >**
- 1 Reducer – Does two passes. In first pass, sums up all **URL_count's** to calculate overall_count. In second pass calculates %'s
 $\xrightarrow{\text{overall}}$ $\xrightarrow{\text{total}}$ $\xrightarrow{\text{overall}}$ $\xrightarrow{\text{percent}}$
Emits multiple <URL, URL_count/overall_count>

Cloud Computing and Distributed Systems **MapReduce**

Now, let us see another MapReduce program to count URL access frequency. Now here the input is a log of accessed URL's from the proxy server and output will be the each URL and the percentage of total accesses for that URL so, this will be a bit difficult. Let us see the details how we can write down a program for this particular application of URL access frequency. So, here first what we will do is that map function will process

the web log and output URL with a value 1. So that means, all the URL's will be given to the map function, URL's will be give raise the input and what it will emit here is URL and 1, that means, for every URL it will emit 1.

Now, the next phase comprises of multiple reducers. So, the first phase what it will do? It will take it will take this particular output. Now reducer number 1, it will take this output URL 1 and then it will count the total URL's, that is it will generate a reducer 1, comma URL and the count. So, with 1 and all the output whatever is generated, so that means, it will generate the total number of URL's present. So, it will generate the total values of URL present.

Now, then another reducer will be basically, done in the second pass which will count the percentage of the URL in the form that this particular reducer will take the URL's, which are omitted by the map function. And for every URL count whatever is output by the reducer in this particular phase, it will divide by the overall count whatever is calculated in the previous pass of the reducer. Therefore, with one map function and 2 times application of a reducer function, we will be able to calculate the count of URL access frequency. So, this particular application has shown that it is possible to cascade multiple MapReduce job to solve an application.

(Refer Slide Time: 32:40)

Applications of MapReduce

(4) Map task's output is sorted (e.g., quicksort)
Reduce task's input is sorted (e.g., mergesort)

Sort

- Input: Series of (key, value) pairs
- Output: Sorted <value>s
- Map – $\langle \text{key}, \text{value} \rangle \rightarrow \langle \text{value}, _ \rangle$ (identity)
- Reducer – $\langle \text{key}, \text{values} \rangle \rightarrow \langle \text{key}, \text{value} \rangle$ (identity)
- Partitioning function – partition keys across reducers based on ranges (can't use hashing!)
 - Take data distribution into account to balance reducer tasks

Cloud Computing and Distributed Systems MapReduce

Now let us see another application, here in this application it will do the sorting. Now the sorting using MapReduce; that means, if a very big file is given: how are you going to

sort using MapReduce is the task. Now here the map tasks outputs they result that is intermediate results in a sorted form, that is using the quick sort already is implemented inside the MapReduce, inside the map function.

Similarly, the reduce task also input in the form of sorted manner that is also done using some sorting algorithm for example, using merge sort. So, using this information let us see how the sorting can be done using MapReduce in a very simple manner. So, let us see that if the input is given to the map. So, it will generate that particular value which is output from the map function. And the second one that is the value of will be any value that is not that important.

So, you know that these particular values are already sorted so, the map will generate these particular keys in a sorted order. Then as far as the reducer is concerned this as a input of this map function and it will generate the same value again. The only thing we have to do is that in the partitioning, we have to provide the range instead of hash function because, hashing will randomize them. Since the output of the map function is already in the form of sorted and we do not want to disturb it so, that we have to provide the range values the range of keys. So, range of keys means the reduce task will be assigned according to the ranges.

So, that this particular range will not be disturbed and the output of this map function will be preserved which is in the sorted list and the output will be sorted. So, this particular example or application of MapReduce could make this sorting algorithm easy why because, it has exploited the way map function is giving the output it is in the sorted order. So, we have to just preserve that sorted order without being disturbed by the partitioning function so, that the entire file will be in a sorted order output.

(Refer Slide Time: 36:06)

Programming MapReduce

Externally: For user

1. Write a Map program (short), write a Reduce program (short)
2. Specify number of Maps and Reduces (parallelism level)
3. Submit job; wait for result
4. Need to know very little about parallel/distributed programming!

Internally: For the Paradigm and Scheduler

1. Parallelize Map
2. Transfer data from Map to Reduce (**shuffle data**)
3. Parallelize Reduce
4. Implement Storage for: Map input, Map output, Reduce input, and Reduce output



(Ensure that no Reduce starts before all Maps are finished. That is, ensure **the barrier** between the Map phase and Reduce phase!)

Cloud Computing and Distributed Systems

MapReduce

Now, let us see the MapReduce scheduling how it is done? Now, as far as programming of a MapReduce is concerned externally for the user we have to only write down the map program, this is very short programs. And they have to specify the map and reduce function and submit the job and wait for the results. The users need not have to know the details of parallel and distributed concepts and the programming.

As far as everything is done internally, so, for internally for the paradigm and the scheduler has to build everything and give the output. So, it has to parallelize the map function execution; that means, it will transfer the data from map to the reduce, that is done by the shuffling that we have seen earlier that a partitioning and shuffling using hash function was being carried out.

Then parallelize the reduce function and implement the storage for map input map output reduce input and reduce output. One thing is there that we have to ensure that, no reducer start before all the maps are finished, that is being done inside MapReduce. So, this ensures that a barrier between the map phase and the reduce phase is there. So, barrier in the sense this is called a barrier is nothing, but a synchronization. So, when all the map function will finish, some are finishing early some are finishing quite late. So, earlier who ever finish they have to wait till all the map function completes their execution then only the next phase called reducer phase will begin afterwards so, this is called a barrier.

In some of the implementations this barriers can also be in some cases being a synchronized; that means, they are not synchronizing may be in the next phase without completing all the reduce phase the next phase can begun.

(Refer Slide Time: 38:34)

The slide has a dark blue header with the title 'Inside MapReduce' in white. Below the header is a section titled 'For the cloud:' in red. It contains a numbered list of 4 steps:

1. Parallelize Map: **easy!** each map task is independent of the other!
 - All Map output records with same key assigned to same Reduce
2. Transfer data from Map to Reduce:
 - Called Shuffle data
 - All Map output records with same key assigned to same Reduce task
 - use **partitioning function**, e.g., $\text{hash}(\text{key}) \% \text{number of reducers}$
3. Parallelize Reduce: **easy!** each reduce task is independent of the other!
4. Implement Storage for Map input, Map output, Reduce input, and Reduce output
 - Map input: from **distributed file system**
 - Map output: to local disk (at Map node); uses **local file system**
 - Reduce input: from (multiple) remote disks; uses **local file systems**
 - Reduce output: to distributed file system

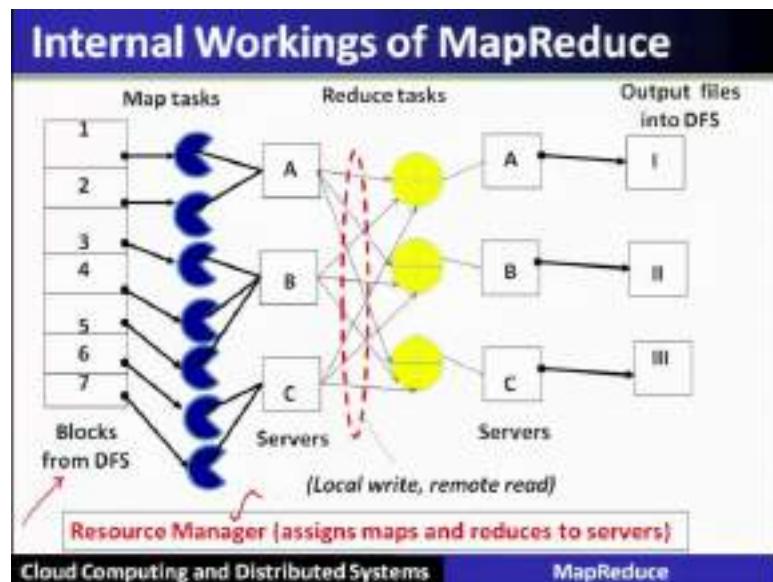
local file system = Linux FS, etc.
distributed file system = GFS (Google File System), HDFS (Hadoop Distributed File System)

At the bottom, there are two navigation buttons: 'Cloud Computing and Distributed Systems' on the left and 'MapReduce' on the right.

So, inside the MapReduce we have seen that parallelize map is quite easy now why because, each task in the map they are all independent. So, they can parallelly execute inside a cloud. To transfer the data from map to reduce a shuffle function is to be called and all map output with the same key is assigned to the same reduce job and this requires to use the partitioning function which is hash based portioning. So, that it will be uniformly distribute across all the servers which are running the reduce task.

Third step is to parallelize reduce is quiet easy why because, reduce is independent of each other, implement the storage for the map input, map output, reduce input, reduce output. So, here we can see that the map input is done from the distributed file system. Map output also uses the local file system and reduce input also from multiple remote disks using the local file system and reduce output node is done through the distributed file system. Distributed file systems which are used here in the MapReduce are either the Google File System or Hadoop Distribute File System.

(Refer Slide Time: 40:11)



Let us see the internal working of the MapReduce. So, first thing is whenever the input file is given. So, this particular input file is splits it is divided into different splits seven different splits are shown over here. Each and ever is split is given to different map tasks now these map tasks are to be assigned a servers or the machines. So, for example, here it is shown that two different map tasks are assigned to on server. A three different map tasks are assigned to the server B and two map tasks are assigned to the server C. So, it depends upon how many servers are there these map task will be assigned.

Similarly, the output of these servers will be shuffled; that means, they will be grouped by the keys and will be given to the reducer functions. So, here this example shows that there are three different reduce task is being are defined over here. And these reduce task will be assigned to different servers. So, there are three different servers are assigned to the reduce task which will be executing these reducers. At the output of these particular servers are stored in a file in a distributed file system, which are given different names or there maybe there could be only file will also it depends upon.

Now, who allocates who assigns these map tasks to the servers? And the reduce task also to the different servers may be the same servers. So, it is the resource manager, it is the resource manager of a distributed file system or a yarn server that we will see scheduler which will basically internally does this.

(Refer Slide Time: 48:24)

The YARN Scheduler

- Used underneath Hadoop 2.x +
- YARN = Yet Another Resource Negotiator
- Treats each server as a collection of **containers**
 - Container = fixed CPU + fixed memory
- Has 3 main components
 - Global Resource Manager (RM)**
 - Scheduling
 - Per-server Node Manager (NM)**
 - Daemon and server-specific functions
 - Per-application (job) Application Master (AM)**
 - Container negotiation with RM and NMs
 - Detecting task failures of that job

Cloud Computing and Distributed Systems MapReduce

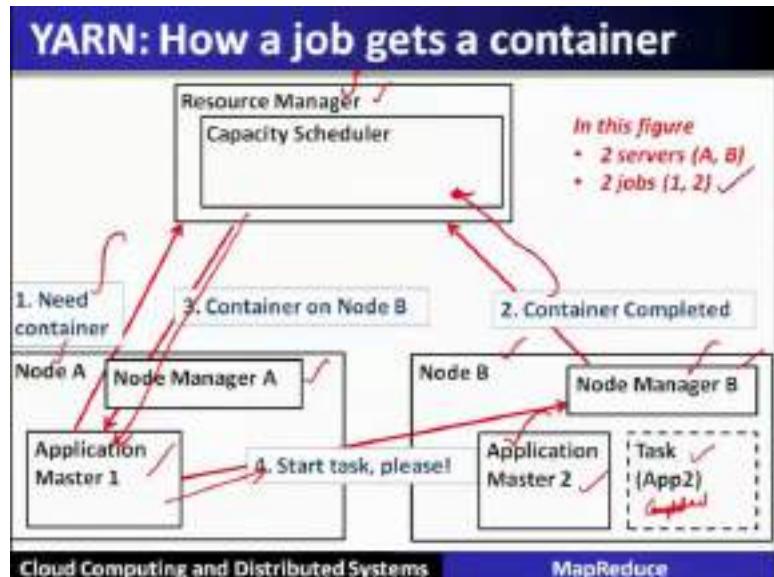
Let us see the yarn scheduler which is combined with the Hadoop 2.0 versions onwards. So, yarn full form is yet another resource negotiator. So, this particular yarn will treat each server as a collection of container. So, it means that, if the server is having some number of course, let us say 3 and it is having 3 GB of internal memory. So that means, one core and 1 GB will form one container so, it has now three containers. So, three different task can be assigned in this manner. So, this yarn scheduler has 3 different components, the use of one component we have seen in the previous slide that is called resource manager. Which assigns these containers to the different tasks that is to the map task and to the reduce task so that is called scheduling.

So, this there is a global resource manager which performs the scheduling of this particular task map or reduce task to this particular machines or the servers or to the container here in this terminology. Then another component of yarn is called the node manager. So, for every server there is a daemon process called node manager which basically, will manage all the server related functions so, that means, a server if it is giving three different containers so the node manager will manage it.

Then per application there is a application manager, application master. So, for every application there is a application master and this particular application master will negotiate with the resource manager and the node manager for allocating their tasks to

the machines. Also the application master is responsible to deal with the task failures at the time of execution of the job.

(Refer Slide Time: 44:45)



Let us see how the yarn will get a job of a container? So, here there is a resource manager and this is the example which is shown that there are two nodes node A and node B. So, the node A is having its manager node A manager and node manager B will manage this server node B. So, there is resource manager which will manage all the nodes which are available in this scenario that is through the yarn.

Now, for there are two applications 1 jobs job one and job 2. So, there are two different application master they will manage these two tasks. Now let us see the example of these interactions with the resource manager. Now let us assume that the application master to its task that is the application 2 task is completed. Therefore, this will return back through the node manager that it is free.

Similarly the application one master negotiates with the resource manager asking for a container, now since this container is available why because node B has completed its task. So, that particular container is now free so it will be assigned back to the master 1 and it will start the task execution.

So, this particular interaction of different resource manager, the node manager and the application master, they will ensure that the resources which are required to execute the applications are done. Let us see the fault tolerance, which is handled in map reduce.

(Refer Slide Time: 46:52)

Fault Tolerance

- Server Failure
 - NM heartbeats to RM
 - If server fails, RM lets all affected AMs know, and AMs take appropriate action
 - NM keeps track of each task running at its server
 - If task fails while in-progress, mark the task as idle and restart it
 - AM heartbeats to RM
 - On failure, RM restarts AM, which then syncs up with its running tasks
- RM Failure
 - Use old checkpoints and bring up secondary RM
 - Heartbeats also used to piggyback container requests
 - Avoids extra messages

Cloud Computing and Distributed Systems MapReduce

So, MapReduce handles different kind of faults at different level. For example, whenever there is a server failure so, this will affect the name node and resource manager, name manager, resource manager and application manager, will be affected in this particular scenario whenever there is a server failures. Similarly, whenever there is a resource manager failure then basically, it uses the secondary resource manager to failover and start working at.

(Refer Slide Time: 47:38)

Slow Servers

Slow tasks are called **Stragglers**

- The slowest task slows the entire job down (why?)
- Due to Bad Disk, Network Bandwidth, CPU, or Memory
- Keep track of "progress" of each task (% done)
- Perform backup (**replicated**) execution of straggler tasks
 - A task considered done when its first replica complete called **Speculative Execution**.

Cloud Computing and Distributed Systems **MapReduce**

These are detected using the heart beats which are contained in the request and also avoid. There is a possibility of slow servers called stragglers. So, the slowest task will slow down the entire job that we have already seen because, it is using the barriers concept. So, slow servers are due to the bad disk, network bandwidth, CPU, memory, there can be many different issues. So, it has to keep track of the progress of the task and performs the backup that is replicated execution of staggered task which is called as a speculative execution.

(Refer Slide Time: 48:18)

Locality

- **Locality**
 - Since cloud has hierarchical topology (**e.g., racks**)
 - GFS/HDFS stores 3 replicas of each of chunks (e.g., 64 MB in size)
 - Maybe on different racks, e.g., 2 on a rack, 1 on a different rack.
 - **Mapreduce attempts to schedule a map task on**
 1. a machine that contains a replica of corresponding input data, or failing that,
 2. on the same rack as a machine containing the input, or failing that,
 3. Anywhere

Cloud Computing and Distributed Systems **MapReduce**

Locality; that means, that the cloud has the hierarchical topology of a data centre that is the servers racks and different racks are connected through the top of racks switch. So, the locality has to be exploited, that means, MapReduce will try to attempt to assign the server for a particular task which is very close to it so, that the network delay can be avoided here in this case.

(Refer Slide Time: 48:58)

Implementation Overview

- Many different implementations of the MapReduce interface are possible. The right choice depends on the environment.
- **For example**, one implementation may be suitable for a small shared-memory machine, another for a large NUMA multi-processor, and yet another for an even larger collection of networked machines.
- Here we describes an implementation targeted to the computing environment in wide use at Google: large clusters of commodity PCs connected together with switched Ethernet.

Cloud Computing and Distributed Systems MapReduce

So, let us see some of the implementation of MapReduce, there are many implementation of MapReduce an interface an interfaces are available. So, let us see one such example.

(Refer Slide Time: 49:10)

Contd...

- (1) Machines are typically dual-processor x86 processor running Linux, with 2-4 GB of memory per machine.
- (2) Commodity networking hardware is used . Typically either 100 megabits/second or 1 gigabit/second at the machine level, but averaging considerably less in overall bisection bandwidth.
- (3) A cluster consists of hundreds or thousands of machines, and therefore machine failures are common.
- (4) Storage is provided by inexpensive IDE disks attached directly to individual machines.
- (5) Users submit jobs to a scheduling system. Each job consists of a set of tasks, and is mapped by the scheduler to a set of available machines within a cluster.

Cloud Computing and Distributed Systems

MapReduce

Here where we assume that the machines are dual processor commodity networks hardware is available and the clusters consist of hundred and thousands of machine is there.

(Refer Slide Time: 49:25)

Distributed Execution Overview

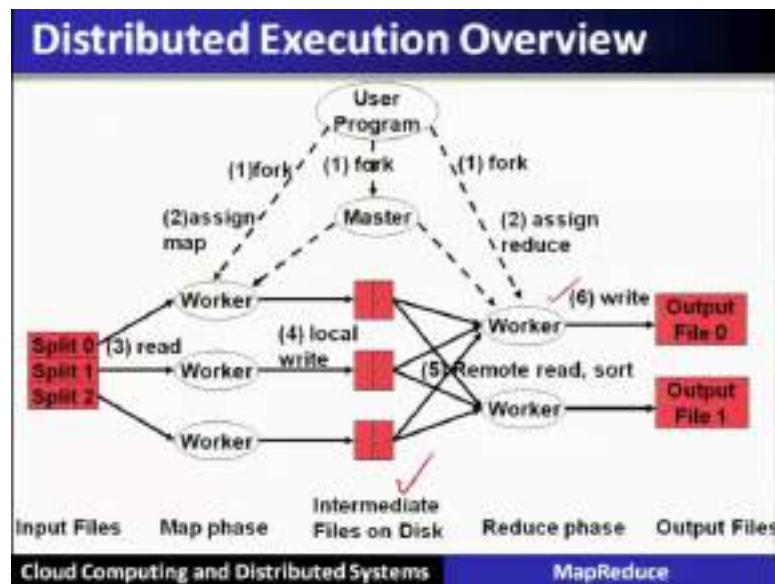
- The **Map invocations** are distributed across multiple machines by automatically partitioning the input data into a set of M splits.
- The input splits can be processed in parallel by different machines.
 -
- **Reduce invocations** are distributed by partitioning the intermediate key space into R pieces using a partitioning function (e.g., $\text{hash}(\text{key}) \bmod R$).
- The number of partitions (R) and the partitioning function are specified by the user.

Cloud Computing and Distributed Systems

MapReduce

And storage is also provided and user can submit the job to this kind of system. Let us see the distributed execution of this MapReduce invocation.

(Refer Slide Time: 49:32)



So, here we can see that the user program, user can submit their job in this particular manner and this particular user also gives the splits that is the input file is divided into different splits. And these particular user program, that is the map phase will generate different workers and these workers are assigned to the different machines. And when these splits argument to the worker they will output and they will write down on the intermediate files on the disks using some distributed file system.

Now, the next task is that these particular intermediate output is given to the input to the next phase that is to the reducer phase. And this reducer phase that is called worker will be assigned by the user program that is to the master. Now, then these particular workers are assigned by the user program that is through the master. Now, then these particular workers are assigned the servers in the reduce phase and they will output through the to the output file in the distributed file system. So, this is the typical scenario of a distributed execution now here we can see the sequence of actions in this distributed execution.

(Refer Slide Time: 50:51)

Sequence of Actions

When the user **program calls the MapReduce function**, the following **sequence of actions** occurs:

1. The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.
2. One of the copies of the program is special- the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task.
3. A worker who is assigned a map task reads the contents of the corresponding input split. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

Cloud Computing and Distributed Systems

MapReduce

So, the program calls the MapReduce function and the sequence of actions, which we have told is being listed over here. For example, MapReduce library will split the user file into different pieces of 16 MB to 64 MB and then one of these copies is called one of the program copies is called a master. And the rest are it will generate the different worker process and these workers are assigned the map task.

(Refer Slide Time: 51:31)

Contd...

4. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function.
 - The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.
5. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.
 - The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

Cloud Computing and Distributed Systems

MapReduce

And into the end each split will be assigned to these workers and periodically they are being buffered and partitioned. These reduce and these particular intermediate results will be stored in the file system.

(Refer Slide Time: 51:51)

Contd...

6. The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function.
 - The output of the Reduce function is appended to a final output file for this reduce partition.
7. When all map tasks and reduce tasks have been completed, the master wakes up the user program.
 - At this point, the MapReduce call in the user program returns back to the user code.

Cloud Computing and Distributed Systems **MapReduce**

Now, the reducer worker is notified by the master and this particular output, intermediate output of the previous stage will be given as the input and it will be reducer phase will give the output.

(Refer Slide Time: 52:03)

Contd...

- After successful completion, the output of the mapreduce execution is available in the R output files (one per reduce task, with file names as specified by the user).
- Typically, users do not need to combine these R output files into one file- they often pass these files as input to another MapReduce call, or use them from another distributed application that is able to deal with input that is partitioned into multiple files.

Cloud Computing and Distributed Systems **MapReduce**

(Refer Slide Time: 52:04)

Master Data Structures

- The master keeps several data structures. For each map task and reduce task, it stores the **state (idle, in-progress, or completed)**, and the identity of the worker machine (**for non-idle tasks**).
- The master is the conduit through which the location of intermediate file regions is propagated from map tasks to reduce tasks. Therefore, for each completed map task, the master stores the locations and sizes of the R intermediate file regions produced by the map task.
- Updates to this location and size information are received as map tasks are completed. The information is pushed incrementally to workers that have in-progress reduce tasks.

Cloud Computing and Distributed Systems

MapReduce

So here the master data structure is like this, that it basically has different states.

(Refer Slide Time: 52:18)

Fault Tolerance

- Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failures gracefully.
- **Map worker failure**
 - Map tasks completed or in-progress at worker are reset to idle
 - Reduce workers are notified when task is rescheduled on another worker
- **Reduce worker failure**
 - Only in-progress tasks are reset to idle
- **Master failure**
 - MapReduce task is aborted and client is notified

Cloud Computing and Distributed Systems

MapReduce

And it will trace through different states internally in the execution.

(Refer Slide Time: 52:27)

Task Granularity

- The **Map phase** is subdivided into M pieces and the **reduce phase** into R pieces.
- Ideally, M and R should be much larger than the number of worker machines.
- Having each worker perform many different tasks improves dynamic load balancing, and also speeds up recovery when a worker fails; the many map tasks it has completed can be spread out across all the other worker machines.
- There are practical bounds on how large M and R can be, since the master must make $O(M + R)$ **scheduling decisions** and keeps $O(M * R)$ **state in memory**.
- Furthermore, R is often constrained by users because the output of each reduce task ends up in a separate output file.

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 52:29)

Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g.,
hash(key) mod R
- Sometimes useful to override
 - E.g., **hash(hostname(URL)) mod R** ensures URLs from a host end up in the same output file

Cloud Computing and Distributed Systems

MapReduce

Fault tolerance we have already covered, locality is being exploited task granularity we have assumed partition function we have covered.

(Refer Slide Time: 52:31)

Ordering Guarantees

- It is guaranteed that within a given partition, the intermediate key/value pairs are processed in increasing key order.
- This ordering guarantee makes it easy to generate a sorted output file per partition, which is useful when the output file format needs to support efficient random access lookups by key, or users of the output find it convenient to have the data sorted.

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 52:38)

Combiners Function (1)

- In some cases, there is significant repetition in the intermediate keys produced by each map task, and the user specified Reduce function is commutative and associative.
- A good example of this is the word counting example. Since word frequencies tend to follow a Zipf distribution, each map task will produce hundreds or thousands of records of the form <the, 1>.
- All of these counts will be sent over the network to a single reduce task and then added together by the Reduce function to produce one number. We allow the user to specify an optional Combiner function that does partial merging of this data before it is sent over the network.

Cloud Computing and Distributed Systems

MapReduce

And ordering guarantees is also there that we have used we have shown in the sorting application and this is the combined function.

(Refer Slide Time: 52:44)

Combiners Function (2)

- The Combiner function is executed on each machine that performs a map task.
- Typically the same code is used to implement both the combiner and the reduce functions.
- The only difference between a reduce function and a combiner function is how the MapReduce library handles the output of the function.
- The output of a reduce function is written to the final output file. The output of a combiner function is written to an intermediate file that will be sent to a reduce task.
- Partial combining significantly speeds up certain classes of MapReduce operations.

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 52:47)

Example: 1 Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
    for each word w in value:
        emit(w, 1)
```

```
reduce(key, values):
// key: a word; values: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 52:51)

Count Illustrated

```
map[key=url, val=contents]:  
    For each word w in contents, emit(w, "1")  
reduce[key=word, values=uniq_counts]:  
    Sum all "1"s in values list  
    Emit result "(word, sum)"
```

see bob run
see spot throw

see 1
bob 1
run 1
see 1
spot 1
throw 1

bob 1
run 1
see 2
spot 1
throw 1

Cloud Computing and Distributed Systems

MapReduce

So, MapReduce more examples are available.

(Refer Slide Time: 52:52)

Example 2: Counting words of different lengths

- The map function takes a value and outputs key:value pairs.
- For instance, if we define a map function that takes a string and outputs the length of the word as the key and the word itself as the value then
 - map(steve) would return 5:steve and
 - map(savannah) would return 8:savannah.

This allows us to run the map function against values in parallel and provides a huge advantage.

Cloud Computing and Distributed Systems

MapReduce

This word count we have already done. Now the counting words of different length, if we want to find out. Then what we will do is? Given a particular input that is then we will written its length and that particular word also. So, it we will written the length and the word both will be emit in the map function.

(Refer Slide Time: 53:27)

Example 2: Counting words of different lengths

Before we get to the reduce function, the mapreduce framework groups all of the values together by key, so if the map functions output the following **key:value pairs**:

3 : the
3 : and
3 : you
4 : then
4 : what
4 : when
5 : steve
5 : where
8 : savannah
8 : research

They get grouped as:

✓ 3 : [the, and, you]
✓ 4 : [then, what, when]
✓ 5 : [steve, where]
✓ 8 : [savannah, research]

Cloud Computing and Distributed Systems MapReduce

So, that is the case then for example, all these particular words, first their length and their words will be written. And then reduce function will combine them using that particular key that is the length of the words. And in this case for every length of word every length different keywords will be combined in the form of a list.

(Refer Slide Time: 53:58)

Example 2: Counting words of different lengths

- Each of these lines would then be passed as an argument to the reduce function, which accepts a key and a list of values.
- In this instance, we might be trying to figure out how many words of certain lengths exist, so our reduce function will just count the number of items in the list and output the key with the size of the list, like:

3 : 3
4 : 3
5 : 2
8 : 2

Cloud Computing and Distributed Systems MapReduce

And this shows that the word that is having the length 3 there are 3 different words and length 4 there are 3 different words, length 5 there are 2 different and length 8 there are 2 different words. So, here we have counted the words of different length.

(Refer Slide Time: 54:10)

Example 2: Counting words of different lengths

- The reductions can also be done in parallel, again providing a huge advantage. We can then look at these final results and see that there were only two words of length 5 in the corpus, etc...
- **The most common example of mapreduce is for counting the number of times words occur in a corpus.**

0

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 54:14)

Example 3: Finding Friends

- Facebook has a list of friends (note that friends are a bi-directional thing on Facebook. If I'm your friend, you're mine).
- They also have lots of disk space and they serve hundreds of millions of requests everyday. They've decided to pre-compute calculations when they can to reduce the processing time of requests. **One common processing request is the "You and Joe have 230 friends in common" feature.**
- When you visit someone's profile, you see a list of friends that you have in common. This list doesn't change frequently so it'd be wasteful to recalculate it every time you visited the profile (sure you could use a decent caching strategy, but then we wouldn't be able to continue writing about mapreduce for this problem).
- We're going to use mapreduce so that we can calculate everyone's common friends once a day and store those results. Later on it's just a quick lookup. We've got lots of disk, it's cheap.

Cloud Computing and Distributed Systems

MapReduce

(Refer Slide Time: 54:16)

Example 3: Finding Friends

- Assume the friends are stored as Person->[List of Friends], our friends list is then:

- A -> B C D
- B -> A C D E
- C -> A B D E
- D -> A B C E
- E -> B C D

Cloud Computing and Distributed Systems

MapReduce

Another example is to find a common friends, list of common friends. So, here every input every person will provide a list of their friends. So, A B C D E they have provided their own list of friends, if this is the input given to the map function.

(Refer Slide Time: 54:29)

Example 3: Finding Friends

For map(A -> B C D) :

- (A B) -> B C D
- (A C) -> B C D
- (A D) -> B C D

For map(B -> A C D E) : (Note that A comes before B in the key)

- (A B) -> A C D E
- (B C) -> A C D E
- (B D) -> A C D E
- (B E) -> A C D E

Cloud Computing and Distributed Systems

MapReduce

Map function will output, for every such list it will give a pair that is A B will list out this particular list of friends. And A C will list out friends and similarly A D. Similarly the person number B will give it is list and this also will be emitted in this particular format.

(Refer Slide Time: 54:56)

Example 3: Finding Friends

For map($C \rightarrow A B D E$):

$(A C) \rightarrow A B D E$
 $(B C) \rightarrow A B D E$
 $(C D) \rightarrow A B D E$
 $(C E) \rightarrow A B D E$
For map($D \rightarrow A B C E$):
 $(A D) \rightarrow A B C E$
 $(B D) \rightarrow A B C E$
 $(C D) \rightarrow A B C E$
 $(D E) \rightarrow A B C E$

And finally for map($E \rightarrow B C D$):

$(B E) \rightarrow B C D$
 $(C E) \rightarrow B C D$
 $(D E) \rightarrow B C D$

Cloud Computing and Distributed Systems

MapReduce

So, when all these list are output.

(Refer Slide Time: 55:00)

Example 3: Finding Friends

- Before we send these key-value pairs to the reducers, we group them by their keys and get:

\checkmark \checkmark *values pair*
 $(A B) \rightarrow (A C D E) (B C D)$
 $(A C) \rightarrow (A B D E) (B C D)$
 $(A D) \rightarrow (A B C E) (B C D)$
 $(B C) \rightarrow (A B D E) (A C D E)$
 $(B D) \rightarrow (A B C E) (A C D E)$
 $(B E) \rightarrow (A C D E) (B C D)$
 $(C D) \rightarrow (A B C E) (A B D E)$
 $(C E) \rightarrow (A B D E) (B C D)$
 $(D E) \rightarrow (A B C E) (B C D)$

Cloud Computing and Distributed Systems

MapReduce

Then as far as the reducer is concerned, what reducer will do? Reducer will now for every keyword which is there. And the list which is basically available in the intermediate they will take the intersection. Intersection of these particular list of common friends.

(Refer Slide Time: 55:25)

Example 3: Finding Friends

- Each line will be passed as an argument to a reducer.
- The **reduce function will simply intersect the lists of values** and output the same key with the result of the intersection.
- For example, **reduce((A B) -> (A C D E) (B C D))** will **output (A B) : (C D)**
- **and means that friends A and B have C and D as common friends.**

Cloud Computing and Distributed Systems

MapReduce

So, intersection will find out the list of common friends and that will be output.

(Refer Slide Time: 55:27)

Example 3: Finding Friends

- The result after reduction is:

(A B) -> (C D)



(A C) -> (B D)

(A D) -> (B C)

(B C) -> (A D E)

Now when D visits B's profile,
we can quickly look up (B D) and
see that they have three friends
in common, (A C E).

(B D) -> (A C E)

(B E) -> (C D)

(C D) -> (A B E)

(C E) -> (B D)

(D E) -> (B C)

Cloud Computing and Distributed Systems

MapReduce

So, this is the result which is shown over here. So, using MapReduce all these particular program become simple, but writing or thinking about this program is not so, trivial.

(Refer Slide Time: 55:39)

Reading

Jeffrey Dean and Sanjay Ghemawat,

"MapReduce: Simplified Data Processing on Large Clusters"

<http://labs.google.com/papers/mapreduce.html>

Cloud Computing and Distributed Systems MapReduce

More details of this MapReduce you can refer to this particular paper by Jeffrey Dean and Sanjay Ghemawat that is the MapReduce simplified data processing on large clusters.

(Refer Slide Time: 55:57)

Conclusion

- The MapReduce programming model has been successfully used at Google for many different purposes.
- The model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing.
- A large variety of problems are easily expressible as MapReduce computations.
- For example, MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems.

Cloud Computing and Distributed Systems MapReduce

So, conclusion MapReduce programming model has been successfully used in the Google and many different purposes.

(Refer Slide Time: 56:08)

Conclusion

- Mapreduce uses **parallelization + aggregation** to schedule applications across clusters
- Need to deal with failure
- Plenty of ongoing research work in **scheduling and fault-tolerance for Mapreduce and Hadoop.**

Cloud Computing and Distributed Systems

MapReduce

So, we have seen that for a large size data set, this particular simple programming paradigm we have also seen the internal details. We have also shown how it is automatically dealing with the parallelization and dealing with the failures. There are many ongoing research work happening in the in the scheduling fault tolerance in the scenarios of MapReduce and Hadoop.

Thank you.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture-20
Introduction to Spark

Introduction to Spark.

(Refer Slide Time: 00:21)

Preface

Content of this Lecture:

- In this lecture, we will discuss the '**framework of spark**', Resilient Distributed Datasets (RDDs) and also discuss some of the applications of spark such as; **Page rank and GraphX**.

Cloud Computing and Distributed Systems Introduction to Spark

Preface content of this lecture, we will cover framework of a spark that is also having the resilient distributed datasets. And we will discuss some of the applications of a spark such as page rank and GraphX.

(Refer Slide Time: 00:47)

Need of Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.
- It is an another system for big data analytics
- **Isn't MapReduce good enough?**
 - Simplifies batch processing on large commodity clusters

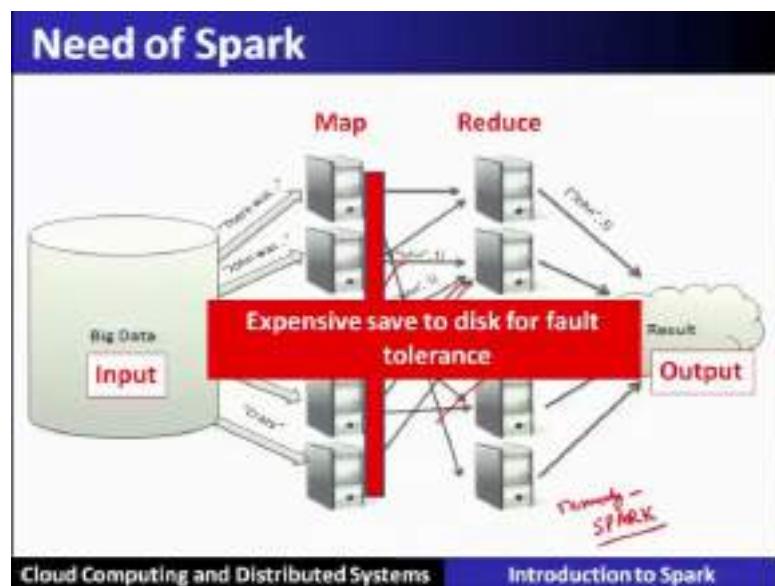
Cloud Computing and Distributed Systems

Introduction to Spark

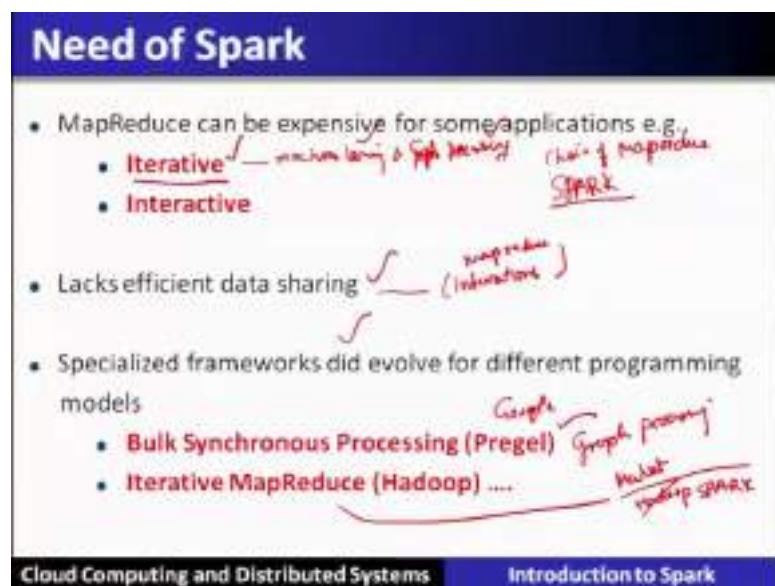
Need of a spark. Apache spark is a big data analytics framework that was originally developed at university of California Berkeley, in 2012. Since then it has gained a lot of attraction both academia and the industry and lot of development is going on in this particular platform.

This particular platform is another system alternative to the Hadoop MapReduce that we have seen in the previous lectures. So, why not the MapReduce still continuing or is not good enough. So, we have seen that MapReduce simplified the batch processing on a large commodity clusters.

(Refer Slide Time: 01:49)



(Refer Slide Time: 01:51)



But it suffers from some of the problems such as the iterative applications, for example, which requires lot of iterations such as machine learning and graph processing. This particular MapReduce performs very badly in these 2 different type of applications. Also in the interactive applications also MapReduce becomes very costly, in the since it slows down.

Let us analyze why this problem is there in the MapReduce. So, let us see that a typical example of a word count. The input data file contains, let us say it is a Wikipedia file. If

it is input, then this will be read into the system that is the MapReduce system and this particular file is divided into the splits. And each split is given to different workers. These workers will then apply the map function. And this map function will output the intermediate results in the form of key value pair.

For example, here, John 1, it will be emitted out of the MapReduce and this particular output will be stored in HDFS file system. Which in cause lot of I/O's; that means, the intermediate data which will be emitted out of map function, will be stored in a file system like HDFS. So, the drawback of storing this file means that lot of I/O's are involved, which will slow down the entire computing.

The next phase which is called the reduced phase will again read from these intermediate key values from the file system. Then it will read from the file system and again incur more your overhead. And this input will be given to the workers which will apply the reduce function on it and the result will be given back. Again through the HDFS using file system. This particular advantage of store is that the intermediate values are made persistent hence achieves the fault tolerance.

Although fault tolerance will be achieved there is no data loss in between the map to the reduce function, but it incurs lot of overhead and most of the applications which basically runs over several iterations of MapReduce again will be slowed down very much. Here it is written that this expensive, a storage to the disk will achieve the fault tolerance, but will be losing in the terms of time the processing takes a lot of time.

So, what is the remedy? Remedy is the emergence of that is spark system for such large scale computing on the cluster. Yet it is going to achieve the speed of in terms of in memory storage between map and reduce or across the iterations. So, with this particular idea in place the need of the spark has given the place. Therefore, the application such as iterative application which require several chain of MapReduce jobs such as the machine learning or the graph processing applications which are called iterative applications MapReduce can be expensive therefore, alternative is the spark.

This is spark is required because the mapreduce lacks the efficient data sharing in the iterations or it is also called the MapReduce iterations. Therefore, a different paradigm than MapReduce is very much needed at this point of time. Before the emergence of a spark and the issues of expensiveness in iterative applications, lot of in between lot of

framework were evolved having different data model such as Googles Pregel data system, which is meant for the graph processing. And the iterative MapReduce they were all created to overcome this particular problem of MapReduce.

Now, once this spark is being available or is being developed all this particular applications are now shifted, for using the spark system for example, to run the mahout for machine learning iterative machine learning in the MapReduce mahout uses earlier the Hadoop file system, but now with the emergence of a spark they are using the spark framework even with the mahout.

(Refer Slide Time: 11:17)

Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Immutable, partitioned collection of records ✓✓
- Built through coarse grained transformations (map, join ...)
- Can be cached for efficient reuse

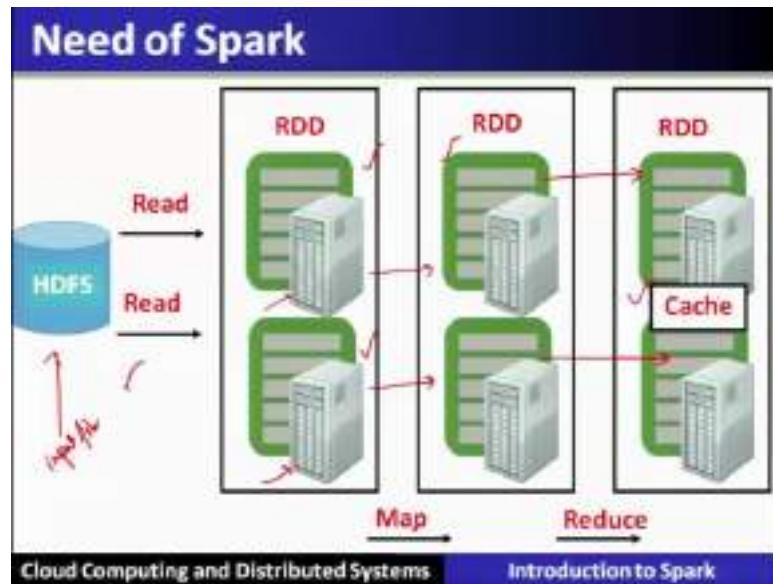
Cache in-memory

Cloud Computing and Distributed Systems Introduction to Spark

So, the solution which was available in the spark is in the form of resilient distributed datasets. So, resilient distributed data set that is called RDDs they are immutable partitioned collection of records.

They are built through the course grade transformations such as map and join and can be cached for efficient reuse. So; that means, this particular resilient distributed datasets can be there in memory operation or can be cached. Therefore, it is able to solve the problem of storing the intermediate results into the file system.

(Refer Slide Time: 12:21)



Let us understand the need of file spark in more detail. So, when we give the input file in the form of HDFS, this will split and these splits are read into the RDDs which will run on different clusters.

Then the map function will be applied and on this particular RDDs, the data will be converted in the form of RDDs and the map function will be applied to the RDDs. It will be transformed using map function and this RDDs will remain in cache in memory; that means, does not require the file system to store after the transformation using map function. Similarly, the data is in the memory there is no use of the file system. Only at the time of reading the file system is being used, then applying the reduce function using in memory will become quite efficient and these situations are speed up 100 times.

Because I/O's used to slow down these operations with in memory operations using RDDs this particular operation becomes very efficient. The output of the MapReduce still will be remain available in the cache. So, that further queries can be on the result can be made efficiently if they are access through the cache. So, this is how the spark system is able to solve this computation over the clusters and also able to compute in a very fast speed.

(Refer Slide Time: 15:01)

Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Immutable, partitioned collection of records ✓
- Built through coarse grained transformations (map, join ...)

Fault Recovery? ✓ *Lineage!* ✓ *Lost*

- Log the coarse grained operation applied to a partitioned dataset ✓
- Simply recompute the lost partition if failure occurs! ✓
- No cost if no failure ✓

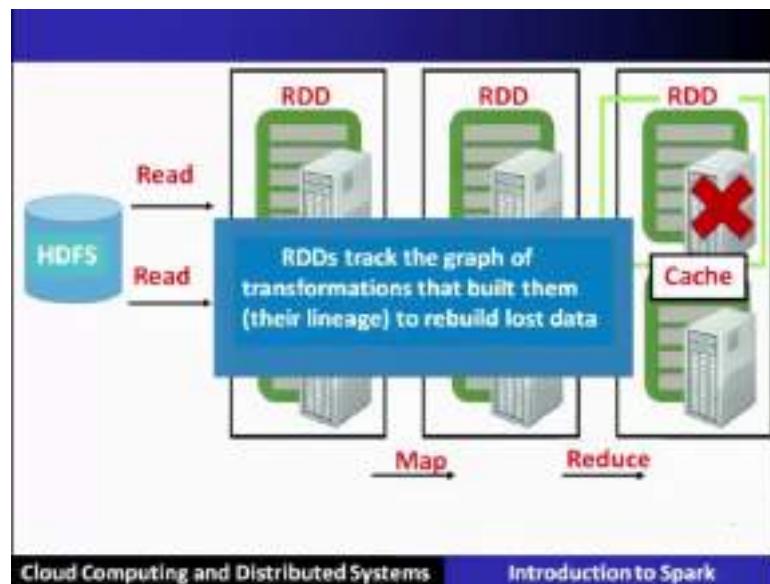
Cloud Computing and Distributed Systems Introduction to Spark

So, resilient distributed data set RDDs. They are immutable partition collection of records and built through the coarse grained transformation some of them we have seen like map join and so on.

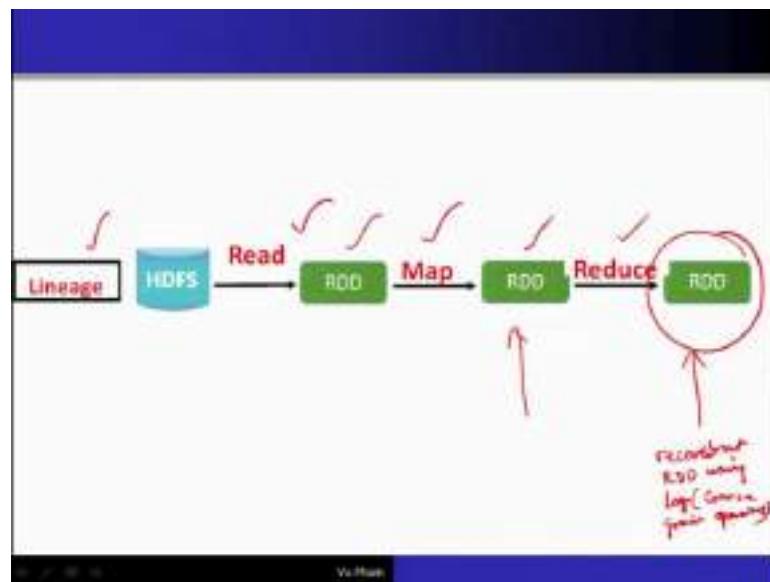
This particular RDDs also ensures fault tolerance. Like we have seen that it will be an in memory computations of RDDs across map and reduce function. So, there is a possibility of failures. Let us see how the failure recovery is supported here in a spark, the failure recovery is supported through an operation which is called lineage. To support the lineage, it will not do the check pointing, but it will do the it will record the logs of the coarse grained operations applied to the partition data sets. So, whenever there is a failure it will simply re compute those lost partitions if the failure occurs. Hence it will not incur any overhead cost of check pointing if there is no failure.

Therefore, the failure recovery in this spark is through the lineage. And also this will be not having any cost. So, basically speed is an important and also it ensures using lineage to reconstruct that RDD and it can recover from the failure also.

(Refer Slide Time: 17:03)



(Refer Slide Time: 17:08)



Let us see in this particular example let us say here this particular RDD is crashed or failed. If it is failed and the previous stage that is the map phase through the log, it will reconstruct the data that is RDDs in this particular form using that particular operation on that specific RDDs not all RDDs are affected.

So, lot of coarse grained operations are good enough to reconstruct the RDDs in case of the failures and the failures are not happening if the failure if there is no failures then

basically there is no overhead hence the computation will be speeded up in this particular scenario. Let us see the lineage how the lineage will be done. So, the data will be read and will be in the form of RDDs. Then the map will be applied and again a reduce will be applied to generate the final RDDs. So, at any stage if it if there is a failure so, from that stage we can reconstruct the RDDs.

So, here it is shown that the RDDs track the graph of the transformations that build them their lineage to rebuild the lost data here that is shown in this particular picture. So, the failure trans failure tolerant or a fault tolerance is also supported using lineage in spark.

(Refer Slide Time: 19:50)

What can you do with Spark?

- **RDD operations**
 - Transformations e.g., filter, join, map, group-by ...
 - Actions e.g., count, print ...
- **Control**
 - **Partitioning:** Spark also gives you control over how you can partition your RDDs.
 - **Persistence:** Allows you to choose whether you want to persist RDD onto disk or not.

*default Map-Reduce
Flexible - Controls*

Cloud Computing and Distributed Systems Introduction to Spark

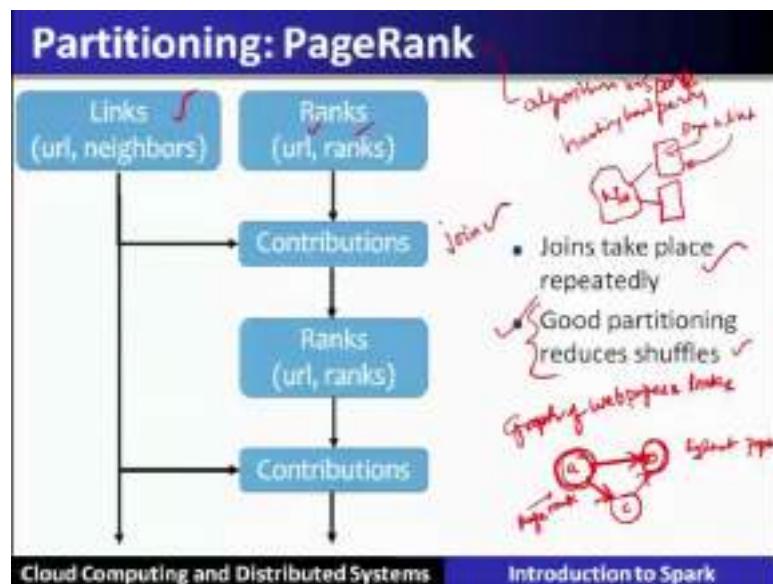
Now let us see what we can do in the spark what are the operations which are supported for RDDs. So, RDDs different operations spark supports for the transformation of RDDs such as it will filter; that means, remove some of the data and whatever is data required that is in the form of filter, it will also allow the join operation map and group by all these transformations are possible in the RDDs.

There are certain actions also, which are supported here in RDD operations such as count and so on. Besides these RDD operations the control of these RDDs are also available. So; that means, for partitioning it will not be automatically done as in case of MapReduce, but here it will be available in the form of the control. So, the partitioning whatever is required in the applications can be controlled and is done. So, spark also

gives you control over how you can partition your RDDs. So, that all is available and it is done in the form of control.

Similarly, the persistence also is available in the form of controls, super systems allows you to choose whether you want to persist the RDDs on the disk or not. So, these 2 things were default embedded into the MapReduce, whereas, here it is in the form of flexible in the form of controls.

(Refer Slide Time: 22:03)



So, let us take an example of partitioning in an application which is called a page rank. Now you know that page rank algorithm requires a graph of web pages and their links. Take this particular example, a b c they represents the web pages and the webpage a refers to the webpage.

Similarly, a refers c and c refers b and so on. So, here we have to find out that page which is having highest popularity in this particular page rank algorithm. This is widely used algorithm in the different search engine like Google uses this PageRanks algorithm. So, this PageRanks algorithm requires these urls and also the links to the neighbours as the input data. And it will compute the ranks that is called a PageRanks using a particular formula, which require these information that is url and their neighbours and compute the rank this is an iterative algorithm; that means, every iteration it computes these PageRanks using join operation; that means, here different links which are referring to a particular webpage will basically a join.

So, join will take repeatedly across all the iterations. Now if these joins requires the partitioning in different clusters. Then lot of network access is required during the join operations. So, if there is a good partitioning so, that all the links which are required in the join operations they can be communicating within a particular cluster. Than that is called a good partitioning which can be done using the hash based partitioning. So, in the hash based partitioning; that means, the page and all its constituents will be mapped to the same cluster to the same node in the cluster. Hence this will reduce automatically the number of shuffles.

So, the number of shuffles across over the iterations using this join operations can be possible to be computed in memory hence the pagerank algorithm, if being implemented on spark will be more efficient in comparison to the MapReduce operations.

(Refer Slide Time: 25:59)

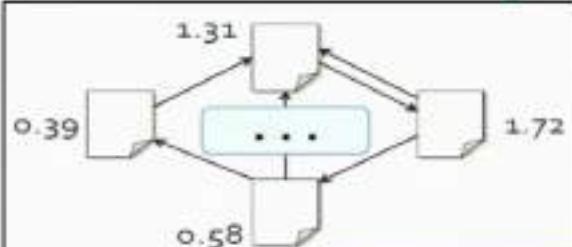


The in comparison to the MapReduce framework. Let us see the example of a page rank. Here, you can see this is the most popular page why because everyone is pointing towards that particular page and a small circle shows that they are not that popular why because nobody is having the reference to that particular page. So, the sizes of this page shows the rank that is the one which is having the highest number of links from many different pages having the highest rank. So, we want to rank all these pages using this particular information.

(Refer Slide Time: 26:54)

Algorithm

Step-1 Start each page at a rank of 1
Step-2 On each iteration, have page p contribute $\text{rank}_p / |\text{neighbors}_p|$ to its neighbors
Step-3 Set each page's rank to $0.15 + 0.85 \times \text{contributions}$



Cloud Computing and Distributed Systems Introduction to Spark

So, here this particular algorithm will have several steps, let us understand those steps then we will explain the PageRank algorithm, step one will start each page with the rank one on each iterations each page will contribute the rank divided by number of neighbors to that neighbor and third step will set the page rank to $0.15 + 0.85$ into contributions. So, for example, this node to find out the page rank of this particular node which is being referred by this page let us say A, B and C they are referring to this particular page. So, PageRank of let us say D is equal to the page rank of A divided by the number of outgoing links here it is 1 plus, this is the page rank in the next iteration.

This is the bageth iteration similarly as far as it will also contribute the page rank of B. So, the page rank of B at the ith iteration divided by how many outgoing links are there this is 1 and this is 2 plus PageRank of C, divided by how many number of outgoing links 1 and 2. So, in every iteration this particular page rank will be calculated till it converges.

(Refer Slide Time: 29:08)

Spark Program

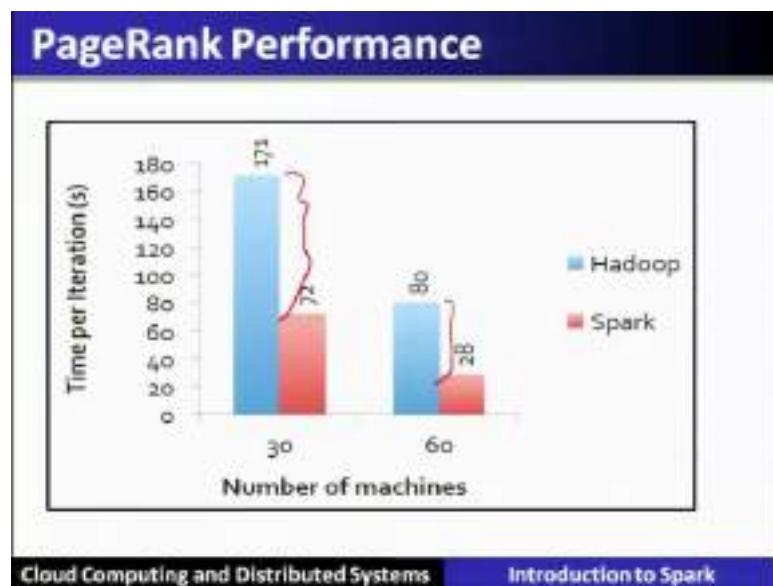
```
val links = // RDD of (url, neighbors) pairs ✓
var ranks = // RDD of (url, rank) pairs ✓
for (i <- 1 to ITERATIONS) {
    val contribs = links.join(ranks).flatMap {
        case (url, (links, rank)) =>
            links.map(dest => (dest, rank/links.size))
    }
    ranks = contribs.reduceByKey(_ + _)
        .mapValues (0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

Cloud Computing and Distributed Systems Introduction to Spark

So, let us see how the spark will be used in this programming of the PageRank, here we have the links that is were which are the outgoing links url and the neighbor pairs and also the previous rank of the previous iterations that also is the url and the ranks these 2 are the inputs for the iterations to be.

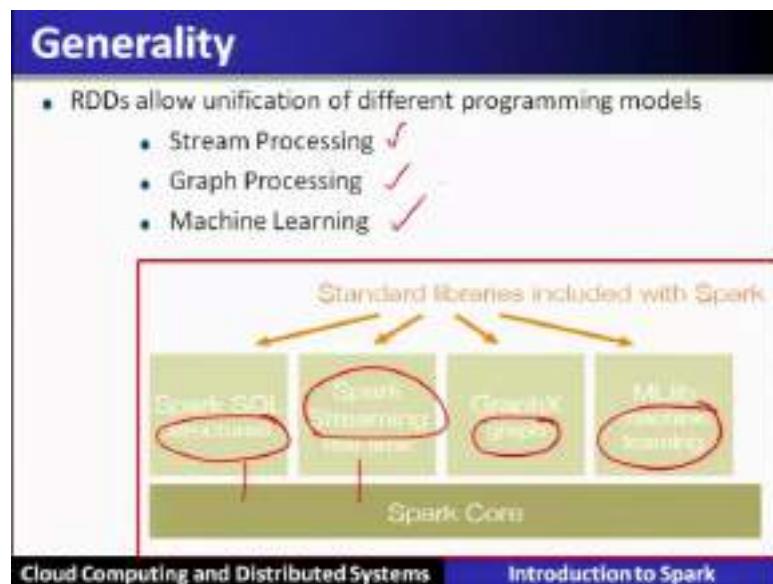
Now, the iterations will begin till it converges. Now as per the joints are concerned it will be done using flat map. And using map function easily this particular computations can be done and it will be reduced and computed the map values in this particular scenario. So, between the map and reduce, it will be the data will be in cache, hence this particular map and reduce function will share the data across iterations or in any iterations. The final ranks will be saved in a text file in between there is no use of file system.

(Refer Slide Time: 30:58)



If you compare the performance of same algorithm using Hadoop and spark, we can see the difference is quite huge.

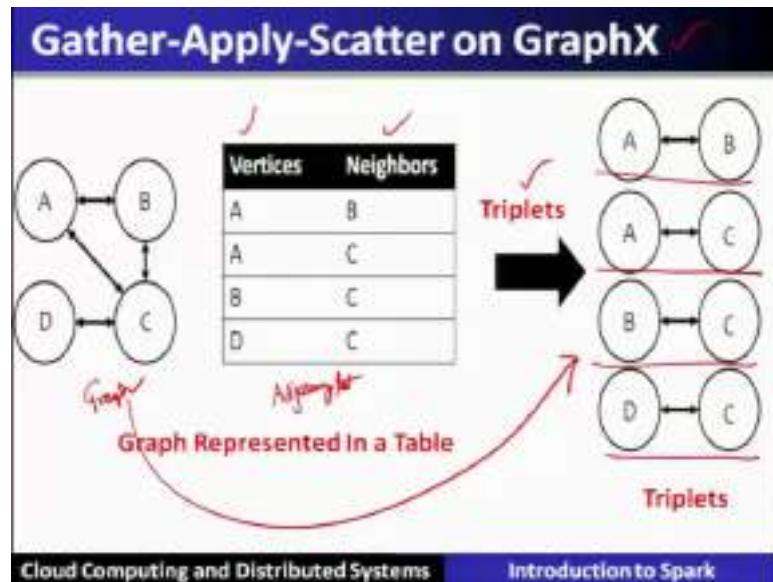
(Refer Slide Time: 31:22)



So, the spark, which supports the RDDs is many times faster than the MapReduce. Hence, it allows many different applications to be supported. There for a spark supports when use different type of libraries which is built over the spark. They are called spark streaming that is a real time streaming applications, then GraphX for graph applications and MLLib for machine learning applications. And structured SQL for database

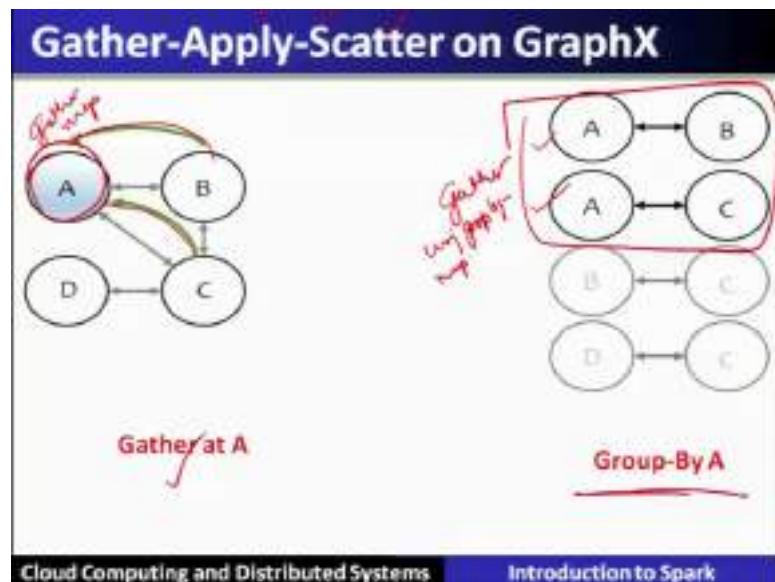
application these are different kind of so, we will. So, then; that means, there are stream processing graph applications and machine learning can easily are available using the libraries over the spark.

(Refer Slide Time: 32:34)



Let us see how the GraphX provides support for graph applications. Let us assume a the graph which is represented by these vertices and their corresponding neighbors, in the form of adjacency list. In a spark this graph can be represented as the triplets. So, triplet means A B C it is a triple. So, it will be represented as A B then A C then B C and then C D these triplets are represented. So, graph is represented in the form of this particular triplets. Having partition, the graph into the triplets, now it will be easier to do the graph processing in this particular triplet forms.

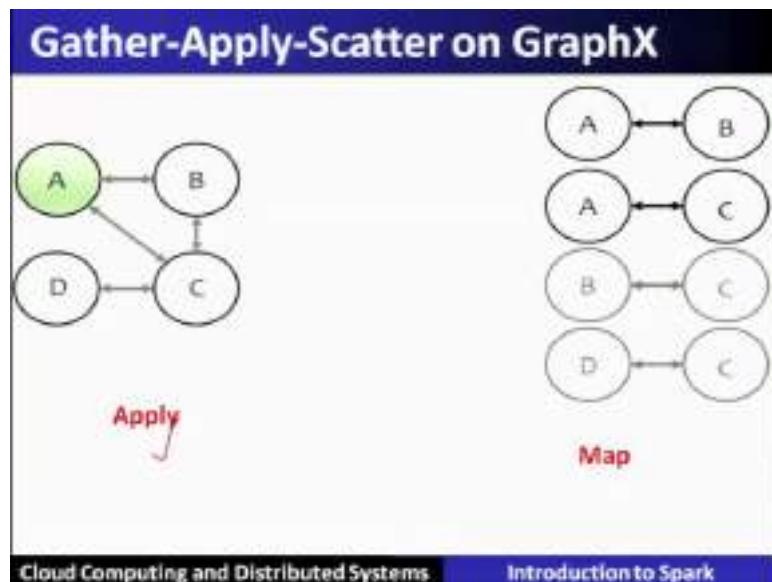
(Refer Slide Time: 33:54)



So, for that it supports GraphX supports the operation which is called gather apply scatter, let us see how this particular triplet supports this gather apply in the scatter. So, gather let us understand for the node A. So, the node A will collect the data from the triplets B and C.

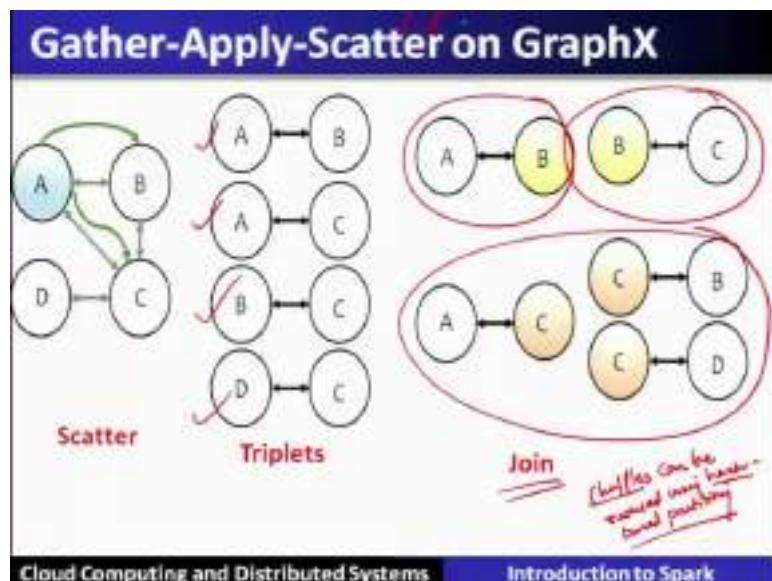
Then it will perform the apply will be quite easy because once the data is collected gathered then it will apply the map function on the data and by scatter; that means, the result of the map function has to be scattered across all different triplets. So, gather is easy using the group by operation which are supported in a spark. So, group by will basically identify these 2 different triplets, and the gather operation using, group by is easy to operate and then will perform the map function on it.

(Refer Slide Time: 35:39)



So, the next phase is called apply. So, once the group by and that is called gather is completed then applying map will be quite easy.

(Refer Slide Time: 35:53)

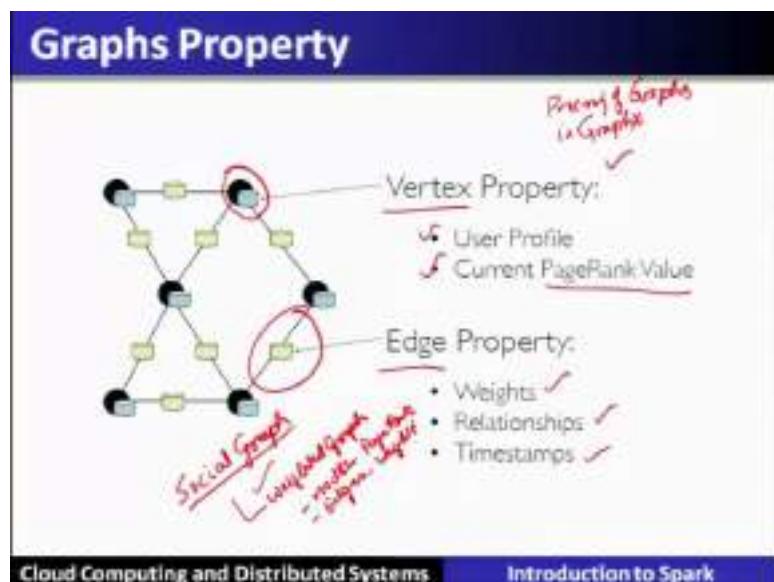


Now, then scatter that is that particular value out of the map has to be given to all it is corresponding neighbors using scatter. Now here we see that when this scatter will happen. So, it will basically affect these 2 triplets. And also it will affect the other triplets also in this particular form. So; that means, if the data is stored across several partitioned

across different clusters. Then this particular operation which is called scatter that is implemented through join and this becomes costlier in that case.

So, let us see that the scatter between A and B, and B and C is quite different than this particular scatter between A and C and C and B and C and D. So, therefore, the join operation will require lot of scatter requires lot of shuffles and here requires the use of good partitioning algorithm which will reduce the number of shuffles in it is scatter operation. So, using hash based partitioning this number of shuffles when this is scatter is applied can be reduced. So, here lot of design controls are available. So, that can be still controlled and good performance can be achieved. So, let us go and more detail of GraphX APIs.

(Refer Slide Time: 38:43)



The graphs APIs let us go through the GraphX APIs which are useful for the graph processing. Now, every graph has some properties, with graph has access supported in the form of APIs which can be used to design the algorithms and form the graph based competitions of the data set. So, different data sets can be represent in the form of a graph and using GraphX they can be easily processed and computed. Let us see some of the properties so, the graph comprises of the vertices and the edges. The vertices are having the properties such as if this particular graph represents the people and their relationship or it is basically called a social graph. Then this vertex will nothing, but it will be having the user profile of a particular person.

And also another important value is about the current PageRank value in this particular scenario; that means, the links to that particular vertex is having how many what is the importance of that particular vertex is it is PageRank value. Similarly, the edges will have the weights on that particular edge. It also represents the relationship between these 2 connections and also the timestamp when these weights are being computed. So, this is a kind of weighted graph, where nodes are also having the PageRank that is the weights and edges also having the weights defined. So, having defined the graph of this particular nature.

(Refer Slide Time: 41:40)

Creating a Graph (Scala)

```

type VertexId = Long

val vertices: RDD[(VertexId, String)] = sc.parallelize(List(
  (1L, "Alice"),
  (2L, "Bob"),
  (3L, "Charlie")))

class Edge[ED](
  val srcId: VertexId,
  val dstId: VertexId,
  val attr: ED)

val edges: RDD[Edge[String]] = sc.parallelize(List(
  Edge(1L, 2L, "coworker"),
  Edge(2L, 3L, "friend")))

val graph = Graph(vertices, edges)

```

The diagram illustrates a graph with three nodes, each represented by a black circle with a number and a blue rectangular label. Node 1 is labeled 'Alice' and has a green box below it labeled 'coworker'. Node 2 is labeled 'Bob' and has a green box below it labeled 'friend'. Node 3 is labeled 'Charlie'. A red curved arrow connects node 1 to node 2, and another red curved arrow connects node 2 to node 3.

Cloud Computing and Distributed Systems Introduction to Spark

Different relationship can be defined here in the GraphX and then RDDs can be evolved and these relationships will exploit the controls and the transformations to process the graph. So, for example, here Alice and Bob their relationship is co worker. So, most of the time they will be together working or basically having the relation of co worker, which will be defined here in the edge. Similarly expires between Bob and Charlie the relationship is friend. So, this can be also defined. So, RDDs will use this particular information in the graph to create the graph and also define all these particular relationships.

(Refer Slide Time: 43:05)

Graph Operations (Scala)

```
class Graph[VD, ED] {
    // Table Views ...
    def vertices: RDD[(VertexId, VD)]
    def edges: RDD[Edge[ED]]
    def triplets: RDD[EdgeTriplet[VD, ED]]
    // Transformations ...
    def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
    def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD, ED2]
    def reverse: Graph[VD, ED]
    def subgraph(spred: EdgeTriplet[VD, ED] => Boolean,
                vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
    // Joins ...
    def outerJoinVertices[U, VD2](
        tbl: RDD[(VertexId, U)])
        (f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
    // Computation ...
    def aggregateMessages[A](
        sendMsg: EdgeContext[VD, ED, A] => Unit,
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]
```

Cloud Computing and Distributed Systems

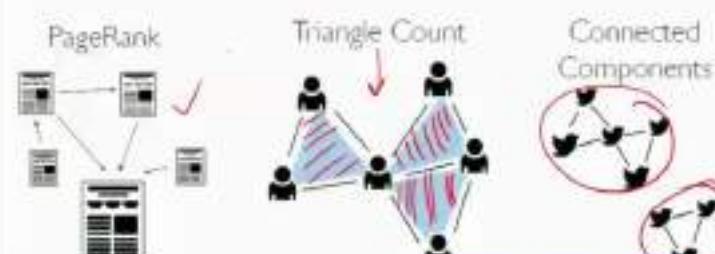
Introduction to Spark

In the GraphX, using the language which is called a scalar. So, scalar provides different API's, where in the vertices edges triplets and different sub graphs then different computations of the sub graph communicate across all this edges all this particular operations are well defined in the form of API's in GraphX, which can be used to program it.

(Refer Slide Time: 43:46)

Built-in Algorithms (Scala)

```
// Continued from previous slide
def pageRank(tol: Double): Graph[Double, Double]
def triangleCount(): Graph[Int, ED]
def connectedComponents(): Graph[VertexId, ED]
// ...and more: org.apache.spark.graphx.lib
```



Cloud Computing and Distributed Systems

Introduction to Spark

So, there are some built in algorithms also which can directly be used. For example, the PageRank algorithm is already built in the GraphX package and will be used directly as an API.

Similarly, to count how many triangles are there for example, here to count how many triangles are there which will give the relationship values. And also in a graph what are the connected components if a graph is disconnected then finding out the connected components all this particular algorithm graph algorithms are already implemented and can be used by just calling the API's.

(Refer Slide Time: 44:33)

The triplets view

```
class Graph[VD, ED] {  
    def triplets: RDD[EdgeTriplet[VD, ED]]  
}  
  
class EdgeTriplet[VD, ED]{  
    val srcId: VertexId, val dstId: VertexId, val attr: ED,  
    val srcAttr: VD, val dstAttr: VD  
}
```

The diagram shows a graph with three nodes labeled 1, 2, and 3. Node 1 is connected to node 2 with an edge labeled 'Alice'. Node 2 is connected to node 3 with an edge labeled 'Bob'. Node 1 is also connected to node 3 with an edge labeled 'Charlie'. An arrow points from this graph to an RDD table on the right. The RDD table has columns 'srcAttr', 'dstAttr', and 'attr'. It contains three rows: one for Alice (srcAttr: Alice, dstAttr: Bob, attr: coworker), one for Bob (srcAttr: Bob, dstAttr: Charlie, attr: friend), and one for Charlie (srcAttr: Charlie, dstAttr: null, attr: null). Red arrows point from the edges in the graph to the corresponding entries in the RDD table.

srcAttr	dstAttr	attr
Alice	Bob	coworker
Bob	Charlie	friend
Charlie		

Cloud Computing and Distributed Systems Introduction to Spark

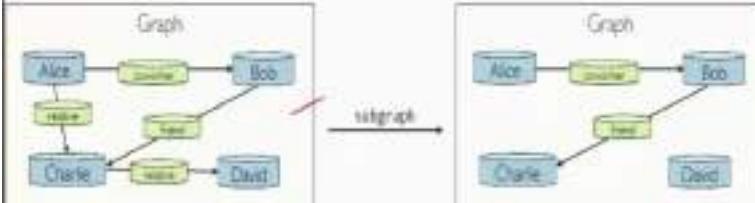
Now, as far as the triplet views are concerned, they can be using triplets the can be modeled in the form of RDDs. For example, the Alice in it is RDD the source and destination that is this is Alice and Bob and their attribute is the coworker; that means they have the relationship which is of type coworker.

Similarly, bob and Charlie there are 2 different nodes in the graph connected with an edge which is having defined this particular relation friend. So, this way the triplets can be defined in the form of RDDs.

(Refer Slide Time: 45:34)

The subgraph transformation

```
class Graph[VD, ED] {  
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,  
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]  
    }  
  
graph.subgraph(epred = (edge) => edge.attr &gt;= "relative")
```



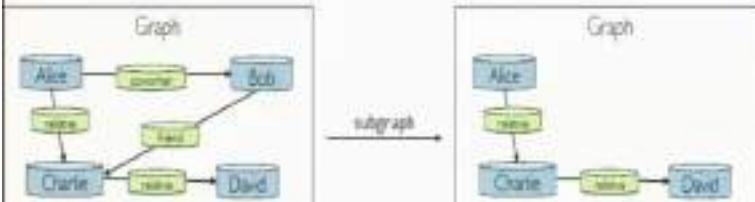
Cloud Computing and Distributed Systems Introduction to Spark

And then once the triplets are defined and various graphs sub graph transformations are also supported in the form of a GraphX.

(Refer Slide Time: 45:54)

The subgraph transformation

```
class Graph[VD, ED] {  
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,  
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]  
    }  
  
graph.subgraph(vpred = (id, name) => name != "Bob")
```



Cloud Computing and Distributed Systems Introduction to Spark

So, for example, out of this complete graph a subgraph can be partitioned and being operated on that part also using subgraph transformations.

(Refer Slide Time: 45:58)

Computation with aggregateMessages

```
class Graph[VD, ED] {  
    def aggregateMessages[A](  
        sendMsg: EdgeContext[VD, ED, A] => Unit,  
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]  
    )  
  
    class EdgeContext[VD, ED, A](  
        val srcId: VertexId, val dstId: VertexId, val attr: ED,  
        val srcAttr: VD, val dstAttr: VD)  
        def sendToSrc(msg: A)  
        def sendToDst(msg: A)  
    )  
  
    graph.aggregateMessages(  
        ctx => {  
            ctx.sendToSrc(1)  
            ctx.sendToDst(1)  
        },  
        _ + _  
    )
```

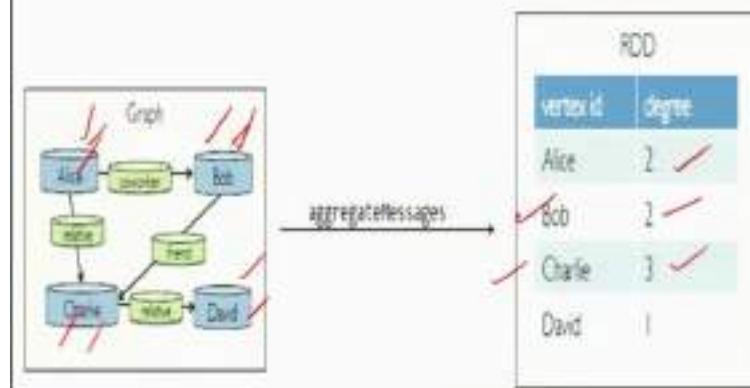
Cloud Computing and Distributed Systems

Introduction to Spark

Similarly, the computation with aggregated messages are also supported, where in the triplets can communicate using send message, send to the source send to the destination.

(Refer Slide Time: 46:22)

Computation with aggregateMessages



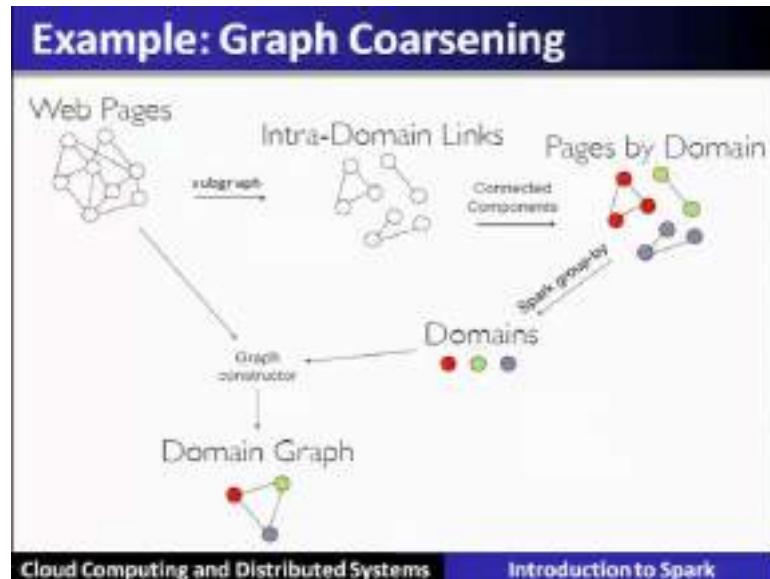
Cloud Computing and Distributed Systems

Introduction to Spark

For example, here see the computation with the aggregated messages. So, let us see this particular Alice has the degree 2. In 2 degree means it is having the relationships with Charlie and Bob in the triplets similarly, as far as the Bob is concerned it is also having the relations with Alice and Charlie and Charlie has 3 relations with Alice Bob and David, David has only one relation with Charlie.

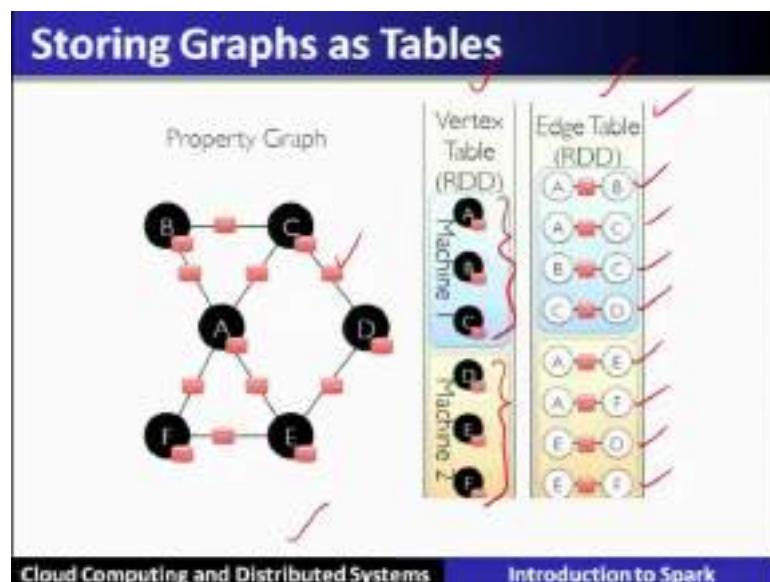
So, these aggregated messages can also be used in the computation, another example of graph coarsening.

(Refer Slide Time: 47:16)



Hear the different type of graphs and they can be operated using different graph coarsening.

(Refer Slide Time: 47:25)

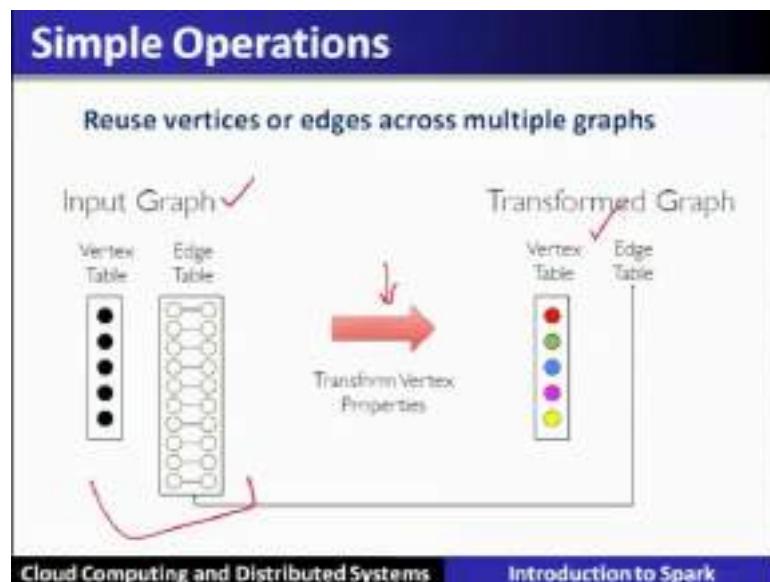


So, let us see how the GraphX works. So, the graph is now stored as a table in GraphX. For example, the example is given as the property graph it will be stored as 2 tables the vertex table and the edge table. So, here the list of vertices which are stored in one

machine and further list of 3 vertices are stored another machine. So; that means, these particular vertices are also partitioned across different machine so, that a throughput can be achieved or when a large scale graph is there then it can be computed over a cluster.

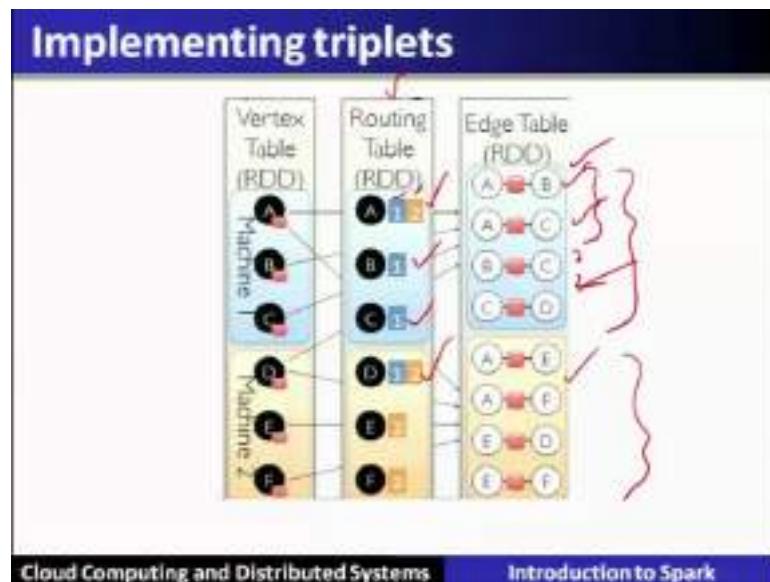
The edges are also stored in another table which is called an edge table they are all RDDs. For example, between A and B this particular triplet is stored A C and B C and C D. They are stored in one machine and A E, A F, E D and E F they are stored another machine. So, we have seen that the graph can be represented in the form of tables vertex table and the edge table they are nothing but RDDs in GraphX.

(Refer Slide Time: 48:57)



Having define these 2 different type of tables this will become an input a graph and then different transformations can be applied on the vertex properties. Similarly, it can be applied later on the graph properties. So, this is shown that if some properties are applied on the vertex table this is the transformed graph. So, there are simple transformations which can be applied.

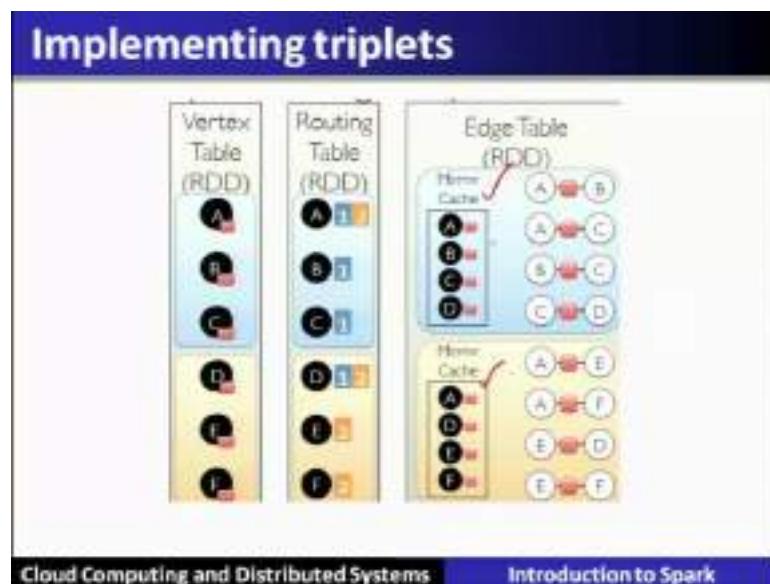
(Refer Slide Time: 49:33)



Now, implementing the triplets to implement the triplets there is one more table involved that is called a routing table that is also an RDD. This will give an information that let us say the vertex A is having the 2 triplets A B and A C. So, A B and A C they are lying in 2 different segments. For example, this particular a is appearing here and also as far as A B is concerned, B is appearing only once, C is also appearing in only one of these clusters machines.

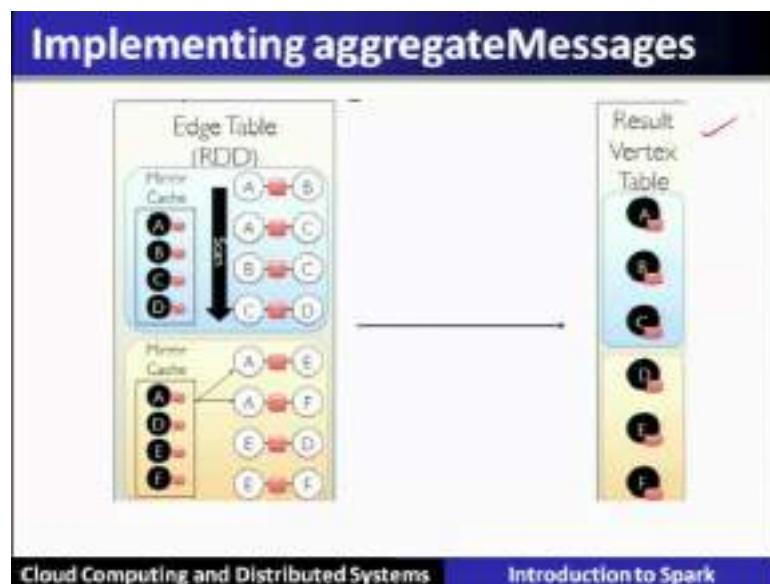
So, all that routing table will give the information that this triplets are either fully stored in one cluster or in many more than one cluster and this particular information is stored in the routing table.

(Refer Slide Time: 50:57)



To implement further the triplets, the edge table will mirror this particular cache of this vertex table RDDs. Similarly, here also it will mirror the cache over here and this will be used in working with the triplets.

(Refer Slide Time: 51:25)



Similarly, the aggregate messages can be easily computed by a scan operations. So, it will be having a mirror cache and the edge table which will scan all these things and it will give the result vertex in another form of table.

(Refer Slide Time: 52:01)

Future of GraphX

- 1. Language support**
 - a) Java API
 - b) Python API: collaborating with Intel, SPARK-3789
- 2. More algorithms**
 - a) LDA (topic modeling)
 - b) Correlation clustering
- 3. Research**
 - a) Local graphs
 - b) Streaming/time-varying graphs
 - c) Graph database-like queries

Cloud Computing and Distributed Systems Introduction to Spark

So, having seen the GraphX applications, different operations which supports the graph computations. Many new languages are coming up to support in the form of API's and more new algorithms are, also being built in the GraphX package also lot of research is going on in this particular support of graphs in graphics such as local graph stream time varying graphs graph database which supports the query operations.

(Refer Slide Time: 52:47)

Other Spark Applications

- i. Twitter spam classification
- ii. EM algorithm for traffic prediction
- iii. K-means clustering
- iv. Alternating Least Squares matrix factorization
- v. In-memory OLAP aggregation on Hive data
- vi. SQL on Spark

Cloud Computing and Distributed Systems Introduction to Spark

There are many other applications which basically uses the spark such as twitter spam classification EM algorithm for traffic prediction K means alternative least square in memory OLAP aggregation and SQL on.

(Refer Slide Time: 53:07)

Reading Material

- Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
"Spark: Cluster Computing with Working Sets"
- Matei Zaharia, Mosharaf Chowdhury et al.
"Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"

<https://spark.apache.org/>



Cloud Computing and Distributed Systems Introduction to Spark

So, there are many further reference materials we have included here the spark cluster computing with working sets and resilient distributed datasets a fault tolerant abstraction in memory cluster computing.

(Refer Slide Time: 53:27)

Conclusion

- RDDs (Resilient Distributed Datasets (RDDs) provide a simple and efficient programming model
- Generalized to a broad set of applications
- Leverages coarse-grained nature of parallel algorithms for failure recovery

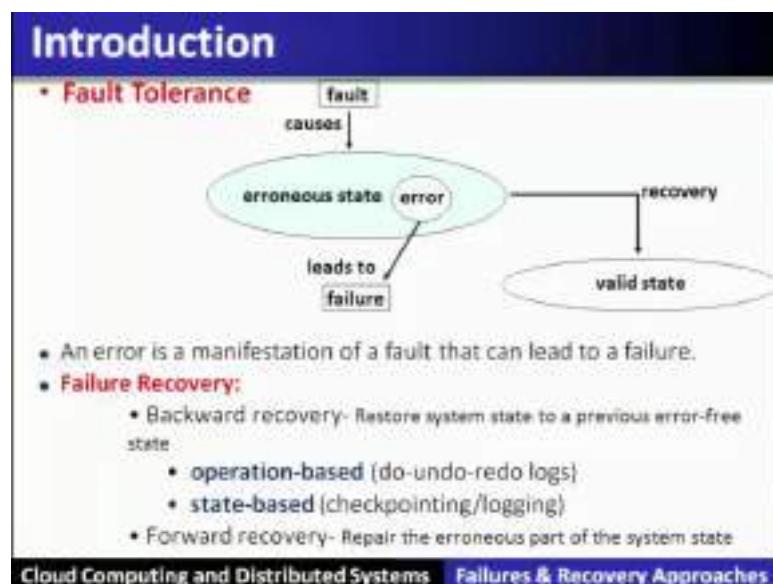
Cloud Computing and Distributed Systems Introduction to Spark

So, we have seen how the spark has overcome from the problems of MapReduce using an RDD that is a resilient distributed datasets, which provide simple and efficient programming model is, spark is 100 times faster than MapReduce operations. Because most of the operations across the iterations it performs in memory without using the distributed file system or basically overcome using the IO's.

We have also seen that this is spark core is used in building various libraries such as for machine learning, it is MLlib for graph processing it is GraphX and so on. We have seen that this is spark leverages coarse grained nature of parallel algorithms for failure recovery. So, failure recovery is not that having an overhead and easily implementing it.

Thank you. Thank you erroneous states is a state where the errors are basically reaching to that state due to the error. So, error is a manifestation of a fault that will lead to a failure, here we will see that in that particular state which is called erroneous state the recovery is required to reach to a valid state, which is an error free state.

(Refer Slide Time: 55:26)



So, the fault tolerance is achieved using recovery of a failure using 2 methods the first one is called forward recovery method where the repair of the erroneous part of a system state is known and it will be carried out to reach or to recover the system from failed state to the varied state that is called forward recovery method. There is another failure recovery method, which is called a backward recovery method. It will restore the system state to a previous error free state. It does not require the knowledge of the erroneous part

of the system state. The backward recovery is of 2 types, that is operation based backward recovery method in which all the coarse grained operational like do undo redo they are logged in a file and they are being operated on. That is why it is called operation based the second type of backward recovery is called is state based here the states are stored or a logged they are also called checkpointing.

So, when a failure happens using this particular checkpoint or a log based logging method it will try to recover to a restore the previous valid state. Interaction with outside world the distributed system of an interacts with the outside world to receive the input data or deliver the outcome of the computation.

(Refer Slide Time: 57:47)

Issues in failure recovery

- The rollback of process P_i to checkpoint $C_{i,1}$ created an orphan message H
- Orphan message H** is created due to the roll back of process P_j to checkpoint $C_{j,1}$.
- Messages C, D, E, and F are potentially problematic

Cloud Computing and Distributed Systems Failures & Recovery Approaches

Issues in failure recovery; the rollback of a process P_i to a check point $C_{i,1}$ created an orphan message H. Orphan message is created due to the rollback of a process P_j to a previous to a checkpoint that is $C_{j,1}$. So, the messages here we are seeing in the example C,D,E,F they are basically the problematic in that sense that they have become the orphan message and it requires a rollback to undo that affect.

(Refer Slide Time: 58:33)

Problem of Livelock

- **Livelock:** case where a single failure can cause an infinite number of rollbacks.
- **The Livelock problem** may arise when a process rolls back to its checkpoint after a failure and requests all the other affected processes also to rollback.
- In such a situation if the rollback mechanism has no synchronization, it may lead to the livelock problem.

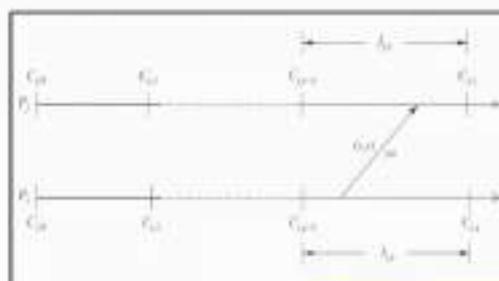
Cloud Computing and Distributed Systems Failures & Recovery Approaches

Problem of livelock; the livelock case where the single failure can cause an infinite number of rollback is called a livelock. The livelock problem may arise when a process roll back to its checkpoint after the failure and request all other affected processes also to rollback. In such a situation if rollback mechanism has no synchronization, which will lead to infinite number of rollback this may lead to the livelock problem.

(Refer Slide Time: 59:08)

Example: Direct dependency tracking technique

- Assume each process P_i starts its execution with an initial checkpoint $C_{i,0}$
- $I_{i,x}$: checkpoint interval, interval between $C_{i,x-1}$ and $C_{i,x}$
- When P_j receives a message m during $I_{j,y}$, it records the dependency from $I_{i,x}$ to $I_{j,y}$, which is later saved onto stable storage when P_j takes $C_{j,y}$



Cloud Computing and Distributed Systems Failures & Recovery Approaches

Different rollback recovery schemes; example of a direct dependency tracking technique. Here we assume that each process P_i starts the execution where the initial checkpoint $C_{i,0}$. Then we will compute the check point interval $I_{i,x}$.

That is nothing but an interval between 2 checkpoints that is $C_{i,x-1}$ and $C_{i,x}$. So, when P_j process receives a message and during that the interval that is j , in j process the interval is y . It records the dependency from the interval of a process i except interval 2 the interval of j or y th interval and which is later stored into the stable storage when P_j takes it as checkpoint $C_{j,y}$. So, along with the check points these intervals which are called dependency tracking is also stored.

Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 21
Introduction to Kafka

(Refer Slide Time: 00:15)

Preface

Content of this Lecture:

In this lecture we will discuss:

- What is Kafka
- What are the use cases for Kafka
- What is Kafka data model
- Kafka architecture
- Types of messaging systems in Kafka
- Importance of brokers in Kafka

Cloud Computing and Distributed Systems Introduction to Kafka

Introduction to Kafka. Preface content of this lecture. We will cover Apache Kafka, its use cases; the data model of Kafka architecture, the type of messaging systems, importance of brokers in Kafka.

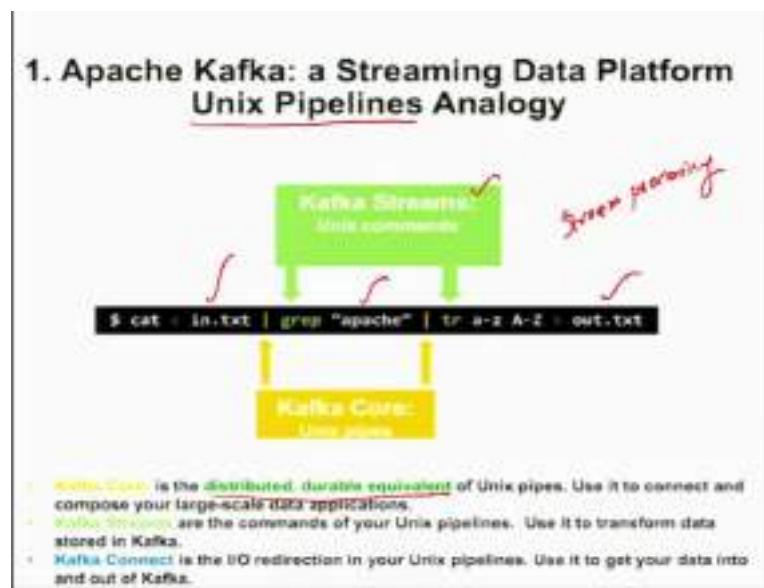
(Refer Slide Time: 00:39)



Let us understand what Kafka is used for so far. We have seen the databases. So, we can turn this particular data is stored as a batch processing application. Why because, data is available in the storage. Hence it is just like the water which is stored in a bottle and then it will be processed the data.

On the other hand, if the data is flowing and you want to tap the water out of that stream, then it is called a streaming. So, how are we processing the streaming kind of data? So, streaming in the sense data is flowing for example, in the network the data is flowing one; that means, in the internet lot of data is flowing all the time. In the real time if we analyze capture the data that is called time series data, and to process that data and store in the database or update in the database is some of the applications of streaming applications. So streaming is becoming a very powerful computing paradigm.

(Refer Slide Time: 02:20)



So, streaming is not new. So, streaming data platform was already there in the Unix pipeline for example, when we say out of a file data when we say grep Apache; that means, out of that particular file stream it will capture all those text. That means, which contains this particular word Apache, and it will be output. So, this is a kind of stream which was provided in the Unix command, but you know that Unix servers were running on one machine as a process.

So, it was limited in the applications in the current scenario, in the current scenario the data is huge. So, this kind of streaming is required to be redefined, and Apache Kafka exactly does that that is called Kafka stream or a stream processing.

So, Kafka is a distributed equivalent of Unix pipes, which is used to connect and compose the large scale data applications that we will see in this part of the discussion.

(Refer Slide Time: 03:53)

Introduction: Apache Kafka

- Kafka is a high-performance, real-time messaging system. It is an open source tool and is a part of Apache projects.
- The characteristics of Kafka are:
 1. It is a distributed and partitioned messaging system. ✓
 2. It is highly fault-tolerant ✓
 3. It is highly scalable. ✓
 4. It can process and send millions of messages per second to several receivers.

Cloud Computing and Distributed Systems Introduction to Kafka

Apache Kafka introduction; so, Kafka is the high performance real time messaging system. In fact, it is a distributed messaging system. So, this particular Kafka is an open source and a part of Apache projects. The characteristics of the Kafka are it is a distributed partition messaging system. We will understand these terms in more details.

It is fault tolerant, highly scalable. It can process and send million of messages per second to several receivers. So, that scale this particular platform is going to support.

(Refer Slide Time: 04:49)

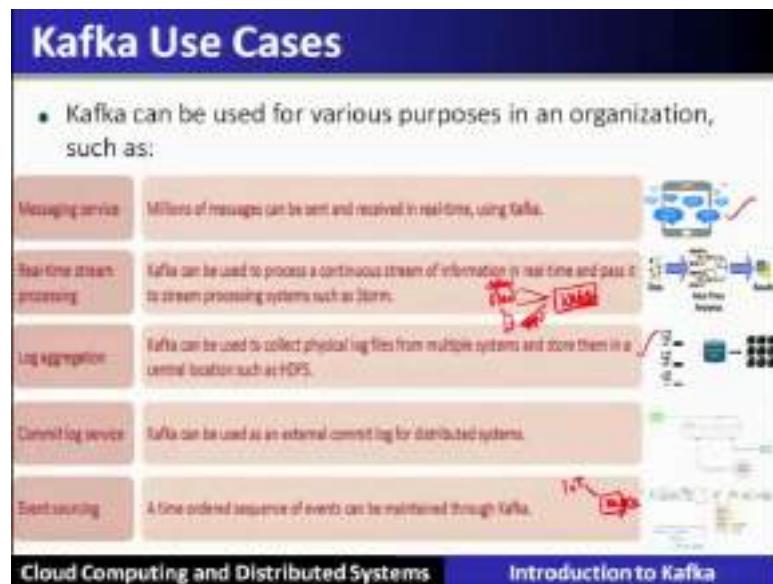
Kafka History

- Apache Kafka was originally developed by LinkedIn and later, handed over to the open source community in early 2011.
- It became a main Apache project in October, 2012. ✓
- A stable Apache Kafka version 0.8.2.0 was released in Feb, 2015.
- A stable Apache Kafka version 0.8.2.1 was released in May, 2015, which is the latest version.

Cloud Computing and Distributed Systems Introduction to Kafka

Let us trace through the brief history of Apache Kafka. So, Apache Kafka was originally developed by LinkedIn. And later handed over to the open source community in 2011. It became Apache project in October 2012. The stable version that is, 0.8.2.0 was released in February 2015. Stable version that is 0.8.2.0 version was released in May 2015. And this becomes continuous in the updates.

(Refer Slide Time: 05:40)



So, Kafka use cases, Kafka can be used for various purposes in any organizations such as it is messaging system billions of messages can be sent and received in the real time using Kafka. Messaging system we have already used in our mobile phones. Similarly, it is also used for real time stream processing. For example, your data, your website is being accessed through different means. For example, mobile apps can also use it through internet client can also use it. And also there are many other route from database it can also access.

So, there are many ways your website is accessing and you want to keep track of them in the real time. Hence, this will become a real time stream processing application. And Kafka can be used to process a continuous stream of information in the real time and pass it to the stream processing system such as storm. Another application is called log aggregation which often is used in any organization. So, Kafka can be used to collect the physical log files from multiple systems, and store them in a central location such as HDFS.

So, for example, there are different proxy servers. And these proxy servers are generating the physical log files. So, from multiple such systems a log can be physically collected. And is being aggregated and stored in HDFS in a real time. Similarly, commit log service. Kafka can be used as an external commit log for distributed system. Event sourcing that is time ordered sequence of events can be maintained through the Kafka that we will see the event sourcing. One such example is an IoT based events which are happening that can be captured using Kafka. And they can be tapped for maintaining the sequence of events into HDFS.

(Refer Slide Time: 08:34)

Apache Kafka: a Streaming Data Platform

Most of **what a business does** can be thought as **event streams**. They are in a

- Retail system: orders, shipments, returns, ...
- Financial system: stock ticks, orders, ...
- Web site: page views, clicks, searches, ...
- IoT: sensor readings, ...
and so on.

Cloud Computing and Distributed Systems Introduction to Kafka

So, most of the businesses can be thought of as event streams processing. For example, the businesses means it is handling the retail system that is various orders shipments returns financial systems, such as stock ticks orders, website uses such as page views clicks searches and monitoring through IoT that is called sensor reading and so on. This all kind of events are happening as the event stream in any business organization. To capture all of it, and get an aggregate view for any organization or for any customer or a user requires this kind of processing called streaming data application.

So, Kafka is the tool for doing this in an present scenario, the businesses are using this kind of event stream processing and Kafka will handle it.

(Refer Slide Time: 10:02)

Enter Kafka

- Adopted at 1000s of companies worldwide

Cloud Computing and Distributed Systems Introduction to Kafka

Therefore, the Kafka is adopted more than thousands of companies worldwide some of the companies which are listed here are already using the Kafka as stream processing Netflix, EBay, airbnb then PayPal, Yahoo, Wikipedia, Salesforce and so on.

(Refer Slide Time: 10:28)

Aggregating User Activity Using Kafka-Example

- Kafka can be used to aggregate user activity data such as clicks, navigation, and searches from different websites of an organization; such user activities can be sent to a real-time monitoring system and hadoop system for offline processing.

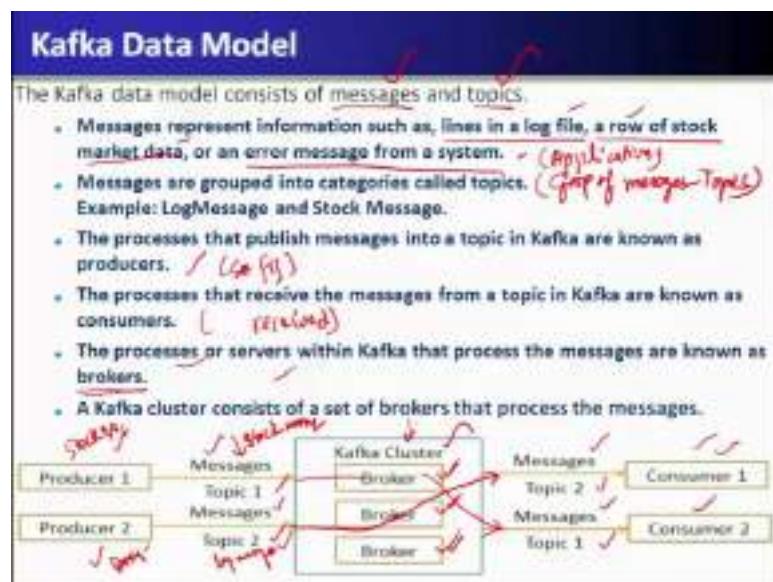
Cloud Computing and Distributed Systems Introduction to Kafka

So, let us see how this aggregation of a particular user activity can be done in Kafka. Let us go through this simple example. So, Kafka can be used to aggregate the user activity data such as clicks navigation, searches from different websites of an organization. Such user activities can be sent to the real time monitoring system, and Hadoop for online

offline processing. So, here let us see that we have a customer portal 1, 2 and 3 who is accessing the upfront the websites and doing navigation having a clicks and search the websites.

All this particular events or activity data will be captured and will be sent to the Kafka Cluster. Kafka Cluster will then process these different streams and capture different type of topics for the real time monitoring, and also out of that some data will be stored in the databases through the Hadoop offline processing.

(Refer Slide Time: 12:06)

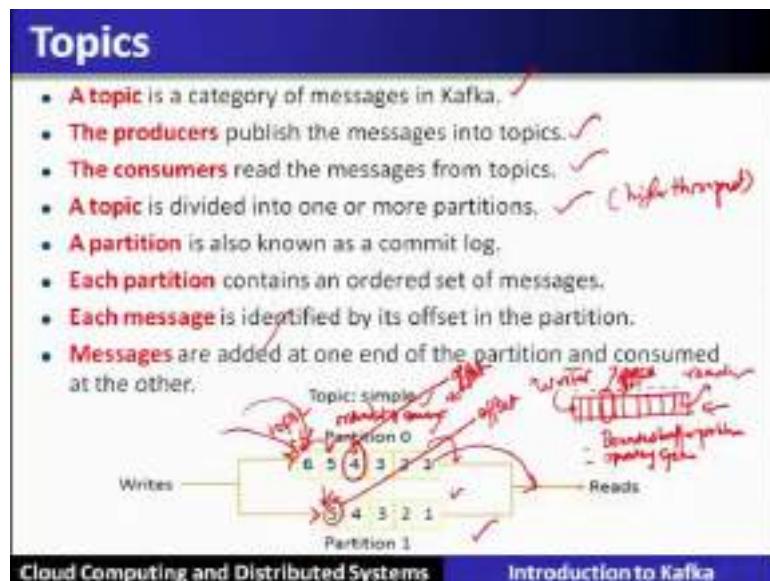


So, to do this let us go in more detail and see, what are the data models which are supported in the Kafka. So, the Kafka data model consists of the messages and the topics. So, messages represent the information's such as lines in a log file; a row in a row of a stock market data or an error message which is generated out of that particular system. So, these messages are generating or this is being generated out of different applications. And hence this different kind of information which we are seeing over here is dependent upon different applications.

But in Kafka it is all called as the messages as an abstraction. These messages are grouped into the categories which are called the topics. So, messages and topics we have just seen, the example is that the log message and the stock message is the example of kind of topics we will see in more detail about this. The processes that publish the messages into the topic in Kafka is known as the producer. Here in the figure below. You

can see there are 2 producers. They will produce the messages and also will be classified under the topic 1, another message which is produced by another producer which is classified as the topic number 2.

(Refer Slide Time: 14:37)



So, the producer could be from by stock exchange, then the message will be the stock messages, and if this is the proxy server of some server or some data center, then here it will generate the log message. So, the processes that receive the message from the topic in the Kafka are known as the consumer. So, these messages will be received or also called as a consumed, they are called consumers who will consume it. So, these terms are used here in Kafka description. The processes or the servers within the Kafka that processes the message are known as Brokers. Here we are seeing here 3 different processes which process these messages and give it to the different customers who are interested in such topics.

For example, topic number 2 is of interest to the customer one. So, it will be sent to the customer number one after the processing through the Broker. So, the Broker will take the message from the producer of a particular topic process it and gives it to the consumer. Similarly, the message one will be handled by let us say a particular Broker and give it to the consumer number 2 who is interested in the message one. So, the processes or the servers within the Kafka that process the messages are known as the

Brokers. So, these Brokers are the processes and they run on the Cluster which is called a Kafka Cluster. So, Kafka Cluster consists of a set of Brokers that process the message.

Let us go in more detail of this data type which is called a topic. So, topic is a category of message in a Kafka. So, the producers publish the messages into the topics and consumer read the message of a particular topic in which he is of interest. So, topic is divided into one or more partitions, why for high throughput. Now partition is also known as commit log each partition contains an ordered set of messages. Each message is identified by its offset in the partition. Messages are added at one end of a partition and consumed at the other end of a partition.

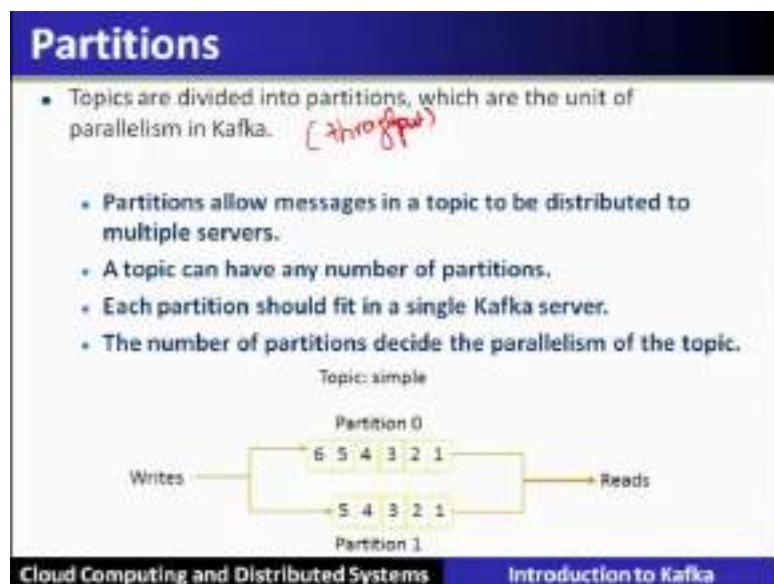
Let us understand this through a simple example. Before going into this example let us understand the concept of a reader and writer problem in a bounded buffer problem. That is called a bounded buffer problem. So, this particular problem you might have studied in operating system courses which are now generalized here in this particular form of messaging system. So, the writer will keep on writing in this particular form of a queue and reader will read in this particular queue and all the messages will be strictly in that particular order in which the writer writes.

Now, if writer is writing and reader is not reading, then eventually this particular bounded buffer will be full after sometime. And once the reader starts reading it, then it will be read. These particular elements are called offset. Now this particular single buffer is now generalized here in the Kafka messaging system which is maintained in the distributed system not in a single server. We will see this particular generalization and understand all the topics. What you mean by the topics?

So, a topic is now partitioned. So, messages are categorized into the partition which is holding the messages of a particular topic. So, this particular let us say that this is. So, this particular topic is divided into 2 different partitions, and the writer can write down for a particular message and according to the topic it will be classified in one of these particular partitions it will be written at the end. Similarly, the consumer will consume from the other end of the partition. So, each partition contains an ordered set of messages, ordered in the sense as they arrive ordering is defined here in the partition and it is strictly ordered.

So, each message in the partition will be identified by an offset. Here the offset is offset number 4 of partition 0. And here this is the offset number 5 of the partition 1. So, if a new message will come to a partition 0. So, it will be finally, consumed here, and it will go to the offset 4 after the consumer consumes 2 more messages. So, this particular offset will know at which point of time which messages are available in to the partition. Again the messages are added at one end; that means, the writer will write down at this end, and the reader will read from the other end that is shown here in the picture; that is very simple let us go and understand about the partition.

(Refer Slide Time: 21:56)



So, the topics are divided into the partition which is the unit of parallelism in Kafka. So, parallelism is for enhancing the throughput by exploiting the parallelism. So, partition allows the messages in a particular topic to be distributed to the multiple servers. So, a topic can have any number of partitions, and each partition should fit in a single Kafka server. So, if the size increases, then you can have another partition and can be stored in another server. So, the number of servers can be scaled in a horizontal scaling to support many partitions here in Kafka. So, the number of partition decides the parallelism of the topic that we have already explained the partition distribution.

(Refer Slide Time: 23:02)

Partition Distribution

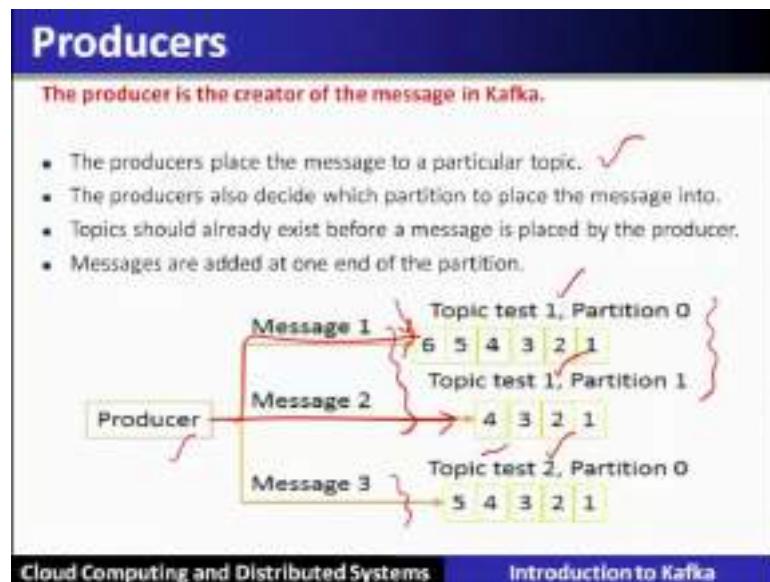
- Partitions can be distributed across the **Kafka cluster**.
- Each Kafka server may handle one or more partitions.
- A partition can be replicated across several servers for fault-tolerance.
- One server is marked as a **leader for the partition** and the others are marked as followers.
- The leader controls the read and write for the partition, whereas, the followers replicate the data.
- If a leader fails, one of the followers automatically become the leader.
- Zookeeper is used for the **leader selection**.

Cloud Computing and Distributed Systems Introduction to Kafka

So, partitions can be distributed across the Kafka servers. Each server may handle one or more partition here we can see in the example that partition number 0 is stored in the server 1 and it has another server 2; which stores a different partition which is called partition 1. It is also possible that 2 different partitions can be stored in one server, or they can be separately stored. So, a partition can be replicated across several servers to for the fault tolerance. So, these petitions are replicated also if Hadoop is used then replication is by default 3.

If it is having more than one replicas, then one server is marked as the leader for the partition and others are the follower. And this leader election will be done through the Zookeeper. This particular leader controls the read and write for the partition where as the replicas will replicate the data. Hence a strict consistency is maintained why because the leader will control the rights also. So, if the leader fails one of the follower can automatically becomes the leader through the leader election.

(Refer Slide Time: 25:03)



Let us understand about the producers. So, producer is the creator of the message in the Kafka. So, producer places the message to a particular topic. So, here we have shown the example of 3 different topic, topic which is called test one and another topic which is called test 2. So, here the producer produces 3 different types of messages, which are classified as the topic test one and another message which is classified as a topic test 2. So, therefore, the producers place the messages to a particular topic the producers also decide which partition to place the message into it.

For example, message one and message 2 belongs to topic test one, but topic test one is maintained in 2 different partitions. So, the producer has to decide that this message will be placed in the partition 1. And message one will be maintained in a partition 0. Topic should already exist before the message is placed by the producer. So, messages are added at one end of the partition that we have already seen. So, the producer will write into the partition at one end of the partition.

(Refer Slide Time: 26:52)

Consumers

The consumer is the receiver of the message in Kafka.

- Each consumer belongs to a consumer group.
- A consumer group may have one or more consumers.
- The consumers specify what topics they want to listen to.
- A message is sent to all the consumers in a consumer group.
- The consumer groups are used to control the messaging system.

Cloud Computing and Distributed Systems Introduction to Kafka

Now, coming to the consumer, consumer is the receiver of the message in the Kafka. So, each consumer belongs to the consumer group. Here in this example we have shown 3 different consumer groups in the boxes. So, the consumer group may have one or more consumers. For example, consumer 1 has 3 different consumers. Consumer 2 has 2 consumers in his group. And consumer 3 has one consumer in the group the consumers specify what topic they want to listen to. So, the message is sent to all the consumers in a particular consumer group. The consumer group is used to control the messaging system. Let us see through more examples.

(Refer Slide Time: 27:47)

Kafka Architecture

Kafka architecture consists of brokers that take messages from the producers and add to a partition of a topic. Brokers provide the messages to the consumers from the partitions.

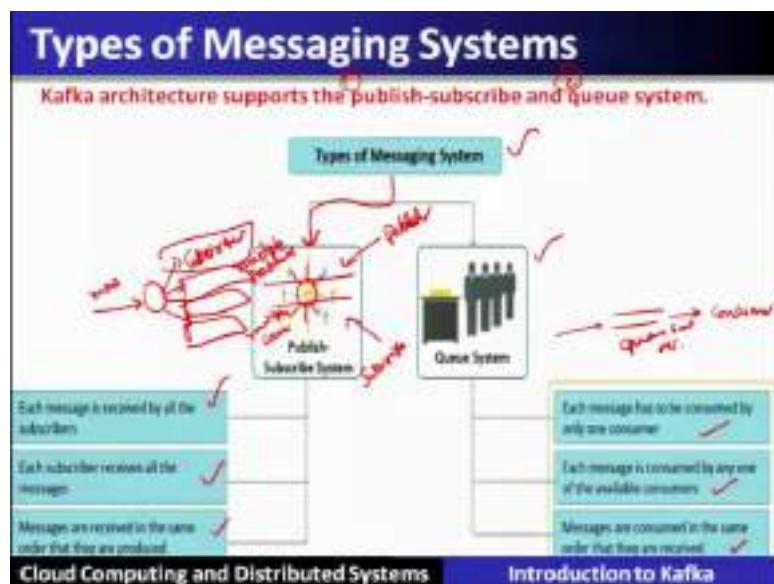
- A topic is divided into multiple partitions.
- The messages are added to the partitions at one end and consumed in the same order.
- Each partition acts as a message queue.
- Consumers are divided into consumer groups.

Cloud Computing and Distributed Systems Introduction to Kafka

So, when a message is come they will be all the messages will be given to the consumer, but these consumers they also have to tell which kind of messages which topic, they are interested in hence once the consumer receives all the messages. It will control to give the topics which are of their interest to them. So, let us understand the Kafka architecture. So, Kafka architecture consist of the Brokers, these are the Brokers. That takes the message from the producer and adds to the partition of a topic. Brokers provide the messages to the consumer from the partition.

A topic is divided into multiple partitions. The messages are added to the partition at one end and consumed from the other end. In the same order each partition acts as a message queue, consumers are divided into the consumer groups.

(Refer Slide Time: 29:29)



So, let us understand the other component in the architecture. So, we have seen that the producer is producing the messages which through the Broker will be stored in these partitions. The Broker is the processes which are running in the Cluster which is called a Kafka Cluster.

Again these partitions also will be running either on the same server or multiple server and they are replicated also. Therefore, several machines are required to run this Kafka Cluster. And this all will be coordinated by the Zookeeper. So, therefore, Zookeeper will ensure the fault tolerance in the Kafka Cluster. The Kafka architecture supports 2 types

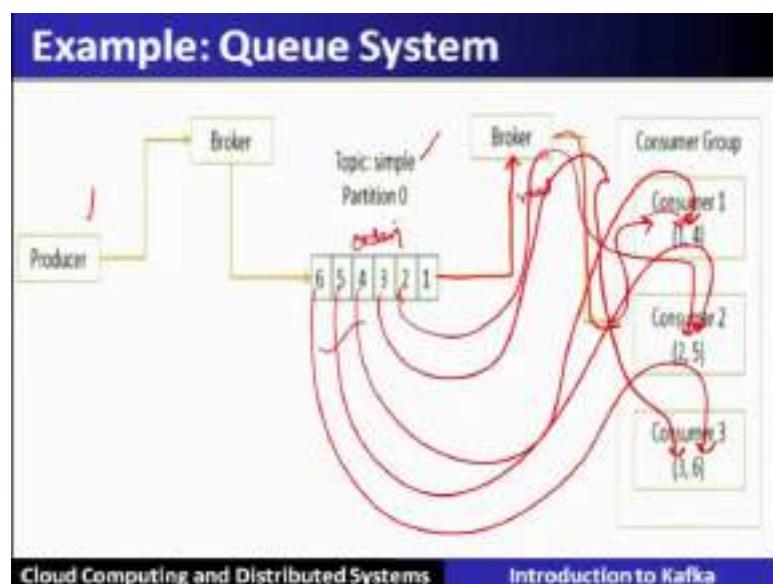
of messaging system. The first one is called publish subscribe messaging system. The other is called queue based messaging system.

So, there are 2 types the first type which is called publish subscribe messaging system. In this particular model, there is a producer which will produce there are several producers, which will be producing the messages. And these messages will be consumed by many consumers; so, multiple producer and multiple consumers. So, they are called publishers and the other system is called subscribe publish subscribe messaging system.

So, here each message in this system is received by all the subscribers. And each subscriber receives all the messages. And messages are received in that particular order that they are being produced. So, therefore, different subscriber will receive all the messages and they will do their applications as per the requirement.

On the other hand, there is another system which is called a queue system. Here the messages each message will be consumed by only one customer or by one consumer. So, each message is consumed by any one of the available consumers. And messages are consumed in the order they are received. So, that becomes a queue based messaging system. Let us see these systems in more details.

(Refer Slide Time: 33:42)



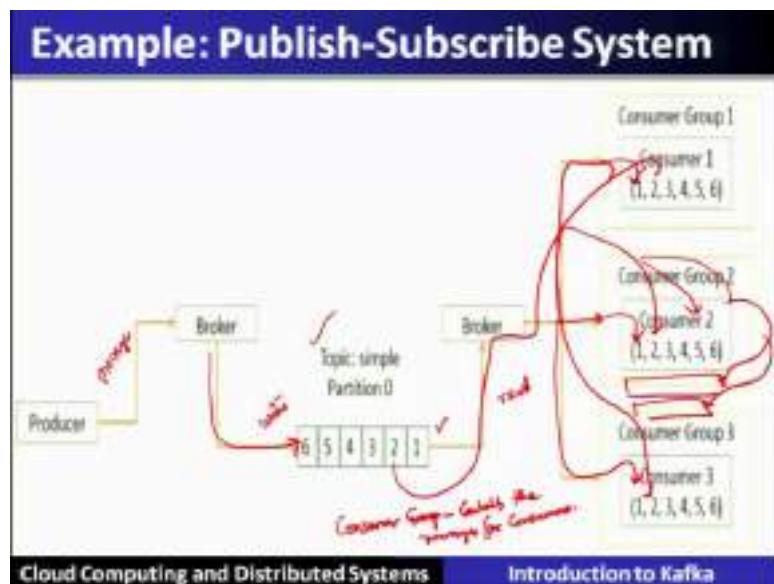
So, here the producer is producing the messages, and using the Broker the message will join in that particular partition of a particular topic let us say some topic. This particular

partition using Broker will read will consume the message that will it will read it, from the other end of the queue and it will give to exactly one consumer. So, here it is given to consumer number 1. It will not be given to all other consumer in a queue based messaging system. Similarly, the message number 2, through the Broker will be read and it will be given to the consumer number 2.

The message number 3 will be read through the Broker and it will be given to the consumer number 3 in this particular sequence. The message number 4 will be given to the consumer 1, message number 5 will be given to the consumer 2 and message number 6 will be given to the consumer number 3.

So, the ordering is strictly maintained and this is the method which is used here in the queue system.

(Refer Slide Time: 35:18)



On the other hand, in publish subscribe system let us see through an example. The producer is producing the messages which through the Broker will be written to the partition of a particular topic at one end of the partition. The message will be read from the other end of a partition, and through the Broker it will be given to all the consumers.

So, here you see that all the consumer will have the message number 1. Similarly, the message 2 will go through the through the Broker and will reach to all the consumers. Now if a particular consumer group has more than one consumer so, on arriving all these

messages at the consumer group this consumer group will decide which message to go to which of these consumers. So, consumer group here controls the messages for the consumers. In case every consumer group has only one consumer. So, it will be given all the messages to that consumer if more than one consumer is there in a particular group. So, it will be controlled through the consumer group and the messages will be delivered between them.

(Refer Slide Time: 37:17)

Brokers

Brokers are the Kafka processes that process the messages in Kafka.

- Each machine in the cluster can run one broker.
- They coordinate among each other using Zookeeper.
- One broker acts as a leader for a partition and handles the delivery and persistence, whereas, the others act as followers.

✓

Cloud Computing and Distributed Systems Introduction to Kafka

Now, comes the Broker. So, Brokers are the Kafka processes that process the messages in the Kafka messaging system. Each machine in the Cluster can run one Broker and they coordinate among each other using the Zookeeper. One Broker will act as the leader for the partition and handles the delivery and persistence; whereas the others act as the followers. Therefore, these Brokers are replicated and one of them is the leader which controls all the read and write to the partitions.

(Refer Slide Time: 38:13)

Kafka Guarantees

Kafka guarantees the following:

1. Messages sent by a producer to a topic and a partition are appended in the same order
2. A consumer instance gets the messages in the same order as they are produced.
3. A topic with replication factor N, tolerates upto N-1 server failures.

Cloud Computing and Distributed Systems Introduction to Kafka

Therefore, Kafka guarantees the following. First one is that the messages sent by the producer to a particular topic. And the partitions are appended in that particular order in which the messages are being sent by the producer. So, the order is maintained at any point of time, and that is guaranteed by the Kafka. Second guarantee which Kafka ensures is that the consumer instance gets the message in the same order they are being produced. So, a strictly order is maintained in the Kafka. So, a topic with the replication factor N tolerates up to N minus 1 different failure.

Now, let us see the replication in the Kafka.

(Refer Slide Time: 39:09)

Replication in Kafka

Kafka uses the primary-backup method of replication.

- One machine (one replica) is called a leader and is chosen as the primary; the remaining machines (replicas) are chosen as the followers and act as backups.
- The leader propagates the writes to the followers. ✓ *Strict consistency*
- The leader waits until the writes are completed on all the replicas.
- If a replica is down, it is skipped for the write until it comes back. ✓
- If the leader fails, one of the followers will be chosen as the new leader; this mechanism can tolerate $n-1$ failures if the replication factor is ' n '. ✓

Cloud Computing and Distributed Systems

Introduction to Kafka

So, Kafka uses primary backup method of replication; that means, one machine one replica is called the leader and is chosen as the primary the remaining machines are chosen as followers and act as a backup. The leader propagates the writes to the followers therefore, it ensures the strict consistency. The leader waits until the writes are completed on all the replicas if the replica is down it is skipped for write until it comes back. If the leader fails one of the followers will be chosen as the new leader and this mechanism can tolerate up to n minus 1 failure if the replication factor is n .

(Refer Slide Time: 40:00)

Persistence in Kafka

Kafka uses the Linux file system for persistence of messages.

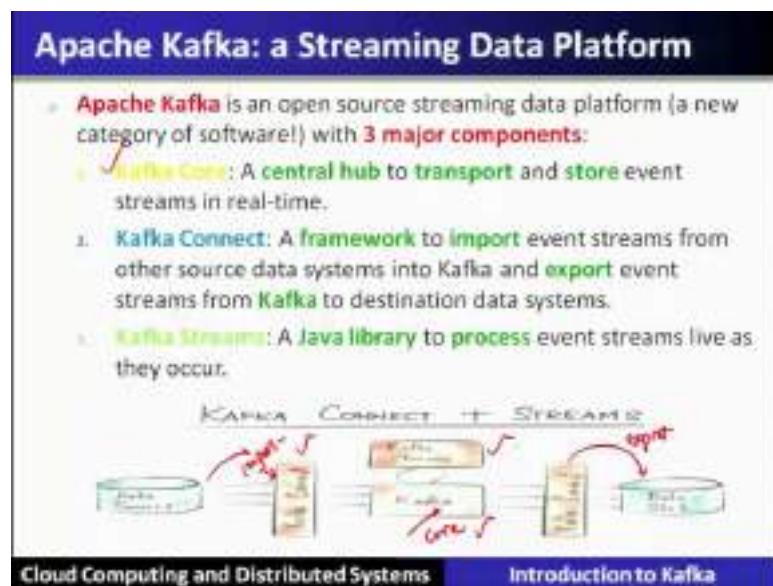
- Persistence ensures no messages are lost.
- Kafka relies on the file system page cache for fast reads and writes.
- All the data is immediately written to a file in file system.
- Messages are grouped as message sets for more efficient writes.
- Message sets can be compressed to reduce network bandwidth.
- A standardized binary message format is used among producers, brokers, and consumers to minimize data modification.

Cloud Computing and Distributed Systems

Introduction to Kafka

Kafka uses the Linux file system for persistence of the messages. So, persistence ensure that no messages are lost Kafka relies on the file system page cache for fast read and write all the data are immediately written to a file system. Messages are grouped as message set for more efficient writes. And messages set can be imposed to reduce the network bandwidth. A standardized binary message format is used among the producers Brokers and consumers to minimize the data modifications.

(Refer Slide Time: 40:38)



In nutshell, Apache Kafka is a streaming data platform; which is an open source streaming data platform with 3 different major components. The first one is called Kafka core. This is the central hub to transport and store the event streams in a real time. That we have already seen.

Then second component is called Kafka connect. This framework is to import the event streams from other sources into the Kafka and export the events from the Kafka to the destination. So, there are 2 different connect. One is for import through the data sources. The other Kafka connect will export to the data sink. Then finally, we have seen the data streams that are called Kafka streams. So, Kafka streams are the java library to process the event streams live as they occur. So, that framework we have discussed here in this lecture.

(Refer Slide Time: 42:02)

Further Learning

- **Kafka Streams code examples**
 - Apache Kafka <https://github.com/apache/kafka/tree/trunk/streams/examples>
 - Confluent <https://github.com/confluentinc/examples/tree/master/kafka-streams>
- **Source Code** <https://github.com/apache/kafka/tree/trunk/streams>
- **Kafka Streams Java docs** <http://docs.confluent.io/current/streams/java/docs/index.html>
- **First book on Kafka Streams (MEAP)**
 - Kafka Streams in Action <https://www.manning.com/books/kafka-streams-in-action>
- **Kafka Streams download**
 - Apache Kafka <https://kafka.apache.org/downloads>
 - Confluent Platform <http://www.confluent.io/download>

Cloud Computing and Distributed Systems Introduction to Kafka

Kafka streams code examples are given in these links and also different resources for further study is mentioned.

(Refer Slide Time: 42:19)

Conclusion

- Kafka is a high-performance, real-time messaging system.
- Kafka can be used as an external commit log for distributed systems.
- Kafka data model consists of messages and topics.
- Kafka architecture consists of brokers that take messages from the producers and add to a partition of a topics.
- Kafka architecture supports two types of messaging system called publish-subscribe and queue system.
- Brokers are the Kafka processes that process the messages in Kafka.

Cloud Computing and Distributed Systems Introduction to Kafka

Conclusion; so, Kafka is high performance real time distributed messaging system. This is highly reliable in the sense that it involves the Zookeeper to maintain the fault tolerance in the system. And also to ensure the horizontal scaling that means, it is highly scalable. We have seen that there are many applications in today's world where a company requires to understand the activity pattern of a particular user; that means, if he

is accessing different websites, what he is doing through the mouse click activities all these are captured through different data sources aggregated it.

And then it will do the real time analysis or it will store in the backup. So, this all is done through the Kafka messaging system. This is a distributed messaging system.

So, Kafka can be used as an external commit log for distributed systems also, this is another application. So, Kafka data model therefore, provides the abstraction of messages and topics. Kafka architecture consists of the Brokers they are nothing but the processors or processes which runs on Kafka Cluster. These processes will read the messages from the producers and maintained it into the partitions which also run on different servers in the Cluster. Kafka architecture also supports the consumers which can take or which can be modeled in 2 different manners one is in the form of queue; that means, exactly one of them can receive the message. The other is called publish subscribe model where in all the consumers will receive in the form of consumer groups, which in turn will divide among themselves according to the topic of interest.

So, publish subscribe model is more general where in different subscribe or a mess or a consumer groups, can be running different kind of applications for example, one such application could be to analyze this real time traffic and then extract the events and stored in the database, maybe (Refer Time: 45:39) database. The other one is it will analyze in the real time and take the action. There may be many different uses of this particular model of consumers.

So, Kafka architecture supports 2 types of messaging system that we have covered publish subscribe and queuing queue system. The Brokers are Kafka processors that process the messages in the Kafka, and they will run on the Cluster which is coordinated through the Zookeeper.

Thank you.



**THIS BOOK IS NOT FOR SALE
NOR COMMERCIAL USE**