

Lesson Preview

Datacenter Technologies

- Brief and high-level overview of challenges and technologies facing data centers
- **Goal:** Provide context for mechanisms from previous lessons

Lesson Preview

Datacenter Technologies

- Multi-tier architectures for Internet services
- Cloud computing
- Cloud and "big data" technologies

1. In this last lesson of the course, I want to provide a brief and high-level overview of some of the technical challenges and technologies that are relevant to today's internet scale applications that are popular. And the underlying data center systems that support their execution. Please note my emphasis on brief and high-level. The space of data center technologies is so vast and so dynamic, that there is no way we can cover it in sufficient technical depth in just one lesson. In fact, there are a number of courses at Georgia Tech that devote a substantial percentage of their content to these topics. Instead what I want to achieve with this lesson, is to make sure I provide you with some context so that you can understand how some of the mechanisms that were discussed earlier in this course, apply to those problems, to those technologies that are in popular use today. Specifically we will describe the typical multi-tier architectures for supporting internet skill services and applications, and their related management challenges. Next we will talk about cloud computing. We will explain what it is and what is the role that it plays in addressing the limitations of some of these traditional approaches. And finally we will briefly comment on some of the key technologies that shape the cloud computing landscape, and that enable us, with scale and flexibility, in how we process large amounts of information.

2. To get us thinking about the importance of the upcoming topics, here are two quick trivia questions. First, how many datacenters are there worldwide in 2015? Second, how much space in square feet is required to house all of the world's datacenters in 2015? These questions are just for fun, so take a couple of guesses.



Datacenter Quiz

How many datacenters are there in the world in 2015?

~ 510,000

datacenters

How much space (in square feet) is required to house all of the world's datacenters (in 2015)?

285.5

million square feet

~~Remember: these questions are for fun Take two guesses!~~

3. Now let's see how you did. Well, to change this number to 2011, for which we have definitive numbers, there are about 510,000 data centers in 2011 that occupied about 285.5 million square feet of space. These numbers were taken from datacenterknowledge.com and were aggregated for 2011, and if you look at the current numbers, you will see estimates that go anywhere from 1 to 1-1/2 billion data centers alone. And of course, that has implications on the million square feet, the estimate that's present currently in the community. Of course, these numbers continue changing all the time.

Internet Services

Internet Service == any type of service provided via web interface

- presentation == static content
- business logic == dynamic content
- database tier == data store

... in multi process configurations →
some form of IPC used, including
RPC / RMI, shared memory ...

- not necessarily separate processes on separate machines
- many available open source and proprietary technologies
- middleware == supporting, integrative or value-added software technologies

proprietary: 專利的

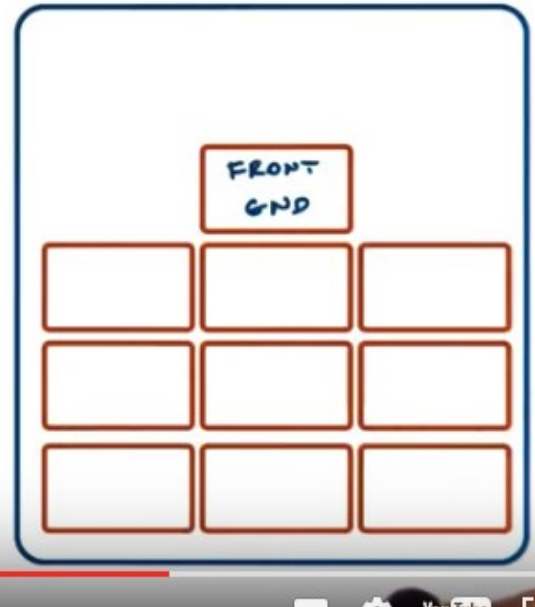
4. In the context of data centers, let's first talk about internet services. An internet service is any type of service that's accessible via web interface. From checking current weather conditions at weather.com to searching or checking email via Google search and mail services, to making online bank transactions or flight reservations. The common way in which these services are used is end users send web requests via web browsers and then receive a response. Most commonly, these types of services are then composed into three components. A presentation component that interfaces with the end users, it's typically responsible for static content related to the web page layout. A business logic component, this is a component that integrates all of the business-specific processing. And this would be all of the dynamic, user-specific content. And then a component that deals with all of the data storage and management, typically in a database. These components are commonly referred to as the presentation, the business and the database tiers. Although these are separate components, they are not always implemented as separate processes running on separate machines. What makes sense will depend on the service complexity, the resource requirements, expected loads. And also in the underlying technologies that are used. For example, the Apache http web server can be used to implement both the presentation logic, to serve static content, and also, as part of the same process, php modules for php processing can be used to generate the dynamic content. There are in fact many open source as well as proprietary solutions that are available that offer technologies for either one of these tiers. To put these applications using these kinds of tiers together, there are various integrated software, or rather middleware, components that are used, that provide commonly used functionality. Messaging services, service configuration and management, security, accounting, some persistence guarantees, recovery management, and many others. One important point to make is that for services that are organized as multiple processes, the inter-process communication between those processes is carried out via some form of IPC, such as RPC or RMI, or from Java based architectures. As well as, some use of optimizations that are based on shared memory, in case that different processes are running from the same machine. So these are some of the examples of the interprocess communication mechanisms that we already discussed in this course. And these are relevant in the context of real world

deployments of Internet services.

Internet Service Architectures

For scale: multi-process, multi-node
=> "scale out" architecture

1. "Boss-worker": front-end distributes requests to nodes
2. "All equal": all nodes execute any possible step in request processing, for any request
3. "Specialized Nodes": nodes execute some specific step(s) in request processing, for some request types



5. For services that need to deal with high or variable request rates, choosing the configuration that involves multiple processes configured on potentially multiple nodes, becomes necessary. This is because the multi-process configurations are the easiest way to deal with scale, to deal with increases in the incoming request rates. What will do in such cases is we'll scale out the service deployment, by launching the same service on multiple machines. And so these are referred to as scale out architectures. A typical architecture could then be organized as follows. A front-end dispatching or load balancing component would route the incoming request to the appropriate or available machine that implements the internet service. This is in some sense similar to the boss-worker pattern that we discussed when we talked about structuring multi-threaded parallel programs. One possibility afterwards is that every single one of the nodes is capable for executing any possible step that's required for the processing of the request. And that they can handle any possible request that may come in from the external users. The other possibility is that the nodes can be specialized to execute only some specific step, or some specific steps in the request processing, or they can be specialized only for certain types of requests. This is again analogous to what we talked about when discussing the boss-worker pattern, that the workers may be general purpose, so they may perform any processing, or they may be specialized for certain tasks.

Internet Service Architectures

For scale: multi-process, multi-node
⇒ "scale out" architecture

1. "Boss-worker": front-end distributes requests to nodes
2. "All equal": all nodes execute any possible step in request processing, for any request
3. "Specialized Nodes": nodes execute some specific step(s) in request processing, for some request types

← functionally homogeneous

← functionally heterogeneous

We can further classify these architectures as those having a property of being functionally homogenous. Like when all of the workers performed, are capable of performing, any processing step. Or, as functionally heterogeneous, like in the case where certain nodes are specialized in certain functions.

Homogeneous Architectures

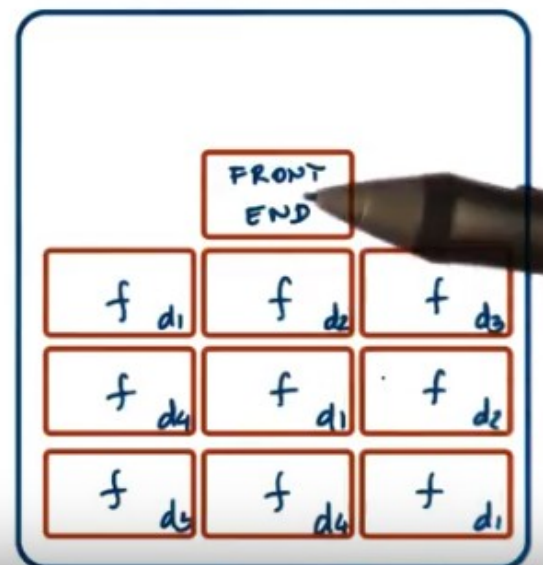
Functionally Homogeneous, each node ...

- can do any processing step

⊕ keeps front-end simple

- doesn't mean that each node has all data; just each node can get to all data

⊖ how to benefit from caching?



6. Functionally homogeneous architectures can be characterized as follows. Any node can process any type of request and can do any of the processing actions that are required in the end to end request processing. Again, this resembles the basic boss worker model. The front end component is the boss, responsible for pushing requests onto the workers. And, all the workers can perform all of the required steps for a request. The benefit of this is that the front-end can be made kept fairly simple. It doesn't have to keep track which one of the nodes can perform which particular action, or can service which types of request. Instead, the front end can simply in a round robin manner, send requests onto the next available node. Or, the front end could be enhanced in some simple way in which it keeps some information regarding the CPU loads, the CPU utilization, on the other nodes in the system. And then, it can use that type of information to direct requests onto the worker nodes. This design doesn't mean that every single one of the nodes has to store all of the data, all of the state that's needed for the internet service. Instead, data may be somehow replicated, or distributed across these nodes. But, with the homogeneous design, every single one of the nodes knows how to get, how to access any type of information that may be necessary for the execution of the service. One downside of this approach is that there is little opportunity to benefit from caching. If the front end is simple, and it just passes requests round robin onto the other nodes. It may not be able, and in fact, it will not be able to explore opportunities like, this particular node, already serviced some state, and therefore it's likely present in its cache. And, it will not need to perform a remote access in order to access that particular information.

Heterogeneous Architectures

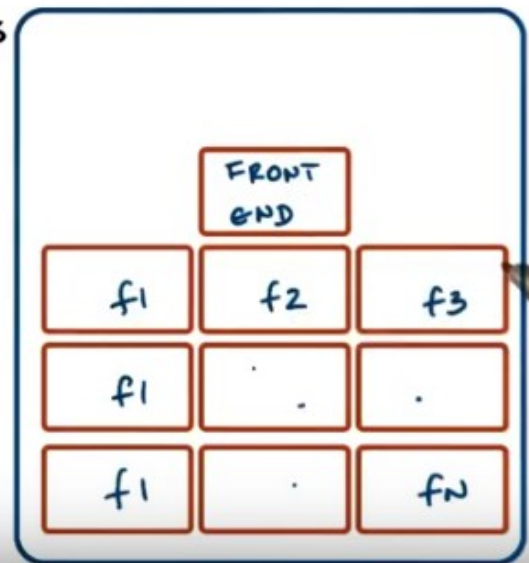
Functionally heterogeneous ...

- different nodes, different tasks/requests
- data doesn't have to be uniformly accessible everywhere

⊕ benefit of locality and caching

⊖ more complex FE

⊖ more complex management



FE: front end

7. The second common architecture for Internet services is referred to as functionally heterogeneous. In these kinds of architectures, different nodes are designated to perform certain functions, or even to see in the particular types of requests. For instance, the request may be divided among the servers based on the request content, such as when the requests are divided based on the alphabetical order of the requested file name. Or for a service such as eBay, servers may be designated to perform browsing

requests versus buying or bidding requests. With this design, it is still possible to have all of the data in some distributed, shared file system and accessible to all of the nodes. But if the functions are somehow designated to be executed only on some subset of the node, then it is possible that not all of the data, not all of the state, will be necessary to be made available across all nodes in the system. So there's some optimization opportunities there. Clearly, one benefit of this approach is that every single node in this system is more specialized for some of the tasks that it performs, or for some of the state that it needs to serve, and therefore, it is more likely that it will take advantage of some benefits like caching and locality. There are also some obvious trade-offs. The first one is that the front end will now need to be more complex because it will need to parse sufficient portion of the incoming request so as to be able to figure out which one of the underlying nodes this particular request should be passed to. Second, the overall management of these systems becomes more complex when nodes fail or when request rates increase, and we need to add more machines, it is important to know what kinds of machines to add. How to configure those machines, what types of requests or tasks are those machines suppose to be designated for. Also even if something changes in the workload pattern and all of a sudden there are more requests of a certain kind, then we have to reconfigure this. We can't just redirect those access of requests to some of the other nodes if they are not capable of processing those kind of actions or serving that kind of data. So that may even be the case if the other nodes are otherwise idling. The result of that is that it is much more easy in these kinds of environments to end up in situations in which some of the nodes are hotspots, and then there are long backlogs of pending requests for those kinds of nodes.

8. Diving into the visual metaphors that we used in this course, I want to ask some questions. How the internet service architecture designs that we just described translate in the context of the toy shop. Consider a toy shop where every worker knows how to build any type of toy, so this is like the homogeneous architecture example. If the rate at which new orders are arriving starts increasing, how will you keep this homogeneous architecture balanced? Note that this quiz is open-ended, and also as a hint, you may want to think about answering the quiz by saying add more of what.



Homogeneous Design Quiz

Consider a toy shop where every worker knows how to build any toy (homogeneous architecture). If higher order rates start arriving you can keep the homogeneous architecture balanced by:

- add more workers (processes)
- add more work benches (servers)
- add more tools, parts... (storage)

bottom line \Rightarrow simple management (still takes time to do this)

Note: This quiz is open-ended and has no single answer; check your answer to see if you are on the right track. Hint: "Add more ..."

9. In this scenario, in order to deal with an increase in the order rates, it would mean that we would need to add more workers, which is analogous to the processes, or add more work benches, analogous to the servers, where those processes need to execute. Or add more tools and parts, so analogous to the storage, the state that's needed for the execution of the service. But we don't have to perform any additional decisions to decide what exactly are those workers intended to do or what specific tools or parts do we need to add. Just a little bit more of everything. The bottom line is that, in this scenario, the management is fairly simple. Still takes some time to do this but we don't have to put a lot of thought into exactly how we need to configure these increased resources.

10. Consider a toy shop where every worker knows how to build some specific toy, so this is analogous to the heterogeneous architecture. In this scenario, if a higher rate of incoming requests for toy orders starts arriving, how would you keep this heterogeneous architecture balanced? Again, this is an open-ended quiz that as a hint, the way you maybe should think about structuring your answers would be add more of x, but what else do you need to do?



Heterogeneous Design Quiz

Consider a toy shop where every worker knows how to build some specific toy (heterogeneous architecture). If higher order rates start arriving you can keep the heterogeneous architecture balanced by:

- profile what kinds of toys are in demand
 - profile what kinds of resources & expertise those require
 - add more of the appropriate type of workers, workbenches and parts
- bottom line => much more complex management

Note: This quiz is open-ended and has no single answer; check your answer to see if you are on the right track. Hint: "Add more ... **but** ..."

11. Now in this scenario again, we'll need to ultimately end up more workers, more work benches, more parts so in a heterogeneous architecture we would need to add more processes, more server node, more storage. However, before we can do that, we first need to profile what kinds of toys are in the end. What kinds of requests are the external users requesting? And then also profile what kinds of resources is required for those particular types of requests for those toys. Only after we perform these profiling tasks will we be able to figure out exactly what kinds of resources need to be added in order to continue maintaining some balance in this heterogeneous design. The bottom line is that the management of these systems is much more complex compared to what's required in a homogeneous architecture. A very popular reference that discusses the design space of these large scale Internet services is Eric Brewer's paper Lessons from Giant Skill Services. This paper also describes some of the trade-offs associated with choices for data replication versus partitioning that we already discussed in the previous two lessons. I have linked the paper in the instructor note.

12. Finally, I want you to consider, again, a toy shop where every worker knows how to build any toy. And in this case, we said that as the rates at which toy orders start coming in increases, the manager keeps scaling up, keeps adding workers and workbenches and parts, etc. My question to you is, until when does this work? As a hint, think about whether endless scaling is possible.



Scale-Out Limitations Quiz

Consider a toy shop where every worker knows how to build any toy (homogeneous architecture). Higher order rates start arriving, so the manager keeps "scaling out": add workers, workbenches, parts... This works until...

- can no longer manage all of the resources
 - can no longer fit more stuff, and staff, in the toy shop
 - cannot find shops to outsource to (only trusts own workers)
- => cloud computing to the rescue!

Note: This quiz is open-ended and has no single answer; check your answer to see if you are on the right track. Hint: Is endless scaling possible?

13. The answer to this is that at some point the manager will simply not be able to control any more resources. There is a limit to the attention that a single person has, and therefore one single manager will not be able to continue controlling all of the different types of resources in the shop. In addition, another limit to the scale maybe that there is a physical limit as to how many different things, how much staff, or how many workers, how much staff can simply fit in the shop. Outsourcing could sometimes be an option, however if the manager only trusts his own workers, then, he will not be able to outsource those tasks, and that's going to compose some limits to the scale. These kinds of limitations exist in the context of internet services as well. Given the size of the data center, there may be a limit on the amount of resources that can be placed there. Given the complexity of the management processes there will be limits as to how many different things can be managed. And if we want to run our own software stacks, our own operating systems, our own software services then we'll clearly be limited by these factors. In order to address these challenges, cloud computing has emerged as a solution that allows us to address some of these limitations regarding the scale that can be supported with existing Internet service solutions.



Cloud Computing & Animoto

Amazon

- provisioned hardware resources for holiday sale season
- resources idle the rest of the year
- "opened" access to its resources via web-based APIs
- third party workloads on Amazon hardware, for a fee

=> Amazon Web Services (AWS)
and Amazon's Elastic Compute Cloud (EC2)

Provision: 準備, 供應

14. Before I talk more about cloud computing, I'd like to illustrate the opportunities that it delivers with an example based on Animoto. This is an example service. And this was a poster child of cloud computing in early years, around 2008, This example that I'll describe next is repeatedly referenced in cloud computing talks, again, especially in those early introductory years. In the mid 2000's, Amazon was already a dominant online retailer servicing large volumes of online sales transactions. Vast majority of these transactions were taking place during the US holiday shopping season between Thanksgiving and Christmas. And to deal with this peak load, Amazon provisioned the hardware resources, so made sure that they've acquired sufficient number of servers for this particular load. What that means is that the rest of the year a lot of these resources were idle. Or they were serviced for other company tasks, for instance, forecasting or other analytics. But regardless, there were still a lot of idle cycles. Now, since Amazon had already developed some software infrastructure to allow these machines to be reprovisioned and used by other services within the company. What they ended up doing in 2006, they opened up those exact same type of API's to the rest of the world. What this did, it allowed third party work load. So not just Amazon work loads, but completely random customers to run their workloads on Amazon's hardware, obviously, for a fee. This was the birth of Amazon's Web Services, or AWS, and Amazon's Elastic Compute Cloud, or EC2.

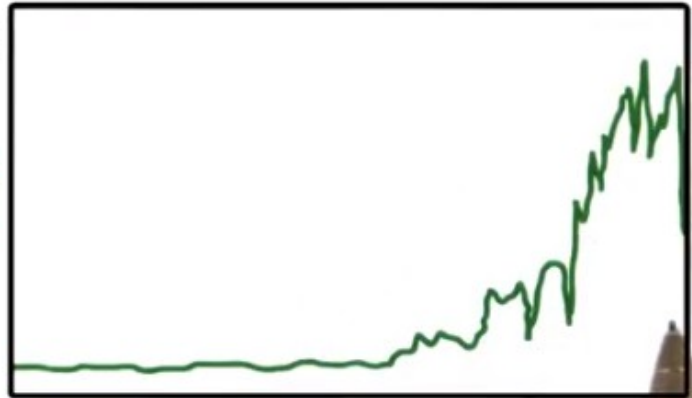


Cloud Computing & Animoto

- Animoto rented "compute instances" in EC2
- In April 2008 became available to Facebook users:
 - Mon 50 → Fri 3400 machines!
 - 750,000 new users in 3 days
- Cannot achieve this with traditional in-house machine deployment and provisioning tools.

Animoto April 2008: Peak EC2 Instances:

- Mon 50, Tues 400, Wed 500, Friday 3400

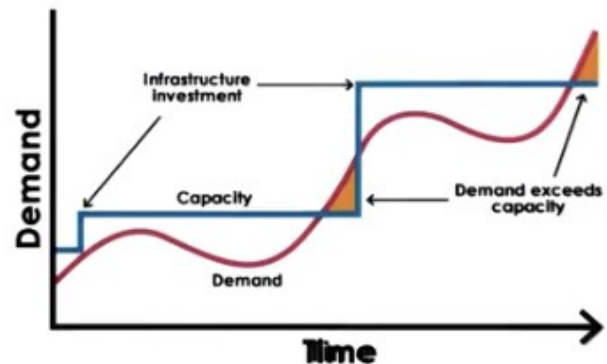


One of the companies that appeared around the same time as Amazon's EC2 cloud was Animoto. Some of you may have used the service. It turns an album of pictures into a video. And although this sounds simple, it's a fairly compute intensive job that involves a lot of image processing steps so that the video appears as smooth as possible. So they decided to focus their resources on the development of the mechanisms that make better videos. And instead of buying and running their own equipment, they chose to rent some of the web provided computer infrastructure that was part of Amazon's compute cloud. The company was doing okay, and it had a relatively steady work load which required about 50 of these Amazon's compute instances. Now these are not physical servers, instead they were virtual machines. They had no control over exactly how Amazon runs these virtual machines, whether they are on the same physical server or many other physical servers. Then, in April 2008, Animoto became available on the Facebook platform. What it means is that it became an option available to Facebook users with a click of a button to turn their timeline photos or albums into cool video. What happened afterwards was the definition of going viral(病毒的). Within 3 days, Animoto signed up 750,000 new users. And from the 50 compute instances, so the 50 machines that it needed on Monday. That number became 400 by Tuesday. So an order of magnitude larger in a day. And then by the end of that week, by Friday, that number of machines was 3,400. Two orders of magnitude increase in the resources, just within that week, just within those four or five days. There's no way they could have been able to respond to this dramatic increase in demand if they had gone with a traditional route of owning and managing their own infrastructure. You just would not be able to bring in, install, wire, configure, et cetera, as many machines in such a short time. Even if by some miracle they actually did have the physical space and could fit all of these machines in the machine room, and then that there was sufficient power to power all that. The only reason that Animoto was able to do this is because they used a cloud-based deployment and then they leveraged the capabilities that cloud computing offers.

Traditional Approach:

- buy and configure resources
⇒ determine capacity based on expected demand (peak)
- when demand exceeds capacity

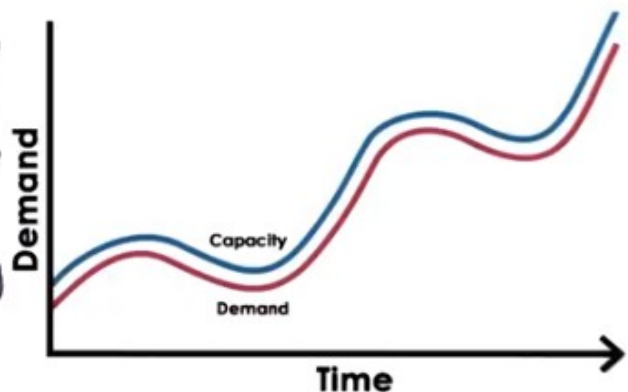
- ⊖ dropped requests
- ⊖ lost opportunity



15. Let's look more formally at what Cloud computing provides. Traditionally businesses would buy and configure the resources that are needed for their services. How many resources should be purchased and configured? So what is the capacity of those resources? Well that typically would be based on some expectations regarding the demand for that particular business, for that particular service, typically considering the peak demand. So if we have a situation in which the Demand follows this pink or this red line, then the business would configure their resources, their capacity, based on this blue line, so that it can tolerate the expected peak in the service demand. Now the expectations turn out to be not quite accurate and the demand ends up exceeding the provision capacity. The business will end up with a situation in which requests have to be dropped and there will be a lost opportunity. In the case of given the very rapid spike in the workload, this lost opportunity would have been tremendous.

Ideal Cloud:

- ⊕ capacity scales elastically with demand
- ⊕ scaling is instantaneous, both up and down
- ⊕ cost is proportional to demand, to revenue opportunity
- ⊕ all of this happens automatically no need for hacking wizardry
- ⊕ can access anytime, anywhere



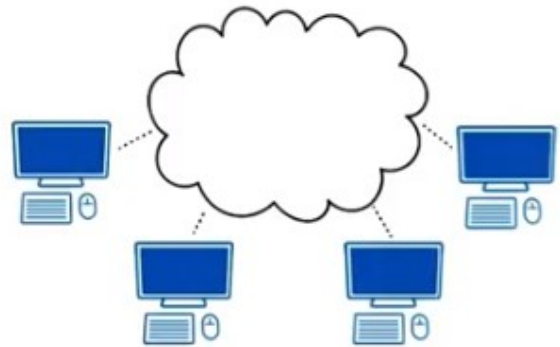
- ⊖ Don't "own" resources

Instead, what we would like would be the ideal case is if the following were to happen. The capacity or

the available resources should scale elastically with the demand and the scaling should be instantaneous. As soon as the demand increases, the capacity should increase too. And then in the other direction, too, as soon as the demand decreases the capacity should decrease as well. Meaning that the cost to operate these resources, the cost to support this service should be proportional to the demand to the revenue opportunity. All of this should happen automatically without the need for some hacking wizardry. And all these resources can be accessed anytime from anywhere. One potential [INAUDIBLE] here is that you wouldn't necessarily own these resources that magically appear on demand. But that may be something that you're willing to compromise on provided that you really do achieve these kinds of benefits. And that there is some kind of proper assurance as to what exactly can possibly go wrong with these resources that you're accessing.

Cloud Computing Requirements : Summarized

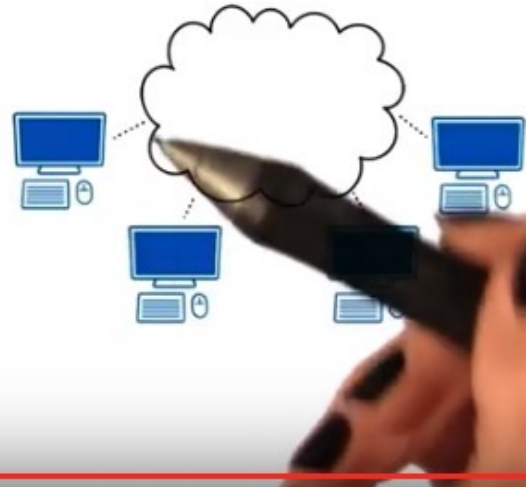
- . On-demand, elastic resources and services
- . Fine-grained pricing based on usage
- . Professionally managed and hosted
- . API-based access



So the goal of Cloud Computing is to provide these capabilities that match the idea scenario as much as possible. The resulting requirements can be distilled as follows. Cloud computing should provide on demand, elastic resources and services. There should be fine-grained pricing based on usage. Not for actual or potentially idle physical servers like in the hosting data centers that were the alternative at the time. All the resources should be professionally managed and hosted. And all of this should be available via APIs that can be used for remote access.

Cloud Computing Overview

- shared resources
 - infrastructure and software/services
- APIs for access & configuration
 - web-based, libraries, command line...
- billing / accounting services
 - many models: spot, reservation, ...
entire marketplace
- typically discrete quantities:
tiny, medium, x-large ...
- managed by (cloud) provider



16. Given these requirements, cloud computing provides the following. First, a pool of shared resources. These can come in as infrastructure, compute storage networking. That doesn't mean that the cloud provider like Amazon, is renting out physical machines, or physical disks. But instead that it's renting out virtual machines that are even potentially interconnected to form some virtual clusters of such [INAUDIBLE] along with some ability to store some amount of state in the underlying storage. Cloud computing can also come in as shared resources that are used by higher level software services, so these are soft resources. For instance, this can correspond to certain popular services like email or database or some processing run times. So it may be easier to just rent all the infrastructure along with the software stack that's appropriately configured, as opposed to, as a customer, to come in and rent actual infrastructure and then deal with the deployment of the service, the configuration and the management of the service. So both of these are options in terms of what it is that cloud computing provides as a resource. These infrastructure software resources are all made available via some APIs for access and configuration. The point is that they need to be accessed and manipulated as necessary remotely, over the internet. This typically includes web-based APIs, also APIs that are wrapped in libraries that can be integrated in popular language runtimes like Java, or command line interfaces, or other types of tools. Providers offer many different types of billing and accounting services. There is an entire marketplace surrounding these cloud services and cloud resources that includes pricing models with spot prices or reservations for future prices or other types of models. One common characteristic is that billing is typically not done based on actual usage. Just because the overheads that are associated with ultra fine grain monitoring and management tend to be pretty high, and instead billing is done based on some discreet step function. For instance, computer resources may come in some number of preconfigured alternatives, like tiny and medium and large and extra large and each of these will have a different cost. So, then the user picks which one of these matches their needs and then pays the corresponding grade when using the VM. And all of this, finally, is managed by the cloud provider via some sophisticated software stack. Common software stacks that are used in this space include the Open Source OpenStack and also the VMWare's vSphere software stack.

Why does cloud computing work?

Law of Large Numbers

- per customer there is large variation in resource needs
- average across many customers is roughly constant

Economies of Scale

- unit cost of providing resources or service drops at "bulk"



17. Two basic principles provide the fundamental theory behind the cloud computing approach. The first is the so-called Law of Large Numbers. This says that once you start considering many customers whose resources may vary over time, the average across all of them will tend to be fairly constant. That's why a cloud computing provider can pick any amount of resources in capacity for the cloud. And with that capacity, can service lots of customers, whose peaks in the expected demand are shifted in time. Second, there is the Principle of Economies of Scale. Essentially, things are cheaper by the dozen. When you start considering how many different customers a cloud provider is able to leverage on a single physical piece of hardware resource. The cost of that hardware resource ends up being amortized. And ends up being an offset by all of the fees that the cloud provider is able to gather from its customers.

Cloud Computing Vision



"If computers of the kind I have advocated become the computers of the future, then computing may some day be organized as a public utility, just as the telephone system is a public utility ... The computer utility could become the basis of a new and important industry."

-- John McCarthy, MIT Centennial, 1961

- computing == fungible utility!
- limitations exist: API lock-in, hardware dependence, latency, privacy, security...



1:58 / 2:06



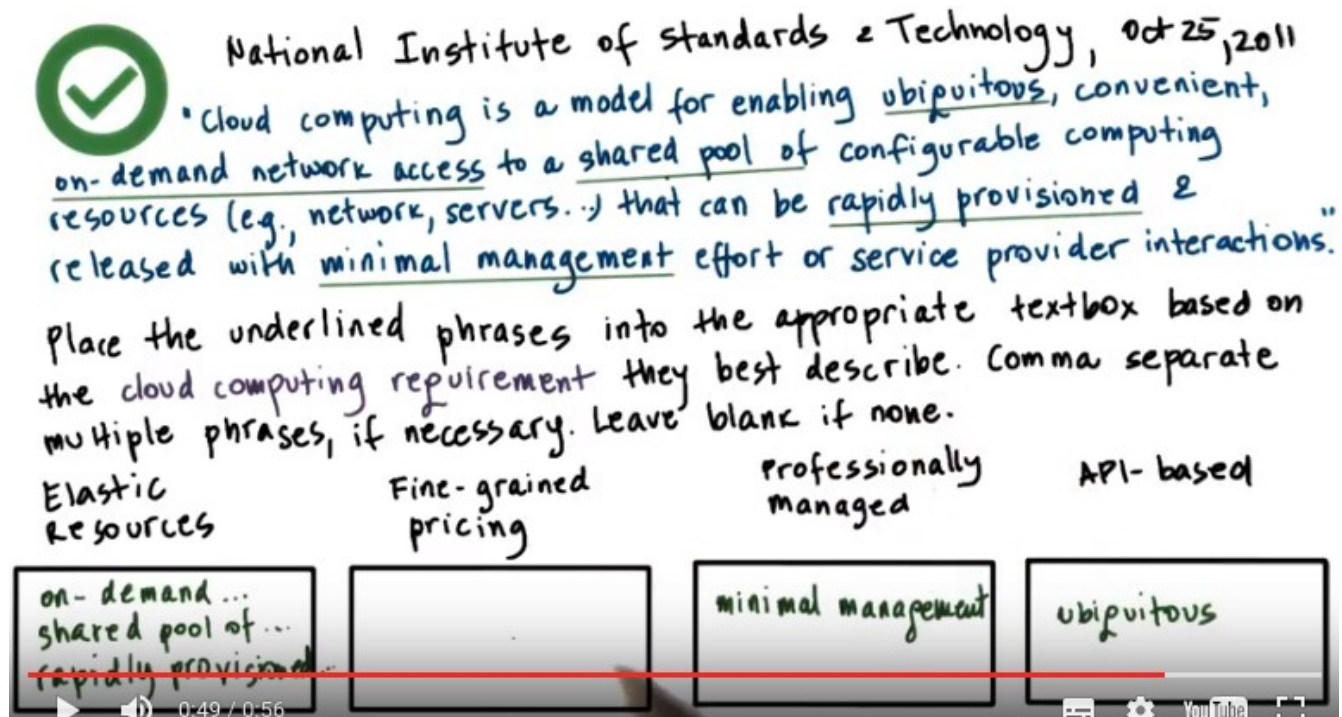
YouTube



fungible: 代替的

18. Interestingly, all the cloud computing is a relatively new trend. As a vision, it existed for a very long time. The oldest description, by John McCarthy, predates the technology by almost half a century, and it appeared in 61. He said the following. If the computers of the kind I have advocated become the computers of the future, then computing may some day be organized as a public utility, just as the telephone system is a public utility. The computer utility could become the basis of a new and important future. If you look at this statement, it describes a vision where computing is a utility just like other utilities. If we need more electricity, we just turn the switch on. We don't worry about where the electricity actually comes from. Based on this vision, cloud computing should turn the IP resource into a fungible utility, where we don't care about what type of CPUs or disk I'll ultimately use or where they are. Virtualization technology is one enabling the process, certainly. Unfortunately there's still limitations before this vision of just seamless computing utility can be realized. For instance, even with virtualization there's still some hardware dependencies that simply cannot be masked. So if software expects a certain type of hardware platform to execute then we cannot just have it run on another type of resource. Then there is the API lock-in where if we're using a particular cloud provider then we're expected to continue using those API's. There isn't a uniform standard across cloud providers. And then, unlike with some of these other utilities, where we're simply bringing this resource, the electricity in, in the case of cloud computing we're putting data, potentially core business services out in the cloud. So some of the privacy and security concerns become much more important. Clearly latency's is an issue given that there is an actual geographic distance related to our access to our

19. Here is a short excerpt from the National Institute of Standards & Technology's document that defines cloud computing. This is a document that was read or published, rather, on October 25, 2011. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources, including network servers, services. It can be rapidly provisioned and released with minimal management effort or service provider interactions. I want you to place the following underline phrases, which there are five of them, into these text boxes, where each text box corresponds to one cloud computing requirement. If you need to list multiple phrases in a single text box, please comma separate them, and if you need to leave a text box empty, or blank rather, then don't write anything in it.



A handwritten note on a green background with a green checkmark icon. The text reads: "National Institute of standards & Technology, Oct 25, 2011". Below this, a definition of cloud computing is written in blue ink: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., network, servers...) that can be rapidly provisioned & released with minimal management effort or service provider interactions." Below the definition, instructions are written in black ink: "Place the underlined phrases into the appropriate textbox based on the cloud computing requirement they best describe. Comma separate multiple phrases, if necessary. Leave blank if none." Below the instructions are four text boxes labeled "Elastic Resources", "Fine-grained pricing", "professionally managed", and "API-based". The first box contains the text "on-demand... shared pool of... rapidly provisioned...". The other three boxes are empty. A red horizontal line is drawn across the boxes. Below the boxes is a video player interface with a progress bar at 0:49 / 0:56 and a YouTube logo.

National Institute of standards & Technology, Oct 25, 2011

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., network, servers...) that can be rapidly provisioned & released with minimal management effort or service provider interactions."

Place the underlined phrases into the appropriate textbox based on the cloud computing requirement they best describe. Comma separate multiple phrases, if necessary. Leave blank if none.

Elastic Resources Fine-grained pricing professionally managed API-based

on-demand... shared pool of... rapidly provisioned... minimal management ubiquitous

ubiquitous: 無所不在的

20. Looking at this definition and the cloud computing requirements that we discussed already, here are how these phrases map to these requirements. When we think about elastic resources, that means that these are on demand network resources, whenever we need them they will get created. They come out of a shared pool of resources, so that's how we can support this requirement, and this provisioning happens rapidly. When we think about API-based, that really means that the resources, the cloud computing resources are ubiquitously available. They need to be professionally managed, so that's the definition, the part of the definition that talks about minimum management effort. There really isn't anything in this statement that talks about the pricing aspect of cloud computing and in principle, there really isn't a solid well-established definition of what cloud computing is, that the entire community subscribes to.

Cloud Deployment Models

Public

- third party customers / tenants

Private

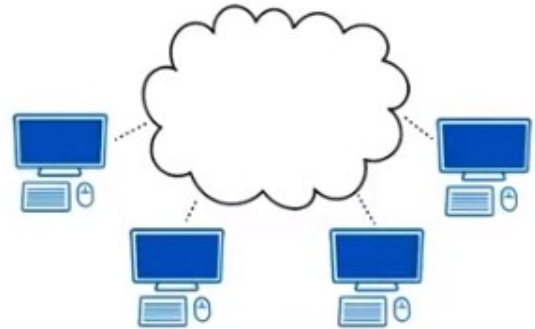
- leverage technology internally

Hybrid (Public + Private)

- failover, dealing with spikes, testing

Community

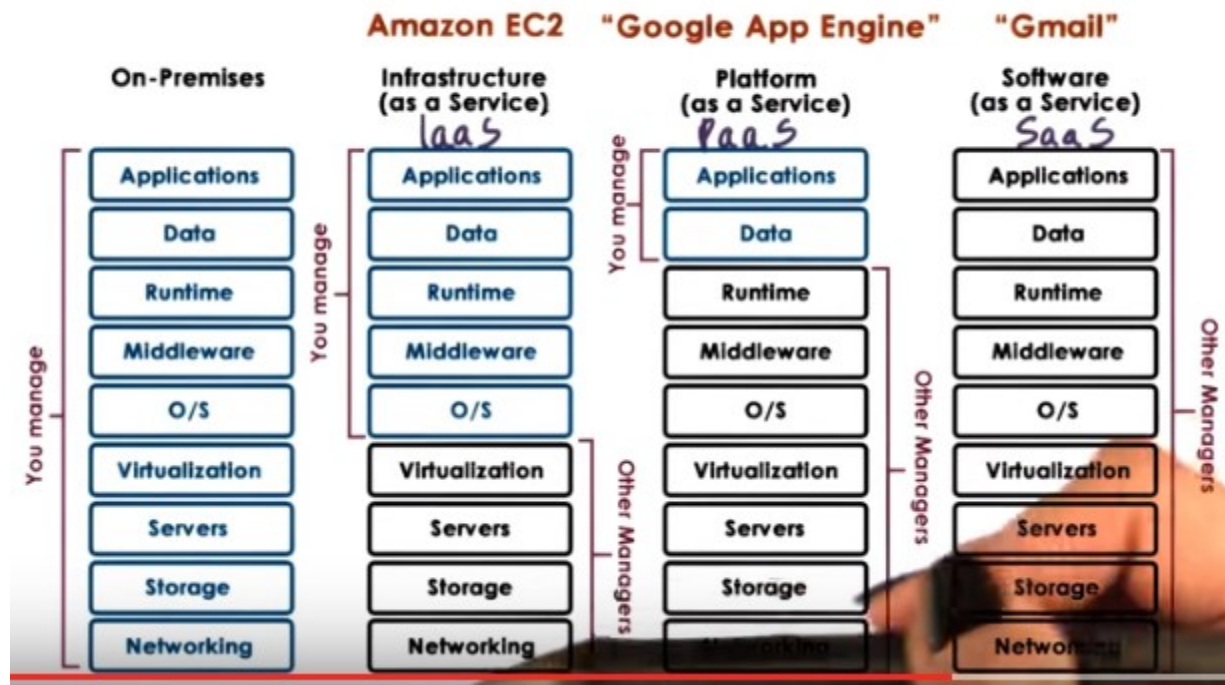
- used by certain type of users



spike: 用尖物刺傷；打亂某人的計劃

21. The National Institute of Standards and Technology, which adapted a definition for cloud computing in 2011 also defined different types of clouds. The first dimension how clouds are classified is based on the deployment models. Clouds can be public like Amazon's EC2 cloud is a public cloud. And this is a cloud where the infrastructure belongs to the cloud provider. However, third party customers or tenants, anyone with a credit card really, can come in and run their services, their infrastructure, on top of Amazon's own hardware. Clouds can be private. I said earlier that Amazon used the same type of technology that they opened up as Amazon Web Services. They used it originally internally to enable elastic and dynamic provisioning of the physical infrastructure across the different types of applications and services that Amazon internally executes. In the case of a private cloud, it is the infrastructure as well as the tenants, the services, the applications, that are run on top of that infrastructure. Everything is owned by the same entity and it's just cloud computing technology is used to enable some of the provisioning and the elasticity that cloud computing offers to applications. Then there is a hybrid cloud as a possible deployment model. This is a scenario where a private cloud is interfaced with some public cloud where these public resources are used either for failover for some additional redundancy, for dealing with spikes. I know of certain companies who choose these public resources to run simulated workloads that would then generate supposed request patterns on top of their, privately run core services. Finally the definition also references that there is a community cloud as a type of cloud. This is really just a public cloud where the third party customers or the third party users of the services or the information that's provided by this community cloud, isn't really just used by arbitrary customers but by a certain type of users.

Separation of Responsibilities



SaaS: software as a service, PaaS 和 IaaS 類似

22. The second way how clouds are commonly differentiated is based on the service model they provide. Here's a great image that depicted the differences between the different service models.

On-Premises:

Without the use of cloud computing, you run your application on premise, and that means you have to take care of all of these different layers yourself. You manage them personally. The applications, the data, the runtime, middleware, O/S, so different components of the software stack including the virtualization technology. And then the lower level, all of the components of the hardware stack.

SaaS (Gmail):

In the other extreme, you can use cloud computing services and then directly leverage the fact that cloud computing can provide you with an application. So this is a model called software as a service. And this resembles what Gmail offers you. You don't have to run your mail server, install quotas, manage security patches, etc., for the employees in your company. You can just force them to use Gmail and, in fact, can just contract Gmail to run the email services for your company. In the case of Gmail, everything is run by Google in this case. It's their hardware, it's their software stack, system software. They own and manage the data, they run the application, configure it. Update it, upgrade it and everything else that's necessary. So this is the software as a service, or SaaS model.

PaaS (Google App Engine):

A second model is where the cloud service can provide you certain APIs that you can access to develop particular types of applications. The cloud service provides a development or execution environment. And that includes the operating system, various libraries, various tools that are necessary for the applications that need to run in that environment. One example is the Google App Engine and this is

called platform as a service offering, or PaaS. In the case of the Google App Engine, the platform as a service offering is that it offers everything that's necessary to develop applications for Android platforms, for instance. Windows Azure was originally purely intended as a pass offering for developing and running .NET applications, and more recently another type of extension was added to the Windows Azure portfolio services.

IaaS (Amazon EC2):

Finally at the lowest level, clouds can provide infrastructure instances, like compute instances that consist of the CPUs, or rather virtual CPUs with accompanying memory, or storage, or the necessary network resource in order to form a cluster of such compute instances. Amazon's compute cloud is an example of such an infrastructure as a service (IaaS) cloud computing model. This can also include other types of hardware resources, like GPUs, so you can rent from Amazon instances with GPUs for instance. Again know that these types of clouds don't really provide you with the physical resources directly. Instead, the resources are virtualized. And so it may turn out that you are using independently some subset of the physical resources available in that cloud system. But it's quite possible and likely that you're sharing the resources with other tenants. An exception to this is that, in particular of the case of Amazon, they do provide some high performance instances. So if you're renting through Amazon, a cluster of VMs or virtual cluster or virtual high performance instance, you will in fact end up at a virtual machine that runs independently(即不跟別人 share) on physical hardware. The same thing happens when it comes to the GPUs. Your GPUs won't be shared in this case. But that's just because of some technical challenges in terms of how to get sufficient performance level in a virtualized environment for these high performance instances or GPU enabled instances that Amazon provides.

Requirements for the Cloud

1. "fungible" resources
2. elastic, dynamic resource allocation methods
3. scale: management at scale, scalable resource allocations
4. dealing with failures
5. multi-tenancy: performance & isolation
6. security

fungible: 代替的

23. From everything that we have said so far, the technologies that are used to enable computer cloud offerings must address a number of requirements. Clouds must provide fungible resources which means that the resources can easily be repurposed to support different customers. Potentially even with

different types of requirements. Otherwise if cloud providers have to allocate and maintain physical resources for every one of their customers, and the exact same type of physical resources as what the customers at least think they require, then the economic opportunity for cloud computing just won't be there. This fungibility is also necessary to deal with some of the heterogeneity that exists within the cloud resources. And we have different generation of servers, maybe even different types of servers. And you don't really want the users to have to deal with that. You don't want them to be exposed to the level of heterogeneity. Clouds must integrate resource management methods that support the premise of cloud computing so they must be able to dynamically adjust the resources that are allocated to cloud users pending underneath. So this needs to be done very flexibly, very elastically. Such management methods must be able to operate at very large scales of thousands and tens of thousands of nodes. The scale is important, both from the perspective of the cloud provider. So how many resources it needs to manage. But also from the perspective of the customers. Customers often look at clouds for their resource needs because otherwise they don't have access to sufficiently large resource pools. So then clouds must be able to provide to potential customers the ability to allocate really large collections of resources for your individual needs to the scalabilities required with respect to that dimension as well. Once scale is introduced, failure has become inevitable, really. If we have some probability that one component in the system will fail, the more such components we put together, the greater the probability that something in that collection will fail. So we have to incorporate mechanisms to deal with failures. Clouds, by definition, are shared resources across multiple tenants, so cloud management has to deal with this multi-tenancy, has to provide mechanisms that guarantee performance and provide isolation across multiple workloads, multiple tenants. We cannot have one misbehaving tenant that will somehow take control over all of the resources and punish and hurt the performance of the remaining tenants. And finally an important concern that must be addressed is isolation of the state that's being accessed in cloud systems. With this respect, clouds need to make guarantees regarding the privacy of their tenants' data and the security of the execution environment that the cloud guarantees. Another aspect of security's also to make sure that their privacy and security guarantees, not just across tenants, but also that the cloud provider isn't going to somehow access or take advantage of the state that is managed by individual tenants. And then also the other way around, that the cloud computing player is protected from certain vulnerabilities that exist within a single tenant. And therefore, that one tenant isn't going to affect the entire cloud platform.

24. I mentioned failures in the previous Morris cell, so let me try to illustrate why our failure is more of a problem in a large scale system like a cloud platform, than if we're not really concerned with scale. Think about the following question. A hypothetical cloud has $N=10$ components, or 10 CPUs, where each CPU has a failure probability of 0.03. My question is, what is the probability that there will be a failure somewhere in this system, in case there are ten components. Then think about also what will happen if there are more components. So they're 100 components, what in that case is the probability of a failure? You should provide the answer in terms of what is the percentage chance.



Cloud Failure Probability Quiz

A hypothetical cloud has $N=10$ components (CPUs). Each has failure probability of $p=0.03$. What is the probability that there will be a failure somewhere in the system?
What if the system has $N=100$ components

% failures ($N=10$)

% failures ($N=100$)

以上答案是正確的。答案頁面太亂，故用的這個頁面。

Prob of failure = $1 - (1 - p)^N$

$1 - (1 - 0.03)^{10} = 0.26$

$1 - (1 - 0.03)^{100} = 0.95$

25. To insert a question, think about this set of formulas. p is our probability that a single component will fail. Then $1-p$ is the probability that a single component will not fail. For there not to be a failure anywhere in this system, so we're trying to answer the inverse of this question. That means that not a single one of the components in the system should fail. If the probability of one component not failing is $1-p$, then the probability of no components in the system failing is $(1-p)$ to the n th. Their total of end components in the system. So this will be the probability that nothing will go wrong in the system. And the question here is what is the probability that something will go wrong? So there will be a failure somewhere in the system, that's clearly 1 minus this result. The formula for the total answer is, on parentheses, 1, minus p to the n . If you do the math, you will see that with this probability, so there's a 3% of something failing and you have ten components. If you do this math, you have a 26% chance that something will go wrong. If you increase the scale and your n becomes 100, then if you do this math you'll realize that with this high probability of failure, 95% of the time you will have a failure at some point. Our cloud systems don't have 10 or 100 components. They have thousands and hundreds of thousands of components. Yes, they may be more stable than this 0.03 probability of failure, but the point is that things will fail. The more components you put together, the more you'll have situations in which something, somewhere, is failing. So you have to have mechanisms that are prepared to deal with those kinds of failures. For instance, your software may have to incorporate mechanisms that include timeouts, so that you can detect that a failure has occurred or to integrate mechanisms for restart and retry in order to be able to recover from failure. You have to incorporate mechanisms for backup, for application, for checkpointing. So you have to accept, you have to embrace the fact that failures will happen, and so you have to build robust software that will be able to

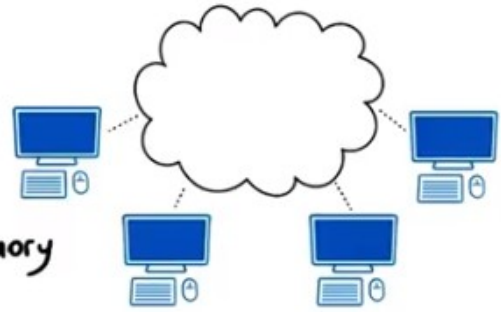
detect and recover or avoid failures from occurring.

Cloud-Enabling Technologies

- Virtualization
- Resource provisioning (scheduling)
mesos, Yarn...
- Big Data processing (Hadoop MapReduce, Spark...)

storage

- Distributed FS ('append only')
- NoSQL, distributed in-memory caches ...



- Software - defined ... networking, storage, datacenters...
- monitoring => real time log processing (Flume, CloudWatch, Log Insight...)

26. Given the requirements that we listed in terms of what a cloud system has to incorporate, several different technologies come into play. First, we already talked about the need for virtualization. This is required in order to address this ability to provide fungible resources that can be dynamically repurposed. Which application they serve and then exactly how are they used, what are the expectations? You don't have to have the exact same CPU or the exact same type of device based on the customer's needs. Then you need some technologies to enable the dynamic provisioning and scheduling of the infrastructure resources. Certain platforms like Meso, or Yarn, Hadoop's Yarn, that serve this role. In order to address the customers' needs for scale, cloud technologies need to provide abilities to process and store large amounts of data. There are a number of big data processing frameworks that are out there. Hadoop MapReduce is one. Spark's another one that is popular. Regarding the data storage layer, cloud computing platforms include things like distributed file systems. These are file systems that typically operate in some append only mode where you're not arbitrarily rewriting data and deleting data and modifying data. And then there are other important technologies that enable the storage, access and manipulation of data at scale. These include a number of NoSQL technologies, which are a departure of traditional relational SQL databases. The ability to store data across distributed memories, as opposed to repeatedly perform disk accesses, and a number of other enhancements. Cloud computing customers, they not only need to get the right amount of resources. However, they need to somehow be isolated from each other so that every one of them thinks that they own their resource slots. For that reason, cloud computing technologies need to include some kind of mechanism that enables these software-defined slices of resources so that costumers are provided with their own software-defined data center or subset of the storage that's controlled and defined by the software or the network resource, etc. And then also, especially given the complexity of these environments, it's important for cloud computing platforms to also use a certain efficient monitoring technologies. In real time to process logs, to detect certain anomalies or failures, but also to provide that as a service to the users that may not just want some batch access to potentially

large data but more of a real time, more of a maybe interactive access to information that's produced from their applications. Some of the technologies that provide this monitoring type functionality or real-time log processing include Flume and CloudWatch, this is the Amazon service, or LogInsight, this is a product that's part of the VMware portfolio. Flume is Open Source, and many, many others.

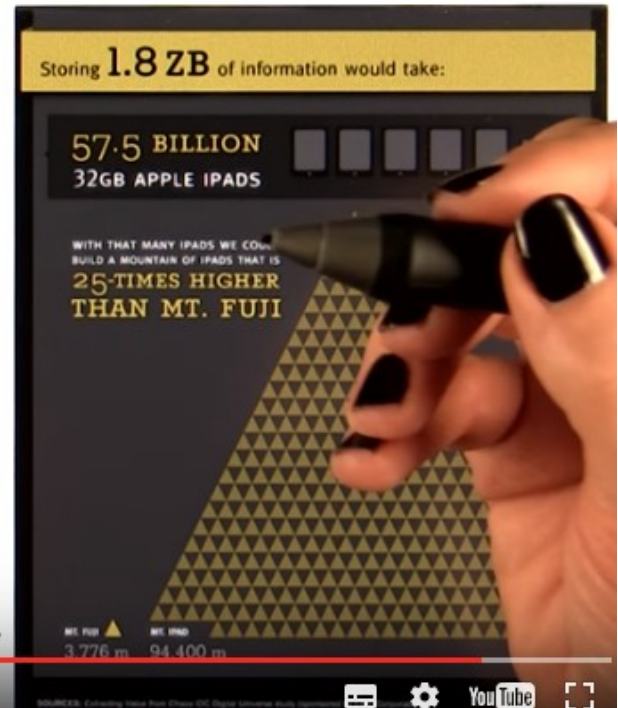
"The Cloud as a Big Data Engine"



27. One benefit of cloud computing is that it empowers anyone to potentially have infinite resources. As long as you can pay, you can have the resources that are required to work on maybe hard problems that involve lots of data and that require lots of processing. Here are some illustrations that I've found pretty informative. To illustrate how much of data are we dealing with in terms of our everyday life in our world. So if we just think about one year, 2011, there were 1.8 zettabytes of data that were created in that one year. This amount of data is equivalent to everyone in the US tweeting 4,320 times a day. Pretty much not doing much else. Or another way to think about it's this amount of data is 200 billion high definition movies that are each about two hours long. If you lived for 47 million years, you'd be just about done watching these 200 billion movies.

"The Cloud as a Big data Engine"

- data storage layer
- data processing layer
- caching layer
- language front-ends (e.g. Querying)
- analytics libraries (e.g. ML)
- continuously streaming data



Or to store this much data, it would require so many of the Apple's 32 gigabit iPads, that if you stack them up together, it would be 25 times higher than Mt Fiji. Pretty powerful these illustrations. And this is all the data from 2011 only. Clearly not every application needs access to all of the data that's generated that's available in the world. But a lot of the applications for scientific discovery, for improving social services, for a number of other types of applications, they do require a lot data. Looking at cloud computing as a big data engine, there are a number of layers that are necessary in order for the cloud to provide that kind of infrastructure and adopt those kinds of services that are needed by these big data applications. Cloud platforms that offer as a platform as a service stack for big data processing at the minimum have to have some layer that enables the storage and access of the data from cross many, many nodes. And also the ability to describe applications in a certain parallel fashion, and ideally to do that easily, so that the data can be processed across potentially many nodes. We talked earlier in this course that access to memory and access to local memory is faster than going to disk. And particularly if you have to go to disk that need these nodes that's attached to another node. So most of the cloud stacks also incorporate some data caching layer where the data can be brought in memory, or maybe even a cross multiple memories, in the distributed system, big data system. I said a cloud platform for big data must incorporate some data processing framework. But commonly, when we talk about small or smaller data, the way we are analyzing it is using things like SQL queries or some other type of querying mechanism. And so often cloud stack, big data stacks, would incorporate some language front end that would make this data processing a little bit more easier. One common thing that people do with big data, that they analyze it. They search for certain patterns, so they perform some data mining operations that often involved common functions, such as some machine learning algorithms. So just as it's common for cloud big data platforms to support popular language front-ends, it's also common for them to support popular analytics libraries. And also, a lot of the types of data that we're interested in isn't just created once, and then that's it, that's a complete data set. Instead, there's continuously generated knowledge. And then potentially, the answers from the analysis on that knowledge are derived continuously and updated over time. So therefore, a common service that needs to be incorporated in big data clouds would be something that provides as streaming

abilities, both as inputs and as outputs.

Example Big Data Stack



Hadoop

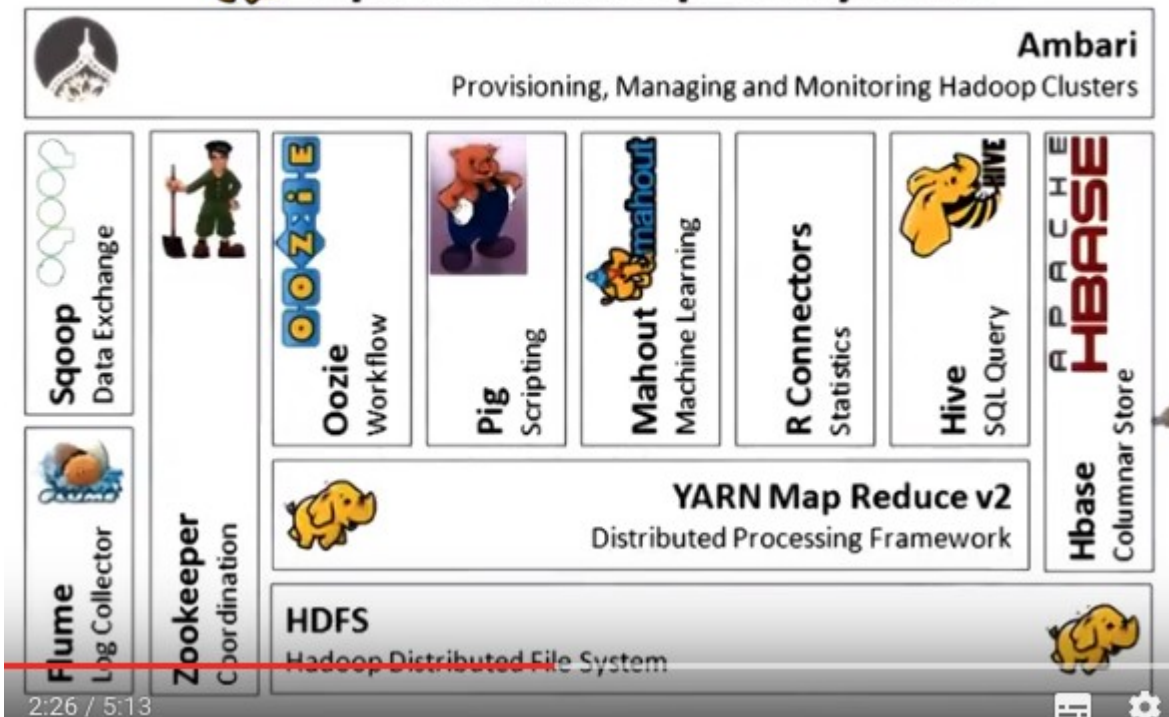


Berkeley Data Analytics Stack (BDAS)

28. So we mentioned, a number of complete technologies that are used in cloud computing and also we mentioned a number of types of functionality that cloud computing technology needs to provide. Again, let me offer a birds eye view of two popular stack for big data that are used in cloud computing platforms. The two concrete stacks I want to show you is the Hadoop Big Data Stack, it's an open source stack and then also the Berkeley Data Analytics Stack, BDAS. This is another open source stack. These are by the way, not the only options. There are number of both propriety stacks and also a number of other open source stacks. For instance, one example is the so-called HPCC stack by LexisNexis. This actually, even predates Hadoop. However, it wasn't necessarily popular or widely used outside of the LexisNexis environment. When it comes to the overall popularity and adoption rate, both in commercial and academic settings or research settings, these two are probably the dominant stacks out there today.

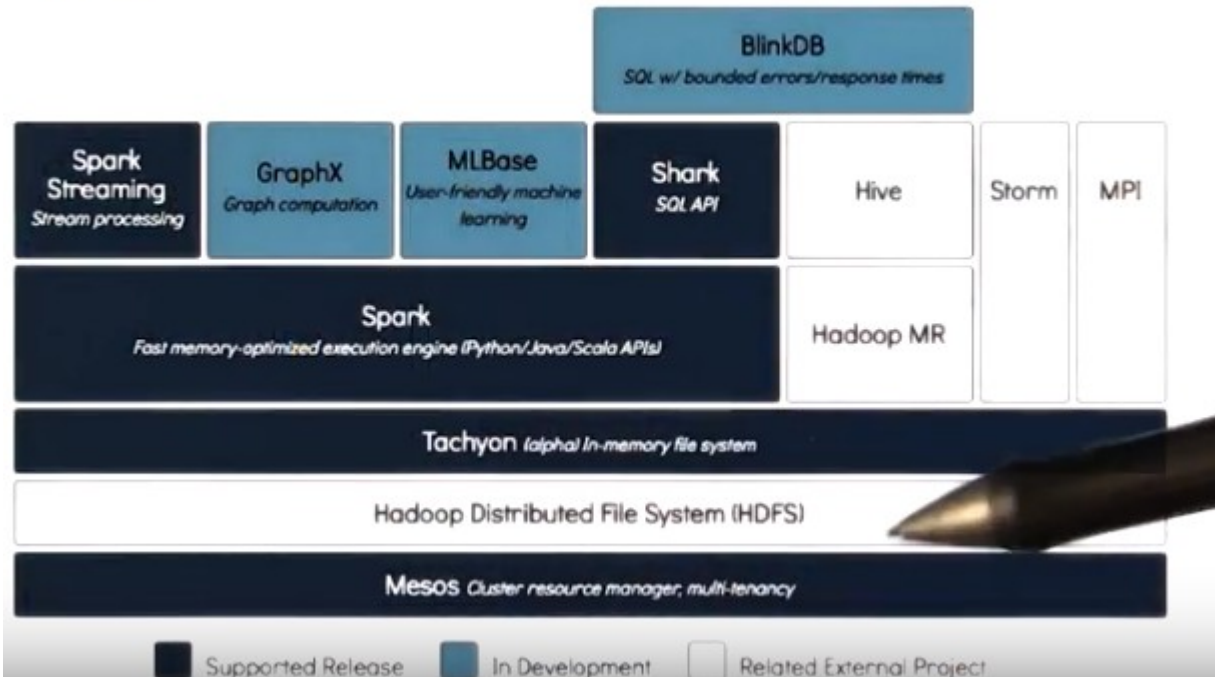


Apache Hadoop Ecosystem



The Hadoop architecture looks like this. At the bottom layer, there is the data storage layer, the Hadoop file system and this takes care of taking large amounts of data and somehow, splitting them and replicating and making sure the data doesn't get lost and making sure that there are no hot spots in the system, etc. Then there is a processing framework that can allow computation to operate on such data and deals with the scheduling of what are the specific nodes that particular data manipulation operations need to be scheduled. Ideally, the scheduling should be done in such a way, so that you don't have to constantly move data from one node to another, depending on what is the task that needs to process it. We're really not going to talk in more detail about the Map Reduce framework. The intent of these is just as an illustration to provide you with an overview of the breadth of technologies in this space. There's a component Hbase and this provides a table like view of the stored data that is more similar to with what we're familiar with in the context of databases, but it is not the same as the relation of database representation of the data as were used in the SQL environment. (此句講另一個 component: Hive); In order to support data processing and data query operations like what SQL databases support, there's a language front end hive that will take an SQL query and then translate it into a number of these Map Reduce operations. That will then operate on top of the data that's stored in this Hbase tables on top of the distributed file system. Order number of higher levels services that would also provide the same kind of things. So end users can use the R environment in order to describe certain rules about how data should be processed and analyzed or we can use machine learning algorithms, one of a collection of many that are supported in the Mahout component that's part of the Hadoop stack. And a number of other technologies that provide more specialized type soft functionality or different kinds of interfaces to the end users. At the lowest level, however, all of these end up being a number of these more primitive operations that can execute on top of the distributed file system where the data is stored. And there are a number of supporting services that are needed in order to deal with locker or coordination or to provide the streaming of data into this distributed environment.

BDAS



And this is what the Berkeley BDAS looks like. There's similarly a component that's the data storage layer and then there is a in memory component that is the in memory file system, so that you can avoid the caches. You can run the regular Hadoop Map Reduce engines on top of this layer or you can run another type of programming model, another processing framework called Spark. There are a number of front-ends for the Spark in terms of the kinds of execution models that are supported, so whether it's in SQLite queries or stream processing or graph processing or machine learning types of operations that should be executed on top of this processing framework. Another component that's important to mention is this Mesos component that's the lowest level scheduling framework, the lowest level resource manager that allows the resources to be partitioned and then used by different types of frameworks. You may have a partition that's a Hadoop partition. A partition that's a Spark partition. A partition that's an MPI partition. And using this stack, the two will be able to basically coordinate and negotiate among them and then you'll have elasticity across these very different types of framework.

Lesson Summary

Datacenter Technologies

- **Challenges for supporting large-scale services and applications**
- **Traditional multi-tier models and tradeoffs**
- **Cloud computing: definitions, challenges, and technologies**

29. In this lesson, we talked about several technologies that are relevant to the data center space. We described the commonly used multi-tier software architecture models, and discussed the management tradeoffs that exists among them. We talked about cloud computing and the role that it plays in addressing some of the scale and cost related challenges to supporting really large-scale applications and services. And we also briefly mention some of the popular technologies that form the cloud computing landscape today.

30. As the final quiz, please tell us what you learned in this lesson. Also, we would love to hear your feedback on how we might improve this lesson in the future.