KTU NOTES

# CLOUD COMPUTING – MODULE 3

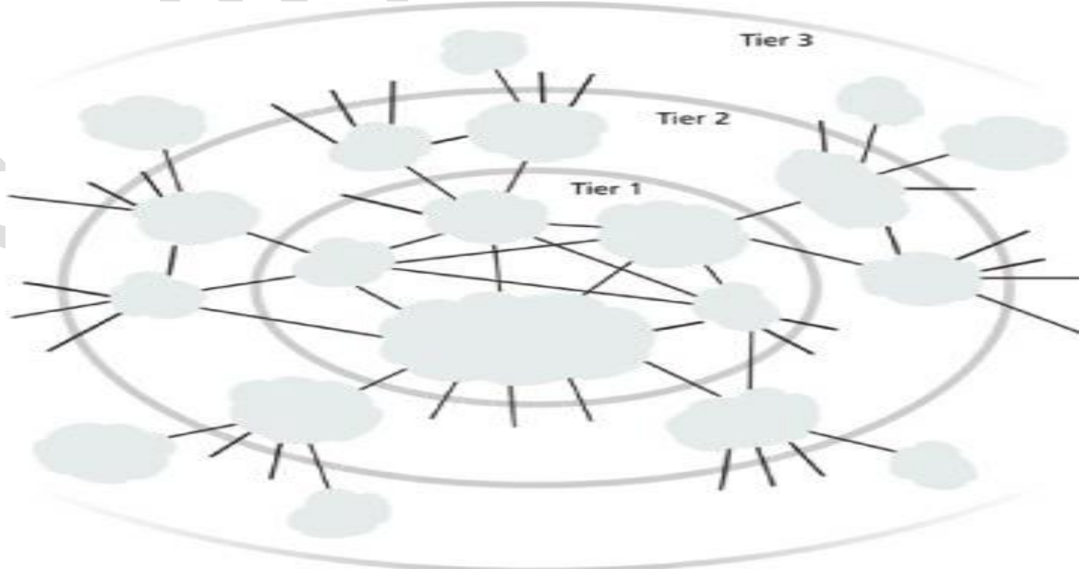
**Syllabus:**

- *PART1: Broadband networks and internet architecture- Internet Service Providers (ISPs), Data center technology, Web technology, Multitenant technology, Service technology. Resource provisioning techniques-static and dynamic provisioning.*
- *PART2: Open-source software platforms for private cloud-OpenStack, CloudStack, Basics of Eucalyptus, Open Nebula, Nimbus.*
- *PART3: Cloud Programming- Parallel Computing and Programming Paradigms. Map Reduce – Hadoop Library from Apache, HDFS, Pig Latin High Level Languages, Apache Spark.*

**PART -1**

## # Internet Service Providers (ISPs)

- ISP stands for Internet Service Provider.
- It is a company that provides access to the internet and similar services such as Website designing and virtual hosting.
- For example, when you connect to the Internet, the connection between your Internet-enabled device and the internet is executed through a specific transmission technology that involves the transfer of information packets through an Internet Protocol route.
- An ISP network interconnects to other ISP networks and various organizations.
- ISPs can freely deploy, operate, and manage their networks in addition to selecting partner ISPs for interconnection.
- Worldwide connectivity is enabled through a hierarchical topology composed of Tiers 1, 2, and 3
- The core Tier 1 is made of large-scale, international cloud providers that oversee massive interconnected global networks, which are connected to Tier 2's large regional providers.
- The interconnected ISPs of Tier 2 connect with Tier 1 providers, as well as the local ISPs of Tier 3.
- Hence, Cloud consumers and cloud providers can connect directly using a Tier 1 provider, as any operational ISP can enable Internet connection.



-

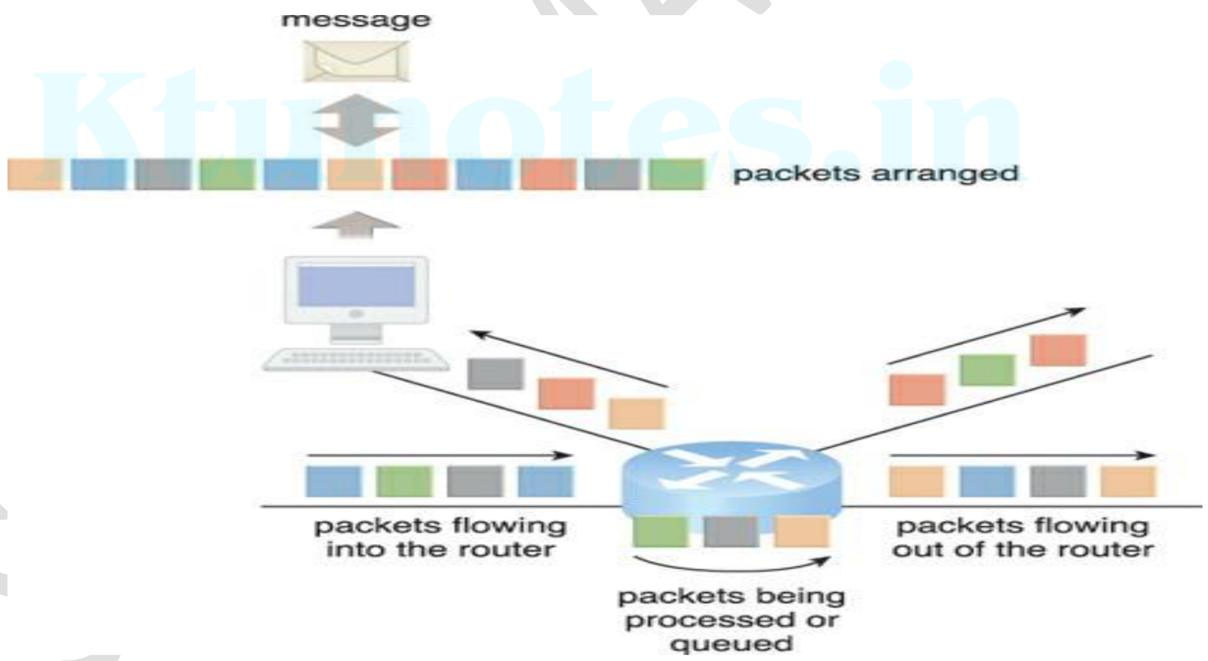
Fig: An abstraction of the internetworking structure of the internet

**Connectionless Packet Switching (Datagram Networks)**

- End-to-end (sender-receiver pair) data flows are divided into packets of a limited size that are received and processed through network switches and routers, then queued and forwarded from one intermediary node to the next.
- Each packet carries the necessary location information, such as the Internet Protocol (IP) or Media Access Control (MAC) address, to be processed and routed at every source, intermediary, and destination node.

**Router-Based Interconnectivity**

- A router is a device that is connected to multiple networks through which it forwards packets. Even when successive packets are part of the same data flow, routers process and forward each packet individually while maintaining the network topology information that locates the next node on the communication path between the source and destination nodes.
- Routers manage network traffic and gauge the most efficient hop for packet delivery, since they are privy to both the packet source and packet destination. The communication path that connects a cloud consumer with its cloud provider may involve multiple ISP networks.
- The Internet's mesh structure connects Internet hosts (endpoint systems) using multiple alternative network routes that are determined at runtime. Communication can therefore be sustained even during simultaneous network failures, although using multiple network paths can cause routing fluctuations and latency.


message
packets arranged
packets flowing into the router
packets flowing out of the router
packets being processed or queued

**Technical and Business Considerations**

- **Connectivity Issues**
  In traditional, on-premise deployment models, enterprise applications and various IT solutions are commonly hosted on centralized servers and storage devices residing in the organization's own data center. End-user devices, such as smartphones and laptops, access the data center through the corporate network, which provides uninterrupted Internet connectivity. TCP/IP facilitates both Internet access and on-premise data exchange over LANs

Organizations using this deployment model can directly access the network traffic to and from the Internet and usually have complete control over and can safeguard their corporate networks using firewalls and monitoring software. These organizations also assume the responsibility of deploying, operating, and maintaining their IT resources and Internet connectivity.

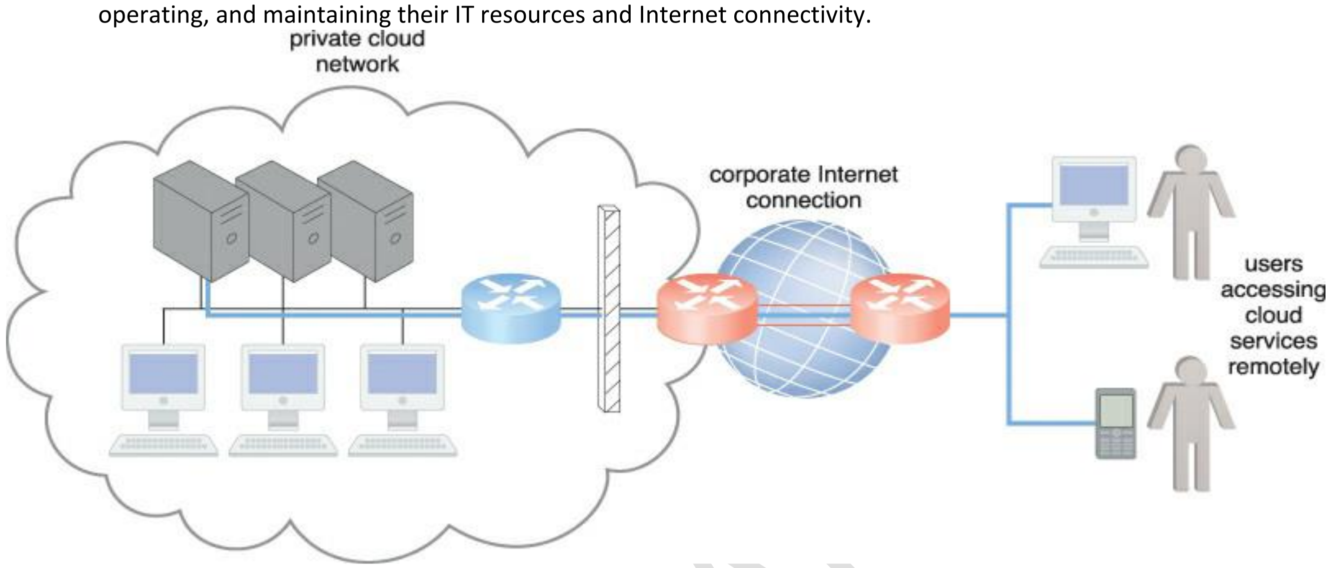

private cloud network
corporate Internet connection
users accessing cloud services remotely

Fig: The internetworking architecture of a private cloud. The physical IT resources that constitute the cloud are located and managed within the organization.

End-user devices that are connected to the network through the Internet can be granted continuous access to centralized servers and applications in the cloud. A salient cloud feature that applies to end-user functionality is how centralized IT resources can be accessed using the same network protocols regardless of whether they reside inside or outside of a corporate network. Whether IT resources are on-premise or Internet-based dictates how internal versus external end-users access services, even if the end-users themselves are not concerned with the physical location of cloud-based IT resources

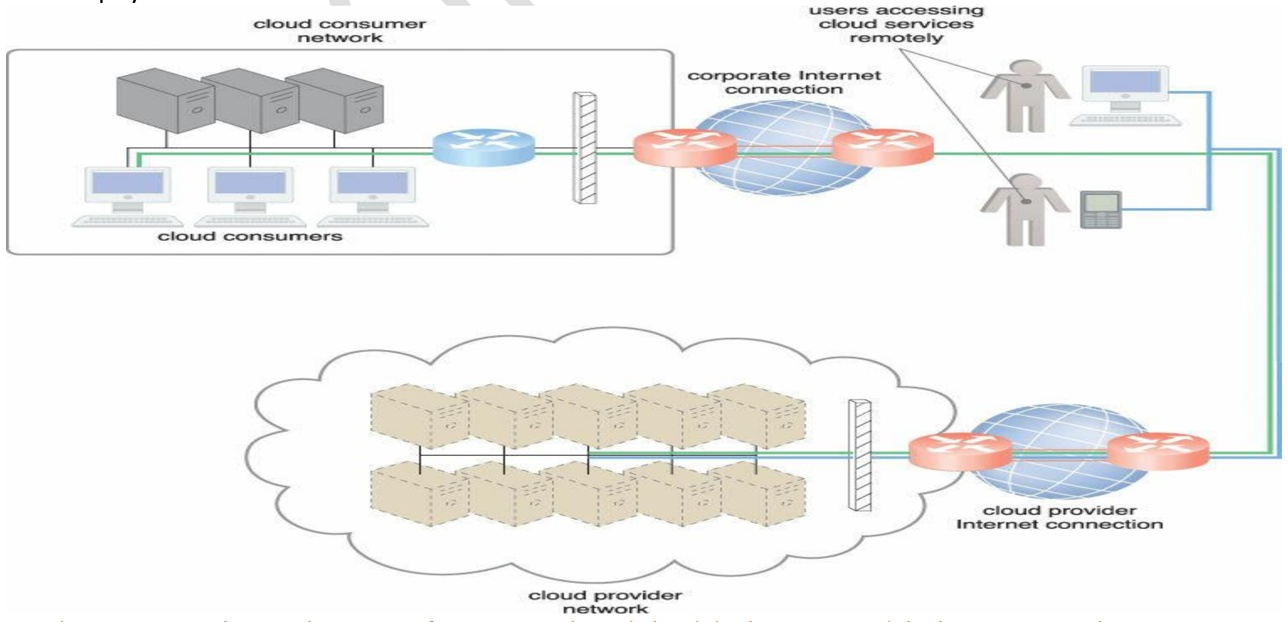
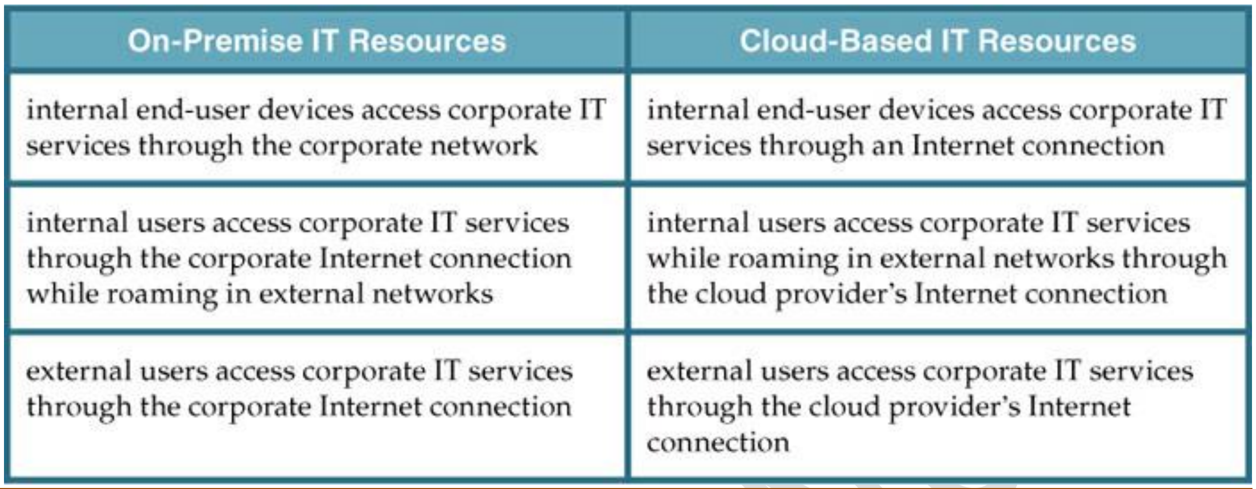

cloud consumer network
users accessing cloud services remotely
corporate Internet connection
cloud consumers
cloud provider Internet connection
cloud provider network

Fig: The internetworking architecture of an Internet-based cloud deployment model. The Internet is the connecting agent between non-proximate cloud consumers, roaming end-users, and the cloud provider's own network

**A comparison of on-premise and cloud-based internetworking:**



| On-Premise IT Resources | Cloud-Based IT Resources |
|---|---|
| internal end-user devices access corporate IT services through the corporate network | internal end-user devices access corporate IT services through an Internet connection |
| internal users access corporate IT services through the corporate Internet connection while roaming in external networks | internal users access corporate IT services while roaming in external networks through the cloud provider's Internet connection |
| external users access corporate IT services through the corporate Internet connection | external users access corporate IT services through the cloud provider's Internet connection |

**Network Bandwidth and Latency Issues**

- End-to-End bandwidth is determined by the transmission capacity of the shared data links that connect intermediary nodes. This type of bandwidth is constantly increasing, as Web acceleration technologies, such as dynamic caching, compression, and pre-fetching, continue to improve end-user connectivity.
- *Latency* is the amount of time it takes a packet to travel from one data node to another. Latency increases with every intermediary node on the data packet's path. Transmission queues in the network infrastructure can result in heavy load conditions that also increase network latency.
- Packet networks with "best effort" quality-of-service (QoS) typically transmit packets on a first-come/first serve basis. Data flows that use congested network paths suffer service-level degradation in the form of bandwidth reduction, latency increase, or packet loss when traffic is not prioritized. The nature of packet switching allows data packets to choose routes dynamically as they travel through the Internet's network infrastructure.

**Cloud Carrier and Cloud Provider Selection**

- The service levels of Internet connections between cloud consumers and cloud providers are determined by their ISPs, which are usually different and therefore include multiple ISP networks in their paths. QoS management across multiple ISPs is difficult to achieve in practice, requiring collaboration of the cloud carriers on both sides to ensure that their end-to-end service levels are sufficient for business requirements.
- Cloud consumers and cloud providers may need to use multiple cloud carriers in order to achieve the necessary level of connectivity and reliability for their cloud applications, resulting in additional costs. Cloud adoption can therefore be easier for applications with more relaxed latency and bandwidth requirements.

## # Web Technology

**Basic Web Technology**

- The World Wide Web is a system of interlinked IT resources that are accessed through the Internet.
- The two basic components of the Web are the Web browser client and the Web server.

- Other components, such as proxies, caching services, gateways, and load balancers, are used to improve Web application characteristics such as scalability and security.
- These additional components reside in a layered architecture that is positioned between the client and the server.

Three fundamental elements comprise the technology architecture of the Web:

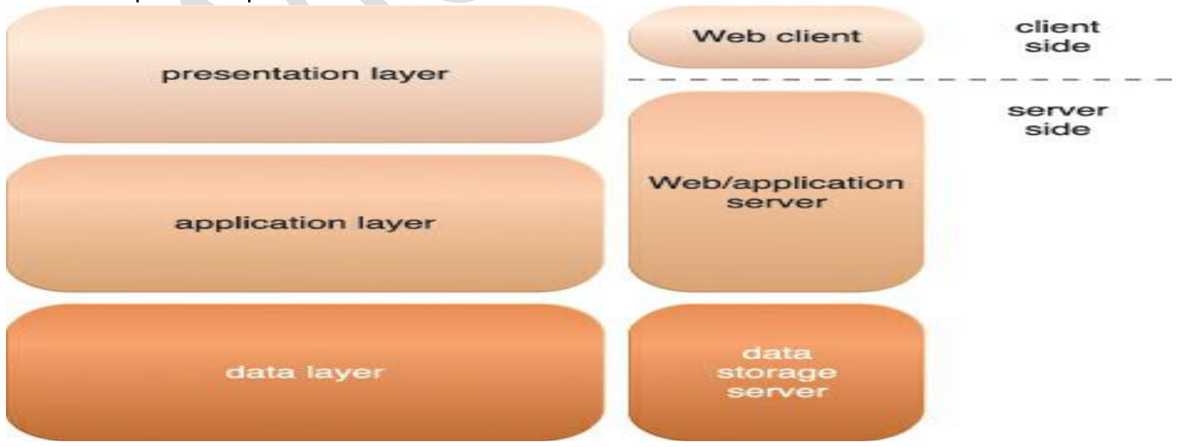
- ***Uniform Resource Locator (URL)*** – A standard syntax used for creating identifiers that point to Web based resources, the URL is often structured using a logical network location.
- ***Hypertext Transfer Protocol (HTTP)*** – This is the primary communications protocol used to exchange content and data throughout the World Wide Web. URLs are typically transmitted via HTTP.
- ***Markup Languages (HTML, XML)*** – Markup languages provide a lightweight means of expressing Web-centric data and metadata. The two primary markup languages are HTML (which is used to express the presentation of Web pages) and XML (which allows for the definition of vocabularies used to associate meaning to Web-based data via metadata).

For example, a Web browser can request to execute an action like read, write, update, or delete on a Web resource on the Internet, and proceed to identify and locate the Web resource through its URL. The request is sent using HTTP to the resource host, which is also identified by a URL. The Web server locates the Web resource and performs the requested operation, which is followed by a response being sent back to the client. The response may be comprised of content that includes HTML and XML statements. Web resources are represented as *hypermedia* as opposed to hypertext, meaning media such as graphics, audio, video, plain text, and URLs can be referenced collectively in a single document. Some types of hypermedia resources cannot be rendered without additional software or Web browser plug-ins

**<u>Web Applications</u>**

A distributed application that uses Web-based technologies (and generally relies on Web browsers for the presentation of user-interfaces) is typically considered a *Web application*. These applications can be found in all kinds of cloud-based environments due to their high accessibility.

Figure presents a common architectural abstraction for Web applications that is based on the basic three tier model. The first tier is called the *presentation layer*, which represents the user-interface. The middle tier is the *application layer* that implements application logic, while the third tier is the *data layer* that is comprised of persistent data stores.



The presentation layer has components on both the client and server-side. Web servers receive client requests and retrieve requested resources directly as static Web content and indirectly as dynamic Web content, which is generated according to the application logic. Web servers interact with application servers in order to execute the requested application logic, which then typically involves interaction with one or more underlying databases.

PaaS ready-made environments enable cloud consumers to develop and deploy Web applications. Typical PaaS offerings have separate instances of the Web server, application server, and data storage server environments.

## # Multitenant Technology

The multitenant application design was created to enable multiple users (tenants) to access the same application logic simultaneously. Each tenant has its own view of the application that it uses, administers, and customizes as a dedicated instance of the software while remaining unaware of other tenants that are using the same application.

Multitenant applications ensure that tenants do not have access to data and configuration information that is not their own. Tenants can individually customize features of the application, such as:

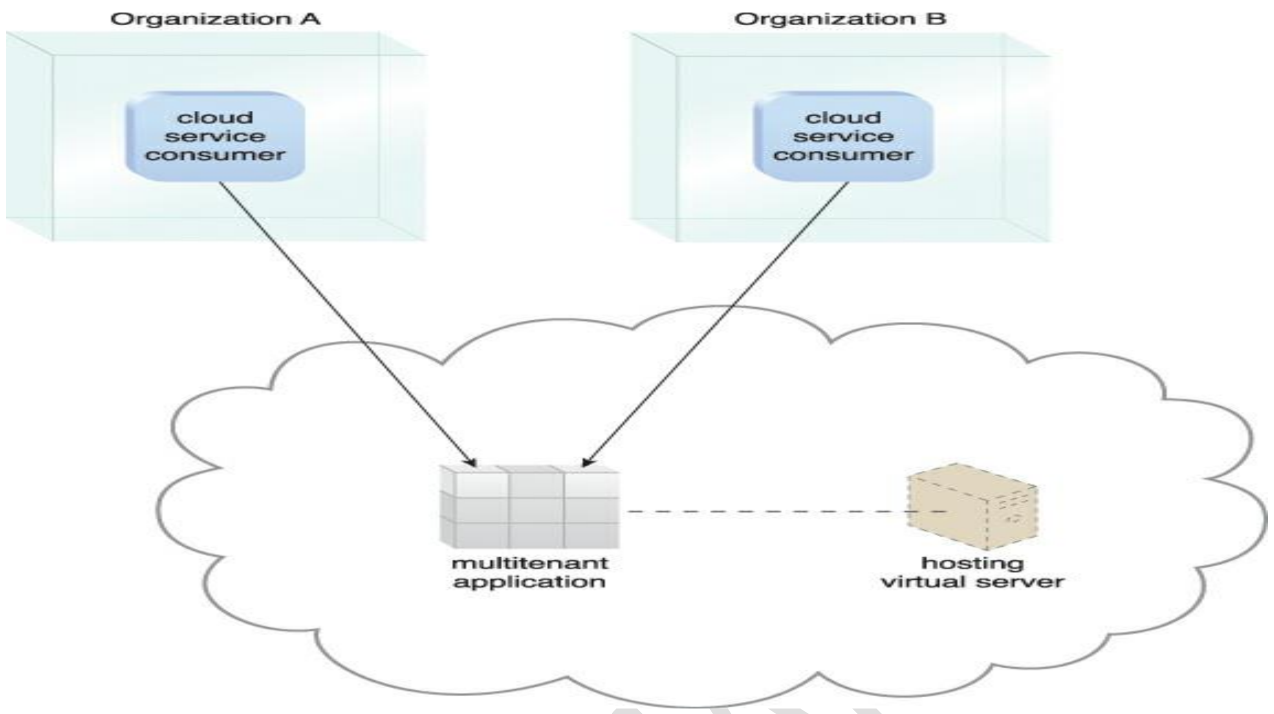
• *User Interface* – Tenants can define a specialized "look and feel" for their application interface.
• *Business Process* – Tenants can customize the rules, logic, and workflows of the business processes that are implemented in the application.
• *Data Model* – Tenants can extend the data schema of the application to include, exclude, or rename fields in the application data structures.
• *Access Control* – Tenants can independently control the access rights for users and groups.

Multitenant application architecture is often significantly more complex than that of single-tenant applications. Multitenant applications need to support the sharing of various artifacts by multiple users (including portals, data schemas, middleware, and databases), while maintaining security levels that segregate individual tenant operational environments.

Common characteristics of multitenant applications include:

• *Usage Isolation* – The usage behavior of one tenant does not affect the application availability and performance of other tenants.
• *Data Security* – Tenants cannot access data that belongs to other tenants.
• *Recovery* – Backup and restore procedures are separately executed for the data of each tenant.
• *Application Upgrades* – Tenants are not negatively affected by the synchronous upgrading of shared software artifacts.
• *Scalability* – The application can scale to accommodate increases in usage by existing tenants and/or increases in the number of tenants.
• *Metered Usage* – Tenants are charged only for the application processing and features that are actually consumed.
• *Data Tier Isolation* – Tenants can have individual databases, tables, and/or schemas isolated from other tenants. Alternatively, databases, tables, and/or schemas can be designed to be intentionally shared by tenants.

A multitenant application that is being concurrently used by two different tenants is illustrated in the figure. This type of application is typical with SaaS implementations.



**Multitenancy vs. Virtualization**
Multitenancy is sometimes mistaken for virtualization because the concept of multiple tenants is similar to the concept of virtualized instances.
The differences lie in what is multiplied within a physical server acting as a host:
- With virtualization: Multiple virtual copies of the server environment can be hosted by a single physical server. Each copy can be provided to different users, can be configured independently, and can contain its own operating systems and applications.
- With multitenancy: A physical or virtual server hosting an application is designed to allow usage by multiple different users. Each user feels as though they have exclusive usage of the application.

## # Data Center Technology
Grouping IT resources in close proximity with one another, rather than having them geographically dispersed, allows for power sharing, higher efficiency in shared IT resource usage, and improved accessibility for IT personnel.
Modern data centers exist as specialized IT infrastructure used to house centralized IT resources, such as servers, databases, networking and telecommunication devices, and software systems.
Data centers are typically comprised of the following technologies and components:

**Virtualization**
Data centers consist of both physical and virtualized IT resources. The physical IT resource layer refers to the facility infrastructure that houses computing/networking systems and equipment, together with hardware systems and their operating systems. The resource abstraction and control of the virtualization layer is comprised of operational and management tools that are often based on virtualization platforms that abstract the physical computing and networking IT resources as virtualized components that are easier to allocate, operate, release, monitor, and control.

**Standardization and Modularity**

Data centers are built upon standardized commodity hardware and designed with modular architectures, aggregating multiple identical building blocks of facility infrastructure and equipment to support scalability, growth, and speedy hardware replacements. Modularity and standardization are key requirements for reducing investment and operational costs as they enable economies of scale for the procurement, acquisition, deployment, operation, and maintenance processes.

**Automation**

Data centers have specialized platforms that automate tasks like provisioning, configuration, patching, and monitoring without supervision. Advances in data center management platforms and tools leverage autonomic computing technologies to enable self-configuration and self-recovery.

**Remote Operation and Management**

Most of the operational and administrative tasks of IT resources in data centers are commanded through the network's remote consoles and management systems. Technical personnel are not required to visit the dedicated rooms that house servers, except to perform highly specific tasks, such as equipment handling and cabling or hardware-level installation and maintenance.

**High Availability**

Since any form of data center outage significantly impacts business continuity for the organizations that use their services, data centers are designed to operate with increasingly higher levels of redundancy to sustain availability. Data centers usually have redundant, uninterruptable power supplies, cabling, and environmental control subsystems in anticipation of system failure, along with communication links and clustered hardware for load balancing.

**Security-Aware Design, Operation, and Management**

Requirements for security, such as physical and logical access controls and data recovery strategies, need to be thorough and comprehensive for data centers, since they are centralized structures that store and process business data.

**Computing Hardware**

Much of the heavy processing in data centers is often executed by standardized commodity servers that have substantial computing power and storage capacity. Several computing hardware technologies are integrated into these modular servers, such as:

- rackmount form factor server design composed of standardized racks with interconnects for power, network, and internal cooling
- support for different hardware processing architectures, such as x86-32bits, x86-64, and RISC
- a power-efficient multi-core CPU architecture that houses hundreds of processing cores in a space as small as a single unit of standardized racks
- redundant and hot-swappable components, such as hard disks, power supplies, network interfaces, and storage controller cards

**Storage Hardware**

Data centers have specialized storage systems that maintain enormous amounts of digital information in order to fulfill considerable storage capacity needs.

Storage systems usually involve the following technologies:

- *Hard Disk Arrays* – These arrays inherently divide and replicate data among multiple physical drives, and increase performance and redundancy by including spare disks.

- *I/O Caching* – This is generally performed through hard disk array controllers, which enhance disk access times and performance by data caching.
- *Hot-Swappable Hard Disks* – These can be safely removed from arrays without requiring prior powering down.
- *Storage Virtualization* – This is realized through the use of virtualized hard disks and storage sharing.
- *Fast Data Replication Mechanisms* – These include *snapshotting*, which is saving a virtual machine's memory into a hypervisor-readable file for future reloading, and *volume cloning*, which is copying virtual or physical hard disk volumes and partitions.

Networked storage devices usually fall into one of the following categories:

- *Storage Area Network (SAN)* – Physical data storage media are connected through a dedicated network and provide block-level data storage access using industry standard protocols, such as the Small Computer System Interface (SCSI).
- *Network-Attached Storage (NAS)* – Hard drive arrays are contained and managed by this dedicated device, which connects through a network and facilitates access to data using file-centric data access protocols like the Network File System (NFS) or Server Message Block (SMB).

NAS, SAN, and other more advanced storage system options provide fault tolerance in many components through controller redundancy, cooling redundancy, and hard disk arrays that use RAID storage technology.

**Network Hardware**

Data centers require extensive network hardware in order to enable multiple levels of connectivity

**Carrier and External Networks Interconnection**

A subsystem related to the internetworking infrastructure, this interconnection is usually comprised of backbone routers that provide routing between external WAN connections and the data center's LAN, as well as perimeter network security devices such as firewalls and VPN gateways.

**LAN Fabric**

The LAN fabric constitutes the internal LAN and provides high-performance and redundant connectivity for all of the data center's network-enabled IT resources. It is often implemented with multiple network switches that facilitate network communications and operate at speeds of up to ten gigabits per second. These advanced network switches can also perform several virtualization functions, such as LAN segregation into VLANs, link aggregation, controlled routing between networks, load balancing, and failover.

**SAN Fabric**

Related to the implementation of storage area networks (SANs) that provide connectivity between servers and storage systems, the SAN fabric is usually implemented with Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), and InfiniBand network switches.

## # Service Technology

Service technology is software that assists customer service teams in achieving customer success that is to provide effective solutions to customers. Formed on the basis of as-a-service cloud delivery model.
Service technologies used are:

- Web Services
- REST Services

- Service agents
- Service middleware

**Web Services**

- Web services include Web Service Description Language (WSDL), XML Schema Definition Language (XML Schema), Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI).
- WSDL- This markup language is used to create a WSDL definition that defines API of a web service.
- XML Schema- describes the structure of an XML document. Messages exchanged by web services must be expressed using XML
- SOAP- used for request and response messages exchanged by web services
- UDDI- This standard regulates service registers in which WSDL definitions can be published as part of a service catalog for discovery purposes.

**REST Services**

- Designed according to a set of constraints that shape the service architecture to emulate the properties of the world wide web.
- REST services share common technical interface called The Uniform Contract established via the use of HTTP methods.
- Six REST design constraints are – Client-Server, Stateless, Cache, Interface/Uniform Contract, Layered System, Code-on-Demand

**Service Agents**

- Are event driven programs designed to intercept messages at runtime. There are active and passive service agents.
- Active service agents perform an action upon intercepting and reading the contents of a message passive on the other hand do not change message contents

**Service Middleware**

- Used primarily to facilitate integration, to sophisticated service middleware platforms designed to accommodate complex service compositions.
- 2 common middleware platforms are Enterprise service bus (ESB) and Orchestration platform

## # Resource Provisioning Techniques

Physical resources can be assigned to the VMs using two types of provisioning approaches like *static* and *dynamic*. In static approach, VMs are created with specific volume of resources and the capacity of the VM does not change in its lifetime. In dynamic approach, the resource capacity per VM can be adjusted dynamically to match work-load fluctuations.

**Static Approach**

- *Static provisioning* is suitable for applications which have predictable and generally unchanging workload demands.
- In this approach, once a VM is created it is expected to run for long time without incurring any further resource allocation decision overhead on the system.
- Resource-allocation decision is taken only once and that too at the beginning when user's application starts running. Thus, this approach provides room for a little more time to take

decision regarding resource allocation since that does not impact negatively on the performance of the system.
- This provisioning approach fails to deal with un-anticipated changes in resource demands. When resource demand crosses the limit specified in SLA document it causes trouble for the consumers.
- Again from provider's point of view, some resources remain unutilized forever since provider arranges for sufficient volume of resources to avoid SLA violation. So this method has drawback from the viewpoint of both provider as well as for consumer.

**Dynamic Approach**

- With *dynamic provisioning*, the resources are allocated and de-allocated as per requirement during run-time. This *on-demand resource provisioning* provides elasticity to the system.
- Providers no more need to keep a certain volume of resources unutilized for each and every system separately, rather they maintain a common resource pool and allocate resources from that when it is required.
- Resources are removed from VMs when they are no more required and returned to the pool. With this dynamic approach, the processes of billing also become as *pay-per-usage* basis.
- Dynamic provisioning technique is more appropriate for cloud computing where application's demand for resources is most likely to change or vary during the execution. But this provisioning approach needs the ability of integrating newly-acquired resources into the existing infrastructure. This gives provisioning elasticity to the system.
- Dynamic provisioning allows system to adapt in changed conditions at the cost of bearing run-time resource allocation decision overhead. This overhead leads some amount of delay in system but this can be minimized by putting upper limit on the complexity of provisioning algorithms.

**Comparison between static and dynamic approaches**

| ***Static Provisioning*** | ***Dynamic Provisioning*** |
|---|---|
| Resource allocation decision can be made once only for an application. | For an application, resource allocation decisions can be made number of times. |
| Resource allocation decision should happen before starting of the application | Decision can be taken even after starting of the application. |
| This approach does not provide scope for elasticity to a system. | It provides scope for elasticity to the system |
| It restricts the scaling. | It enables the scaling. |
| It does not introduce any resource allocation decision overhead. | It incurs resource allocation decision overhead on system. |
| Resource once allotted cannot be returned | Allotted resources can be returned again. |
| It is suitable when load pattern is predictable and more or less unchanging. | It is suitable for applications with varying workload. |
| It introduces under-provisioning and overprovisioning of resource problems. | It resolves the under-provisioning and overprovisioning of resource problems. |

**PART-2**

## # Open Cloud Services

Open-source cloud community generally focusses on private cloud computing arena.

### ->Eucalyptus

Eucalyptus is an open-source Infrastructure-as-a-Service (IaaS) facility for building private or hybrid cloud computing environment. It is a linux-based development that enables cloud features while having installed over distributed computing resources. The name ‘Eucalyptus’ is an acronym for ‘Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems’. Eucalyptus started as a research project at the University of California, United States and the company ‘Eucalyptus Systems’ was formed in the year of 2009 in order to support the commercialization of the Eucalyptus cloud. In the same year, the Ubuntu 9.04 distribution of Linux OS was included Eucalyptus software into it.

Eucalyptus Systems went into an agreement with Amazon during March 2012, which allowed them to make it compatible with Amazon Cloud. This permits transferring of instances between Eucalyptus private cloud and Amazon public cloud making them a combination for building hybrid cloud environment. Such interoperable pairing allows application developers to maintain the private cloud part (deployed as Eucalyptus) as a *sandbox* for executing prominent codes. Eucalyptus also offers a storage *cloud API* emulating as Amazon’s storage service (Amazon S3) API.

### ->OpenNebula

OpenNebula is an open-source Infrastructure-as–a-Service (IaaS) implementation for building public, private and hybrid clouds. Nebula is a Latin word which means as ‘cloud’. OpenNebula started as a research project in the year of 2005 and its first release was made during March 2008. By March 2010, the prime authors of OpenNebula founded C12G Labs with the aim of providing value-added professional services to OpenNebula and the cloud is currently managed by them.

OpenNebula is freely available, subject to the requirements of the *Apache License version 2*. Like Eucalyptus, OpenNebula is also compatible with Amazon cloud. Consequently, the distributions of Ubuntu and Red Hat Enterprise later included OpenNebula integrating into them.

### ->Nimbus

Nimbus is an open-source IaaS cloud solution compatible with Amazon’s cloud services. It was developed at University of Chicago in United States and implemented the Amazon cloud’s APIs. The solution was specifically developed to support the scientific community. The Nimbus project has been created by an international collaboration of open-source contributors and institutions. Nimbus code is licensed under the terms of the Apache License version 2.

### ->OpenStack

OpenStack is another free and open-source IaaS solution. In July 2010, U.S.-based IaaS cloud service provider Rackspace.com and NASA jointly launched the initiative for an open-source cloud solution called ‘OpenStack’ to produce a ubiquitous IaaS solution for public and private clouds. NASA donated some parts of the Nebula Cloud Platform technology that it developed. Since then, more than 200 companies (including AT&T, AMD, Dell, Cisco, HP, IBM, Oracle, Red Hat) have contributed in the project. The project was later taken over and promoted by the OpenStack Foundation, a non-profit organization founded in 2012 for promoting OpenStack solution. All the code of OpenStack is freely available under the Apache 2.0 license.

**->Apache CloudStack**

Apache CloudStack is another open-source IaaS cloud solution. CloudStack was initially developed by Cloud.com, a software company based in California (United States) which was later acquired by Citrix Systems, another USA based software firm, during 2011. By next year, Citrix Systems handed it over to the Apache Software Foundation and soon after this CloudStack made its first stable release. In addition to its own APIs, CloudStack also supported AWS (Amazon Web Services) APIs which facilitated hybrid cloud deployment.

**PART-3**

## # Parallel Computing and Programming Paradigms

A distributed computing system is a set of computational engines connected by a network to achieve a common goal of running a job or an application. A computer cluster or network of workstations is an example of a distributed computing system. Parallel computing is the simultaneous use of more than one computational engine (not necessarily connected via a network) to run a job or an application. It has several advantages.

- From the users' perspective, it decreases application response time;
- from the distributed computing systems' standpoint, it increases throughput and resource utilization

the system should include the following

**Partitioning**

This is applicable to both computation and data as follows:

**Computation partitioning** This splits a given job or a program into smaller tasks. Partitioning greatly depends on correctly identifying portions of the job or program that can be performed concurrently.

**Data partitioning** This splits the input or intermediate data into smaller pieces. Similarly, upon identification of parallelism in the input data, it can also be divided into pieces to be processed on different workers.

**Mapping** This assigns the either smaller parts of a program or the smaller pieces of data to underlying resources. This process aims to appropriately assign such parts or pieces to be run simultaneously on different workers and is usually handled by resource allocators in the system.

**Synchronization** Because different workers may perform different tasks, synchronization and coordination among workers is necessary so that race conditions are prevented and data dependency among different workers is properly managed.

**Communication** Because data dependency is one of the main reasons for communication among workers, communication is always triggered when the intermediate data is sent to workers.

**Scheduling** For a job or program, when the number of computation parts (tasks) or data pieces is more than the number of available workers, a scheduler selects a sequence of tasks or data pieces to be assigned to the workers. It is worth noting that the resource allocator performs the actual mapping of the computation or data pieces to workers, while the scheduler only picks the next part from the queue of unassigned tasks based on a set of rules called the scheduling policy.

### Motivation for Programming Paradigms

Simplicity of writing parallel programs is an important metric for parallel and distributed programming paradigms. Others are

 to improve productivity of programmers

 to decrease programs' time to market
 to leverage underlying resources more efficiently
 to increase system throughput,
 to support higher levels of abstraction

The loose coupling of components in these paradigms makes them suitable for VM implementation and leads to much better fault tolerance and scalability.

**What is a Twister?**

Twister is a lightweight MapReduce runtime we have developed by incorporating these enhancements
 Distinction on static and variable data
 Configurable long running (cacheable) map/reduce tasks
 Pub/sub messaging based communication/data transfers
 Efficient support for Iterative MapReduce computations (extremely faster thanHadoop or Dryad/DryadLINQ)
 Combine phase to collect all reduce outputs
 Data access via local disks
 Lightweight (~5600 lines of Java code)

## **# MapReduce**

- It is a data processing tool.
- Came into existence in order to overcome the disadvantages of traditional computing.
- MapReduce eliminated the idea of centralized systems.
- In MapReduce instead of using a single system, tasks are distributed between multiple systems and parallel these tasks can be updated, monitored, and processed.
- MapReduce is an algorithm that divides the task into smaller parts and assign them to many computers and collects the results from them which when integrated generates data sets.
- MapReduce is by far the most powerful realization of data-intensive cloud computing programming. It is often advocated as an easier-to-use, efficient and reliable replacement for the traditional data intensive programming model for cloud computing.

## **MAPREDUCE PROGRAMMING MODEL**

MapReduce is a software framework for solving many large-scale computing problems. The MapReduce abstraction is inspired by the Map and Reduce functions, which are commonly used in functional languages such as Lisp. The MapReduce system allows users to easily express their computation as map and reduce.

- The map function, written by the user, processes a key/value pair to generate a set of intermediate key/value pairs:

  map (key1, value1) - list (key2, value2)
- The reduce function, also written by the user, merges all intermediate values associated with the same intermediate key:

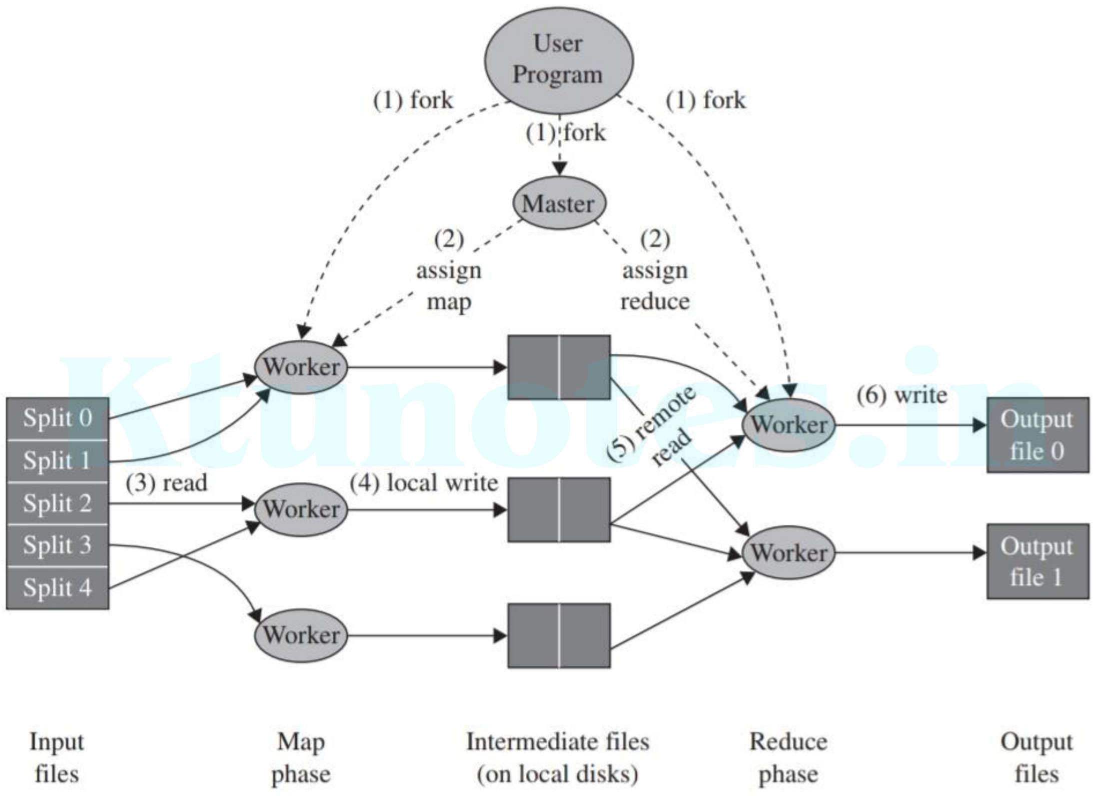
  reduce (key2, list (value2)) - list (value2)

## **MapReduce Features**

- Data-Aware. When the MapReduce-Master node is scheduling the Map tasks for a newly submitted job, it takes in consideration the data location information retrieved from the GFS-Master node.

- Simplicity. As the MapReduce runtime is responsible for parallelization and concurrency control, this allows programmers to easily design parallel and distributed applications.
- Scalability. Increasing the number of nodes (data nodes) in the system will increase the performance of the jobs with potentially only minor losses.
- fault Tolerance and Reliability. The data in the GFS are distributed on clusters with thousands of nodes. Thus any nodes with hardware failures can be handled by simply removing them and installing a new node in their place. Moreover, MapReduce, taking advantage of the replication in GFS, can achieve high reliability by (1) rerunning all the tasks (completed or in progress) when a host node is going off-line, (2) rerunning failed tasks on another node, and (3) launching backup tasks when these tasks are slowing down and causing a bottleneck to the entire job.

## **MapReduce Execution Flow**



The MapReduce library in the user program first splits the input files into M pieces of typically 16 to 64 megabytes (MB) per piece. It then starts many copies of the program on a cluster. One is the "master" and the rest are "workers." The master is responsible for scheduling (assigns the map and reduce tasks to the worker) and monitoring (monitors the task progress and the worker health).

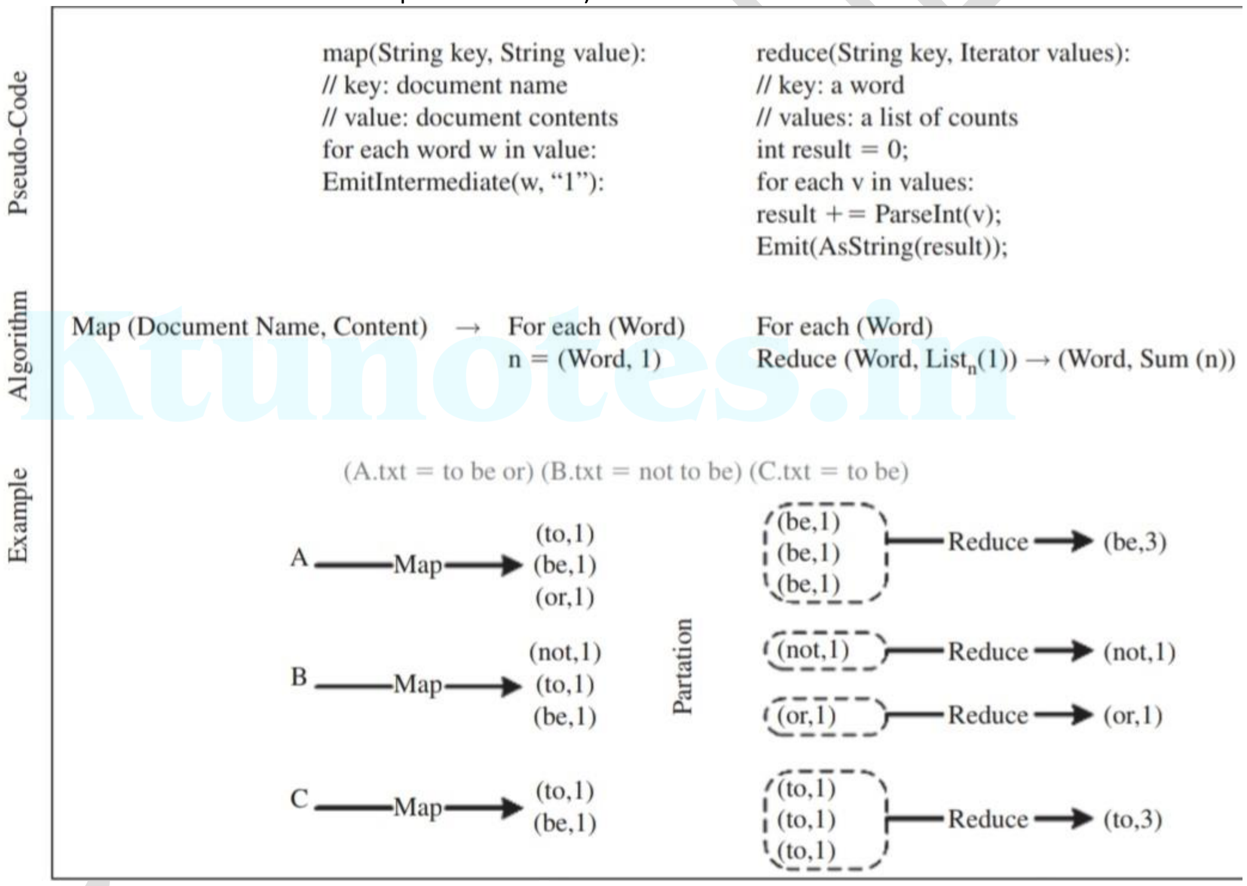
When map tasks arise, the master assigns the task to an idle worker, taking into account the data locality. A worker reads the content of the corresponding input split and emits a key/value pairs to the user-defined Map function. The intermediate key/value pairs produced by the Map function are first buffered in memory and then periodically written to a local disk, partitioned into R sets by the partitioning function.

The master passes the location of these stored pairs to the reduce worker, which reads the buffered data from the map worker using remote procedure calls (RPC). It then sorts the intermediate keys so that all occurrences of the same key are grouped together. For each key, the worker passes the corresponding intermediate value for its entire occurrence to the Reduce function. Finally, the output is available in R output files (one per reduce task).

## The Wordcount Example

As a simple illustration of the Map and Reduce functions, Figure shows the pseudo-code and the algorithm and illustrates the process steps using the widely used "Wordcount" example. The Wordcount application counts the number of occurrences of each word in a large collection of documents.

The steps of the process are briefly described as follows: The input is read (typically from a distributed file system) and broken up into key/value pairs (e.g., the Map function emits a word and its associated count of occurrence, which is just "1"). The pairs are partitioned into groups for processing, and they are sorted according to their key as they arrive for reduction. Finally, the key/value pairs are reduced, once for each unique key in the sorted list, to produce a combined result (e.g., the Reduce function sums all the counts emitted for a particular word).



Pseudo-Code
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1"):
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
Algorithm
Map (Document Name, Content) → For each (Word) n = (Word, 1)
For each (Word) Reduce (Word, List_n(1)) → (Word, Sum (n))
Example
(A.txt = to be or) (B.txt = not to be) (C.txt = to be)
A Map (to,1) (be,1) (or,1)
B Map (not,1) (to,1) (be,1)
C Map (to,1) (be,1)
Partation
(be,1) (be,1) (be,1) Reduce (be,3)
(not,1) Reduce (not,1)
(or,1) Reduce (or,1)
(to,1) (to,1) (to,1) Reduce (to,3)

## # Hadoop Library from Apache

Hadoop is an open source implementation of MapReduce coded in Java by Apache. This implementation uses the *Hadoop Distributed File System (HDFS)* as its underlying layer. It has two fundamental layers: the MapReduce engine and HDFS. The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.

**HDFS:** It is a distributed file system that organizes files and stores their data on a distributed computing system.

**HDFS Architecture**

HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves). To store a file in this architecture, HDFS splits the file into fixed-size blocks and stores them on workers. The mapping of blocks to DataNodes is determined by the NameNode. The NameNode (master) also manages the file system's metadata and namespace. Each DataNode, usually one per node in a cluster, manages the storage attached to the node. Each DataNode is responsible for storing and retrieving its file. Each DataNode is responsible for storing and retrieving its file blocks.

**HDFS Features**

Distributed file systems have special requirements, such as performance, scalability, concurrency control, fault tolerance, and security requirements, to operate efficiently. However, it does not need all the requirements for HDFS, beacuse it only executes specific types of applications. Two important characteristics of HDFS to distinguish it from other generic distributed file systems.

**HDFS Fault Tolerance**

Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception. Hadoop considers the following issues to fulfill reliability requirements of the file system:

**Block replication**

HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.

**Replica placement**
The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS. Although storing replicas on different nodes (DataNodes) located in different racks across the whole cluster provides more reliability, it is sometimes ignored as the cost of communication between two nodes in different racks is relatively high in comparison with that of different nodes located in the same rack. Therefore, sometimes HDFS compromises its reliability to achieve lower communication costs.
**Heartbeat and Blockreport messages**
Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly, while each Blockreport contains a list of all blocks on a DataNode.
**HDFS Operation**
The control flow of the main operations of HDFS on files is the interaction between the user, the NameNode, and the DataNodes.
**Reading a file** To read a file in HDFS, a user sends an “open” request to the NameNode to get the location of file blocks. For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. Upon receiving such information, the user calls the read function to connect to the closest DataNode containing the first block of the file. After the first block is streamed, the established connection is terminated and the same process is repeated for all blocks of the requested file.
**Writing to a file** To write a file in HDFS, a user sends a “create” request to the NameNode to create a new file in the file system namespace. If the file does not exist, the NameNode notifies the user and allows him to start writing data to the file by calling the write function. The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode. Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.The procedure will repeat for all the blocks of a file.

## # Apache Spark
Apache Spark is an open-source cluster computing framework. Its primary purpose is to handle the real-time generated data.
Spark was built on the top of the Hadoop MapReduce. It was optimized to run in memory whereas alternative approaches like Hadoop's MapReduce writes data to and from computer hard drives. So, Spark process the data much quicker than other alternatives.

### Features of Apache Spark
**Fast** - It provides high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.
**Easy to Use** - It facilitates to write the application in Java, Scala, Python, R, and SQL. It also provides more than 80 high-level operators.
**Generality** - It provides a collection of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.
**Lightweight** - It is a light unified analytics engine which is used for large scale data processing.
**Runs Everywhere** - It can easily run on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud.

## <u>Usage of Spark</u>

**Data integration:** The data generated by systems are not consistent enough to combine for analysis. To fetch consistent data from systems we can use processes like Extract, transform, and load (ETL). Spark is used to reduce the cost and time required for this ETL process.

**Stream processing:** It is always difficult to handle the real-time generated data such as log files. Spark is capable enough to operate streams of data and refuses potentially fraudulent operations.

**Machine learning:** Machine learning approaches become more feasible and increasingly accurate due to enhancement in the volume of data. As spark is capable of storing data in memory and can run repeated queries quickly, it makes it easy to work on machine learning algorithms.

**Interactive analytics:** Spark is able to generate the respond rapidly. So, instead of running pre-defined queries, we can handle the data interactively.