USN :

CMR Institute of Technology, Bangalore
III - INTERNAL ASSESSMENT

Semester: 4-CBCS 2018
Subject: CLOUD COMPUTING (18MCA444)
Faculty: Ms Moumita Roy

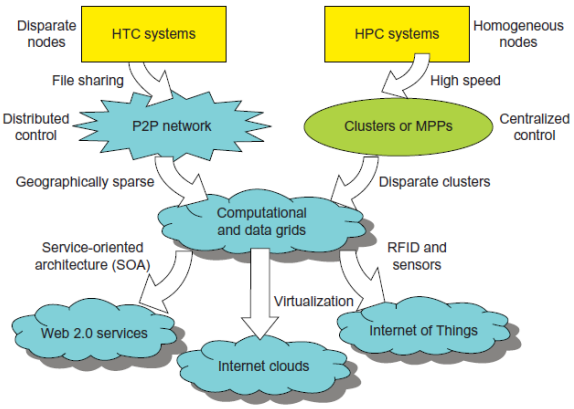
Date: 15 Jun 2021
Time: 02:00 PM - 03:30 PM
Max Marks: 50

PART A					
Answer any1 question/s					
Q.No		Marks	CO	PO	BT/CL
1	Discuss HPC and HTC in details.	10	CO1	PO1,PO2	L1
2	Give the architecture of P2P systems. What are the major categories of P2P network?	10	CO1	PO1,PO2	L2
PART B					
Answer any1 question/s					
Q.No		Marks	CO	PO	BT/CL
3	Discuss the performance metrics and dimensions of scalability for parallel and distributed systems.	10	CO2	PO2	L2
4	Write and explain Amdahl's law and Gustafson's Law	10	CO2	PO2	L3
PART C					
Answer any1 question/s					
Q.No		Marks	CO	PO	BT/CL
5	What is VMM? Explain XEN architecture with suitable diagram.	10	CO3	PO2,PO3	L2
6	Explain para virtualization technique.	10	CO3	PO2,PO3	L2
PART D					
Answer any1 question/s					
Q.No		Marks	CO	PO	BT/CL
7	List and explain data centre management issues.	10	CO4	PO1,PO2	L2
8	Discuss five public cloud offerings of PaaS.	10	CO4	PO2	L2
PART E					
Answer any1 question/s					
Q.No		Marks	CO	PO	BT/CL
9	Explain Google File System.	10	CO4	PO2,PO3	L2
10	Explain SQL Azure and Azure Table.	10	CO4	PO2,PO3	L2

Part 1

1. Discuss HPC and HTC in details.

Ans:



According to the specific nature of the problem, a variety of categories for task computing have been proposed over time. These categories do not enforce any specific application model but provide an overall view of the characteristics of the problems. They implicitly impose requirements on the

infrastructure and the middleware. Applications falling into this category are high-performance computing (HPC), high-throughput computing (HTC), and many-task computing (MTC).

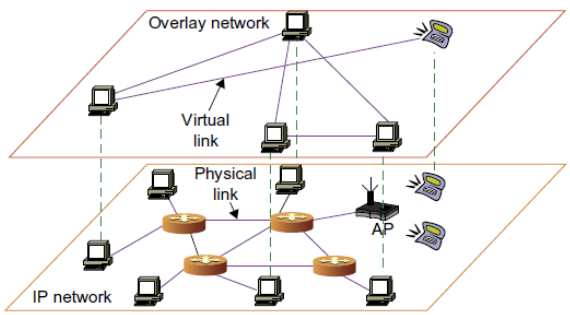
According to the specific nature of the problem, a variety of categories for task computing have been proposed over time. These categories do not enforce any specific application model but provide an overall view of the characteristics of the problems. They implicitly impose requirements on the infrastructure and the middleware. Applications falling into this category are high-performance computing (HPC), high-throughput computing (HTC), and many-task computing (MTC).

High-throughput computing (HTC) is the use of distributed computing facilities for applications requiring large computing power over a long period of time. HTC systems need to be robust and to reliably operate over a long time scale. Traditionally, computing grids composed of heterogeneous resources (clusters, workstations, and volunteer desktop machines) have been used to support HTC. The general profile of HTC applications is that they are made up of a large number of tasks of which the execution can last for a considerable amount of time (i.e., weeks or months). Classical 7.1 Task computing 213examples of such applications are scientific simulations or statistical analyses. It is quite common to have independent tasks that can be scheduled in distributed resources because they do not need to communicate. HTC systems measure their performance in terms of jobs completed per month.

2. Give the architecture of P2P systems. What are the major categories of P2P network?

In a P2P system, every node acts as both a client and a server, providing part of the system resources. Peer machines are simply client computers connected to the Internet. All client machines act autonomously to join or leave the system freely. This implies that no master-slave relationship exists among the peers. No central coordination or central database is needed. In other words, no peer machine has a global view of the entire P2P system. The system is self-organizing with distributed control.

The figure below shows the architecture of a P2P network at two abstraction levels. Initially, the peers are totally unrelated. Each peer machine joins or leaves the P2P network voluntarily. Only the participating peers form the physical network at ar[illegible] es not use a dedicated interconnection network. The ph[illegible] arious Internet domains randomly using the TCP/IP and N[illegible] nd topology dynamically due to the free membership in t[illegible]



Overlay Networks

Data items or files are distributed in the participating peers. Based on communication or file-sharing needs, the peer IDs form an overlay network at the logical level. This overlay is a virtual network formed by mapping each physical machine with its ID, logically, through a virtual mapping as shown in the figure. When a new peer joins the system, its peer ID is added as a node in the overlay network. When an existing peer leaves the system, its peer ID is removed from the overlay network automatically. Therefore, it is the P2P overlay network that characterizes the logical connectivity among the peers.

There are two types of overlay networks: unstructured and structured. An unstructured overlay network is characterized by a random graph. There is no fixed route to send messages or files among the nodes. Often, flooding is applied to send a query to all nodes in an unstructured overlay, thus resulting in heavy network traffic and nondeterministic search results. Structured overlay net-works follow certain connectivity topology and rules for inserting and removing nodes (peer IDs) from the overlay graph. Routing mechanisms are developed to take advantage of the structured overlays.

System Features	Distributed File Sharing	Collaborative Platform	Distributed P2P Computing	P2P Platform
Attractive Applications	Content distribution of MP3 music, video, open software, etc.	Instant messaging, Collaborative design and gaming	Scientific exploration and social networking	Open networks for public resources
Operational Problems	Loose security and serious online copyright violations	Lack of trust, disturbed by spam, privacy, and peer collusion	Security holes, selfish partners, and peer collusion	Lack of standards or protection protocols
Example Systems	Gnutella, Napster, eMule, BitTorrent, Aimster, KaZaA, etc.	ICQ, AIM, Groove, Magi, Multiplayer Games, Skype, etc.	SETI@home, Geonome@home, etc.	JXTA, .NET, FightingAid@home, etc.

Part 2

3. Discuss the performance metrics and dimensions of scalability for parallel and distributed systems.

Here, we are mostly interested in **metrics** that measure the **performance** of **parallel** applications. Speedup is a measure of **performance**. It measures the ratio between the sequential execution time and the **parallel** execution time. Efficiency is a measure of the usage of the computational capacity.

how much benefit an application or a software system can gain from parallelism. In particular, what we need to keep in mind is that parallelism is used to perform multiple activities together so that the system can increase its throughput or its speed. But the relations that control the increment of speed are not linear. For example, for a given n processors, the user expects speed to be increased by n times. This is an ideal situation, but it rarely happens because of the communication overhead.

Here are two important guidelines to take into account:

- Speed of computation is proportional to the square root of system cost; they never increase linearly. Therefore, the faster a system becomes, the more expensive it is to increase its speed (Figure 2.8).
- Speed by a parallel computer increases as the logarithm of the number of processors (i.e., $y = k^*log(N)$). This concept is shown in Figure 2.9.

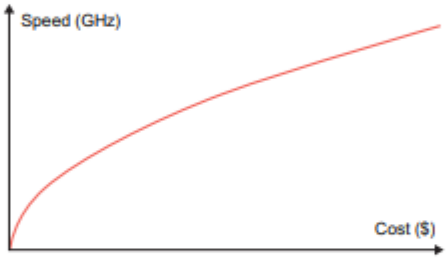


FIGURE 2.8

Cost versus speed.

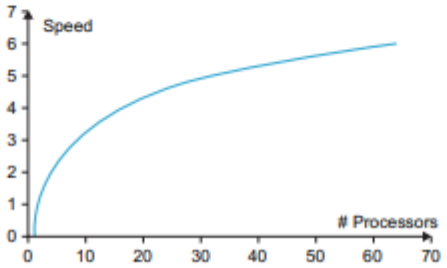


FIGURE 2.9

Number processors versus speed.

Scalability is an important indicator in **distributed computing** and **parallel computing**. It describes the ability of the **system** to dynamically adjust its own **computing** performance by changing available **computing** resources and scheduling methods.

A **parallel** architecture is said to be **scalable** if it can be expanded (reduced) to a larger (smaller) **system** with a linear increase (decrease) in its performance (cost). ... **Scalability** is used as a measure of the **system's** ability to provide increased performance, for example, speed as its size is increased.

We can measure the **scalability** of a **distributed system** in three main ways: size **scalability**, geographical **scalability**, and administrative **scalability**. These three **forms** of measuring how a **system** scales are often referred to as **scalability** dimensions.

4. Write and explain Amdahl's law and Gustafson's Law.

Consider the execution of a given program on a uniprocessor workstation with a total execution time of T minutes. Now, let's say the program has been parallelized or partitioned for parallel execution on a cluster of many processing nodes. Assume that a fraction α of the code must be executed sequentially, called the sequential bottleneck. Therefore, (1 − α) of the code can be compiled for parallel execution by n processors. The

total execution time of the program is calculated by T + (1 − α)T/n, where the first term is the sequential execution time on a single processor and the second term is the parallel execution time on n processing nodes.

All system or communication overhead is ignored here. The I/O time or exception handling time is also not included in the following speedup analysis. Amdahl's Law states that the speedup factor of using the n-processor system over the use of a single processor is expressed by:

$$\text{Speedup} = S = T/[\alpha T + (1-\alpha)T/n] = 1/[\alpha + (1-\alpha)/n]$$

The maximum speed up of n is achieved only if the sequential bottleneck α is reduced to zero or the code is fully parallelizable with α = 0. As the cluster becomes sufficiently large, that is, n → ∞, S approaches 1/α, an upper bound on the speedup S. Surprisingly, this upper bound is independent of the cluster size n. The sequential bottleneck is the portion of the code that cannot be parallelized. For example, the maximum speedup achieved is 4, if α = 0.25 or 1 − α = 0.75, even if one uses hundreds of processors. Amdahl's law teaches us that we should make the sequential bottle-neck as small as possible. Increasing the cluster size alone may not result in a good speed up in this case.

Problem with Fixed Workload

In Amdahl's law, we have assumed the same amount of workload for both sequential and parallel execution of the program with a fixed problem size or data set. This was called fixed-workload speedup by Hwang and Xu. To execute a fixed workload on n processors, parallel processing may lead to a system efficiency defined as follows:

$$E = S/n = 1/[\alpha n + 1 - \alpha]$$

Very often the system efficiency is rather low, especially when the cluster size is very large. To execute the aforementioned program on a cluster with n = 256 nodes, extremely low efficiency E = 1/[0.25 × 256 + 0.75] = 1.5% is observed. This is because only a few processors (say, 4) are kept busy, while the majority of the nodes are left idling.

Gustafson's Law

To achieve higher efficiency when using a large cluster, we must consider scaling the problem size to match the cluster capability. This leads to the following speedup law proposed by John Gustafson (1988), referred as scaled-workload speedup in [14]. Let W be the workload in a given program. When using an n-processor system, the user scales the workload to W′ = αW + (1 − α)nW. Note that only the parallelizable portion of the workload is scaled n times in the second term. This scaled workload W′ is essentially the sequential execution time on a single processor. The parallel execution time of a scaled workload W′ on n processors is defined by a scaled-workload speedup as follows:

$$S' = W'/W = [\alpha W + (1-\alpha)nW]/W = \alpha + (1-\alpha)n$$

This speedup is known as Gustafson's law. By fixing the parallel execution time at level W, the following efficiency expression is obtained:

$$E' = S'/n = \alpha/n + (1-\alpha)$$

For the preceding program with a scaled workload, we can improve the efficiency of using a 256-node cluster to E' = 0.25/256 + 0.75 = 0.751. One should apply Amdahl's law and Gustafson's law under different workload conditions. For a fixed workload, users should apply Amdahl's law. To solve scaled problems, users should apply Gustafson's law.

Part 3

5. What is VMM? Explain XEN architecture with suitable diagram.

A fundamental element of hardware virtualization is the hypervisor, or virtual machine manager (VMM). It recreates a hardware environment in which guest operating systems are installed. There are two major types of hypervisor: Type I and Type II.

Xen is an open-source initiative implementing a virtualization platform based on paravirtualization. Initially developed by a group of researchers at the University of Cambridge in the United Kingdom, Xen now has a large open-source community backing it. Citrix also offers it as a commercial solution, XenSource. Xen-based technology is used for either desktop virtualization or server virtualization, and recently it has also been used to provide cloud computing solutions by means of Xen Cloud Platform (XCP). At the basis of all these solutions is the Xen Hypervisor, which constitutes the core technology of Xen. Recently Xen has been advanced to support full virtualization using hardware-assisted virtualization. Xen is the most popular implementation of paravirtualization, which, in contrast with full virtualization, allows high-performance execution of guest operating systems. This is made possible by eliminating the performance loss while executing instructions that require special management. This is done by modifying portions of the guest operating systems run by Xen with reference to the execution of such instructions. Therefore it is not a transparent solution for implementing virtualization. This is particularly true for x86, which is the most popular architecture on commodity machines and servers. A Xen-based system is managed by the Xen hypervisor, which runs in the highest privileged mode and controls the access of guest operating system to the underlying hardware. Guest operating systems are executed within domains, which represent virtual machine instances. Moreover, specific control software, which has privileged access to the host and controls all the other guest operating systems, is executed in a special domain called Domain 0. This is the first one that is loaded once the virtual machine manager has completely booted, and it hosts a HyperText Transfer Protocol (HTTP) server that serves requests for virtual machine creation, configuration, and termination. This component constitutes the embryonic version of a distributed virtual machine manager, which is an essential component of cloud computing systems providing Infrastructure-as-a-Service (IaaS) solutions. Many of the x86 implementations support four different security levels, called rings, where Ring 0 represent the level with the highest privileges and Ring 3 the level with the lowest ones. Almost all the most popular operating systems, except OS/2, utilize only two levels: Ring 0 for the kernel code, and Ring 3 for user application and nonprivileged OS code. This provides the opportunity for Xen to implement virtualization by executing the hypervisor in Ring 0, Domain 0, and all the other domains running guest operating systems—generally referred to as Domain U—in Ring 1, while the user applications are run in Ring 3. This allows Xen to maintain the ABI unchanged, thus allowing an easy switch to Xen-virtualized solutions from an application point of view. Because of the structure of the x86 instruction set, some instructions allow code executing in Ring 3 to jump into Ring 0 (kernel mode). Such operation is performed at the hardware level and therefore within a virtualized environment will result in a trap or silent fault, thus preventing the normal operations of the guest operating system, since this is now running in Ring 1. This condition is generally triggered by a subset of the system calls. To avoid this situation, operating systems need to be changed in their implementation, and the sensitive system calls need to be reimplemented with hypercalls, which are specific calls exposed by the virtual machine interface of

Xen. With the use of hypercalls, the Xen hypervisor is able to catch the execution of all the sensitive instructions, manage them, and return the control to the guest operating system by means of a supplied handler. Paravirtualization needs the operating system codebase to be modified, and hence not all operating systems can be used as guests in a Xen-based environment. More precisely, this condition holds in a scenario where it is not possible to leverage hardware-assisted virtualization, which allows running the hypervisor in Ring -1 and the guest operating system in Ring 0. Therefore, Xen exhibits some limitations in the case of legacy hardware and legacy operating systems. In fact, these cannot be modified to be run in Ring 1 safely since their codebase is not accessible and, at the same time, the underlying hardware does not provide any support to run the hypervisor in a more privileged mode than Ring 0. Open-source operating systems such as Linux can be easily modified, since their code is publicly available and Xen provides full support for their virtualization, whereas components of the Windows family are generally not supported by Xen unless hardware-assisted virtualization is available. It can be observed that the problem is now becoming less and less crucial since both new releases of operating systems are designed to be virtualization aware and the new hardware supports x86 virtualization.

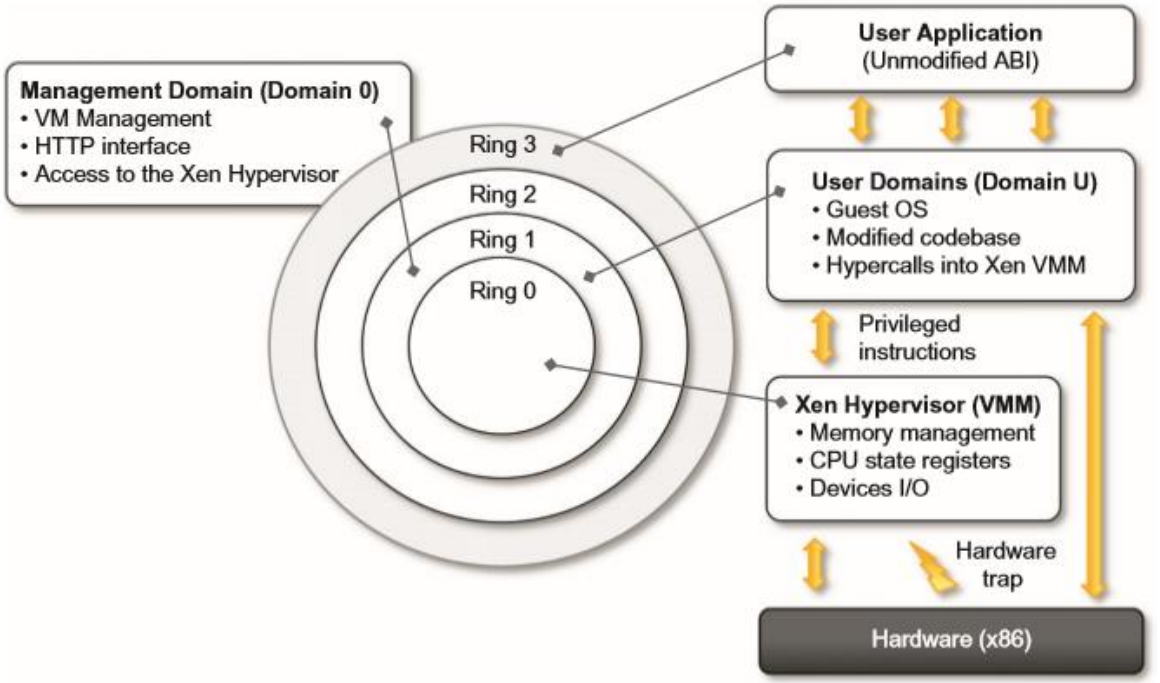


FIGURE 3.11

Xen architecture and guest OS management.

6. Explain para virtualization technique.

Paravirtualization. This is a not-transparent virtualization solution that allows implementing thin virtual machine managers. Paravirtualization techniques expose a software interface to the virtual machine that is slightly modified from the host and, as a consequence, guests need to be modified. The aim of paravirtualization is to provide the capability to demand the execution of performance-critical operations directly on the host, thus preventing performance losses that would otherwise be experienced in managed execution. This allows a simpler implementation of virtual machine managers that have to simply transfer the execution of these operations, which were hard to virtualize, directly to the host. To take advantage of such an opportunity, guest operating systems 3.3 Taxonomy of virtualization techniques 85need to be modified and explicitly ported by remapping the performance-critical operations through the virtual machine software interface. This is possible

when the source code of the operating system is available, and this is the reason that paravirtualization was mostly explored in the opensource and academic environment. Whereas this technique was initially applied in the IBM VM operating system families, the term paravirtualization was introduced in literature in the Denali project [24] at the University of Washington. This technique has been successfully used by Xen for providing virtualization solutions for Linux-based operating systems specifically ported to run on Xen hypervisors. Operating systems that cannot be ported can still take advantage of paravirtualization by using ad hoc device drivers that remap the execution of critical instructions to the paravirtualization APIs exposed by the hypervisor. Xen provides this solution for running Windows-based operating systems on x86 architectures. Other solutions using paravirtualization include VMWare, Parallels, and some solutions for embedded and real-time environments such as TRANGO, Wind River, and XtratuM.

Part 4

7. List and explain data centre management issues.

The demand for new on-demand technology services and the cost of deploying and managing them continue to skyrocket. In order to manage deepening costs, complex deployments, and ensure reliability and uptime, data center managers need access to information and data that isn't always readily available. These are the top five challenges data center managers face.

Challenge 1: Maintaining Availability and Uptime

If you're using spreadsheets or homegrown tools to manage your server information, you probably already know the information stored can be outdated, inaccurate, or incomplete. This can prove challenging when unplanned downtime requires troubleshooting, or when attempting to map the power chain.

Challenge 2: Improving Utilization of Capacity (Power, Cooling, Space)

In a dynamic data center it is almost impossible to understand how much space, power, and cooling you have; predict when will you run out, which server is the best for a new services, and just how much power is needed to ensure uptime and availability.

Challenge 3: Reporting Reduced Operating Expenses

It's not enough to implement solutions that reduce operating expenses, you also have to prove it. According to Uptime institute, "Going forward, enterprise data center managers will need to be able to collect cost and performance data, and articulate their value to the business in order to compete with third party offerings."

Challenge 4: Managing Energy Usage & Costs

According to a NY Times article, "Most data centers, by design, consume vast amounts of energy in an incongruously wasteful manner…online companies typically run their facilities at maximum capacity around the clock…as a result, data centers can waste 90 percent or more of the electricity they pull off the grid."

Challenge 5: Improving Staff Productivity

Non-automated or manual systems require facilities and IT staff to spend an extraordinary amount of time logging activities into spreadsheets. This takes away time that can be spent making strategic decisions for the data center and improving service offerings.

8. Discuss five public cloud offerings of PaaS.

Platform-as-a-Service (PaaS) solutions provide a development and deployment platform for running applications in the cloud. They constitute the middleware on top of which applications are built. Application management is the core functionality of the middleware. PaaS implementations provide applications with a runtime environment and do not expose any service for managing the underlying infrastructure. They automate the process of deploying applications to the infrastructure, configuring application components, provisioning and configuring supporting technologies such as load balancers and databases, and managing system change based on policies set by the user. Developers design their systems in terms of applications and are not concerned with hardware (physical or virtual), operating systems, and other low-level services. The core middleware is in charge of managing the resources and scaling applications on demand or automatically, according to the commitments made with users. From a user point of view, the core middleware exposes interfaces that allow programming and deploying applications on the cloud. These can be in the form of a Web-based interface or in the form of programming APIs and libraries. The specific development model decided for applications determines the interface exposed to the user. Some implementations provide a completely Web-based interface hosted in the cloud and offering a variety of services. It is possible to find integrated developed environments based on 4GL and visual programming concepts, or rapid prototyping environments where applications are built by assembling mash-ups and user-defined components and successively customized. Other implementations of the PaaS model provide a complete object model for representing an application and provide a programming language-based approach. This approach generally offers more flexibility and opportunities but incurs longer development cycles. Developers generally have the full power of programming languages such as Java, .NET, Python, or Ruby, with some restrictions to provide better scalability and security. In this case the traditional development environments can be used to design and develop applications, which are then deployed on the cloud by using the APIs exposed by the PaaS provider. Specific components can be offered together with the development libraries for better exploiting the services offered by the PaaS environment. Sometimes a local runtime environment that simulates the conditions of the cloud is given to users for testing their applications before deployment. This environment can be restricted in terms of features, and it is generally not optimized for scaling. PaaS solutions can offer middleware for developing applications together with the infrastructure or simply provide users with the software that is installed on the user premises. In the first case, the PaaS provider also owns large datacenters where applications are executed; in the second case, referred to in this book as Pure PaaS, the middleware constitutes the core value of the offering. It is also possible to have vendors that deliver both middleware and infrastructure and ship only the middleware for private installations.

Part 5

9. Explain Google File System.

Google File System is a proprietary distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware. The last version of Google File System code-named Colossus was released in 2010.

GFS is enhanced for Google's core data storage and usage needs (primarily the search engine), which can generate enormous amounts of data that must be retained; Google File System grew out of an earlier Google effort, "BigFiles", developed by Larry Page and Sergey Brin in the early days of Google, while it was still located in Stanford. Files are divided into fixed-size *chunks* of 64 megabytes, similar to clusters or sectors in regular file systems, which are only extremely rarely overwritten, or shrunk; files are usually appended to or read. It is also designed and

optimized to run on Google's computing clusters, dense nodes which consist of cheap "commodity" computers, which means precautions must be taken against the high failure rate of individual nodes and the subsequent data loss. Other design decisions select for high data throughputs, even when it comes at the cost of latency.

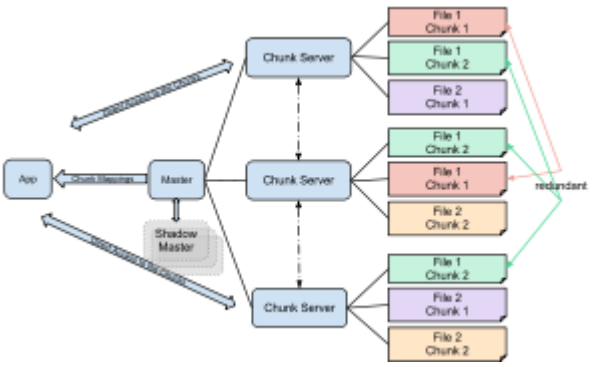
A GFS cluster consists of multiple nodes. These nodes are divided into two types: one *Master* node and multiple *Chunkservers*. Each file is divided into fixed-size chunks. Chunkservers store these chunks. Each chunk is assigned a globally unique 64-bit label by the master node at the time of creation, and logical mappings of files to constituent chunks are maintained. Each chunk is replicated several times throughout the network. At default, it is replicated three times, but this is configurable. Files which are in high demand may have a higher replication factor, while files for which the application client uses strict storage optimizations may be replicated less than three times - in order to cope with quick garbage cleaning policies.

The Master server does not usually store the actual chunks, but rather all the metadata associated with the chunks, such as the tables mapping the 64-bit labels to chunk locations and the files they make up (mapping from files to chunks), the locations of the copies of the chunks, what processes are reading or writing to a particular chunk, or taking a "snapshot" of the chunk pursuant to replicate it (usually at the instigation of the Master server, when, due to node failures, the number of copies of a chunk has fallen beneath the set number). All this metadata is kept current by the Master server periodically receiving updates from each chunk server ("Heart-beat messages").

Permissions for modifications are handled by a system of time-limited, expiring "leases", where the Master server grants permission to a process for a finite period of time during which no other process will be granted permission by the Master server to modify the chunk. The modifying chunkserver, which is always the primary chunk holder, then propagates the changes to the chunkservers with the backup copies. The changes are not saved until all chunkservers acknowledge, thus guaranteeing the completion and atomicity of the operation.

Programs access the chunks by first querying the Master server for the locations of the desired chunks; if the chunks are not being operated on (i.e. no outstanding leases exist), the Master replies with the locations, and the program then contacts and receives the data from the chunkserver directly (similar to Kazaa and its supernodes).

Unlike most other file systems, GFS is not implemented in the kernel of an operating system, but is instead provided as a userspace library.



10. Explain SQL Azure and Azure Table.

Microsoft Azure SQL Database is a managed cloud database (PaaS) provided as part of Microsoft Azure.

A cloud database is a database that runs on a cloud computing platform, and access to it is provided as a service. Managed database services take care of scalability, backup, and high availability of the database. Azure SQL Database is a managed database service which is different from AWS RDS which is a container service. Microsoft Azure SQL Database includes built-in intelligence that learns app patterns and adapts to maximize performance, reliability, and data protection. It was originally announced in 2009 and released in 2010.

Key capabilities include:

- Continuous learning of your unique app patterns, adaptive performance tuning, and automatic improvements to reliability and data protection
- Scaling as needed, with virtually no app downtime
- Management and monitoring of multitenant apps with isolation benefits of one-customer-per-database
- Leverage open-source tools like cheetah, sql-cli, VS Code and Microsoft tools like Visual Studio and SQL Server Management Studio, Azure Management Portal, PowerShell, and REST APIs
- Data protection with encryption, authentication, limiting user access to the appropriate subset of the data, continuous monitoring and auditing to help detect potential threats and provide a record of critical events in case of a breach

Azure Table storage is a service that stores non-relational structured data (also known as structured NoSQL data) in the cloud, providing a key/attribute store with a schemaless design. Because Table storage is schemaless, it's easy to adapt your data as the needs of your application evolve. Access to Table storage data is fast and cost-effective for many types of applications, and is typically lower in cost than traditional SQL for similar volumes of data.

You can use Table storage to store flexible datasets like user data for web applications, address books, device information, or other types of metadata your service requires. You can store any number of entities in a table, and a storage account may contain any number of tables, up to the capacity limit of the storage account.

What is Table storage

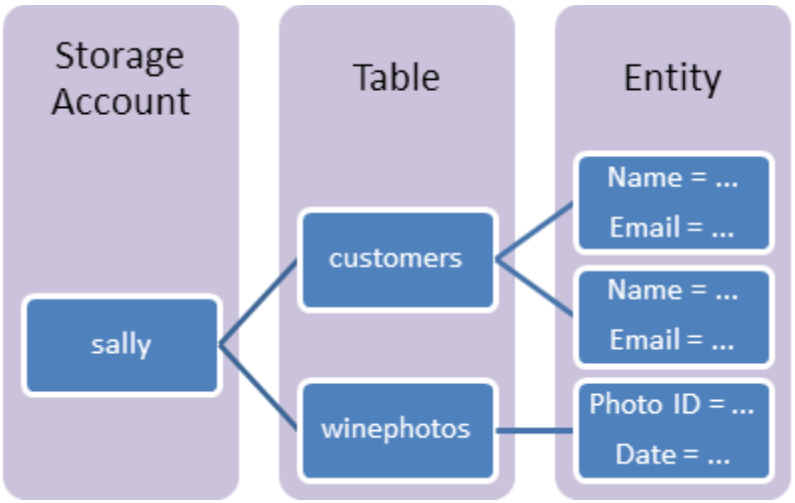
Azure Table storage stores large amounts of structured data. The service is a NoSQL datastore which accepts authenticated calls from inside and outside the Azure cloud. Azure tables are ideal for storing structured, non-relational data. Common uses of Table storage include:

- Storing TBs of structured data capable of serving web scale applications
- Storing datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
- Quickly querying data using a clustered index
- Accessing data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries

You can use Table storage to store and query huge sets of structured, non-relational data, and your tables will scale as demand increases.

Table storage concepts

Table storage contains the following components:



- **URL format:** Azure Table Storage accounts use this format: `http://<storage account>.table.core.windows.net/<table>`

  Azure Cosmos DB Table API accounts use this format: `http://<storage account>.table.cosmosdb.azure.com/<table>`

  You can address Azure tables directly using this address with the OData protocol.

- **Accounts:** All access to Azure Storage is done through a storage account.
- **Table**: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties.
- **Entity**: An entity is a set of properties, similar to a database row. An entity in Azure Storage can be up to 1MB in size. An entity in Azure Cosmos DB can be up to 2MB in size.
- **Properties**: A property is a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has three system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition.