COMP5349 Notes

GENERAL INFORMATION

Topic	Type	Marks	Due Date
Lab practice	Individual	30%	Continuous
Project	Individual	20%	Week 12
Final Exam	Individual	50%	Exam Period

EXAM INFORMATION

Exam is for 2 hours.
Closed book exam with no resources allowed. Only blank sheets are allowed.
Exam is for 100 marks and the distribution is given below:

Question	Description	Marks
1-4	Short answer conceptual questions not based on scenario from all over. (NO GUEST LECTURE, NO AMAZON LAMBDA)	25
5	Given an IAM policy, explain the permissions defined in the policy. • Challenge labs will make it easy.	8
6	Given a diagram of VPC, Subnets, SGs, EC2 instances, route table and SG rules answer several questions based on this most of which are related with network access. • Challenge labs will make it easy. • Definitions of each of the stuff, VPC, Subnets, SG Rules, etc.	12
7	Given a scenario of a Spanner deployment (architecture, sample tables, sample transactions given) answer questions on spanner data placement and replication, TrueTime API, transaction coordination, commit time consensus using Paxos, etc. • Given scenario of Spanner deployment answer questions • Sub-questions are: • How to use Paxos to commit timestamps.	25
8	Given a dynamo cluster with number of nodes and other settings, answer questions on data distribution, vector clock and read/write handling. • Sub-questions of data distributions, vector clock, read/write handling	15
9	Given a scenario of Aurora database, answer questions of storage node distribution and various LSNs. • Describe the situation based on the scenario.	15

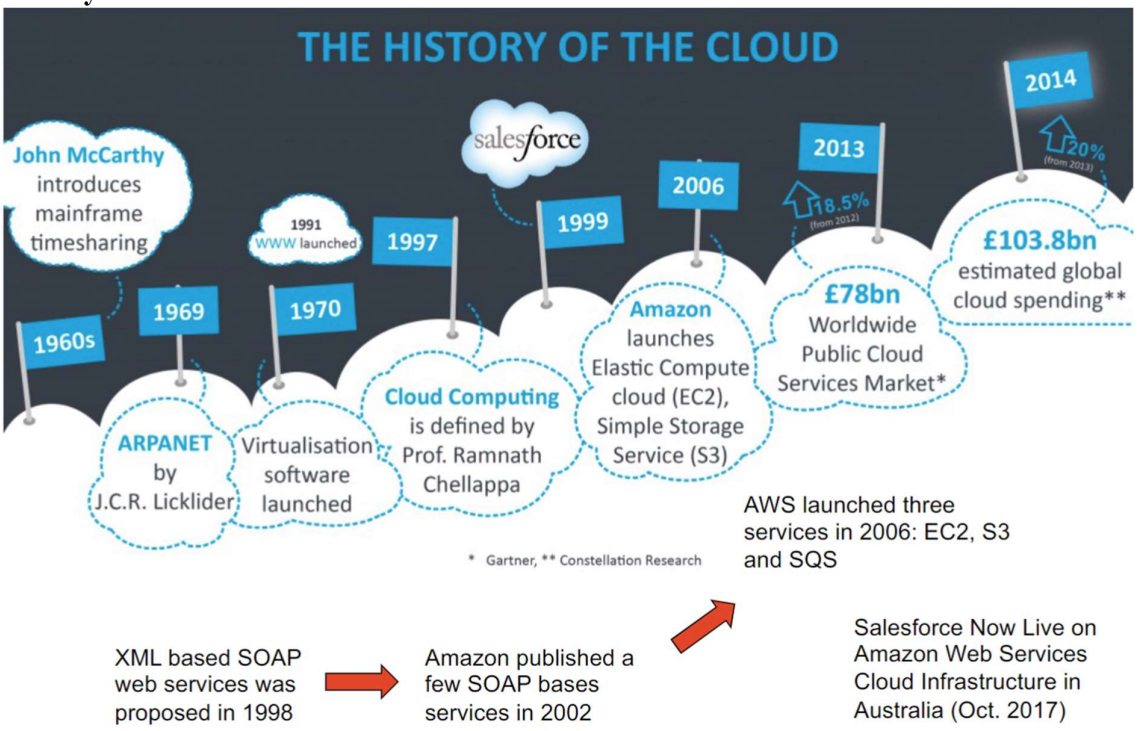
WEEK 1

Introduction

Cloud computing can be seen as a way of renting/sharing IT resources through the internet. There are major cloud computing providers other than Amazon Web Services, these are Microsoft and Google.

Any IT-related resources controlled by a firm including physical resources associated with IT such as IT infrastructure, databases, networks and software packages and applications, as well as non-physical (human) resources such as organization-wide IT expertise and IT managers' skills is called as IT resources.

History of Cloud



Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Cloud services were initially divided into three different models which were:

- IaaS: Infrastructure as a Service

  The consumer uses "fundamental computing resources" such as processing power, storage, networking components or middleware. The consumer can control the operating system, storage, deployed applications and possibly networking components such as firewalls and load balancers, but not the cloud infrastructure beneath them. The specification is like the general spec when you purchase a computer. These include CPU speed, number of cores, memory, etc. Initially the pricing was pay-per-hour but now it has become pay-per-second.
- PaaS: Platform as a Service

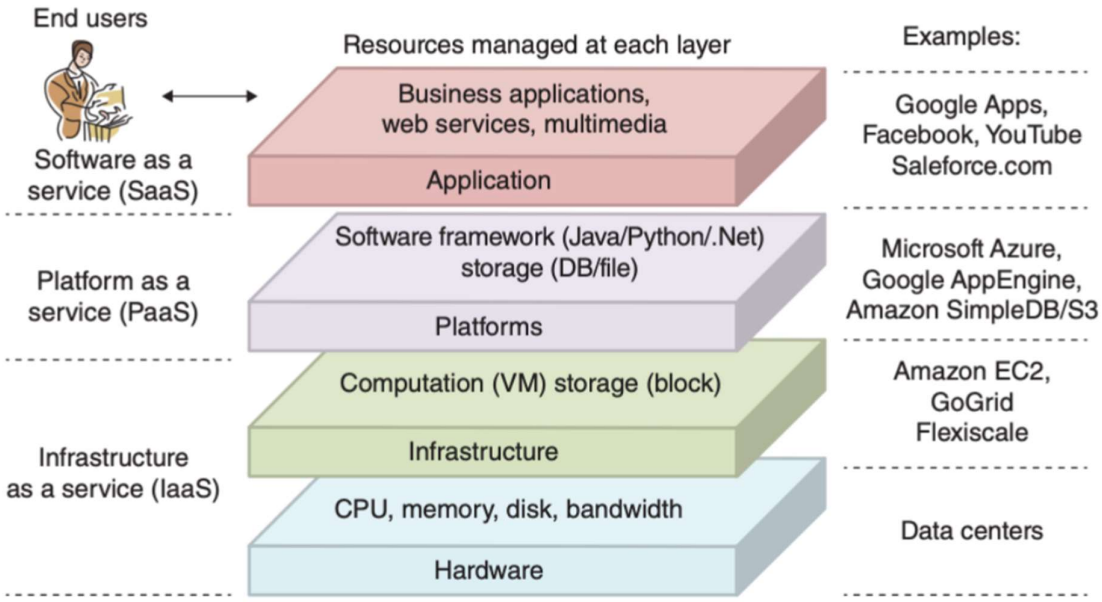
The consumer uses a hosting environment for their applications. The consumer controls the applications that run in the environment (and possibly has some control over the hosting environment), but does not control the operating system, hardware, or network infrastructure on which they are running. The platform is typically an application framework. PaaS is priced somewhere in between of both the other services where it can be either pay-per-hour or subscription based.

- SaaS: Software as a Service

  The consumer uses an application, but does not control the operating system, hardware, or network infrastructure on which it's running. The service specification depends on the actual application, it could be the number of user account supported, the size of storage, etc. The pricing is usually subscription based.

Today we have got many more services available from the Cloud. One such service is RaaS: Ransomware as a Service. Many providers are not restricted by a single service model and many services can not be categorized easily. Other services have their own way of describing, charging, and enabling technologies.

Typical Cloud Architecture



Incentives to Cloud Providers

IT resource utilization is usually very low. Server's CPU, memory, IO, networking are in idle state most of the time and hence the providers can let the same resource be used by multiple people at the same time hence maximising on the profit earned. And most of these Cloud providers are companies with a large amount of IT facilities.

Incentives to Cloud Users

Cloud users benefit from the cost models as they just need to pay for what and how much they have used. This lifts the operational management burden from the users as they will not need to install softwares, upgrade softwares, etc. as it is already done by the cloud providers.

Cloud provisioning problems

It is very hard to predict the usage and to provide enough cloud resources to the users.

WEEK 2

Cloud Storage Types

- Storage as SaaS

  Many cloud storage services act like an extension to the local hard disk. They provide similar features as the local file system. They also have some cloud features which make them different from the local file system such as sharing, collaborative editing, and versioning. A few examples of storage as a service are Google drive, One Drive, drop box, iCloud, etc.

- Storage as IaaS/PaaS

  There are also large services provided as independent infrastructure or as part of the other infrastructure or platform. They have much lesser UI features which makes them a bit difficult to use. They are a bit restrictive as well. A few examples of such storage services are AWS S3, Azure services, Google cloud storage, etc.

AWS & S3

Amazon Web Services provide storage services as well. They store data files as objects in a bucket and every object is addressable. This feature makes the amazon simple storage service a pseudo directory structure. Internally it uses a simple key value abstraction where the key is the address, and the value is the object. The problem with this is that one cannot preview the file online and would have to download the file locally to view the file. There is no random update functionality and hence if the file needs to be updated, it simply needs to be replaced with the updated version of the file. AWS Storage provides various ways to control access, replication, and retention options. The early version of AWS was built on Dynamo, an internal storage service used in Amazon in early 2000s. Today the internal structure is way more complicated than it was when it was first introduced.

Dynamo

Dynamo was used as an internal storage to support S3. It was motivated by the requirement of the main business as an online retailer. It is a decentralized peer-to-peer system where all nodes take the same role. Many services in Amazon only needed primary key access to the data and needed both scalability, and availability for their business model. The ecommerce model required a guaranteed latency and high availability and this led to the creation of a new design which considered the simple key value model. Strong consistency was sacrificed for high availability and conflict resolution was executed during the read instance instead of the write. Another feature was the incremental scalability of the service. A summary of the Dynamo techniques is given below:

Problem	Technique	Advantage
Partitioning	**Consistent Hashing**	**Incremental Scalability**
High Availability for writes	**Vector clocks** with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	**Sloppy Quorum** and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	**Gossip-based membership** protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Dynamo Partitioning Algorithm

A partition or shard key is used to determine where to put the data. A partition or shard key can be a user account, bucket name, or object key. It needs to be unique so that it uniquely identifies a single file. There is an algorithm to decide which data goes into which partition. There are 2 types of partition:

- Range partition

  Range partition requires keys to be of the same type so we can define a range of keys for each partition.
- Random (Hash) partition

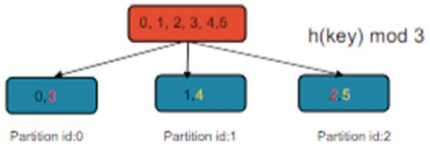
  Hash partition is more flexible as key distribution is based on its hash value instead of the raw value of the object. There are many options to determine the key placement such as:
  - Modulo function

    More popular and adopted by a lot of systems. If the size of the cluster is constantly changing, repartitioning becomes a problem in this method. The modulo function of key's hash value will redistribute large number of keys to different partitions. An example of the modulo function partitioning is:

The calculation goes as follows:

0 mod 2 is 0 hence it goes in 0.
1 mod 2 is 1 hence it goes in 1.
2 mod 2 is 0 hence it goes in 0.
3 mod 2 is 1 hence it goes in 1.

Facing an increment in the keys, the new example would be:

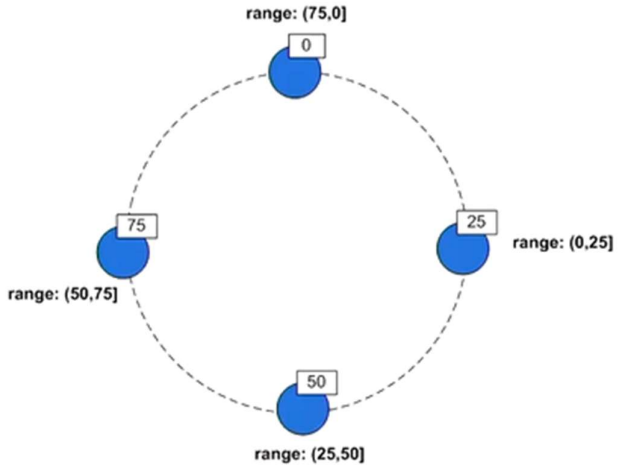


0 mod 3 is 0 hence it goes in 0.
1 mod 3 is 1 hence it goes in 1.

2 mod 3 is 2 hence it goes in 2.
3 mod 3 is 0 hence it goes in 0.
4 mod 3 is 1 hence it goes in 1.
5 mod 3 is 2 hence it goes in 2.

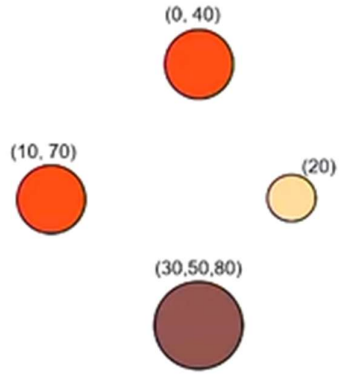
- Consistent hashing (Used in Dynamo)

  Used in Dynamo and is more complicated to implement but it has its own benefits such as incremental scalability. It is a special kind of hashing such that when a hash table is resized, only k/n keys need to be remapped on average, where k is the number of keys and n is the number of slots. It does not identify each partition as a number in [0, n-1]. The output range of a hash function is treated as a fixed circular space, meaning that the largest hash value wraps around to the smallest hash value. It circles back. Each partition represents a range in the ring space identified by its position value (token) and the hash of a data record's key will uniquely locate in a range. In a distributed system, each node represents one partition or a number of partitions if a virtual node is used. Each node in the Dynamo cluster is assigned a token representing its position in the ring and each node is responsible for the region in the ring between it and its predecessor node. An example is given below:



Assuming the ring space is 0-99. Suppose a key is hashed to the number 31. The node with token 50 will be responsible for the key. This is called the coordinator of this key.

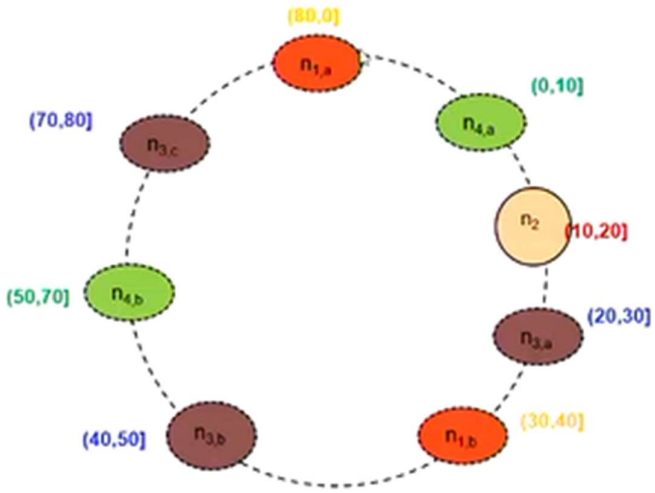
Possible issues of the basic consistent hashing algorithm are that the position is randomly assigned and hence cannot guarantee balanced distribution of data on nodes. It also assumes that nodes have similar capacities, meaning that each node is handling one partition. In the real implementation, the virtual nodes are used wherein multiple partitions per node can be made. Suppose the following virtual nodes are given:



Then the nodes will be given as:

N0 → N10 → N20 → N30 → N40 → N50 → N70 → N80

So, if a key with hash value 31 is entered, the virtual node N40 will be responsible for it. It would be represented as:



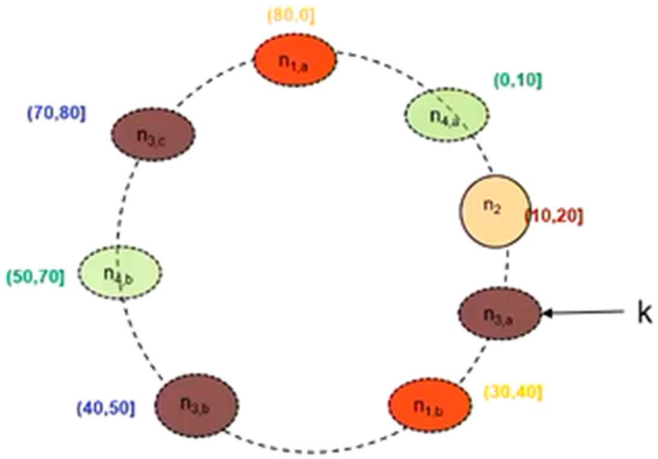
Dynamo Replication

Replication is essential for high availability and durability of the nodes. A few key terms in the concept are:

- Replication factor
  The replication factor is equivalent to the number of nodes where the data are replicated.
- Coordinator
  The coordinator node serves the client requests and replicates the data on the next N-1 clockwise successor nodes.
- Preference List
  In Dynamo, every node has a list of nodes on which it replicates the data. This list is known as the node's preference list. It contains more than the N successor nodes and skips similar virtual nodes.

Each key and its data is stored in the coordinator node as well as the N-1 clockwise successor nodes in the ring. The list of nodes that is responsible for storing a particular key is called the preference list. This preference list contains more than N nodes to allow for node failures. Some of the nodes are used as temporary storage or can be computed on the fly by any

node in the system. Note that with the use of virtual nodes, it is possible that the first N successor positions for a particular key may be owned by less than N distinct physical nodes. Meaning that the physical nodes might be partitions of the same virtual node and hence these nodes would be skipped to store memory in 2 distinct different physical nodes. For example, the preference list of the following would be:



N3 → N1 → N4 → N2

Membership and Failure detection

As it is a peer-to-peer system, each node in the cluster is aware of the token range handled by its peers. This is done by using a gossip protocol. The Gossip Protocol is a simple idea. At regular intervals (in Dynamo this is usually every second), nodes will choose another known node in the cluster at random and send some information. If the nodes don't agree on the information, they will reconcile the differences. This pattern repeats until every node in the cluster eventually has the same information.

Failure detection is also achieved through gossip protocol. Through the gossip protocol a local knowledge is gained. Nodes do not have to agree on whether a node is really dead. It is used to handle temporary node failure to avoid communication cost during read/write. A permanent node departure is handled externally.

Adding a storage node

To add a node the node must have a token not being currently used by any other node and then it can join the ring previously formed by the ring. As soon as the new node is joined, new ranges are formed.

Read/Write with replication.

In a typical primary/secondary replication environment, write happens on the primary and may propagate to the replica immediately and wait for all to ack before declaring success, or lazily and declare success after the master finishes write. Read may happen on the primary (strong consistency) or at one replica (may get stale data). In an environment where there is no designated primary/coordinator, other mechanisms need to be used to ensure certain level of consistency. It needs to be ensured that order of concurrent writes and how many replicas to contact before answering/acknowledging.

Concurrent writing in Peer-to-Peer setting

Different clients may try to update an item simultaneously. If nothing special is done, system could end with a split-brain. We need to ensure the order. Timestamp is a typical way to order concurrent writes. Based on the timestamp, the higher timestamp wins. Node ignores a write whose timestamp is lower than the value already present. This is also known as the Thomas Write Rule. Once the ignore is done, a new issue of clock synchronisation takes place.

Quorums

Suppose each write is initially done to W replicas (out of N). other replicas will be updated later. The typical way to find the current value of an item when reading is the traditional quorum approach in which we look at R replicas. In this timestamp of each value is considered and the value with the highest timestamp as the result of read is chosen. If $W > N/2$ and $R+W > N$, then this works properly. Any write and read will have at least one site in common as quorums intersect. Any read will see the most recently completed write and there will be at least one replica that is among the W written and among the R read.

Dynamo Mechanisms

- Vector Clock

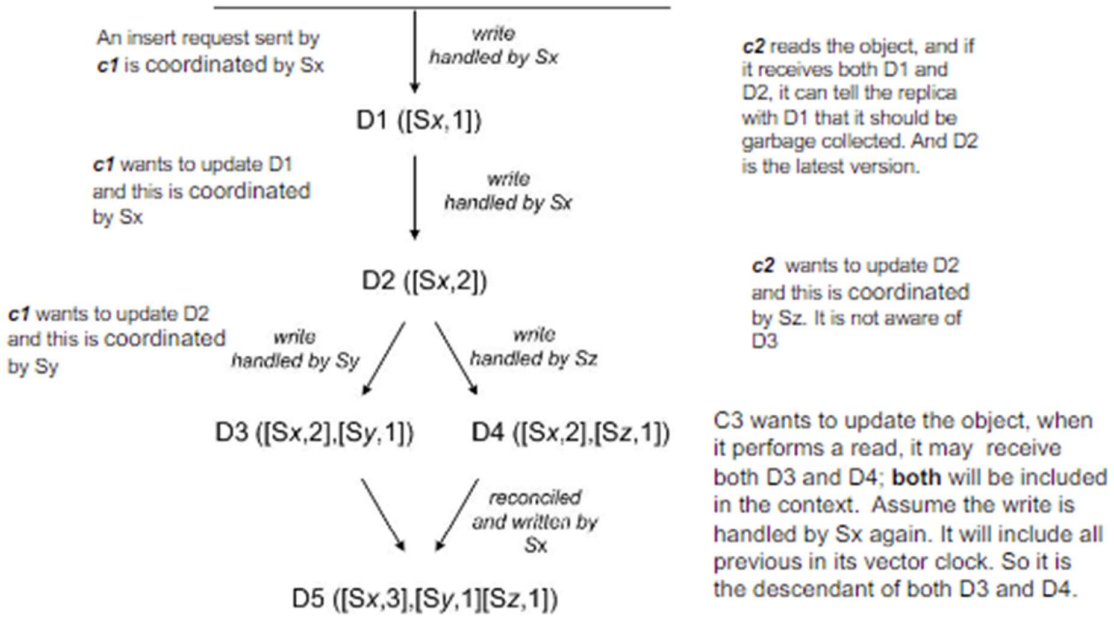
  It is a way to implement the timestamp concept in a distributed system. It is an aid to resolve version conflicts. Any node is eligible to receive the client read/write request and the node receives the request can direct the request to the node that has the data and is available. In dynamo a node handling a read or write operation is known as the coordinator. Typically, this is the first among the top N nodes in the preference list. A vector clock is effectively a list of (node, counter) pairs. One vector clock is associated with every version of every object. Node refers to the node that coordinator writes and counter remembers how many times this node has coordinated the write. Examples:
  - [(N0, 3), (N1, 1)] → The version of the object has been coordinated 3 times by N0 and once by N1. Hence it has been updated a total of 4 times.
  - [(N1, 5), (N2, 1), (N3, 2)] → The version of the object has been coordinated 5 times by N1, once by N2, and twice by N3. Hence it has been updated a total of 8 times.

  Coordinator variation is caused by temporary node failure. Vector clock is not designed for deriving coordination history of a single version. It is designed to determine casual ordering between a pair of clocks belonging to 2 versions of the same object.

  One can determine whether two versions of an object are on parallel branches or have a causal ordering, by examine their vector clocks. If the counters on the first object's clock are less-than-or-equal to all the nodes in the second clock, then the first is an ancestor of the second and can be forgotten. Otherwise, the two changes are in conflict and require reconciliation. Example for this is:
  - [(N0, 1), (N1, 1)] < [(N0, 2), (N1, 1), (N3, 1)] → The second version is 2 updates ahead of the first one.
  - [(N0, 1), (N1, 1)]??[(N0, 2), (N2, 1)]
    - The 2 versions branch off after the initial insert with [(N0, 1)].
    - The left version is then updated once, coordinated by N1.
    - The right version is updated twice, coordinated once by N2 and once by N0.

Each replica stores a copy of an object. The copies stored on different replica may be of different versions as indicated by its vector clock. This happens because any node hosting the replica may be temporarily unavailable and miss some update requests. Both read/write may contact multiple replicas and obtain multiple versions.

In Dynamo, when a client wishes to update an object, it must specify which version it is updating. This is done by passing the context it obtained from an earlier read operation, which contains the vector clock information. Upon processing a read request, if Dynamo has access to multiple branches that cannot be syntactically reconciled, it will return all the objects at the leaves, with the corresponding version information in the context. An update using this context is considered to have reconciled the divergent versions and the branches are collapsed into a single new version.



The size of vector clock is relatively small. Only the node coordinates the write request has an entry in the vector clock. If the replication factor N is 3, most vector clocks contain up to 3 entries. In rare situation when the top N nodes in the preference list is not available, another node will coordinate the write and add an entry in the vector clock. A vector clock truncate mechanism is used to remove the oldest entry. Truncating may remove histories that needed in reconciliation.

- Sloppy Quorum + Hinted hand off

  It is a mechanism to achieve the always writable feature and eventually consistency.

WEEK 3

Introduction

Cloud computing allows users to rent various internet based resources from providers on hourly/secondly, daily, monthly, yearly, etc. base. A few general models are used to describe the spectrum of cloud computing. IaaS and many PaaS use virtualization technology to present one or many virtual machines to the user. Virtualization technology is relatively standard and mature particularly with respect to CPU allocation and memories.

Virtualization

Virtualization in cloud computing lets one run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments. A few early virtualization examples are:

- Virtual memory
  - Provide more accessible memories to the CPU than those physically presented on the board.
  - An important component of the modern operating systems as it provides various benefits such as expansion and increasing code reusability.
  - The actions involved in the process are address translation, data transferring, data placement strategy for swapping.
- Mainframe Virtualization
  - IBM was the first one to release the mainframe virtualization solution in 1972.
  - The overall architecture of mainframe virtualization looks like modern system virtual machines.
  - It solves the problem of Operating system migration triggered by hardware release.
  - It was initialized by the time-sharing technologies of 1960s.
- Hot standby router protocol
  - This protocol solves the problem of single point failure because one host could define only a default gateway.
  - By using this protocol, a virtual IP and preconfigured priority is used to decide which physical router would act as the default gateway.
  - Other routers with the same virtual IP can take over if the default one fails.

A few common features of virtualization are:

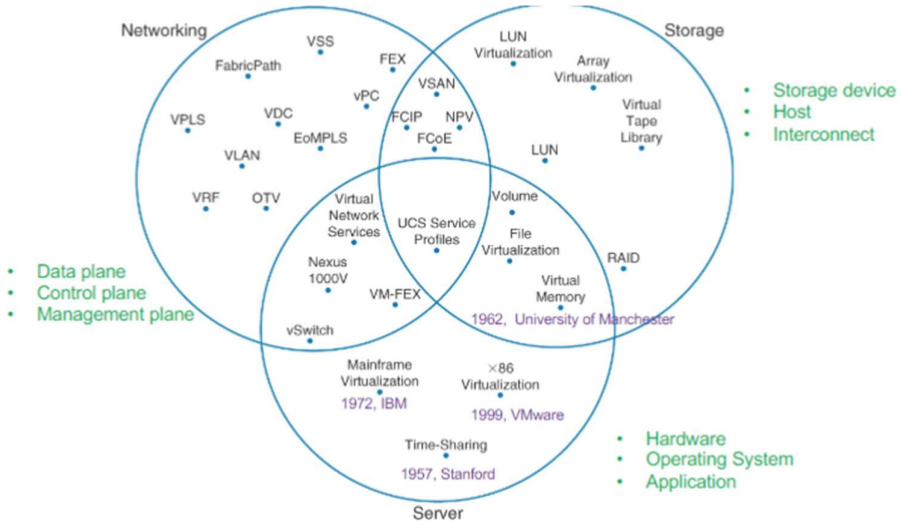
- Emulation

  Emulation, as name suggests, is a technique in which Virtual machines simulates complete hardware in software. In this technique pre-existing IT resources were emulated. These resources could be memory, mainframes or even IP addresses.
- Transparency

  The consumers of the resources which include CPU, Mainframe users, network hosts, etc. cannot distinguish between real and emulated resources.
- Benefits

  Compared with directly using actual resources, virtualization brings various benefits such as memory expansion, resource optimization, high availability, etc.

An overview of the Virtualization Technology Areas



Motivations for Server Virtualization

- Server Sprawl

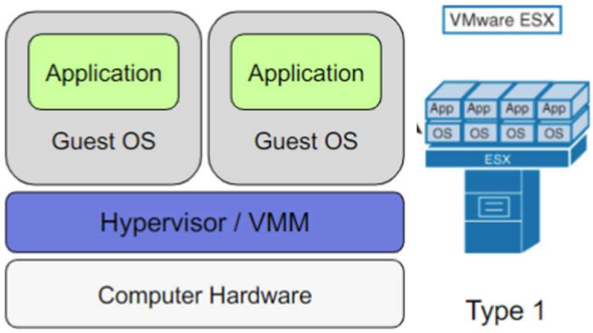
  A large number of underutilized servers is a major problem in many IT departments. The main cause for this underutilization maybe the heterogeneity of operating systems, requirements by vendors to run their applications in isolation, mergers, or other integration projects, etc. A solution to this problem was server consolidation. This was done by pooling heterogenous services together and to do so several virtual machines were run on the same shared hardware coordinated by a hypervisor also known as a virtual machine manager/monitor. The hypervisor abstracts/hides physical computing platform. This technique allows to share commodity hardware without sacrificing security and performance.

A virtual machine is a software defined computer that runs on a physical computer with a separate operating system and computing resources. The physical computer is called the host machine and virtual machines are guest machines. Multiple virtual machines can run on a single physical machine. Virtual machines are abstracted from the computer hardware by a hypervisor.

Hypervisors

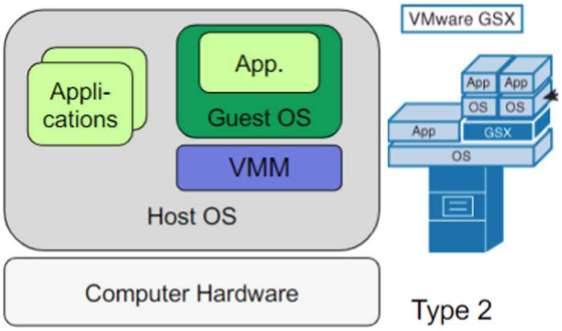
The hypervisor is a software component that manages multiple virtual machines in a computer. It ensures that each virtual machine gets the allocated resources and does not interfere with the operation of other virtual machines. It performs a lot of traditional operating system tasks such as resource sharing, device management and virtual storage management. There are 2 types of hypervisors:

- Type 1: Bare metal Virtual Machine (Native)



In this type of hypervisor, the virtual machine is run on the hardware which supports multiple virtual machines which support multiple operating systems.

- Type 2: Hosted Virtual Machine

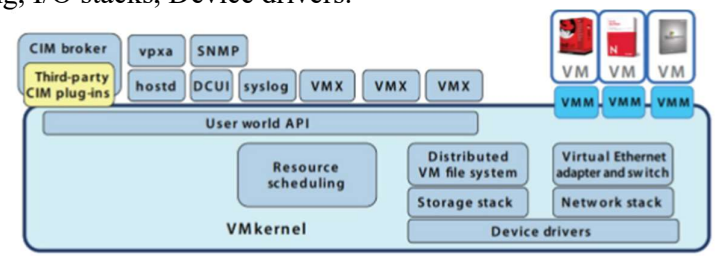


In this type of hypervisor, it runs as a part of a process over which other set of operating systems and applications are run.

The role of the hypervisor is to provide an environment identical to the physical environment and to provide that environment with minimal performance cost. It retains complete control of the system resources. Each virtual machine is presented with a fraction of the resources of the physical host. For example, a host may have 256 GB of physical memory installed in its frame but a guest Virtual Machine may get only 64 GB. A hypervisor abstracts the hardware resources and controls access to them to ensure each virtual machine gets allocated resources. Thus, all the access to hardware resources goes through the hypervisor. These hardware resources include memory, storage, network, etc. A few examples of Hypervisors are:

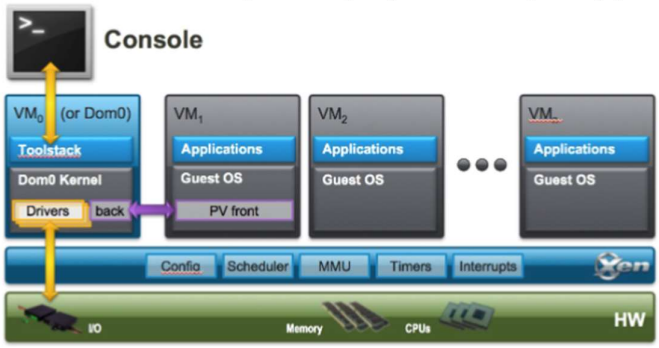
- VMWare ESXi

  VMkernel is a POSIX-like operating system which provides certain functionality like that found in other operating systems, such as process creation and control, signals, file system, and process threads. It is designed specifically to support running multiple virtual machines and provides such core functionality as: Resource scheduling, I/O stacks, Device drivers.



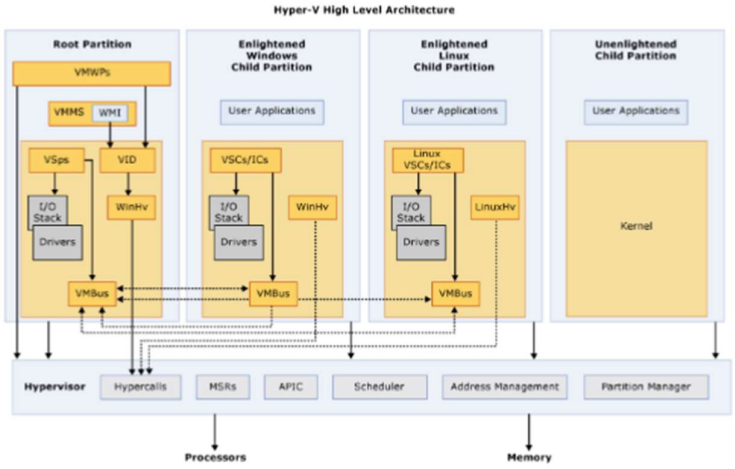
- Xen

Xen hypervisor is a thin software layer that runs directly on the hardware and is responsible for managing CPU, memory and interrupts. The control domain is a specialised virtual machine for handling I/O and for interacting with other virtual machines. It also exposes a control interface to the outside world through which the system is controlled. The guest domains/virtual machines are allocated to the users and a set of tool stack and console is provided in the form of an admin interface for creating, destroying, and configuring guest domains.



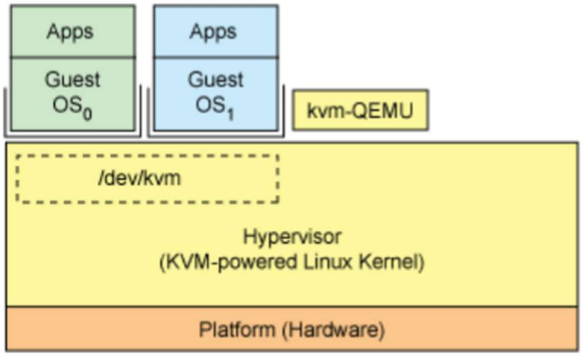
- Hyper-V

This hypervisor is a layer of software that sits on top of the hardware. Its primary job is to provide isolated execution environments called partitions. The hypervisor controls and arbitrates access to the underlying hardware. The partition is basically a virtual machine running on the hypervisor. The root partition is similar to Xen's domain 0 which manages I/O and communicates with the other partitions. All the other guest Virtual Machines are called child partitions.



- KVM

This hypervisor performs a lot of traditional operating system tasks. The full form of KVM is Kernel Based Virtual Machine and its development started in mid-2006. It turns Linux kernel into a hypervisor with the help of a few modules such as the KVM, QEMU (Quick Emulator which is a generic emulator that virtualizes I/O devices as well), and libvert (Provides a common management layer to manage virtual machines running on hypervisors). Both google cloud platform and AWS use KVM based virtualization. What this does is that it turns the kernel into a hypervisor which means it allows other operating systems to be booted on it and act as guest operating systems. It acts as a process, but it has much better isolation compared to a regular process.



There are some challenges to hypervisor implementation, and these are:

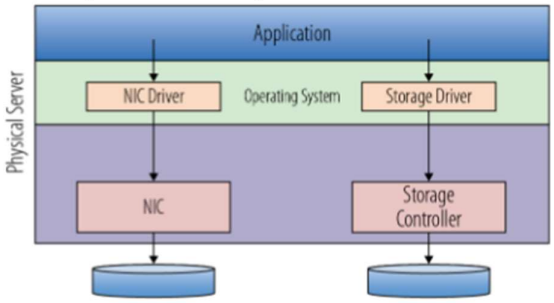
- Each Virtual Machine runs its own OS (called as the guest OS to differentiate it from the regular one) which assumes full control of all hardware resources. This is not true as the guest OS has only a portion of the hardware to use in virtualization. The OS is designed to be the most privileged software system on a machine and this feature does not make it as privileged as it was before.
- An extra layer of software is introduced to the performance overhead, this makes it as:
$$OS(ring\ 0) + APP\ (ring\ 3) \longrightarrow VMM + OS + APP$$
- The relationship between the VM and the OS need to carefully be managed.
- Typically, the VMM runs on ring 0 (the most privileged level), OS runs of ring 1 and APP runs on ring 3.

There are multiple types of instructions, these are:

- Privileged instruction
  One that traps when it is in user mode and does not trap when its in the system mode. An example of this is the CPU timer.
- Sensitive instruction
  Instructions that interact with the hardware. For example, memory address translation.
- Normal instruction

Virtual Machines

Virtual machines can be viewed as a software computer running an operating system and associated applications. The software computer runs on top of a hypervisor on a physical server. A virtual machine is described by a set of files which includes the emulated hardware definition file, virtual disk data and VM BIOS. The design to describe a VM through a set of files makes it very easy to save, copy, and clone VMs across physical servers. Fundamentally a hardware manages a lot of instructions and executes them on the CPUs. Application's request for hardware need to go through the OS for safety and security reasons. Those measures are built by having a protection ring mode at the processor level.



There are 2 modes which are:

- User Mode

  In this mode only certain resources are available and can run regular instruction like arithmetic operations. It can access user memory.
- Kernel/Privileged Mode

  In this mode all the resources are available and it can run regular as well as privileged instructions such as memory address mapping, load system registers, perform I/O operations, etc. It can access user memory as well as kernel memory.

Efficient Virtualization

When all sensitive instructions are also privileged meaning that they trap when they are not being executed in the system mode. This leads to full virtualization by running guest OS is user mode and VMM in system mode. Hence when guest OS attempts to run a sensitive instruction in user mode, it will be trapped to VMM which is responsible for the management of hardware globally. When some sensitive instructions are not privileged, meaning they don't trap automatically and are referred to as critical instructions. The existence of such critical instructions makes virtualising in x86 based system very hard in early days.

Translation/Emulation

Translation/Emulation can be done by using VMWare products. They try not to modify guest OS. It relies on techniques such as binary translation to trap and virtualize the execution of certain instructions. These instructions are discovered statically or dynamically during run-time and replaced with traps into the hypervisor that are to be emulated in software. A binary translation can incur a large performance overhead.

Paravirtualization

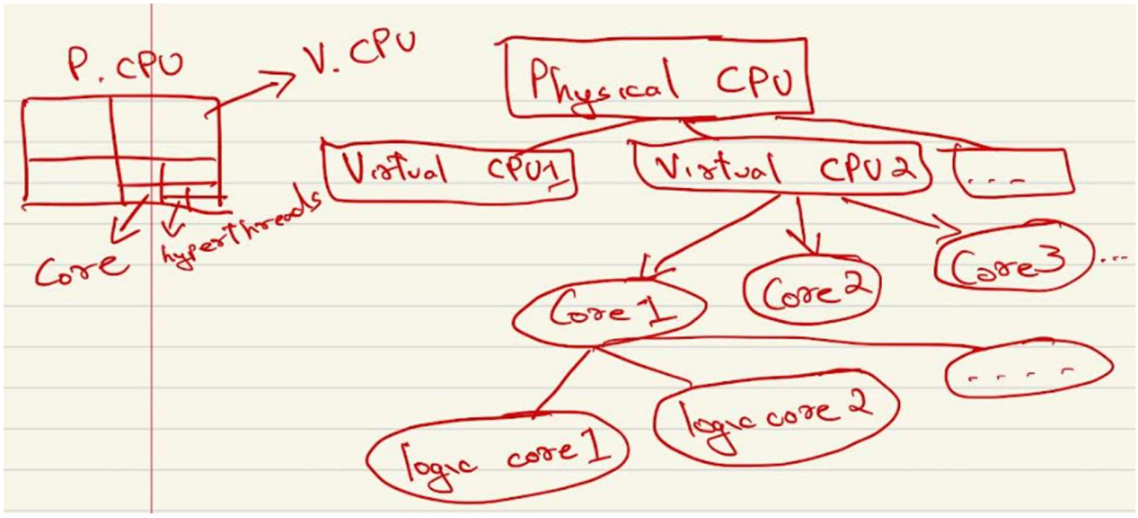
In this the hypervisor runs in ring 0 as usual where the kernel runs and the guest kernel is moved to ring 1. User apps runs in ring 3. The guest OS is modified for privileged instructions and this guest OS is aware that it is running on a VM extent. The software process validates first were allowed then executes on behalf of the guest OS.

Hardware Assisted Full Virtualization

Hypervisor runs in the new mode altogether. OS requests are trapped to hypervisor without the binary translation of Para-visualization. OS modification is not required since certain instructions will be trapped automatically to hypervisor. KVM is a hardware assisted full virtualization.

VR Resource Management

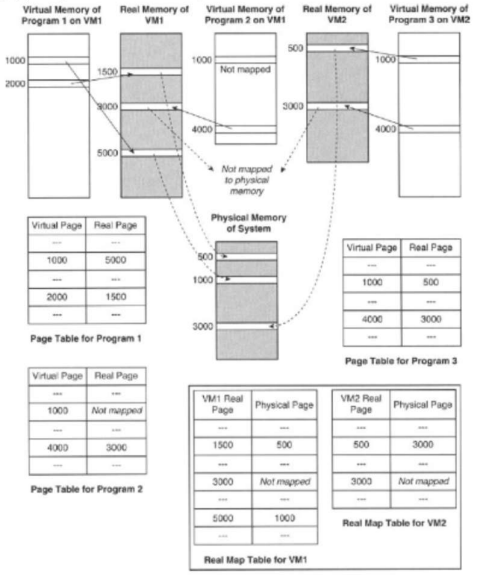
Physical CPU cores are time shared by virtual CPUs in server virtualization. The dynamic scheduling of virtual CPUs on physical CPU is managed by hypervisor following a customized scheduler algorithm. A host server may be configured with multiple processor chips each with multiple cores and the core may support hyperthreading to have two or more logic cores for one physical core.



When ordering Virtual Machine with Amazon or Google, we can specify the number of Virtual CPUs we want. The Virtual CPU count of a VM gives us the maximum number of threads that the VM can run at any given moment. A VM can run on any and all of the host CPUs over a period of time and some hypervisor may support pinning. For a VM with multiple vCPUs the hypervisor needs to schedule each vCPU on a different physical CPU. Hence the maximum number of vCPU you can assign to a single VM is bound by the hardware capacity, meaning the number of cores the computer has. Most applications are not designed to run on multiple threads. Configuring a VM with too many vCPUs may not have any benefits hence a very typical setting is a VM with just 2 vCPUs.

Virtualizing Memory Management

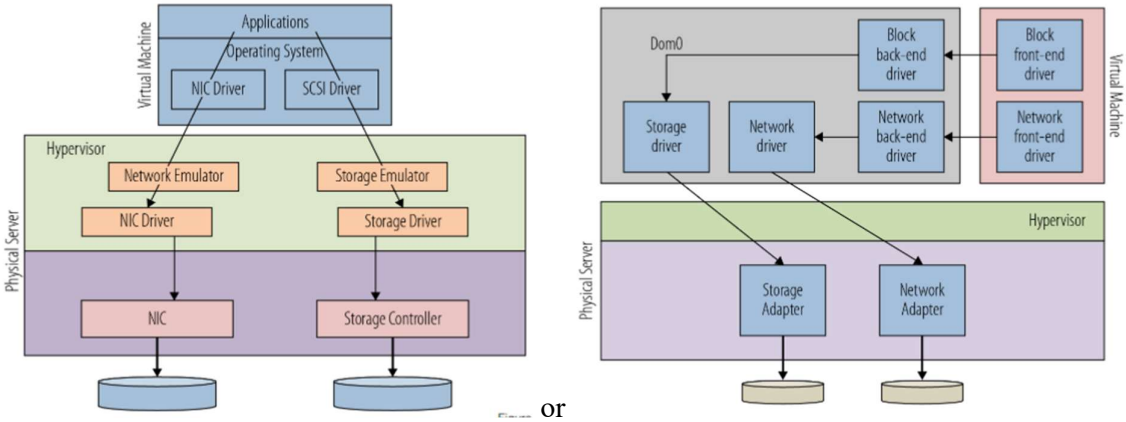
Memory is falls under the category of partitionable resources as they can be partitioned. OS is designed to manage mapping of virtual memory and physical memory through TLB and page tables. Guest OS usually gets a dedicated portion of the physical memory to ensure strong isolation and it does not know any other part of the memory and should not interfere with other VM's memory. The guest OS should not even know which part of the physical memory it is using and only the VMM has access to physical memory and can manage this mapping. The guest OS does a 2-level mapping of memory.



Virtualizing I/O

I/O refers to all input/output devices such as the disk, network, printer, monitor, mouse, etc. On one hand there are many devices controlling them in different ways whereas on the other hand OS have efficient ways to deal with I/O by providing standard interface and by loading various drivers.

A disk and a network are usually virtualized. A disk is a partitioned device, each VM can get a partition of it exposed as a virtual disk. An external disk driver can be mounted easily. A network adapter is a shared device which can be time shared by each VM. A virtual network adapter is presented to the VM. The request is passed to and managed by the Virtual Machine Management. An example of an I/O pathway is given below:



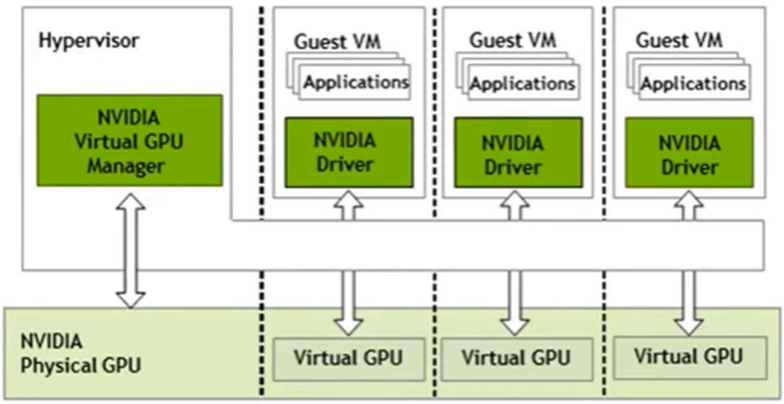
or

GPU vs CPU

CPU stands for the Central processing unit. It has several cores and typically low latency. It is good for serial processing but can do only a handful of operations at once.

GPU stands for Graphics processing unit. It has many cores and has a typically high throughput value. It is good for parallel processing and can-do thousands of operations at once.

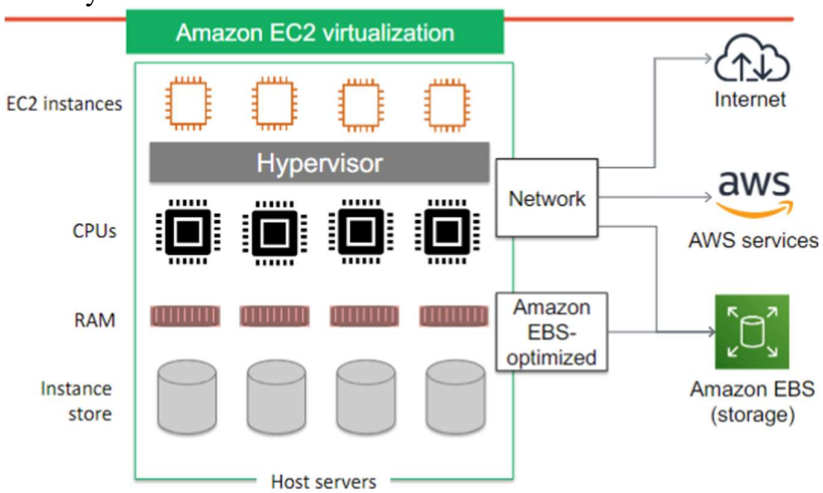
A GPU can be virtualized. An entire physical GPU is directly assigned to a VM. This GPU is accessed by corresponding drivers running in the VM. It is not shared among other VMs. If a physical GPU is to be shared among multiple VMs, it needs a special virtual GPU management software to slice the GPU resources among VMs.



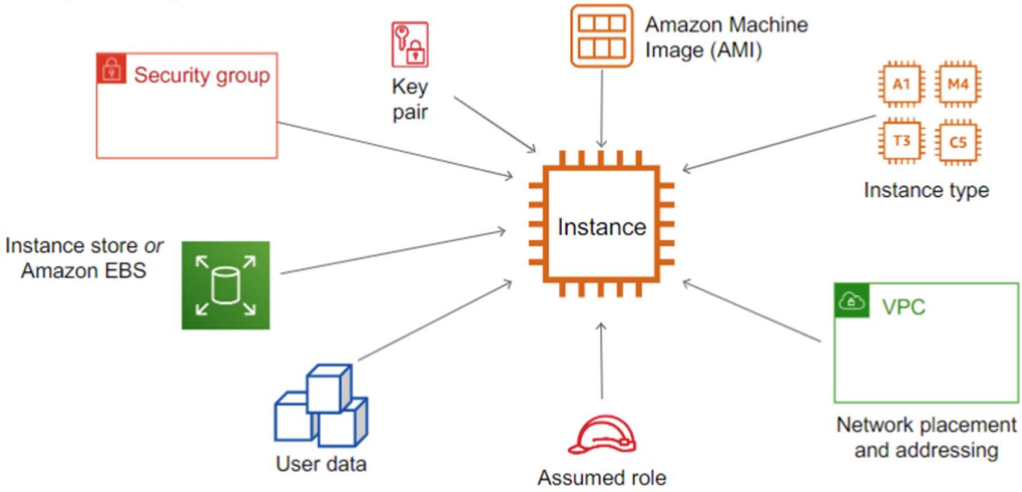
WEEK 4

Introduction

Amazon EC2 provides resizable compute capacity in the cloud. Amazon EC2 provides virtual machines (servers) and provisions servers in minutes. It can automatically scale capacity up or down as needed and even enables you to pay only for the capacity that you use. An EC2 instance is a virtual machine that runs on a physical host. One can choose different configurations of CPU and memory capacity and it supports different storage options such as instance store and Amazon Elastic Block Store (Amazon EBS). Amazon EC2 also provides network connectivity.

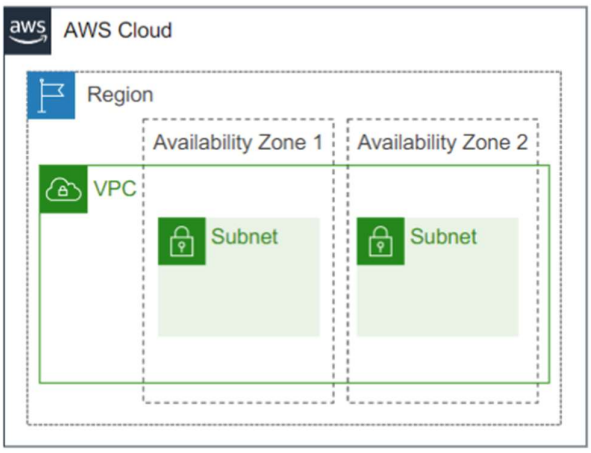


Provisioning an EC2 instance requires a lot of things. A few essential instances launch configuration parameters are:



Virtual Private Cloud (VPC)

An EC2 instance is always launched in a VPC and a subnet. There is a default VPC and a public subnet per region for any user account. This can be used to quickly launch an instance. VPCs are logically isolated from other VPCs and dedicated to user's own AWS account. It belongs to a single AWS region and can span multiple availability zones. Subnets on the other hand is a range of IP addresses that divide a VPC and belongs to a single availability zone. It can be classified as public or private.

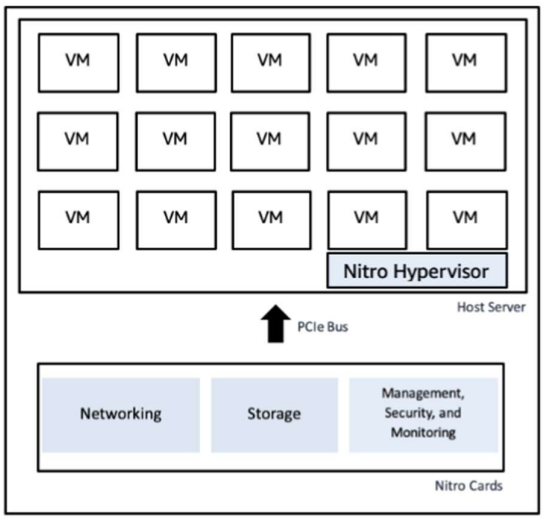


Persistent and Ephemeral Resources

Ephemeral storage is the volatile temporary storage attached to your instances which is only present during the running lifetime of the instance. In the case that the instance is stopped or terminated, or underlying hardware faces an issue, any data stored on ephemeral storage would be lost. Persistent storage on the other hand keeps the data stored on it safe even against a system crash. An example of persistent storage is Elastic Block storage. A few examples of persistent resources are elastic block storage volumes, security groups, key pairs, etc. An example of Ephemeral resources is EC2 instances which could have resources like IP address, memory, local storage (instance storage), etc.

AWS Nitro System

Nitro is the current virtualization system used in amazon to provision EC2 and all related services. AWS used Xen hypervisor in the early days and then moved into its own virtualization system Nitro in 2017. Nitro itself is an example of microservice architecture with a very thin HVM based hypervisor which is mainly used for partitioning and scheduling CPU and memory. Hardware support for all other resources, including monitoring and management are built into Nitro cards and domain 0 (Domain 0 is Xen hypervisor) was thus no longer required.



The Nitro architecture looks like the above diagram. There is a hypervisor based on the KVM which is a Linux kernel. The internals are not open-sourced and hence not much

information is available. There exists a nitro card for networking, storage and management, security, and monitoring. The first thing offloaded from the physical CPU is the networking, then storage and so on.

A typical Nitro system consists of a hypervisor which is KVM based and accesses the CPU and memory. It also has one or more nitro cards and security chips. A few examples of nitro cards are:

- Networking

  A nitro card for VPC. It has a network interface for EC2. It looks like a network adapter and provides its own API hence making the driver independent of actual hardware capacity making upgrading a lot easier.
- Storage

  A nitro card for EBS. It provides a NVMe controller, and it also provides encryption support. It also provides NVM to remote control storage. A nitro card is also allotted for instance storage where instance storage has been on and off in AWS. Networking and storage evolved along a different path. Nitro card offer local storage when I/O throughput is faster than network throughput. When network throughput is greater than storage throughput, offering network drive makes more sense and hence local storage comes back with SSD and NVMe.
- Management
- Monitoring
- Security

AWS Bare metal instance

It is possible because of nitro systems and microservices and offloading everything to hardware. Bare metal instances do not use nitro hypervisor but still use the nitro cards to access VPC, EBS and other AWS services. There is not much performance difference between hypervisor supported and bare metal instances. Usually use instances backed by hypervisor in most cases.

WEEK 5

Introduction

Cloud hosted databases are those databases which are installed and managed on existing database servers on Virtual Machines. These databases are fully managed database services based on existing database engines. A few examples being: AWS RDS, MongoDB Atlas, etc. A few cloud native databases are Azure cosmos DB, Google Bigtable, Google Spanner, Amazon DynamoDB, Amazon Aurora, etc.

A cloud native database is designed and built to incorporate cloud features such as scalability, elasticity, high availability, etc. Early versions of cloud native databases used NoSQL or simplified versions of SQL models, but now full cloud scale SQL databases are supported by large providers.

Google Spanner

Google's earliest cloud database was Bigtable, and it was a table like structure with only key based API. It was optimized for cloud features such as scalability, elasticity, faulty tolerance, etc. but not designed as a general-purpose storage system. It became difficult to use for OLTP applications.

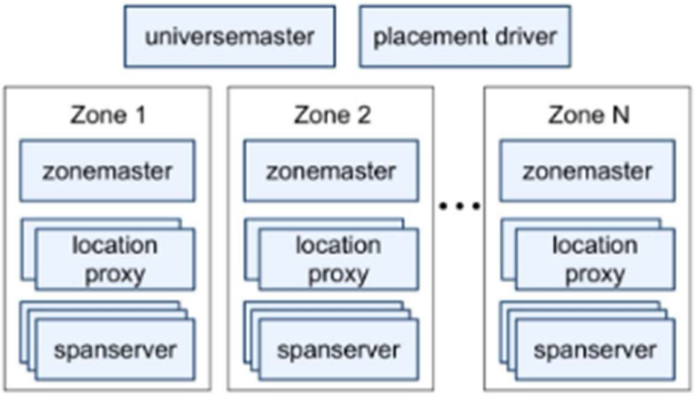
Google then came up with a semi-relational data model on top of Bigtable called Megastore. The performance for which was unfortunately not ideal. It lacked many traditional database features but is still used by many well-known google applications such as Gmail, Picasa, Calendar, etc.

Google's latest take on cloud database came in the form of Google spanner which places its focus on managing cross-datacentre replicated data. It was initially used as a backend database for Google's advertising services but then was used for other services as well. It uses a temporal multi-version database system where data is stored in schematized semi-relational tables where data version is automatically timestamped with its commit time. The main challenges of implementing database of this scale is:

- Distributed join is very expensive and to avoid this expense pre-join at the data model level is used.
- Distributed transaction is hard to coordinate and TrueTime API is used to ensure external consistency.

Google Spanner Structure

A spanner deployment is called a universe and a universe consists of many zones. A zone is the rough analogue of a Bigtable cluster and represent the replication location. Zones are a unit of physical location and there are one or more than one zones in a data centre. Other than the zones, there are 2 special servers which are the Universe master and placement driver which are singletons. There can only be a handful of universes such as the development, test, and production universe.



A zone level server will have 3 types of components. It has one Zonemaster that allocates data to the spanserver, another type of component in the zone level server. There are many spanservers which have their main function to serve data to their clients. A few location proxies' servers also exist to help clients locate the spanserver assigned to their data.

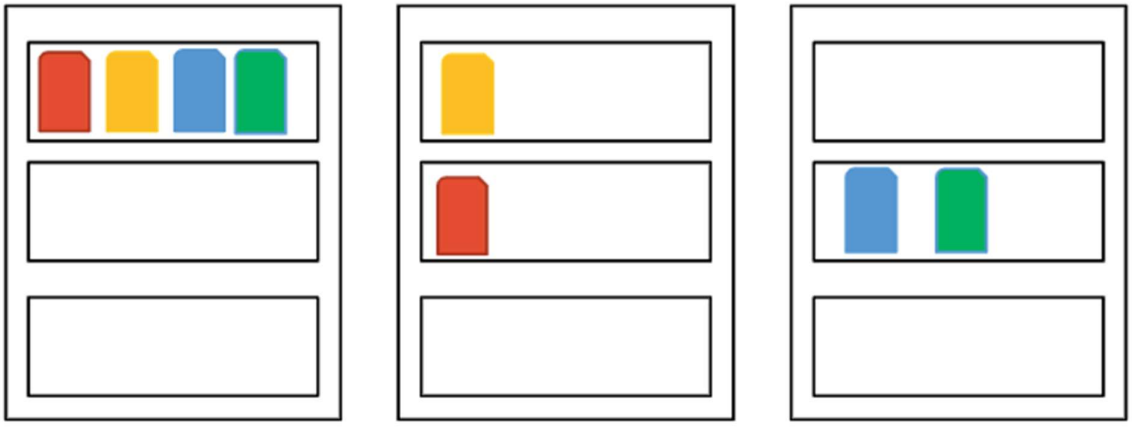
Each spanserver manages between 100 to 1000 tablets wherein a tablet is a data partition unit. Actual data files and logs are stored on 'append only' Colossus file system where the replication of the data files and logs are managed by Colossus.

Data Replication

Here are 2 levels of replication which are locally within the zone and other one being across zones. The replication which occurs locally within the zone is managed outside Spanner by the underlying file system of Colossus whereas for the replication taking place across zones, it is managed by the Spanner.

The Universe Master is primarily a console to monitor zone status for interactive debugging and the placement driver handles data movement across the zones. The placement driver is responsible for load balancing and satisfying replication constraints.

Cross Zone Data replication



Assuming we have 3 different zones with 3 different spanservers in each of them. We have got the 4 tablets which are coloured in one of the spanservers in Zone 1. Zone 2 will have replicas of 'red' and 'yellow' tablets in 2 distinct spanservers and Zone 3 will have replicas of 'blue' and 'green' tablets in 1 spanserver.

Tablet replication

Tablet replication across zones is managed by Paxos algorithm. The set of all replicas of a tablet forms a Paxos group and there are many Paxos group in the entire universe. The leader uses locks to implement concurrent write and act as a transaction manager for distributed transactions. It also reads access data that is sufficiently up to date. The number of Paxos groups in the whole universe are equivalent to the number of tablets managed by the spanner deployment.

Coordination activities with the tablet

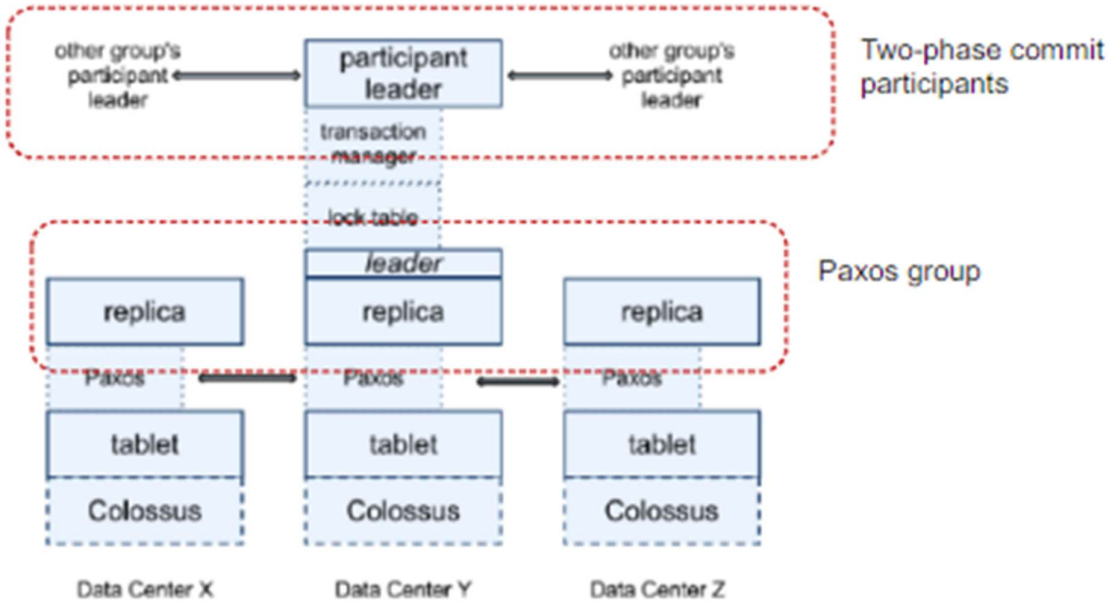
A universe is expected to receive many transactions and concurrent transaction might also be needed to be executed in consistent order in the replicas involved. There are a couple of activities involved such as:

- Ensuring consensus on the transaction order among all the replicas. – Ensured using Paxos algorithm.
- Execute the transactions to the agreed order including concurrent ones. – Ensured using lock mechanisms.

Distributed transaction coordination activities

Each transaction may contain a few queries and most of the transactions contain queries that involve a single Paxos group. In most of the cases, queries target the same tablet, and no extra coordination is needed as the consistency of the replicas are guaranteed by Paxos. A small number of transactions contains queries which lie across Paxos groups and in this case a distributed transaction occurs. This distributed transaction is coordinated by a 2-phase count, each involved group's participant leader would join the protocol.

Transaction Management

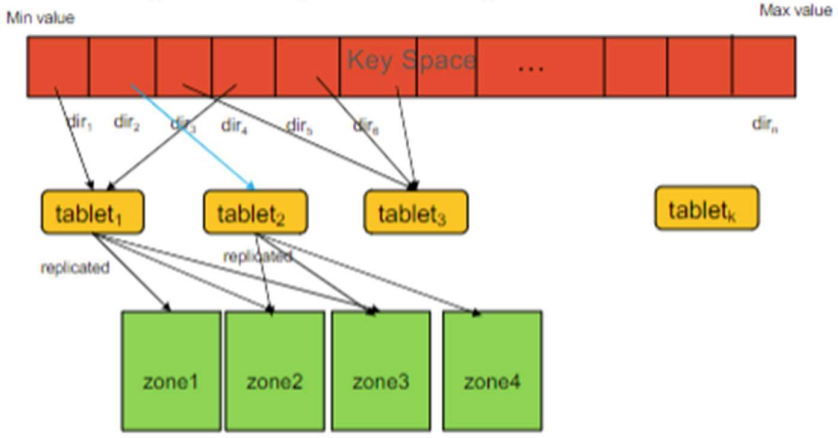


When different tablets are involved in a transaction and for each tablet there are 3 replicas that means there are 3 spanservers. Hence if a transaction involves 2 tablets, it involves a total of 6 spanservers. Only one of these spanservers will join the transaction which would be the leader. Meaning that one spanserver will participate in the transaction as the leader to ensure the execution of the transaction.

Remember that each tablet is managed by a Paxos group and has leaders and the leaders from different Paxos groups use 2pc to manage the transaction and each leader then coordinates its replica's update within the Paxos group.

Google Spanner Data model

Data in a tablet is organized as directories. A directory is a bucket like abstraction representing a set of continuous keys in the tablet that shares a common prefix. A directory is the unit of data[1] placement and a unit of data movement between Paxos groups. Following is an image to understand spanner data partition and replication:



[1] Data > Tablet > Directory

- Each directory contains a continuous range of keys sharing common prefixes.
- Each tablet contains several directories.
- Each tablet does not contain a continuous range of keys.
- Each tablet is replicated to several zones managed by the Spanner.
- Within the zone, replication is managed by the underlying file system: Colossus.

Each table is required to have one or more columns specified as the primary key as this is the key that gets referenced in transactions like DBMS. Each table defines a mapping from the primary key columns to the other columns. Table can have hierarchies defined by the client when creating the tables and this hierarchy defines the key-value pairs in directories and tablets.

Google Spanner TrueTime API

Spanner assigns globally meaningful commit timestamps to transactions. These timestamps reflect the serialization order which satisfies external consistency. If a transaction T1 commits before transaction T2, the timestamp of T1 is smaller than that of T2. The commit timestamp does not necessarily equal to the actual commit time and is like the vector clock in Dynamo. The timestamp is assigned based on a strict 2-phase locking for writing transactions. A timestamp is assigned only when a lock is held. Hence the timestamp order respects global wall-time order.

Timestamps are picked by the transaction coordinator which is the leader in this case. The leaders are the spanservers which are distributed globally. The clocks on different servers are not synchronized. Hence to solve this problem of unsynchronized timestamps, a global wall-clock time with bounded uncertainty is introduced. In this the earliest and latest timestamp is given with the most minimum value for error.

WEEK 6

Paxos Algorithm Introduction

It is a distributed consensus algorithm. The main idea is to get all nodes to agree on something. Consistency is a special case of this consensus problem. Assuming a collection of processes that can propose values. A consensus algorithm ensures that a single value among the proposed values is chosen and shared among the processes. If no value is proposed, then no value will be chosen.

Requirements for correct consensus

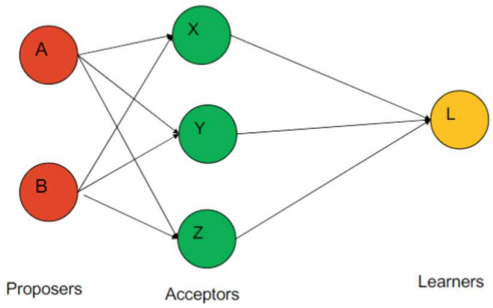
- Safety requirements
  Only a single value that has been proposed may be chosen and a process never learns that a value has been chosen until it has.
- Liveness requirements
  Some proposed value will be eventually chosen and that if a value is chosen then a node will eventually learn the value.

Paxos Algorithm

A node/process can play any number of roles in the algorithm. There are 3 roles for the participants of the algorithm. These are:

- Proposers
  A proposer will send a proposed value to a set of acceptors.
- Acceptors
  An acceptor may accept the proposed value. A value is chosen only when many acceptors have accepted a value.

- Learners
  A learner will learn the chosen value.



Assumptions in Paxos

- The communication is asynchronous that means that a node can send a message to another node without waiting for acknowledgement. The message can be duplicated, lost, but can never be corrupted.
- Nodes can fail but after failure will eventually restart. And even after restart will remember some information.

Choosing a value

A naïve condition would be that each acceptor can accept only one value and that a value will be chosen only if most of the acceptors accept that value. There are certain undesirable scenarios such as having no accepted values, or no majority agreed values. This will happen only if the requirement of consensus is breached.

- Liveness requirement
  Based on this requirement, an acceptor must accept the first proposal it receives. The potential issue with this is that different acceptors might accept different proposals with no majority. The solution for this would be to allow the acceptor to accept multiple proposals. Each proposal is numbered which makes it easy to not the issuing order.
- Safety requirement
  Based on this requirement, a value is chosen when a single proposal with that value has been accepted by most of the acceptors.

If a proposal (P2) with a value (v) is chosen, then every higher numbered proposal such as (P3, P4, …) will have the same value as (v). If a proposal (P2a) with value (v) is chosen then every higher numbered proposal such as (P3, P4, …) accepted by any acceptor has the value (v). If a proposal (P2b) with value (v) is chosen, then every higher numbered proposal issued by any proposer has the value (v). For any proposal (P2c), any value (v) and number (n) is issued then there is a set (S) consisting of the majority of the acceptors such that either no acceptor in (S) has accepted a proposal numbered less than (n) or (v) is the value of the highest numbered proposal accepted by an acceptor.

Basically, this means that if a proposer proposes a value which is accepted, then any other proposals will have the same value. If a value is chosen, then all the proposers will propose the same value. This is made sure by making sure than if a value (v) is chosen from a proposal (P) with a number (P4) then the acceptors will not accept any values from a proposal numbered less than 4.

2 Phases of Paxos Algorithm

- Phase 1

  This is the "Prepare and Promise" phase. In this phase the proposer lets the acceptor know the current proposal number (E.g., P2, P3, etc.). The proposer the gets a promise from the acceptor that it will not accept any proposals less than it and that the acceptor will learn the value of any accepted proposal with the highest proposal number. The acceptor can accept several proposals with different numbers and values.

  In this phase the proposer creates a proposal identified with a number "n" and sends a prepare request to the quorum of acceptors. The "n" should be higher than the previous proposal. If the acceptor receives the prepare request with number "n" greater than any proposal it has got, it responds to the proposer with a promise to not accept any more proposals numbered lover than "n". It will not respond if the number "n" is smaller than another accepted proposal with a higher number.

- Phase 2

  This is the "Accept" phase. In this phase the proposer sends the complete proposal with the proposal number and value to the acceptor. The value could be a value already accepted by the acceptor or chosen by the proposer if no value has been accepted yet. An acceptor can accept or ignore the proposal based on the response in Phase 1.

  In this phase the proposer receives response from most acceptors and then sends an accept request to each of those acceptors for a proposal numbered "n" with a value "v" where "v" is the value of the highest numbered proposal among the responses. If an acceptor receives an accept request for a proposal numbered "n", it accepts the proposal unless it has already responded to a prepare request of a higher numbered proposal numbered "n".

Concurrent Request Scenario

Paxos is designed with concurrent requests in mind. It is also designed to tolerate message loss. The requests may arrive in different orders and request, or response may be lost or ignored. Under this scenario, if an acceptor receives a prepare request with number (n) greater than any other prepare requests it saw, it will respond with the promise to not accept any other proposals numbered less than (n). Through this, acceptors can ignore any prepare requests with a number (n) smaller than the promised number. Usually, after phase 1, multiple proposers will receive responses from most of the acceptors. Some acceptors might have even made more than one promises.

Chosen Value

A value is chosen at a proposal number (n) if and only if most of the acceptor nodes accept that value in phase 2 of the algorithm. A value can be chosen without any nodes in the system knowing the fact for some time and hence the chosen value and knowing the value is different. Each acceptor whenever it receives a proposal responds to all learners by sending the proposal. The simple way would need (n*m) messages (n being the number of acceptors and m being the number of learners). To avoid unnecessary message passing, it is possible to have a single distinguished learner. All acceptors will send accept messages to this learner which then informs other learners of the value chosen.

A proposer can make multiple proposals but only a single value will be chosen in a single instance of Paxos algorithm. Sometimes, it will be a good decision for a proposer to abandon a proposal and this would not affect the outcome of the algorithm.

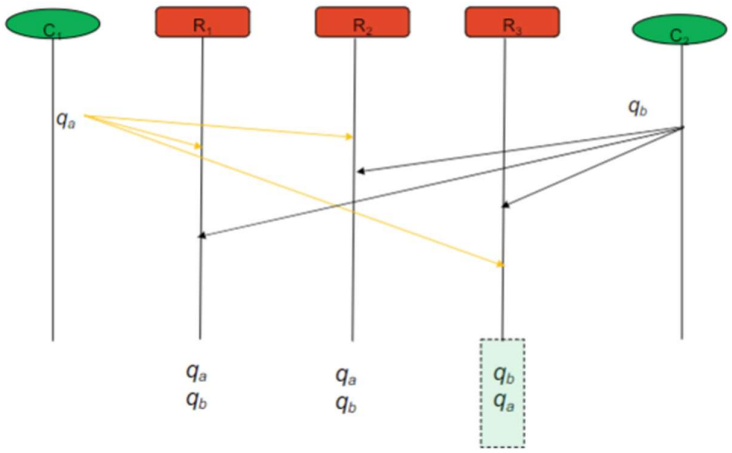
Multiple Proposal and progress

A proposer can make multiple proposals but only a single value will be chosen in a single instance of Paxos Algorithm. Sometimes, it is a good idea for a proposer to abandon a proposal and this won't affect the correctness. Now, multiple proposers are sending proposals to the acceptors. Acceptor will send a promise to not accept any proposals from lower numbered proposals. But if the acceptor has accepted any values, it will send a promise to the higher numbered proposal with the value the acceptor has already accepted along with the proposal number. If the proposer receives a response from the acceptor, it will send an accept request with the value (v) to each of the acceptors.

In a system with 2 proposers, it is possible that each keep sending proposals to the acceptor. Only the higher numbered proposal will be accepted, whereas the other would be ignored by the acceptors. To guarantee progress, a distinguished proposer must be selected as the only one to try issuing proposers. This distinguished proposer will act like a coordinator in a replicated system.

Paxos Algorithm in Replicated systems

A single instance of Paxos algorithm only allows one value to be chosen. Any real system would involve many servers and the data can be updated by multiple clients. Each replica server performs client commands in some sequence. Each replica server can be viewed as a deterministic state machine and the state machine has a current state. It performs a step by taking a command as input and producing an output and a new state. For stronger consistency, the system needs to ensure that all or majority of the replica servers execute their client commands and execute it in the same order. Consider the following system with 3 replica servers and 2 clients who are running queries in the following order:



In the above example the Replica servers get the orders in the following order:

- Replica 1: Qa, Qb.
- Replica 2: Qa, Qb.
- Replica 3: Qb, Qa.

Multiple Paxos (almost infinite) is used in this case. A single Paxos instance corresponds to a particular command (query). Assuming a set of fixed servers, each server plays all the 3 roles. A single server is elected as a distinguished proposer. Each Paxos instance is numbered and represents the i^{th} command the system to run. For example, if Replica 2 is the distinguished proposer, the queries are all sent to the distinguished proposer and runs an infinite number of Paxos instances. It runs the 1^{st} Paxos to get an agreement with all replicas that the first command to run is Qa, at the same time another Paxos is run to make all replicas know

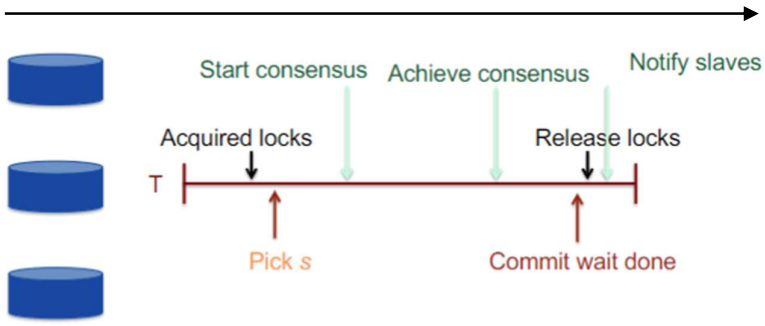
the second command to run is Qb. Any replica which knows the chosen value, will go ahead, and execute the command accordingly. If a replica learns the value of the second Paxos first, then it waits until it learns the value of the first Paxos.

Leadership Change

In case the distinguished proposer fails, a new distinguished proposer (leader) has been elected. The new leader will be one of the replica servers and would know most of the commands that have been chosen but may have slightly different knowledge than other replicas. The new leader will execute the instances of the knowledge gaps and if there are any replica which have accepted values, it will execute the second phase of these instances to confirm the chosen value. After the execution steps, there might still be some gaps. The new leader can either assign commands or assign a special no-op value to the Paxos. Then these Paxos with no-op value have their second phase executed and then continues to run as normal. A newly chosen leader will execute phase 1 for infinitely many instances of the consensus algorithm. It can run infinite instances efficiently by using the same proposal number for all instances. If an acceptor has received a phase 2 message, then it responds with a short message otherwise it sends a message OK.

Paxos in Spanner

In Spanner (Google Database), we have the concept of tablets, and a tablet can be replicated on multiple zones. Each Spanner tablet represents a Paxos group of Spanservers managing the replica of that tablet. So each Spanner will have multiple Paxos groups which is equal to the number of tablets that it manages. Each Paxos group will implement multiple Paxos algorithm to ensure strict timestamp order of transactions within a group. Only the Paxos leaders (distinguished Proposer) acquires locks and pick timestamps for transaction involving data in Paxos groups. The timestamp is shared with the replicas using the Paxos algorithm. Within each Paxos group, there is a strict order of timestamps assigned to transactions and transactions affecting different data and managed by different Paxos groups can have the same timestamp.



2PC for Distributed Transaction

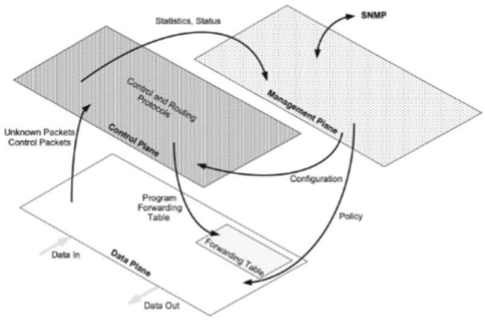
Distributed transactions involve more than one Paxos groups and each leader picks a timestamp based on its lock acquiring time. Transaction coordinator receives these timestamps as part of the 2Phase Commit message and the transaction coordinator picks the largest timestamp as the timestamp of the transaction. This guarantees that the transaction time is the time after all locks have been acquired. It also ensures order of the transactions within a Paxos group.

WEEK 7

Network Virtualization

One data centre may accommodate a lot of physical servers and some server may host up to 20 Virtual Machines. The internal network of a data centre could interconnect and host more than a million users. Traditional network technologies are unreliable with communication links whereas the data centre has highly reliable communication links. There is a lot of traffic on these networks, and these can be Intra-data, East-West traffic (Internal server traffic), and North-South traffic (Traffic which leaves the data centre). The protocols that were designed to achieve robustness in the wide-area internet today requires over 30% of their CPU cycle to rediscover and recalculate routes. Physical network topology in the data centre is highly static and only changed under strict centralized control. Agility is required because virtual machines in a data centre come and go frequently.

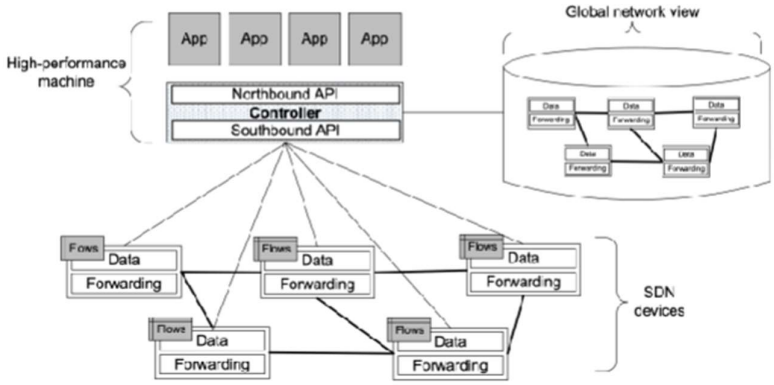
Traditional Switch Architecture



The data plane is responsible for the transmission of packets with the help of the forwarding table. The control plane is responsible for keeping the current information in the forwarding table so that the data plane can independently handle as much traffic as possible. The management plane is used by network administrators to configure and monitor the switch. The base topology in a data centre is centrally provisioned and most topology changes are done programmatically and intentionally. Any changes in this topology takes time to converge a set of consistent forwarding tables. In contrast, the topology inside a data centre is quite stable and under strict local administrative control and the knowledge of the topology is already centralized and controlled by the same administrators. Therefore, any changes in the topology can be quickly updated and a consistent set of forwarding tables can be obtained.

Software Defined Network

A common standard of software defined network is called open floor. An example of this is given below:



This network model does not really need a control plan but requires a control planner. The data planner consists of only the Data and the forwarding. The control plan is centralized.

SDN has the functionality for deciding what to do with each incoming packet of data. The SDN controller maintains a view of the entire network that it controls and presents an abstraction of the network resources to the applications running on top of it. It requires high performance computation to calculate the optimal flow of data based on the knowledge of the entire network. SDN Application is not a user application but is responsible for managing the flow of entries that are programmed on the network devices using the controller's API to manage flows.

AWS VPC

VPC stands for Virtual Private Cloud. The VPC can host subnets (private or public) on which EC2 instances can be launched. Each VPC can host a range of subnets and each subnet can host a range of IP addresses. There are different IP addresses that are reserved for some private functionality. There can be multiple subnets in a VPC, and each subnet can run across different availability zones. Each subnet will have instances in it. Every VPC has, by default a public route table, that gives it an internet connection. A VPC is a logical network and subnet are logical range of IP addresses. When an EC2 instance is created, it is created on the physical host. The physical host can hold multiple instances of varied sizes such as t3.2xlarge, t3.xlarge, t3.large, t3.medium, t3.small, etc.

Multi-tenancy is a special feature of cloud computing which means that multiple resources can share the same physical hardware. The latest virtualization technology is Nitro, and it has different layers in its architecture. All the network communication happening on the physical host will go through the Nitro network card. A virtual router will encapsulate the VPCs. Each VPC has its own unique ID. This allows it to differentiate the incoming and outgoing information despite having the same IP address.

Amazon has mapping services which knows the location of the VMs in the physical machines and the VPC and subnet routes. The virtual router will communicate with the mapping service for any information. For non-VPC destination routing, the Device might need to connect to the internet, or VPN, etc. A network gateway is a device or node that connects disparate networks by translating communications from one protocol to another. Amazon internet gateway is a virtual gateway that allows instances with public Ips to access the internet. This gateway allows a two-way traffic. Amazon NAT (Network Address Translation) gateway on the other hand is a service that allows instances with no public IPs to access the internet. This gateway allows only 1 way traffic (outgoing). The Blackfoot edge device handles everything.

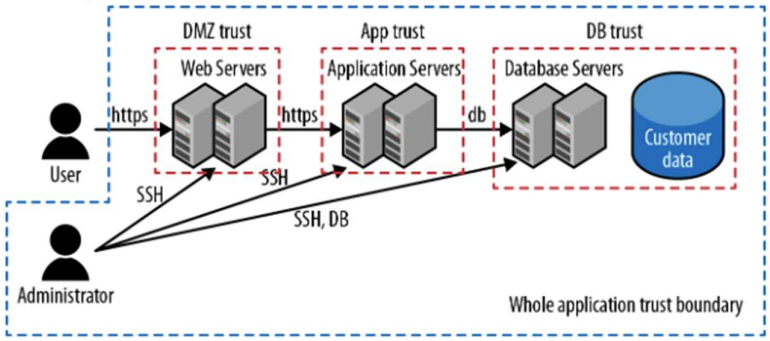
WEEK 8

Principles of Cloud Security

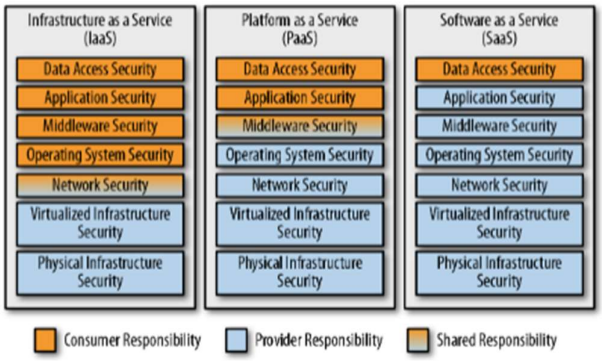
There are multiple cloud security principles, one of which is the Principle of Least Privilege. The principle states that a security architecture is designed so that each entity is granted minimum system resources and authorizations that the entity needs to perform its function. In most cases, the access policy should be denied by default. The cloud console is the ultimate front end of all cloud resources. This needs to be guarded and protected properly.

Another principle is called Defence in depths. This principle states that one should create multiple layers of overlapping security controls so that if one fails another one can catch the attackers. There needs to be a balance between over-protection and under-protection. To achieve this, the weak points need to have overlapping security features.

To understand security requirements, a diagram is created with the IT assets and the trust boundaries of these assets. Trust boundaries define the boundary within which any object can access each other and have a similar type of protection. Hence, if one object within the trust boundary is breached, all other objects within that trust boundary can also be said to have been breached. Usually, the diagram looks like the following:



Cloud utilises a shared responsibility model. In the traditional on-premises model, all the security is the organisation's responsibility, but in the cloud model, it is a shared responsibility between the consumer and the service provider. The shared responsibilities are shown below:



Data Asset Management and Protection

Data Assets are classified usually into 3 levels. These are:

Low

- Information may or may not be intended for public release, the impact would be low or negligible if it were released
- Examples:
  - Server's public IP address
  - Application log data with no personal data, secrete or value to attackers

Moderate

- This information should not be disclosed outside the organization without the proper nondisclosure agreements
- Examples:
  - System design documents
  - Personal information

High

- Vital information that may cause significant harm if disclosed
- Examples
  - Admin credentials like aws root account
  - Sentsitive data

There are different methods of protecting data in the cloud. These are:

- Tokenization

  These are typically used in credit card information management. Instead of storing the credit card information, tokens are generated for the credit card. It is not encryption but quite alike encryption in motion (being transmitted across a network), while in use (processed on computer's CPU and held in RAM) or wile at rest (on persistent storage).

- Compression and Encryption
  The most complicated part of encryption data is the handling on encryption keys. The encryption key is generated on the fly and is not saved but the data needs to be encrypted and stored and that is the most difficult task.

Simple key management service is to generate a key, encrypt the data with the key, store the key into the Knowledge Management System, and then store the data to the disk. The problem with this simple key management system is that it can not manage too many keys. This brings us to use 2-Level keys. In this a single master key is provided which can decrypt all the other keys which in turn can be used to decrypt the encrypted data.

Another protection method is encryption. Usually to be chosen between Server-side and Client-side encryption. Server-side encryption is when the cloud provider manages the process of encryption/decryption, but the providers have access to the unencrypted data. Client-side encryption is much more secure as providers do not have access but harder to manage.

Cloud Access Management and Protection

All physical assets are controlled, managed, and protected by the cloud providers. The assets are of different types such as Computing assets (VMs and containers), Storage assets (Block storage, file storage, cloud databases, etc.), Network assets (VPCs and subnets), etc.

Identity and Access Management

Each entity needs an identity. The process of verifying that identity is called authentication. Access management is about ensuring that entities can perform only the tasks they need to perform. The process of checking what access an entity should have is called authorization. IAM has these 2 as the main functionalities. More details about these are given below:

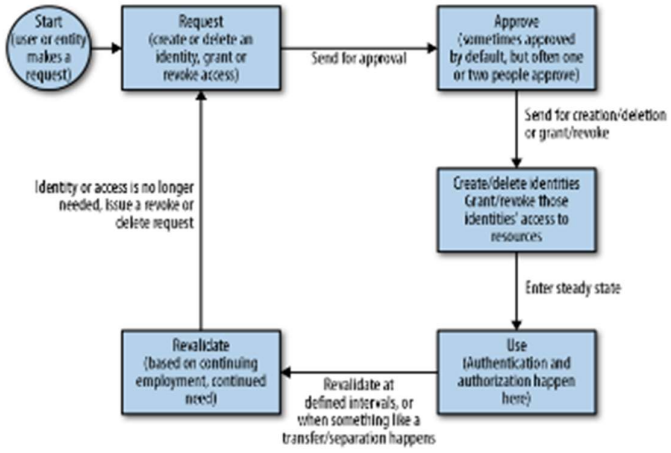
- Authorization
  The principles for this are the same as we saw before, the principle of least privilege under which each entity is granted minimum system resources and authorizations that the entity needs to perform its function. Another principle at work here is the principle of separation of duties in which a task is completed by more than one person. There are 3 types of centralized authorization, these are:
  - Policy Enforcement Point
    Implemented in the application. The application will not let you perform the function if you do not have access for it and this access is checked by the application through the policy decision point.
  - Policy Decision Point
    Implemented in the Centralized authorization system which checks the user identity against their policy document and decide whether to give access or not.
  - Policy Administrative Point
    Implemented in the centralized authorization system and is usually a web user interface where you can tell the system who is allowed to do what.
- Authentication

A database stores all the identities and a protocol is used to authenticate users and verify their identity based on this database. There are different types of authentications, such as:

- Authenticating organisation's employees with your cloud providers. Cloud IAM is used for this case.
- Authenticating organisation's customers with your cloud applications. Customer Identity Management System is used for this case.
- Authenticating organisation's employees with your cloud applications. Customer Identity Management System is used for this case.

Automated revalidation is the automatic removal of all resources once the environment is closed. Revalidation requires human judgement.

An IAM life cycle is given below:



AWS IAM

In a typical identity management system, we have 3 types of users. These are the root users with the user id 0, system users with the user id between 1-999, and local users with a user id >1000. Most of the Linux OS has a sudo group to represent users with the admin privilege whereas more groups can be created. The root user is disabled by default, even in AWS EC2 instances. In a typical identity management system in application, we have got account users with privileges and roles allotted to them, these privileges can be granted to the users and roles can be set.

IAM Components

There are 2 types of users, the account users and the IAM users. The IAM user is defined in the AWS account and the IAM group is a collection of IAM users that have been granted identical authorization. IAM policies defines which resources can be accessed and the level of access to each resource. IAM role is a mechanism to grant temporary access for making AWS service requests. IAM role can be assumed by a person, application, or a service.

IAM Groups and Roles

Groups and Roles refer to a similar mechanism of representing a collection of permissions that can be granted to different users. A user can be assigned to multiple groups or roles. The groups and roles all represent a collection of permissions. IAM roles are used to

handle the cloud specific authorization scenarios such as allowing EC2 instances to access S3, provide access to externally authenticated users, provide access to third parties, etc.

IAM Roles

The following are the characteristics of an IAM role:

- Provide temporary security credentials.
- Not uniquely associated to one person.
- Is assumable by a person, application, or a service.
- Is often used to delegate access.

IAM roles can be used to provide AWS resources to access AWS services, provide access to externally authenticated users, provide access to third parties, etc.

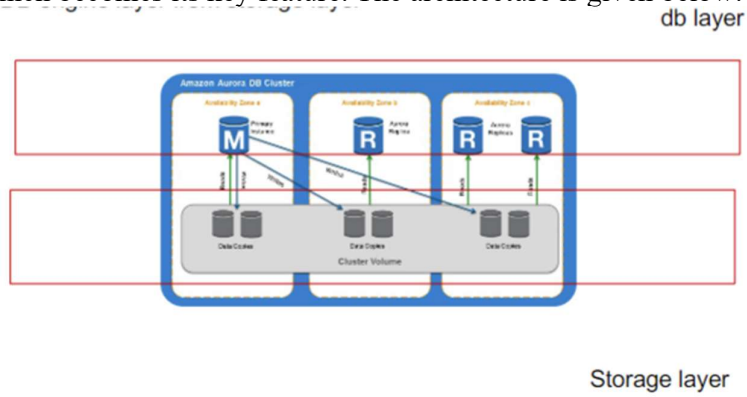
Access Controls

- Attribute Based Access Control (ABAC)
  A lot of cloud providers use ABAC which grants access based on the attributes of the resources, suers, actions, or environment. It is a much more flexible way of granting accesses. AWS provides ABAC using tags as attributes and the tag-based control is specified in the condition element of the policy.
- Role Based Access Control (RBAC)
  This is the traditional approach to grant access controls to the users. It grants users specific permissions based on the job function and creates a distinct IAM role for each permission combination. Permissions are updated by adding access for each new resource. This becomes time consuming to keep updating policies.

WEEK 9

Aurora Architecture

Amazon aurora is a relational database service attempting to build SQL in a distributed environment. It runs on modified MySQL/PostgreSQL. The main idea of Aurora is to offer a service with full SQL support but with high scalability. It is an innovative way of disaggregation of compute and storage architecture. Aurora has a separate DB engine layer and storage layer which becomes its key feature. The architecture is given below:



More details about each layer are given below:

- DB Engine Layer
  From the client perspective, the DB layer is quite like the MySQL configuration. It can have a primary (writer) and multiple replicas (readers) of Database. The actual data is not stored on the instance that runs the DB engine. Aurora supports a much larger data

volume than a traditional RDBMS can (1 writer and 15 readers). The latest update committed to the database will be reflected in the memory of all available DB instances.

- Storage Layer
  The storage layer is hidden from the client perspective. It is a group of EC2 instances with local SSD and are scattered in different Availability Zones. They persist and replicate the data for the database. The database is partitioned in the storage layer.

Durability at Scale

Replication is a typical way to achieve durability as well as availability and scalability. Typical replication factor in most distributed storage system is 3. Most systems use the quorum mechanism to ensure data consistency. In this mechanism for any request, 2 replicas should respond and there will always be an overlapping replica. In large scale distributed systems with multiple availability zones, replicas are distributed in different Availability Zones as well to minimize concurrent failure.

Amazon Aurora: log as database

To answer a write request, a database engine usually needs to append the write ahead log and update the actual data file. Several other data must also be written in practice such as the binary log, the double-write, metadata file, etc. During the write operation, only redo logs are sent across the network by primary instance. Replicate instances receive redo logs to update their memory. Storage nodes receive redo logs, they will store the log for some time before the update the database page. Primary instance waits for majority acknowledgements to consider the write as successful.

IO traffic in Aurora Storage Nodes

The steps for IO traffic in Aurora Storage Nodes are:

- Receive the log record and add it to an in-memory queue.
- Persist record on the disk and acknowledge.
- Organise records and identify gaps in the log since some batches may be lost.
- Gossip with peers to fill in the gaps.
- Coalesce log records into new data pages.
- Periodically stage log and new pages to S3.
- Periodically garbage collects old versions.
- Periodically validate the CRC codes on the pages.

The log marching forward

To fill in the gaps in the storage nodes, the system relies on the log sequence number. The log sequence number is attached to each log. It is assigned by the primary instance, and it can generate log sequence numbers to different instances. A database primary instance handles multiple transactions at any time point and operations in the transaction are either all executed or not at all. Log record of the last operation in the transaction represents a possible consistency point. Storage service might have different storage status at any time.

Combining DB view and storage view

During crash recovery, the storage service determines the highest Log Sequence Number for which it can guarantee availability of all prior log records (Volume Complete Log Sequence Number). The database layer can specify a few Log Sequence Numbers as a consistency point called the Consistency Point Log Sequence Number. VDL (Volume Durable Log Sequence Number) is the highest Consistency Point log sequence name than or equal to Volume Complete

Log Sequence. Primary instances gossip with replica instance to establish common view of VDL (Volume Durable Log Sequence) and other information.

Using the normal write operation, storage nodes gossip with peers to fill in the gaps of missing logs after it acknowledges the write request. The write request is considered successful after the write quorum is satisfied and the database can advance its VDL During the normal read operation, the DM engine will search its memory first and may return without any disk IO involved. When there is a missing data, it needs to real data from storage nodes. Read quorum is not established consensus. Database specifies a read point represented by VDL and it then select the storage node is complete with respect to VDL.

Replication and LSN example

Assume an aurora database has 15 GB of data and is partitioned into 2 groups, PG1 and PG2. It has one primary and 4 replicas in 3 Availability Zones. The primary instance is in Availability Zone (AZ) 1 and 2 replicas each in AZ2 and AZ3. There are 10 storage nodes in 3AZs distributed as AZ1:3, AZ2:3, AZ3:4. The replication is as follows:

- PG1 is replicated in SN_{11}, SN_{12}, SN_{21}, SN_{22}, SN_{31}, SN_{32},
- PG2 is replicated in SN_{11}, SN_{13}, SN_{22}, SN_{23}, SN_{33}, SN_{34}

Following is a sample of transactions:

- X targets PG1; It has the following redo logs
  - PG1: 1001, 1002, 1003
- Y targets both PG1 and PG2; It has the following logs
  - PG2: 1004, 1005
  - PG1: 1006, 1007

The 4 replicas are the destinations for all the redo logs.
Therefore

- 1001, 1002, 1003, 1006, & 1007 will go to SN_{11}, SN_{12}, SN_{21}, SN_{22}, SN_{31}, SN_{32},.
- 1004 & 1005 will go to SN_{11}, SN_{13}, SN_{22}, SN_{23}, SN_{33}, SN_{34}.

If a write request is expressed in redo log 1001, the primary will send the redo log to all read replicas and SN_{11}, SN_{12}, SN_{21}, SN_{22}, SN_{31}, SN_{32},. If 4 out of the 6 responds, the write is considered successful. Read replicas only use the log to update the in-memory data.

Assuming the SN_{11} is the node of interest and has received logs 1002, 1003, 1001, and 1006. As each storage node only processes a subset of the logs, each log has a backtrack link identifying the previous log of that PG.

Through the database view, the consistence point is 1003 (representing the last log transaction of transaction X. and 1007 as the last log transaction of transaction Y. The storage view shows us that PG1 has received 1001, 1002, 1003 & 1006. PG2 has received 1004 & 1005. VCL is 1006 but is in the middle of transaction Y. VDL is 1003.

WEEK 11

Containerization

It is a type of virtualization technique. Operating system level virtualization is called containerization. It refers to an operating system feature in which the kernel allows the

existence of multiple isolated user-space instances. These instances are called containers (partitions, virtualization engines, etc.).

Linux Kernel

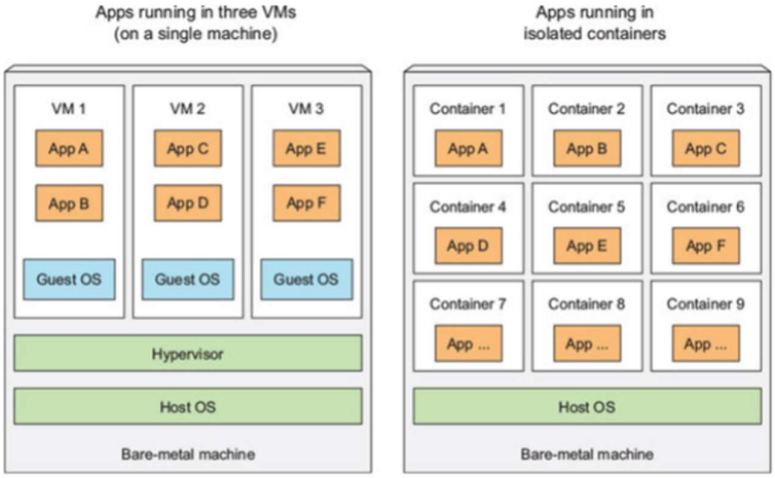
Linux kernel is the most basic component of a Linux Operating System. It does 4 basic jobs: memory management, process management, device drivers, and system calls and security. A Linux system is composing of the kernel and system libraries and tools. There are 2 features of Linux kernel which enables containerization. These features are:

- Namespace
  Namespace refers to the different resources managed by the OS. The goal is to run a lot of containers on a single OS and to create an illusion of a full computer system for each container, a copy of the kernel resources is given to the containers. Each container will then have a root directory, a set of process IDs, an independent set of user IDs, etc. To avoid conflict, the OS provides a feature called namespace which provides containers their own view of the system. There are different kinds of Namespace, such as:
  - Mount: This deals with file systems. It allows each container to have its own root and manage its own mount points.
  - Process IDs: Each container has its own numbering starting at 1. When PID 1 goes away, all other processes exit immediately. PID namespaces are nested, and the same process may have different PIDs in different namespaces.
  - User ID: It provides user segregation and privilege isolation. There is mapping between the container UID to host UID.
- Cgroup
  Control groups is another kernel feature which allows the OS to allocate resources to the containers. It will also monitor and limit the usage of the resources which may include memory, CPU, File, network, etc.

Container Runtimes

Runtime is the life cycle of a program. Container runtime has similar responsibilities as a Java Runtime Environment. It enables the containers to run by setting up namespaces and cgroups. It can also be viewed as the counterpart of a hypervisor. Low level container runtimes focus on just running the containers. High level container runtimes contain lot of other features like defining image formats, managing images, etc. An example of it is Docker.

Container vs Virtual Machine



There are a lot of similarities between containers and Virtual Machines but there is a huge difference between them. Each virtual machine runs of a guest operating system whereas the containers do not need that many Operating systems.

Container vs Server Virtualization

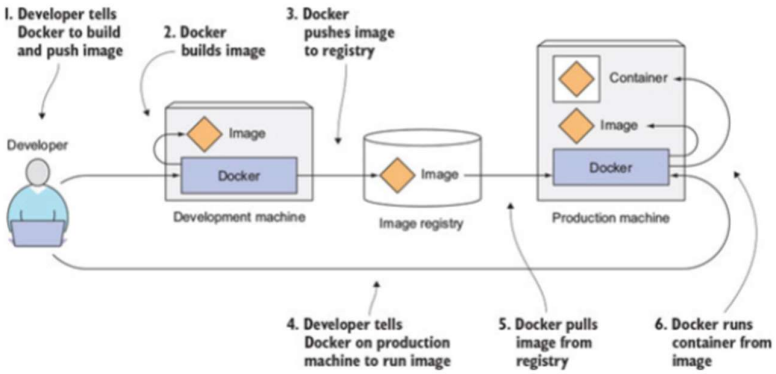
A virtual machine is running a full-fledged OS inside it. It will take some time to boot up the OS. In contrast, the container does not need that many resources as it does not have an OS inside of it and if it isn't working, it will not use any resources. Compared to a VM, the container is much faster to start. VM has very good isolation and security compared to which containers provide a reasonable level of isolation and security using kernel techniques such as namespace and control groups.

Docker

It is the most famous container runtime. It is a packaging and development system built on the container technology. There is a large ecosystem of various components. The container in the docker ecosystem is presented as a black box where docker users do not need to know how it works inside. The users mainly use the dependency management and deploy everywhere features of Docker. Docker has 3 main concepts associated with it:

- Images:

  A docker based container image is something you package your application and its environment into. It is defined in Dockerfile, which is a text document containing a sequence of instructions to build and run your application. In this text document, dependencies are declared, environment variables and configurations are set, etc. These are composed of read-only layers. Each command put in the Dockerfile would be saved as a read-only layer. Base layers can be shared by many images. When an image is loaded on the container to run, the container adds a thin writable layer on top of it. All 'writes' to the container are stored in this layer. This container layer will be deleted when the container is deleted and can be used only as a temporary storage. If multiple images are using the same base layer, only one copy of the base layer is required in the container. When the container starts, only a new writable layer is added on top of the existing image layers. If an image is shared between multiple containers, each container will have its own write layer for the image.
- Registries: Docker registry is a repository that stores docker images and facilitates easy sharing of those images between different people and computers.
- Containers: A docker based container is a regular Linux container created from a Docker based container image. A running container is a process running on the host running on Docker. The actual containers use many OS technologies to provide isolation and to allocate resources. This is achieved by using various features provided by the Linux Kernel such as Namespaces, Cgroups, etc.



The above is a process of how the concepts work. The developer will create an image on Docker, this image once created is pushed (saved) into the image registry which then gets stored on a production machine which can run the process (image).

Apart from storing data within the writable layer of a container, there are other preferred ways to persist data. Some of these are:

- Bind mount: An early version storage option where it allowed the client to mount any file or directory on the host machine into a container.
- Volumes: A more secure version of storage option wherein it uses a designated location in the host machine as a container storage managed completely by Docker.
- Tmpfs: It is a rarely used option which uses host memory to simulate storage.

Docker also provides a few options for networking. It provides a few features such as:

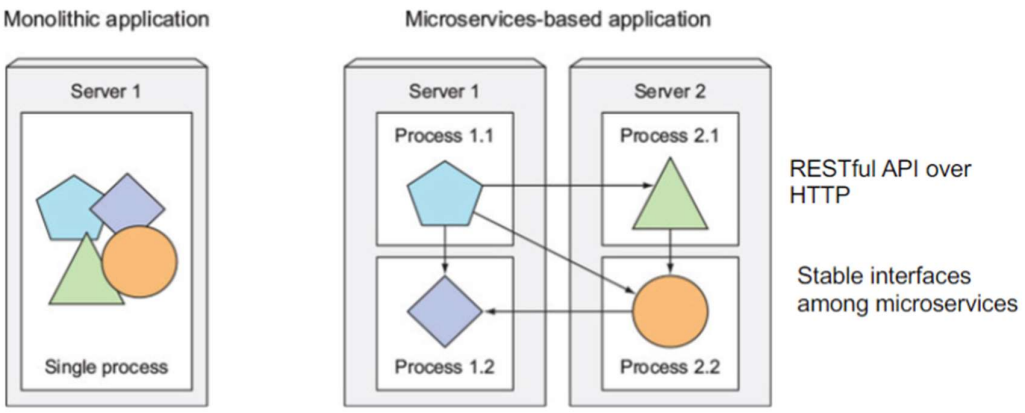
- Host
  Host driver is the simplest option. It is not very secure as it removes the isolation between the container and the host. The container is treated in the same way as a process in the host. It connects directly to the network interface of the host. This option prevents the client from running multiple web servers on the same host. The command lien is given below:

```
docker run -d --name nginx-1 -net=host nginx
```

- Bridge
  Docker manages its own private network. In the bridge method, a software bridge is added to the host. The mapping from the private address to the public address is done through NAT (Network Address Translation) gateways.

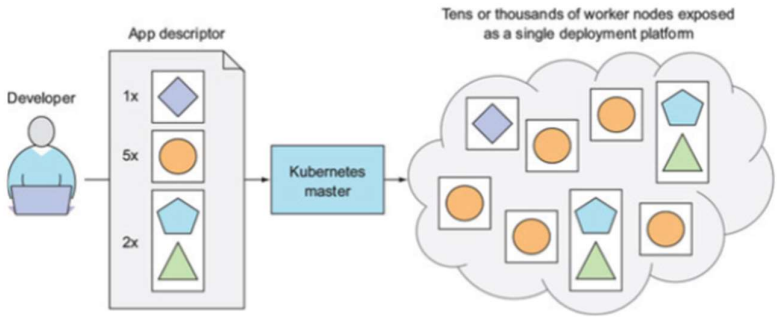
Kubernetes

Google has been running applications in containers since early 2000s. The predecessor of Kubernetes is Borg and Omega, Google's internal container-oriented cluster management system. Kubernetes was introduced in 2014 which helps in achieving better utilization of the infrastructure. Software architecture has evolved from a monolithic architecture to micro-services-based application as seen below:



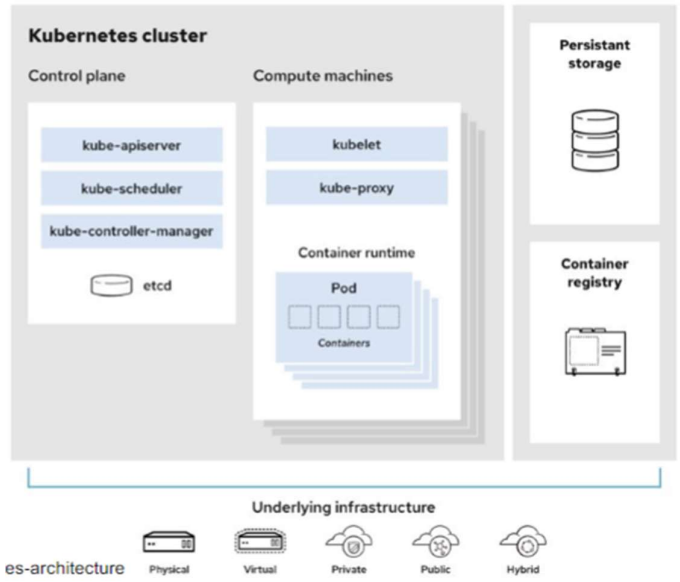
The monolithic architecture will not work well for complicated applications as even the failure of one process will lead to the entire group of processes to be run again. Today we use microservice based applications wherein the application is divided into multiple processes and these processes run on multiple servers. This ensures that even if one process fails, other processes continue to run, and the failed process can be corrected without affecting the others.

Each microservice has its own development and release cycle. They are likely to have conflicting environment requirements, and this is managed by container technology. When the system consists of so many microservices, automated management is crucial as services need to communicate with each other for a smooth flow. Kubernetes is an application automatic deployment system. The basic use case of Kubernetes is to deploy container based on the application as a distributed system. Docker focuses on single service deployment. Most production systems are deployed as distributed systems instead of the monolithic application approach. A simple illustration of Kubernetes is given below:



Kubernetes expose the whole data center as a single deployment platform

Architecture of Kubernetes cluster



The control plane has several things which is responsible for the overall control of the environment.

The computing machines (Kubernetes pod) is part of the Kubernetes Cluster. A pod is the unit of scheduling in Kubernetes. It is a resources envelope in which one or more containers run. Containers that are part of the same pod are guaranteed to be scheduled together on the same machine and can share state via local volumes. One Kubernetes pod can run multiple Pods. A Kubelet utilizes Cgroups and Traffic control to reserve computing resources for the Pod. Kube-proxy maintains network rules on the nodes. Container runtime (For example Docker) is the software responsible for running the containers inside the pod.

Running an application in Kubernetes

The basic steps to run an application in Kubernetes is given below:

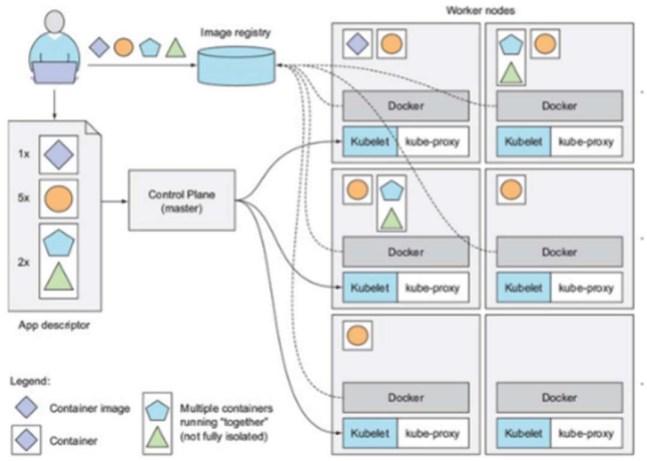
- Package the application into one or more container images. (Similar to creating images in Docker)
- Push those images into the image repository. (Similar to pushing the images into an image repository in Docker)
- Post a description of the application to the Kubernetes API server (Similar to providing a read only text to Docker). This description should include the following:
    - Images that represent the component of an application.
    - Relationship between images.
    - Desirable copies of the components supposed to run.

In docker we had the Dockerfile to package the application, in Kubernetes, we have the declarative configuration object. The docker image has a step-by-step instruction which is immutable once created, this is called imperative configuration. Kubernetes uses Declarative configuration in which it defines the desired state of the system. Kubernetes is responsible to ensure the actual state matches the desirable one and the user does not have to provide any information here as Kubernetes does it by itself. An example for the difference between Imperative Configuration and Declarative configuration is given below:

Example of having three replicas of some software

- Imperative configuration: `run w on n1; run w on n2; run w on n3`
- Declarative configuration: `replica of w equals 3`

The basic working mechanism of Kubernetes is as follows:



WEEK 12

Cloud Services

Something as a Service is called XaaS. The main competing services was IaaS and PaaS around late 2000s. IaaS was the first model embraced by the market. The early cloud users wanted to recreate the same local computing environment on the cloud and the downside was that developers needs to manage the VM to set up the environment. This needed the developers to have a lot of administrative knowledge. The success of cloud gave users more trust on the technology and users then started to give up control for simpler operation. Hence, what started happening was that providers did more, and client did less.

Function as a Service

It is not suitable for every kind of use cases. Serverless computing originated as a design pattern for handling low duty cycle workloads such as processing in response to infrequent changes to files stored in cloud. Foe example sending images from the phone application to the cloud. In 2015, Amazon released AWS Lambda Service where the user writes a function, and whenever the function is involved, Amazon would start the necessary environment and run it. From the perspective of the user, the servers are not being used but in the background the servers are being used. The user is charged based on the actual execution time.

	Characteristic	*AWS Serverless Cloud*	*AWS Serverful Cloud*
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.0000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible

Table 2: Characteristics of serverless cloud functions vs. serverful cloud VMs divided into programming and system administration categories. Specifications and prices correspond to AWS Lambda and to on-demand AWS EC2 instances.

There are critical distinctions between Serverless and Serverful cloud, these features of Serverless cloud are:

- Computation and storage are scaled independently, computation is stateless, and state is saved somewhere else.
- User provides a piece of code, and the cloud automatically provides the resources to execute that code.
- This enables the user to pay in proportion to resources used instead of for resources allocated.

Serverless Computing

A function usually runs in a container or other type of sandbox with limited resources launched by the provider. In comparison to PaaS, serverless computing is much better at autoscaling and billing, it has stronger isolation and has better platform flexibility. The following is a comparison between container orchestration enabled by Kubernetes and Serverless computing:

Kubernetes	Serverless Computing
Similar short lived computing environments and much less limitations	Offloads more responsibilities to the providers.
Programmer has more control and hence more responsibilities	Fine grained charging, charged proportional to resources used.

Serverless is attractive due to various reasons. It gives customers the perspective that they have unlimited resources. The customers are attracted towards the programming productivity and cost saving as it charges based on resources used. Providers look at the business growth brought about by bringing in new customers and helping existing customers use more service types. By having more predictable resource usage, they can utilize the resources much more efficiently. Cloud providers can also utilize less popular computers.

The biggest limitation of Serverless computing is storage, but that makes it appropriate for certain things. Serverless relies on cloud storage services for maintaining states, certain

applications have fine-grained state sharing needs and that may suffer with latency caused by retrieving and storing states and the frequent querying will lead to higher costs. Object storages like AWS S3, Google cloud storage, etc. has low storage cost but high access cost and high access latency. Key-Value databases such as Amazon DynamoDB, Google Cloud Datastore, etc. are expensive and take a long time to scale up. Another point to remember is that the in-memory storage is still in its infancy.

There are also limitations on coordination. There is no solution to support workload that needs coordination among various serverless functions as these functions are very small. The notification mechanisms need to be managed by the developers as well meaning that the developers will need to write customized code for each notification.

Another limitation is by communication where serverless architecture may have poor performance for standard communication patterns. Functions need to communicate on the run on different VMs and a lot more messages are needs for standard communication primitives.

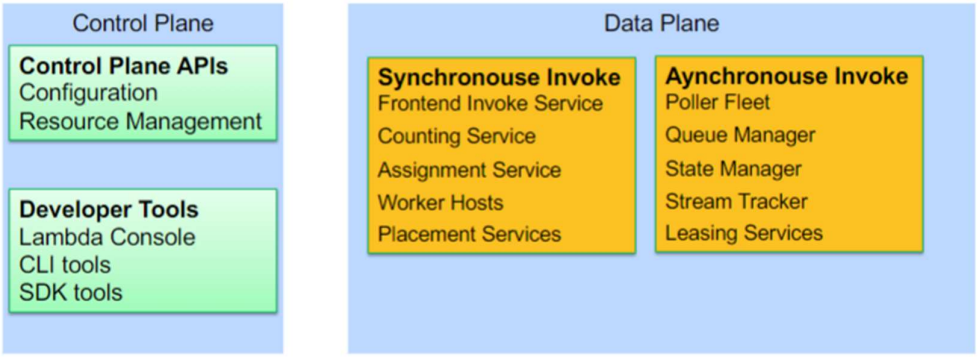
Another limitation is on performance predictions. There could be start latency (cold start) which is the time taken for starting the cloud function, initializing the software environment, application specific initializations, etc. Providers try to minimize this by having a warm environment. Performance variations can be caused due to the underlying hardware as well.

Serverless computing need to overcome multiple challenges which have been given below:

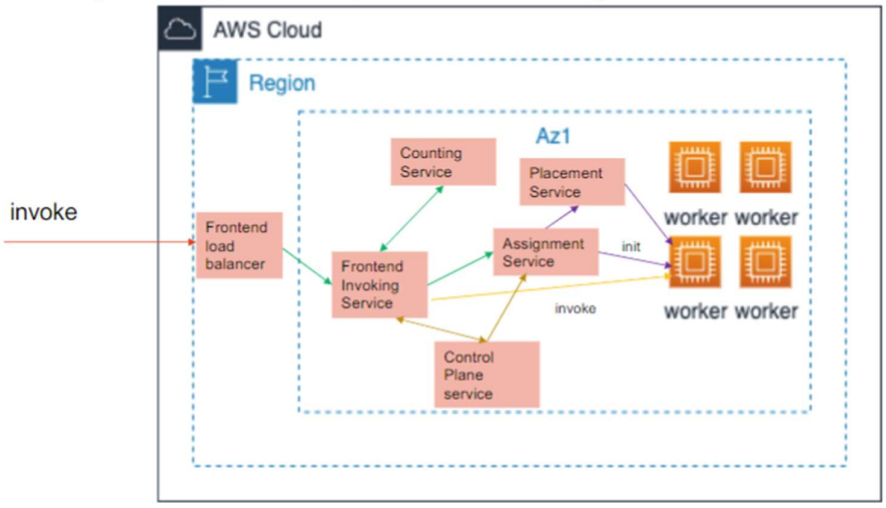
- Abstraction challenge
  In the current serverless offering, the developer specifies the cloud functions memory size and execution time limit, but not other resource needs. Serverless computing should give the option of giving developers more freedom to specify CPU and other resources, but this will make it hard for provider to schedule efficiently and make it hard for the developer to estimate efficiently.
- Data Dependencies
  Serverless platforms have no knowledge of data dependency among functions. Serverless computing could allow users to specify computation group consisting of functions to the provider.
- System Challenge
  There are system challenges on the storage and coordination service fronts. They can come up with cache like ephemeral storage and high-performance durable storage. They should also minimize the startup time by developing new lightweight isolation mechanisms.
- Network Challenge
  Effective communication is lacking in serverless computing. Effective communication can be achieved by application and data aware functions scheduling mechanism. For this, more information needs to be shared with the cloud provider.

Amazon Lambda Architecture

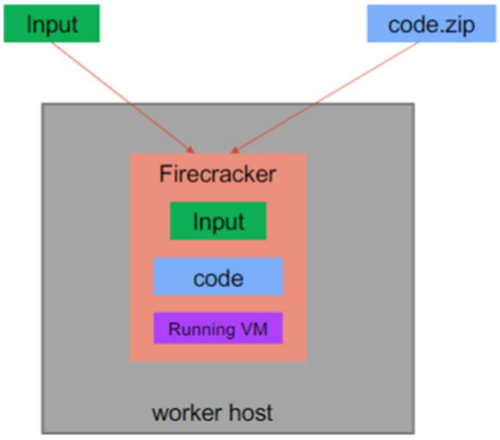
It has a control plane and a data plane. The control plane has the control plane API and developer tools. Data plane provides 2 types of invocation: the synchronous invoke and the asynchronous invoke. A snapshot of the AWS Lambda architecture is given below:



A function invoke process (synchronous) looks the following:



An invoke comes into the AWS cloud. As the region supports multiple availability zones, the frontend load balancer receives this invoke after careful consideration through a particular availability zone. Each availability zone has several workers (For example an EC2 instance). Several internal components work together as seen above, to pick a suitable worker and set up the environment. The worker looks like the following:



A worker host is like the Bare-metal instance. This does not work on a physical host, but a VM that is the Bare-metal Instance. The firecracker is a KVM (Kernel Based Virtual Machine) based Hypervisor. This does not need a lot of resources and is a minimalized VM. It enjoys all the VM features. The worker needs an input and the code zip file. The code is stored

and downloaded from S3. The input on the other hand would be an event object due to invoke. The VM will run the code, and then the VM will be removed after execution.

EXAMPLES

`"s3:x-amz-server-side-encryption-aws-kms-key-id": "true"`	This means that the object has server-side encryption, meaning that the provider is handling the encryption/decryption process. This means that the provider has access to the data.
`"aws:ResourceTag/Project": "DataAnalytics"`	Specifies the ABAC control structure which uses tag and value pair.

EXAM BASED NOTES

Dynamo and Vector Clock

Suppose we have a Dynamo ring that consists of five nodes. Each node is assigned one or two tokens in the ring space [0,99]. The respective token(s) for each node are as follows:

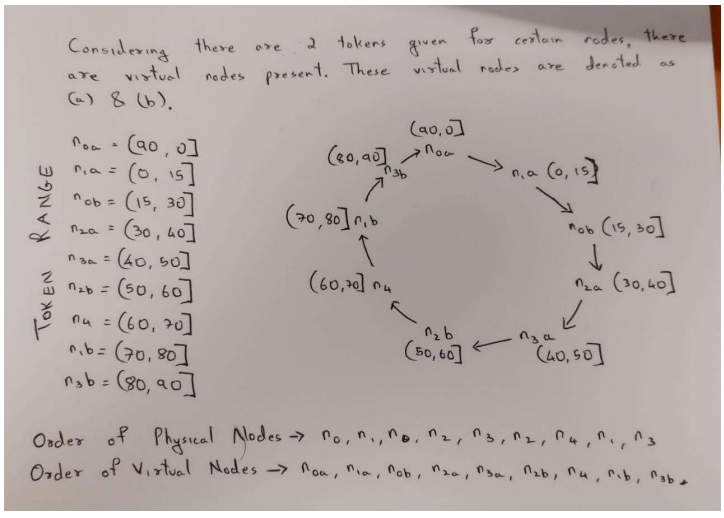
- n_0: 0, 30
- n_1: 15, 80
- n_2: 40, 60
- n_3: 50, 90
- n_4: 70

We store data about university faculties in this Dynamo instance. The data is keyed with the faculty. Table 1 shows a few sample keys and their corresponding hashes.

Table 1: Sample keys and hashes

Key	Arts	Business	Education	Engineering	Law	Medicine	Science
Hash	31	93	29	13	71	47	43

- What are the token range(s) of each node? What is the order of physical nodes in the ring?



- Suppose node n3 receives a request to insert a record with key "Law". Which node will it forward the request to?
  n3 will forward the request to **n1**, which is the coordinator node for key "Law". n1 is responsible for the hash range (70,80].
- Which other nodes will eventually get a copy of the inserted data?
  n3 and **n0** will also receive a copy of the data. They are the two clockwise successor nodes of n1 (the coordinator) in the ring. n3 is responsible for the token range(80,90], n0 is responsible for the token range (90,0].

We assume that the consistency configuration (N,R,W) of the system has the value (3,2,2). The preference list consists of 4 nodes. Answer the following questions:

- The record with key "Engineering" currently has three replicas all with the vector clock([n1,1]). Suppose node n1 is temporarily not available between the time points t1 and t2. A write request on key "Engineering" is received by node n4 during this period. Which node will coordinate the write? What will the vector clock be after the write request is processed?
  node n0 is the next on the preference list and it will be **the coordinator** of the write request. It will add a new pair to the vector clock of "Engineering" record. The vector clock becomes ([n1, 1], [n0,1]). It then sends the data together with the new vector clock to **n2 and n3** (As n2 and n3 fall right after n0, the coordinator node and W=2)**.** If either n2 or n3 replies, the write is considered as successful.
- Suppose n1 comes back to life after t2 and receives a read request for key "Engineering". How will this request be processed and what will be returned to the client?
  There are several possible scenarios. Assuming n3 has replied in the previous question:
  i. **Node n3** sends the hinted data to n1 **before the read request.** This means n1 has the data with vector clock **([n1, 1], [n0,1])**. It will send the read request to **n0 and n2** (The next 2 physical nodes in line) and wait until at least one of them replies. If **n0 or n2 replies**, it will be the version with vector clock ([n1, 1], [n0,1]). n2 may reply with the new data or old data; it depends on if the write is successful on n2**. If n2 returns the old version, n1 can instruct it to write the new data.**
  ii. **Node n3** has **not sent** the hinted data to n1. n1 now has the old version with **vector clock([n1,1]).** It will send read request to n0 and n2 (as they are next in line) and wait until at least one of them replies. If any of them reply with the new data, with vector clock ([n1, 1], [n0,1]). n1 can tell from the vector clock that the version it maintains has been overwritten by the new version and will replace it with the new version. The new version will be sent back to the client as the reply. If n2 replies with the old version because it did not perform the write successfully in the previous question, n1 now has two versions with the vector ([n1,1]), it will send back the old version as reply. This illustrates the eventual consistency feature of sloppy quorum.

Spanner Deployment with Paxos algorithm

Question 1: Paxos Concurrent Proposals and No Message Loss

We use the same Paxos agents as in lecture 6: two proposers A and B, three acceptors X, Y and Z and a learner L. The message sequence between various agents is shown in Figure 1.

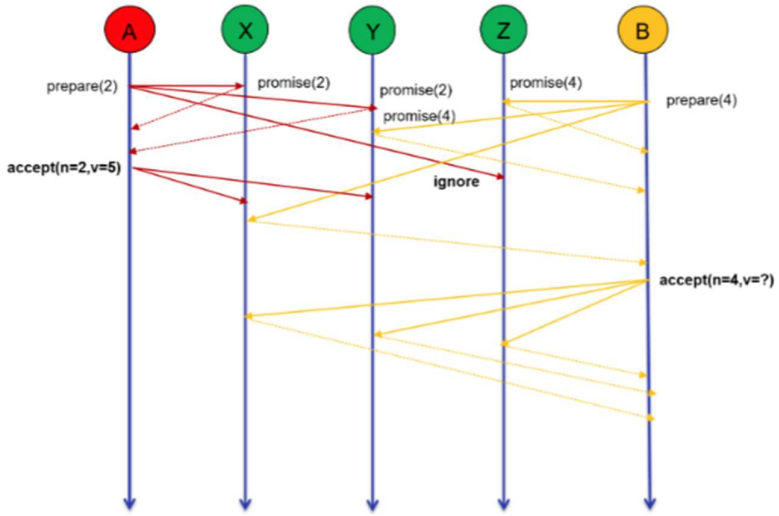


Figure 1: Concurrent Proposal from Proposer A and B

- How would acceptor X and Y respond to the accept message from proposer A.
  Acceptor X would accept the proposal. Acceptor Y would ignore because it has already promised not to accept any proposal with number less than 4.
- How would acceptor X respond to the prepare message from Proposer B?
  It will respond with a promise not to accept any proposer with number less than 4; It will also include proposal(n=2, v=5) in the response.
- What would be the value in Proposal B's accept message?
  The value should be 5, based on response from acceptor x.
- How would all acceptors respond to proposer B's accept message?
  All of them will accept the proposal (n=4, v=5).

Question 2: Paxos Concurrent Proposal with Message Loss

Again, we use the same Paxos agents as in lecture 6: two proposers A and B, three acceptors X, Y and Z and a learner L. Assume that both proposers have completed phase 1 as described in lecture 6. The accept message sent from proposer A arrived successfully in all acceptors. The accept message sent from proposer B with proposal:n=4,v=10, only reached acceptor X, the other two messages are lost. The message sequence is shown in Figure 2. The sequence also includes a second prepare message as described in part b).

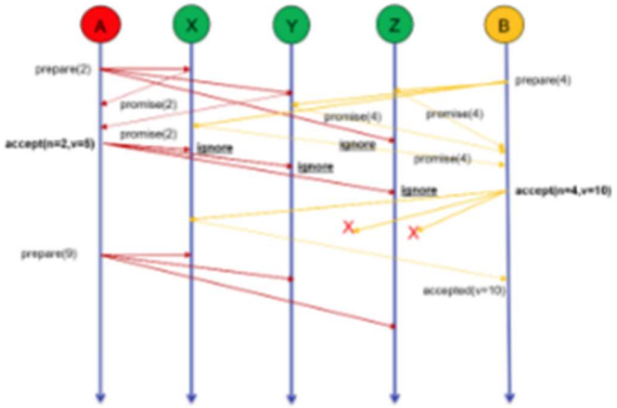


Figure 2: Paxos with Concurrent Proposal and Mesage Loss

- Does the system have any value chosen at this moment?
  No. Since only one acceptor receives the accept message and responds. The value is not chosen.
- Now proposer A wants to make a new proposal with value 15. It chooses a proposal number 9 and sends a prepare message to all acceptors. How would each acceptor respond (not respond)?
  Acceptor x would respond with the promise that it will not accept any proposal with number less than 9. It also attached the proposal it accepted in the response which is: {n=4,v=10}. Both Y and Z would respond with the promise that they will not accept any proposal with number less than 9
- Assuming all responses arrive at proposer A; following the algorithm, would the proposer continue with phase 2 by sending out an accept message? If so, what would be the proposal it requests to accept in this phase?
  Since all acceptors respond and there is no message loss, the proper would go ahead with phase 2. One acceptor responds with the proposal:(n=4, v=10),the proposer would send a an accept message with proposal:(n=9, v=10).
- Assuming no message loss in phase 2, will there be a chosen value and what would be the chosen value?
  With no message loss, there will be a chosen value and the value is 10.

Question 3: Spanner Architecture and Data Model

Consider a Spanner schema that stores user data along with their friends and posts. The schema includes a table called `Users`, which is defined as a directory table. This table has two descendant tables, namely `Friends` and `Posts`. The primary key for the `Users` table is `userID`, while the primary keys for the `Friends` and `Posts` tables are `(userID, friendID)` and `(userID, postID)`, respectively.

Below are sample data of two users

Users(1)
Friends(1, 20)
Friends(1, 35)
Posts(1, 1)
Users(2)
Friends(2, 1)
Friends(2, 5)
Posts(2, 100)

Let us assume that directory 111 contains the data for user 1. This directory is replicated on three Spanservers, namely Spanserver 1_a in zone 1, Spanserver 2_b in zone 2, and Spanserver 3_c in zone 3. Similarly, directory 222 contains the data for user 2, and it is replicated on Spanserver 1_b in zone 1, Spanserver 2_b in zone 2, and Spanserver 3_b in zone 3.