Disclaimer

This document summarizes the lecture slides of *Cluster and Cloud Computing* and thus is for the use of students enrolled in COMP90024 only.

This document is distributed on an "as is" basis, without warranties or conditions of any kind, either express or implied. No further support will be provided for this document.

Lecture 1

Terminology

- Centralized system
  - Single physical (centralized) system. All resources (processors, memory and storage) fully shared and tightly coupled within one integrated OS
- Parallel system
  - All processors either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Inter-process communication through shared memory or through some form of message passing
- Distributed system
  - Multiple autonomous computers with their own private memory, communicating through some form of message passing over a computer network.

Cloud Characteristics

1. on-demand self-service
   - A consumer can provision computing capabilities as needed without requiring human interaction with each service provider.
2. network access
   - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous client platforms.
3. resource pooling
   - The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model potentially with different physical and virtual resources that can be dynamically assigned and reassigned according to consumer demand.
4. rapid elasticity
   - Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly upon demand.
5. measured service
   - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of services.

勞力是無止境，活著多好 不需要靠物證，也不以高薪高職高級品搏尊敬。

陀飛輪

Lecture 3 Distributed and Parallel Computing Systems

Acronyms

- HTC High Throughput Computing
- HPC High Performance Computing
- ALU Arithmetic Logic Unit
- FPU Floating-Point Unit

Compute Scaling

- Vertical Computational Scaling
  - Have faster processors
    * n GHz CPU $\rightarrow 2n$ GHz $\Rightarrow$ 2x faster
    * Easy to do but costs more
  - Limits of fundamental physics / matter
- Horizontal Computational Scaling
  - Have more processors
    * Easy to add more; cost increase not so great
    * Harder to design, develop, test, debug, deploy, manage, understand
    * Transistor count still rising
    * Clock speed flattening sharply
  - HTC far more important than HPC for most folks

Options

- Single machine multiple cores
  - Typical laptop / PC / server
- Loosely coupled cluster of machines
  - Pooling / sharing of resources
    * Dedicated vs Available only when not in use by others
    * Web services
- Tightly coupled cluster of machines
  - Typical HPC /HTC set-up (SPARTAN)
    * Many servers in same rack / server room (often with fast message passing interconnects)

- Widely distributed clusters of machines
- Hybrid combinations of the above
  - Leads to many challenges with distributed systems
    * Shared state
    * Message passing paradigms - dangers of delayed / lost messages

Limitations

$T(1)$ time for serial computation
$T(N)$ time for N parallel computations
$S(N)$ speed up

Proportion of speed up depends on parts of program that cannot be parallelized.

Amdahl's Law

If 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20 times, no matter how many processors are used, i.e. if the non-parallelizable part takes 1 hour, then no matter how many cores you throw at it, it will not complete in less than 1 hour.

Oversimplification

Consider a program that executes a single loop, where all iterations can be computed independently, i.e. code can be parallelized. By splitting the loop into several parts, e.g. one loop iteration per processor, each processor now has to deal with loop overheads such as calculation of bounds, test for loop completion etc. This overhead is replicated as many times as there are processors. In effect, loop overhead acts as a further (serial) overhead in running the code.

Amdahl's law also assumes a fixed problem size. It cannot predict length of time required for some jobs, e.g. state space exploration or differential equations that do not solve.

$$T(1) = \sigma + \pi$$
$$T(N) = \sigma + \frac{\pi}{N}$$

$$S(N) = \frac{T(1)}{T(N)} = \frac{\sigma + \pi}{\sigma + \frac{\pi}{N}} = \frac{1 + \frac{\pi}{\sigma}}{1 + \frac{\pi}{\sigma}\frac{1}{N}}$$
$$\frac{\pi}{\sigma} = \frac{1 - \alpha}{\alpha}$$

$$S(N) = \frac{1 + \frac{1-\alpha}{\alpha}}{1 + \frac{1-\alpha}{\alpha}\frac{1}{N}} = \frac{1}{\alpha + \frac{1-\alpha}{N}} \approx \frac{1}{\alpha}$$

α the proportion of the program that cannot be parallelized and must be executed sequentially

$$S(N) = \frac{T(1)}{T(N)} = \frac{T(1)}{\alpha T(1) + (1-\alpha)T(1)\frac{1}{N}} = \frac{1}{\alpha + \frac{1-\alpha}{N}} \approx \frac{1}{\alpha}$$

Gustafson's Law

Gustafson's law proposes that programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve. Therefore, if faster equipment is available, larger problems can be solved within the same time. The law redefines efficiency, due to the possibility that limitations imposed by the sequential part of a program may be countered by increasing the total amount of computation.

$$T(1) = \sigma + N\pi$$
$$T(N) = \sigma + \pi$$

$$S(N) = \frac{T(1)}{T(N)} = \frac{\sigma + N\pi}{\sigma + \pi} = \frac{\sigma}{\sigma + \pi} + \frac{N\pi}{\sigma + \pi} = \frac{1}{1 + \frac{\pi}{\sigma}} + \frac{N\frac{\pi}{\sigma}}{1 + \frac{\pi}{\sigma}}$$

$$S(N) = \frac{1}{1 + \frac{1-\alpha}{\alpha}} + \frac{N\frac{1-\alpha}{\alpha}}{1 + \frac{1-\alpha}{\alpha}} = \alpha + N(1-\alpha) = N - \alpha(N-1)$$

Computer Architecture

- CPU for executing programs
  - ALU, FPU, . . .
  - Load / store unit
  - Registers (fast memory locations)
  - Program counter (address of instruction that is executing)
  - Memory interface
- Memory that stores / executes programs and related data
- I/O systems (keyboards, networks)
- Permanent storage for reading / writing data in / out of memory
- The balance of all of these are of key importance (especially for HPC systems)
  - Superfast CPUs starved of data
- There are many ways to design / architect computers
  - Different flavors suitable to different problems

Flynn's Taxonomy

SISD

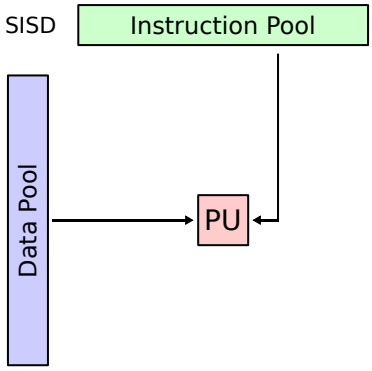


Figure 1: Single Instruction Single Data

- Sequential computer which exploits no parallelism in either the instruction or data streams
- Single control unit (CU/CPU) fetches single Instruction Stream from memory. The CU/CPU then generates appropriate control signals to direct single processing element to operate on single Data Stream, i.e. one operation at a time.
- Basic idea of von Neumann computer (pretty much obsolete)

MISD

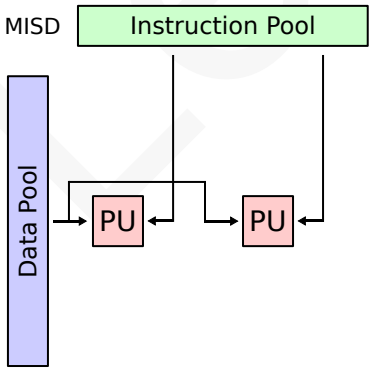


Figure 2: Multiple Instruction Single Data

- Parallel computing architecture where many functional units (PU/CPU) perform different operations on the same data
- Examples include fault tolerant computer architectures, e.g. running multiple error checking processes on same data stream

SIMD

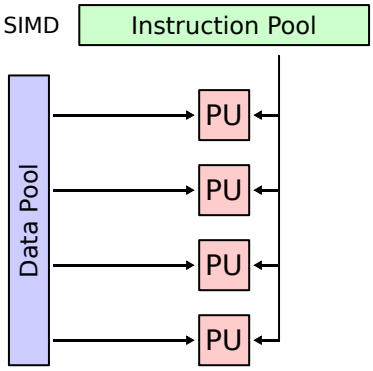


Figure 3: Single Instruction Multiple Data

- multiple processing elements that perform the same operation on multiple data points simultaneously
- focus in on data level parallelism, i.e. many parallel computations, but only a single process (instruction) at a given moment.
- many modern computers use SIMD instructions, e.g. to improve performance of multimedia use such as for image processing
- Digital Signal Processor?

MIMD

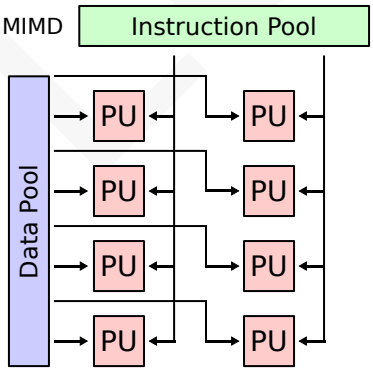


Figure 4: Multiple Instruction Multiple Data

- number of processors that function asynchronously and independently
- at any time, different processors may be executing different instructions on different pieces of data
- machines can be shared memory or distributed memory categories
  - depends on how MIMD processors access memory
- most systems these days operate on MIMD e.g. HPC

Approaches for Parallelism

1. Implicit vs Explicit
2. Hardware
3. Operating system
4. Software / Applications

Implicit vs Explicit Parallelization

- Implicit Parallelism
  - Supported by parallel languages and parallelizing compilers that take care of identifying parallelism, the scheduling of calculations and the placement of data
  - Pretty hard to do
- Explicit Parallelism
  - In this approach, the programmer is responsible for most of the parallelization effort such as task decomposition, mapping tasks to processors, inter-process communications
  - This approach assumes user is the best judge of how parallelism can be exploited for a particular application

Hardware Parallelization

Hardware Threading CPU

Cache - much faster than reading / writing to main memory; instruction cache, data cache (multi-level) and translation lookaside buffer used for virtual-physical address translation

Parallelization by adding extra CPU to allow more instructions to be processed per cycle. Usually shares **arithmetic units**. Heavy use of one type of computation can tie up all the available units of the CPU preventing other threads from using them.

Multi-Core

Multiple cores that can process data and perform computational tasks in parallel. Typically share same **cache**, but issue of cache read / write performance and cache coherence. Possibility of cache stalls (CPU not doing anything whilst waiting for caching), many chips have mixture (L1 cache on single cores; L2 cache on pairs of cores; L3 cache shared by all cores); typical to have different cache speeds and cache sizes (higher hit rates but potentially higher latency).

Symmetric Multiprocessing (SMP)

Two or more identical processors connected to a single, shared main **memory**, with full access to all I/O devices, controlled by a single OS instance that treats all processors equally. Each processor executes different programs and works on different data but with capability of sharing common resources (memory, I/O devices, . . .). Processors can be connected in a variety of way: buses, crossbar switches, meshes. More complex to program since need to program both for CPU and inter-processor communication (bus).

Non-Uniform Memory Access (NUMA)

Non-uniform memory access provides speed-up by allowing a processor to access its own local memory faster than non-local memory. Improved performance as long as data are localized to specific processes / processors. Key is allocating memory / processors in NUMA friendly ways, e.g. to avoid scheduling / locking and (expensive) inter-processor communication.

Operation System Parallelism Approaches

- Most modern multi-core operating systems support different “forms” of parallelization
  - Parallel vs Interleaved semantics
- Compute parallelism
  - Processes
    * Used to realize tasks, structure activities
  - Threads
    * Native threads: Fork, Spawn, Join
    * Green threads: Scheduled by a virtual machine instead of natively by the OS
- Data parallelism
  - Caching (cache coherency)
  - OS implies on “a” computer

Software Parallelism Approaches

- Many languages now support a range of parallelization / concurrency features
  - Threads, thread pools, lock, semaphores
- Many languages developed specifically for parallel / concurrent systems
- Key issues that need to tackled
  - Deadlock - processes involved constantly waiting for each other
  - Livelock - processes involved in livelock concurrently change with regard to one another, but none are progressing

Message Passing Interface

- widely adopted approach for message passing in parallel systems
- mappings to major languages C, C++, Python, Java
- support point-point, broadcast communications
- key MPI functions
  1. `MPI_Init` initiate MPI computation
  2. `MPI_Finalize` terminate computation
  3. `MPI_Comm_size` determine number of processors
  4. `MPI_Comm_rank` determine my process identifier
  5. `MPI_Send` send a message
  6. `MPI_Recv` receive a message

Erroneous Assumptions of Distributed Systems

Starch BLT

A BLT sandwich contains starch and BLT (bacon, lettuce and tomato).

1. The network is secure
2. Time is ubiquitous
3. There is one administrator
4. The network is reliable
5. Transport cost is zero
6. The network is homogeneous
7. Bandwidth is infinite
8. Latency is zero
9. Topology does not change

Parallelization Paradigms

1. Master Slave Model
2. Single-Program Multiple-Data
3. Pipelining
4. Divide and Conquer
5. Speculative Parallelism

Mater Slave Model

- Master decomposes the problem into small tasks, distributes to workers and gathers partial results to produce the final result.
- Realized in many ways of different levels of granularity, e.g. threads through to web service workflow definition and enactment.

Single-Program Multiple-Data

- Commonly exploited model
  - Bioinformatics, MapReduce
- Each process executes the same piece of code, but on different parts of the data
- Data is typically split among the available processors
- Data splitting and analysis can be done in many ways

Data pipelining

- Suitable for applications involving multiple stages of execution, that typically operate on large number of data sets.

Divide and Conquer

- A problem is divided into two or more sub problems, and each of these sub problems are solved independently, and their results are combined
- 3 operations: split, compute and join
- Master-worker / task-farming is like divide and conquer with master doing both split and join operation

Speculative Parallelism

- Used when it is quite difficult to achieve parallelism through the previous paradigms
- Problems with complex dependencies - use “look ahead” execution
- Consider a (long running) producer P and a consumer C such that C depends on P for the value of some variable V. If the value of V is **predictable**, we can execute C speculatively using a predicted value in parallel with P.
  - If the prediction turns out to be correct, we gain performance since C does not wait for P anymore.
  - If the prediction is incorrect (which we can find out when P completes), we have to take corrective action, cancel C and restart C with the right value of V again.

Lecture 4

Limitations of Parallel Computation

Correctness in parallelization requires synchronization (locking). Synchronization and atomic operations causes loss of performance, communication latency.

Lecture 5 Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resource (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud Models

- Deployment Models
  - Public
  - Private
  - Hybrid

  - Community
- Delivery Models
  - Software as a Service
  - Platform as a Service
  - Infrastructure as a Service

Deployment Models

Public Clouds

- Pros
  - Utility computing
  - Can focus on core business
  - Cost-effective
  - “Right-sizing”
  - Democratization of computing
- Cons
  - Security
  - Loss of control
  - Possible lock-in
  - Dependency of cloud provider continued existence

Private Clouds

- Pros
  - Control
  - Consolidation of resources
  - Easier to secure
  - More trust
- Cons
  - Relevance to core business?
  - Staff / management overheads
  - Hardware obsolescence
  - Over / under utilization challenges

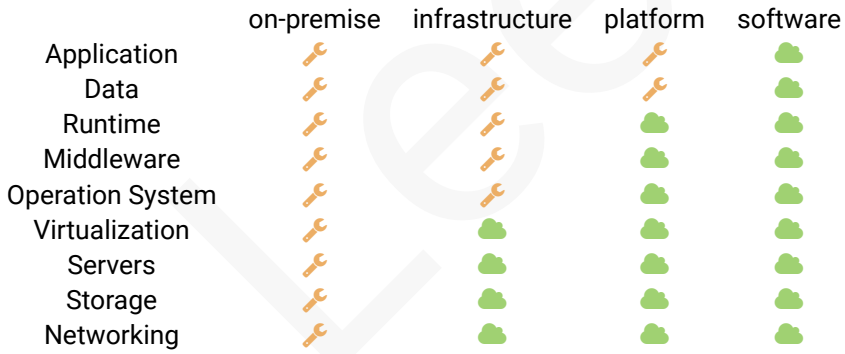
Hybrid Clouds

- Examples
  - Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems)
  - VMware vCloud Hybrid Service
- Pros
  - Cloud-bursting: use private cloud but burst into public cloud when needed

- Cons
  - How do you move data / resources when needed?
  - How to decide (in real-time) what data can go to public cloud?
  - Is the public cloud compliant with PCI-DSS (Payment Card Industry - Data Security Standard)?

Deliverry Models

- IaaS
  - AWS
  - Azure
  - Alibaba Cloud
- PaaS
  - Google App Engine
  - Heroku
  - OpenShift
- SaaS
  - Gmail
  - Salesforce.com
  - Microsoft Office 365



Common Terms (not in slides)

- Machine Image
  - a stored image / template from which a new virtual machine can be launched
- Instance
  - a running virtual machine based on some machine image
- Volume
  - attachable Block Storage, which is the equivalent of a virtual disk drive
- Object Storage
  - a large store for storing simple binary objects + metadata within containers
- Security groups
  - a means of specifying firewall rules
- Key-pairs
  - public / private key pairs for accessing virtual machine

Automation

- Deploying complex cloud systems requires a lot of moving parts
  - Easy to forget what software you installed, and what steps you took to configure the system
  - Error-prone, can be non-repeatable
  - Snapshots are monolithic - provide no record of what has changed
- Automation
  - Provides a record of what you did
  - Codifies knowledge about the system
  - Makes process repeatable
  - Makes it programmable - “Infrastructure as Code”

Tools

- Cloud-focused
  - Boto, CloudFormation, Heat
- Shell scripts
  - Bash, Perl
- Configuration Management
  - Configuration management refers to the process of systematically handling changes to a system in a way that it maintains integrity over time.
  - Automation is the mechanism used to make servers reach a desirable state, previously defined by provisioning scripts using tool-specific languages and features.
  - Puppet, Ansible

Ansible Features

- Easy to learn
  - Playbooks in YAML, Templates in Jinja2, Inventory in INI file
  - Sequential execution
- Minimal requirements
  - No need for centralized management servers / daemons
  - Single command to install `pip install ansible`
  - Uses SSH to connect to target machine
- Idempotent (repeatable)
  - Executing N times no different to executing once
  - Prevents side-effects from re-running scripts
- Extensible
  - Write your own modules
- Supports push or pull

  - Push by default but can use cron job to make it pull
- Rolling updates
  - Useful for continuous deployment / zero downtime deployment
- Inventory management
  - Dynamic inventory from external data sources
  - Execute tasks against host patterns
- Ansible Vault for encrypted data
- Ad-hoc commands
  - When you need to execute a one-off command against your inventory
  - `ansible -i hosts -u ubuntu -m shell "reboot"`
- Ansible Tower: Enterprise mission control for Ansible
  - Dashboard, System tracker, . . .

原來尚有理想 沒法再攀上，想贈那獎 都不及去頒上。
難道歲月 多少課也可白上，唯獨告別路途 要懂怎去走上。

完

Lecture 6 Big Data and CouchDB

Volume, Velocity, Variety, Veracity

1. Volume
   - volume (giga, tera, peta, exa, zetta)
2. Velocity
   - the frequency of new data being brought into the system and analysis performed
3. Variety
   - the variability and complexity of data schema. The more complex the data schema you have, the higher the probability of them changing along the way, adding more complexity.
4. Veracity
   - the level of trust in the data accuracy (*provenance*); the more diverse sources you have, the more unstructured they are, the less veracity you have.

While relational DBMSs are extremely good at ensuring consistency, they rely on normalized data models that, in a world of big data (think about Veracity and Variety) can no longer be taken for granted.

Therefore, it makes sense to use DBMSs that are built upon data models that are not relational (i.e. tables and relationships amongst tables and the entities they describe)

The Tale of Clusters

- Distributed databases are run over clusters, i.e. sets of connected computers
- Clusters are needed to
  - Distribute the computing load over multiple computers, e.g. to improve availability
  - Store multiple copies of data, e.g. to achieve redundancy

MongoDB & CouchDB

MongoDB Cluster Architecture

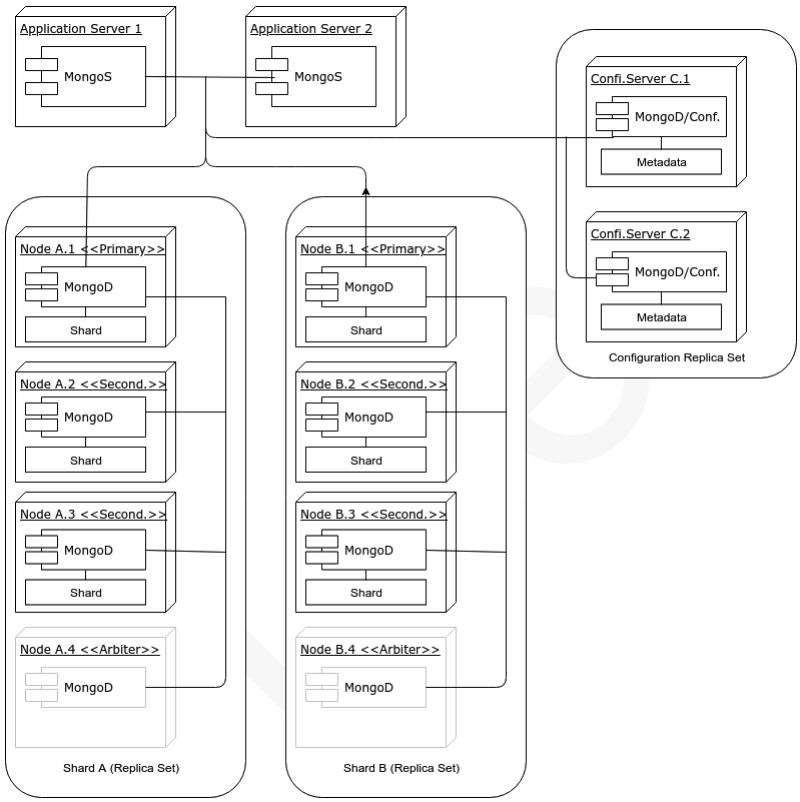


Figure 5: MongoDB Cluster

- Sharding is done at the *replica set* level, hence it involves more than one cluster (a shard is on top of a replica set)
- Only the *primary node* in a replica set answers write requests, but read requests can - depending on the specifics of the configuration - be answered by every node (including secondary nodes) in the set
- Updates flow only from the primary to the secondary
- If a primary node fails, or discovers it is connected to a minority of nodes, a secondary of the same replica set is elected as the primary
- Arbiters (MongoDB instances without data) can assist in breaking a tie in elections
- Data are balanced across replica sets
- Since a quorum has to be reached, it is better to have an odd number of voting members （The arbiter in this diagram is only illustrative)

CouchDB Cluster Architecture

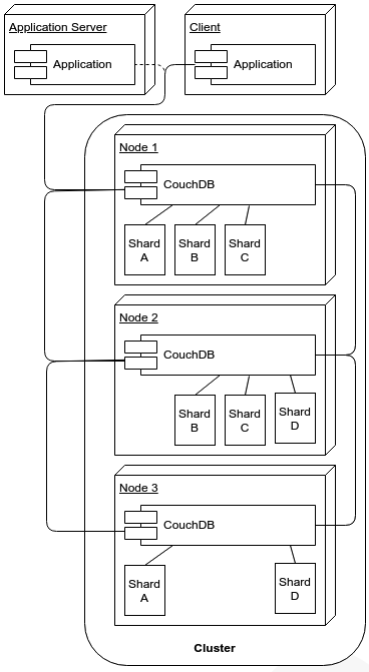


Figure 6: CouchDB Cluster

- All nodes answer requests (read or write) at the same time
- When a node does not contain a document (say, a document of Share A is requested to Node 2), the node request it from another node (say, Node 1) and returns it to the client
- Nodes can be added / removed easily, and their shards are re-balanced automatically upon addition / deletion of nodes
- In this example, there are 3 nodes, 4 shards and a replica number of 2

MongoDB vs CouchDB

- MongoDB clusters are considerably more complex than CouchDB ones
- MongoDB clusters are less available, as - by default - only primary nodes can talk to clients for read operations, (and exclusively so for write operations)
- MongoDB software routers (MongoS) must be embedded in application servers, while any HTTP client can connect to CouchDB
- Losing two nodes out of three in the CouchDB example architecture, means losing access to one quarter of data, while losing two nodes in the MongoDB example architecture implies losing access to half the data (although there are ten nodes in the cluster instead of three)
- Some features (such as unique indexes or geo-spatial indexes) are not supported in MongoDB sharded environments

While ConcuDB uses MVCC, MongoDB uses a hybrid two-phase commit (for replicating data from primary to secondary nodes) and Paxo-like in supporting network partition strategies.

From the MongoDB 3.6 documentation: A network partition may segregate a primary into a partition with a minority of nodes. When the primary detects that it can only see a minority of nodes in the replica set, the

primary steps down as primary and becomes a secondary. Independently, a member in the partition that can communicate with a majority of the nodes (including itself) holds an election to become the new primary.

INFO90002 Recap

CAP

- In the presence of a partition, one is then left with two options: consistency or availability.
  - When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up-to-date due to network partitioning.
  - When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up-to-date due to network partitioning.

ACID

1. Atomicity
   - A transaction is treated as a single, indivisible, logical unit of work. All operations in a transaction must be completed; if not, then the transaction is aborted.
2. Consistency
   - Constraints that hold before a transaction must also hold after it.
   - (multiple users accessing the same data see the same value)
3. Isolation
   - Changes made during execution of a transaction cannot be seen by other transactions until this one is completed.
4. Durability
   - When a transaction is complete, the changes made to the database are permanent, even if the system fails.

BASE

- Basically Available
  - This constraint states that the system does guarantee the availability of the data: there will be a response to any request. But data may be in an inconsistent or changing state.
- Soft state
  - The state of the system could change overtime - even during times without input there may be changes going on due to eventual consistency.
- Eventual consistency
  - The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it needs to, sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

Consistency, Availability, Partition-tolerance

- Consistency
  - every client receiving an answer receives the same answer from all nodes in the cluster
- Availability
  - every client receives an answer from any node in the cluster
- Partition-tolerance
  - the cluster keeps on operating when one or more nodes cannot communicate with the rest of the cluster

CAP Theorem

- While the theorem shows all three qualities are symmetrical, Consistency and Availability are at odds when a Partition happens
- "Hard" network partition may be rare, but "soft" ones are not (a slow node may be considered *dead* even if it is not); ultimately, every partition is detected by a *timeout*
- Can have consequences that impact the cluster as a whole, e.g. a distributed join is only complete when all sub-queries return
- Traditional DBMS architectures were not concerned with network partitions, since all data were supposed to be in a small co-located cluster of servers
- The emphasis on numerous *commodity server*, can result in an increased number of hardware failures
- The CAP theorem forces us to consider trade-offs among different options

Classification of Distributed Processing Algorithms

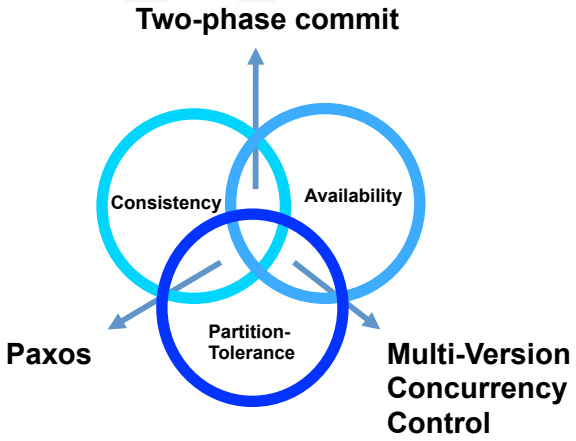


Figure 7: CAP Theorem

Two-phase commit (Consistency & Availability)

- This is the usual algorithm used in relational DBMSs and MongoDB, it enforces consistency by:
  - locking data that are with the transaction scope
  - performing transactions on write-ahead logs

  - completing transactions (commit) only when all nodes in the cluster have performed the transaction
  - aborting transactions (rollback) when a partition is detected
- This procedure entails the following
  - reduced availability (data lock, stop in case of partition)
  - enforced consistency (every database is in a consistent state, and all are left in the same state)
- Therefore, two-phase commit is a good solution when the cluster is co-location, less then good when it is distributed

Paxos (Consistency & Partition-Tolerance)

- This family of algorithms is driven by consensus and is both partition-tolerance and consistent
- In Paxos, every node is either a *proposer* or an *accepter*
  - an proposer proposes a value (with a timestamp)
  - an accepter can accept or refuse it (e.g. if the accepter receives a more recent value)
- when a proposer has received a sufficient number of acceptance (a quorum is reached), and a confirmation message is sent to the accepter with the agreed value
- Paxos clusters can recover from partitions and maintain consistency, but the smaller part of a partition (the part that is not in the quorum) will not send responses, hence the availability is compromised

MVCC (Availability & Partition-Tolerance)

- Multi-Version Concurrency Control is a method to ensure availability (every node in a cluster always accepts requests), and some sort of recovery from a partition by reconciling the single databases with *revisions* (data are not replaced, they are just given a new revision number)
- In MVCC, concurrent updates are possible without distributed locks (in *optimistic locking* only the local copy of the object is locked), since the updates will have different revision numbers; the transaction that completes last will get a higher revision number, hence will be considered as the *current value*
- In case of cluster partition and concurrent requests with the same revision number going to two partitioned nodes, both are accepted, but once the partition is solved, there would be a conflict. Conflict that would have to be solved somehow (CouchDB returns a list of all current conflicts, which are then left to be solved by the application.)

Sharding

- Sharding is the partition of a database “horizontally”, i.e. the database rows (or documents) are partitioned into subsets that are stored on different servers. Every subset of rows is called a *shard*
- Usually the number of shards is larger than the number of replicas, and the number of nodes is larger than the replica numbers (usually set to 3)
- The main advantage of a sharded database lies in the improvement of performance through the distribution of computing load across nodes. In addition, it makes it easier to move data files around, e.g. when adding new nodes to the cluster.
- The number of shards dictates the (meaningful) number of nodes; the maximum number of nodes is equal to the number of shards
- There are different sharding strategies, most notably:

  - Hash sharding: to distribute rows evenly across the cluster
  - Range sharding: similar rows (say, Tweets coming for the same area) that are stored on the same node (or subset of nodes)

Replication and Sharding

- Replication is the action of storing the same row (or document) on different nodes to make the database fault-tolerant
- Replication and sharding can be combined with the objective on maximizing availability while maintaining a minimum level of data safety.

MapReduce

- MapReduce is particularly suited to parallel computing of the SIMD type.
- The first step (Map) distributes data across machines, while the second (Reduce) hierarchically summarizes them until the result is obtained.
- Apart from parallelism, its advantage lies in moving the process to where data are, greatly reducing network traffic.
- MapReduce is the tool of choice when operations on big datasets are to be done due to its horizontal scalability.

CouchDB

Features

- Document-oriented DBMS, where documents are expressed in JavaScript Object Notation (JSON)
- HTTP ReST API
- Web-based admin interface
- Web-ready: since it talks HTTP and produces JSON (it can also produce HTML or XML), it can be both the data and logic tier of a three-tier application, hence avoiding the marshaling and unmarshaling of data objects
- Support for MapReduce algorithms, including aggregation at different levels
- JavaScript as the default data manipulation language
- Run Mango queries (MongoDB query language), which can use indexes for better performance
- Schema-less data model with JSON as the data definition language
- Support for replication
- Support for sharding
- Support for clustering

Lecture 7.1 Service-oriented Architecture

Acronyms

- SOA Service-Oriented Architecture
- ROA Resource-Oriented Architecture
- UML Unified Modeling Language
- SOAP Simple Object Access Protocol
- ReST Representational State Transfer
- RPC Remote Procedure Call

Architecture

- A system architecture, is the way different software components are distributed on computing devices, and the way in which they interact with each other
- Architectures are often difficult to describe in words, hence diagrams are often used
- A standard graphic way to describe architectures is through the Unified Modeling Language deployment diagram

Why SOA

When an architecture is completely contained with the same machine, components communicate through *function calls* or *object instantiations*.

However, when components are distributed, function calls and object instantiations cannot always be used directly.

Therefore, components (more properly, *systems*) have to interact in more loosely-coupled ways. *Services* are often used for this.

Every system in a SOA should be considered as autonomous, but network-reachable and inter-operable through services.

SOA Core Ideas

1. A set of services that a business wants to provide to their customers, partners, or other areas of an organization
2. An architectural pattern that requires a service provider, mediation and service requester with a service description
3. A set of architectural principles, patterns and criteria that address characteristics such as modularity, encapsulation, loose coupling, reuse and composability
4. A programming model complete with standards, tools and technologies that supports web services, ReST services or other kinds of services
5. A middleware solution optimized for service assembly, orchestration, monitoring and management

SOA Principles

Grass and Cl_2

11 letters: both grass and chlorine gas are green.

1. Service granularity
    - A design consideration to provide optimal scope at the right granular level of the business functionality in a service operation.
2. Service reusability
    - Logic is divided into services with the intention of promoting reuse.
3. Service abstraction
    - Beyond descriptions in the service contract, services hide logic from the outside world.
4. Standardized service contract
    - Services adhere to a communications agreement, as defined collectively by one or more service-description documents.
5. Service statelessness
    - Services minimize resource consumption by deferring the management of state information when necessary.
6. Service autonomy
    - Services have control over the logic they encapsulate.
7. Service normalization
    - Services are decomposed and / or consolidated to a level of normal form to minimize redundancy. In some cases, services are denormalized for specific purposes, such as performance optimization, access and aggregation.
8. Service discoverablility
    - Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
9. Service composability
    - Services are effective composition participants, regardless of the size and complexity of the composition.
10. Service location transparency
    - The ability of a service consumer to invoke a service regardless of its actual location in the network.
11. Service loose coupling
    - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.

SOAP/WS vs ReST

1. SOAP/WS is built upon the paradigm of the *Remote Procedure Call*; particularly, a language independent function call that spans another system
2. ReST is centered around *resources*, and the way they can be manipulated (added, deleted, etc.) remotely
3. ReST is more of an architectural style of using HTTP than a separate protocol, while SOAP/WS is a stack of protocols that covers every aspect of using a remote service, from service discovery, to service description, to the actual request / response
4. ReST makes use of the different HTTP Methods (`GET`, `POST`, `PUT`, `DELETE`, etc)

從何時開始忌諱空山無人 從何時開始怕遙望星塵
原來神仙魚橫渡大海會斷魂 聽不到世人愛聽的福音

任我行

Lecture 7.3 REST Web Services & ROA Architecture

Actions

POST $\neq$ Create and PUT $\neq$ Update

Create

CouchDB: `PUT /{db}/{docid}`
creates a new named document, or creates a new revision of the existing document.

CouchDB: `POST /{db}`
creates a new document in the specified database.

ElasticSearch: `PUT /{index}/{type}/{id}/`

ElasticSearch: `POST /{index}/{type}/`

HTTP Methods

- `GET`, `OPTIONS`, `HEAD`
  - safe: do not change, repeating a call is equivalent to not making a call at all
- `PUT`, `DELETE`
  - idempotent: effect of repeating a call is equivalent to making a single call
- `POST`
  - neither safe nor idempotent

孩童只盼望歡樂 大人只知道期望 為何都不大懂得努力體恤對方
成人只寄望收穫 情人只聽見承諾 為何都不大懂得努力珍惜對方

Shall We Talk

Lecture 7.4 Code Versioning Systems

Git

Create

Clone an existing repository
```
git clone ssh://user@domain.com/repo.git
```

Create a new local repository
```
git init
```

Local Changes

Add all current changes to the next commit
```
git add .
```

Commit previously staged changes
```
git commit
```

Branches & Tags

Switch HEAD branch
```
git checkout <branch>
```

Create a new branch based on your current HEAD
```
git branch <new-branch>
```

Delete a local branch
```
git branch -d <branch>
```

Mark the current commit with a tag
```
git tag <tag-name>
```

Branches & Tags

Download all changes from <remote>, but don't integrate into HEAD
```
git fetch <remote>
```

Download changes and directly merge/integrate into HEAD
```
git pull <remote> <branch>
```

Publish local changes on a remote
```
git push <remote> <branch>
```

Delete a branch on the remote
```
git branch -dr <remote/branch>
```

Publish your tags
```
git push --tags
```

Merge & Rebase

Merge <branch> into your current HEAD
```
git merge <branch>
```

開始時捱一些苦 栽種絕處的花 幸得艱辛的引路甜蜜不致太寡

苦瓜

Lecture 8 Big Data Analytics

Challenges

A framework for analyzing big data has to distribute both data and processing over many nodes, which implies:

- Reading and writing distributed datasets
- Preserving data in the presence of failing data nodes
- Supporting the executing of MapReduce tasks
- Being fault-tolerant (a few failing compute nodes may slow down the processing, but not stop it)
- Coordinating the execution of tasks across a cluster

All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework. W

Hadoop Distributed File System

The core of Hadoop is a fault tolerant file system that has been explicitly designed to span many nodes.

HDFS blocks are much larger than blocks used by an ordinary file system (4 kB versus 128 MB)

- Reduced need for memory to store information about where the blocks are (metadata)
- More efficient use of the network (with a large block, a reduced number of connections need to be kept open)
- Reduced need for seek operations on big files
- Efficient when most data of a block have to be process

HDFS Architecture

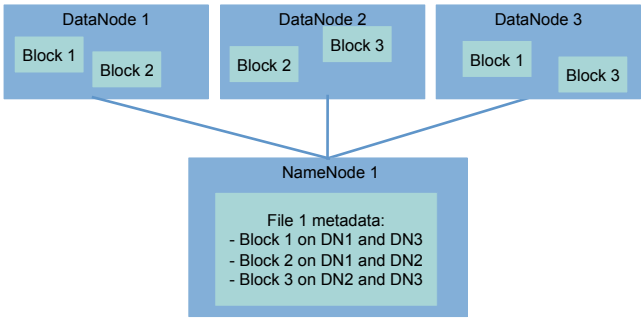


Figure 8: HDFS Architecture

A HDFS file is a collection of blocks stored in datanodes with metadata (such as position of those blocks) that is stored in namenodes.

Why Spark?

- While Hadoop MapReduce works well, it is geared towards performing relatively simple jobs on large datasets
- However, when complex jobs are performed (say, machine learning or graph-based algorithms), there is a strong incentive for caching data in memory and in having finer-grained control on the execution of jobs
- Apache Spark was designed to reduce the latency inherent in the Hadoop approach for the execution of MapReduce jobs
- Spark can operate within the Hadoop architecture, using YARN and Zookeeper to manage computing resources and storing data on HDFS

Resilient Distributed Dataset

RDDs are the way data are stored in Spark during computation, and understanding them is crucial to writing programs in Spark

1. Resilient
    - data are stored redundantly, hence a failing node would not affect their integrity
2. Distributed
    - data are split into chunks, and these chunks are sent to different nodes
3. Dataset
    - a dataset is just a collection of objects, hence very generic

Properties

- RDDs are immutable, once defined, they cannot be changed (this greatly simplifies parallel computations on them, and is consistent with the functional programming paradigm)
- RDDs are transient, they are meant to be used only once, then discarded (but they can be cached, if it improves performance)

- RDDs are lazily-evaluated, the evaluation process happens only when data cannot be kept in an RDD, as when the number of objects in an RDD has to be computed, or an RDD has to be written to a file (these are called *actions*), but not when an RDD are transformed into another RDD (these are called *transformations*)

這個剎那宇宙 拒絕永久
世事無常還是未看夠 還未看透

夕陽無限好

Lecture 9.1 Virtualization

Acronyms

- VMM Virtual Machine Monitor
- POPF Pop Flags
- SMSW Store Machine Status Word
- SGDT Store Global Descriptor Table register
- SLDT Store Local Descriptor Table register
- DMA Direct Memory Access

Terminology

- Virtual Machine Monitor / Hypervisor
  - The virtualization layer between the underlying hardware (e.g. the physical server) and the virtual machines and guest operating systems it supports.
    * The environment of the VM should appear to be the same as the physical machine
    * Minor decrease in performance only
    * Appear as though in control of system resources
- Virtual Machine
  - A representation of a real machine using hardware / software that can host a guest operating system
- Guest Operating System
  - An operating system that runs in a virtual machine environment that would otherwise run directly on a separate physical system

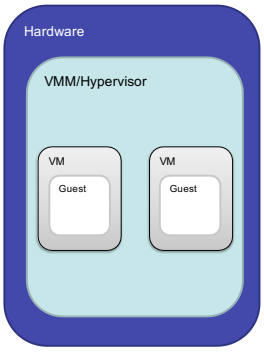
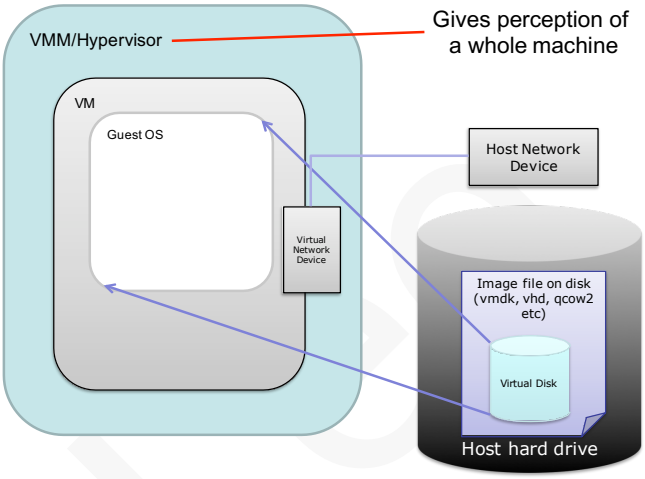


Figure 9: Hardware, VMM, VM and Guest OS

What Happens in a VM



- VHD (Virtual Hard Disk) represents a virtual Hard Disk Drive (HDD). May contain what is found on a physical hard disk, such as disk partitions and a file system, which in turn can contain files and folders.
- VMDK (Virtual Machine Disk) described containers for virtual hard disk drives to be used in virtual machine like VMware.
- qcow (QEMU Copy On Write) is a file format for disk image files used by QEMU. It uses a disk storage optimization strategy that delays allocation of storage until it is actually needed. qcow2 is an updated version of the qcow format.
- Guest OS apps “think” they write to hard disk and translated to virtualized host hard drive by VMM.
- Which one is determined by image that is launched.

Motivation

- Server consolidation
  - Increased utilization
- Personal virtual machines can be created on demand
  - No hardware purchase needed
  - Public cloud computing

- Security / Isolation
  - Share a single machine with multiple users
- Hardware independence
  - Relocate to different hardware

Classification of Instructions

1. Privileged Instructions
   - instructions that trap if the processor is in user mode and do not trap in kernel mode
2. Sensitive Instructions
   - instructions whose behavior depends on the mode or configuration of the hardware
     - different behaviors depending on whether in user or kernel mode e.g. POPF interrupt (for interrupt flog handling)
3. Innocuous Instructions
   - instructions that are neither privileged nor sensitive
     - read data, add numbers etc

Sensitive $\subset$ Privileged

For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions. x86 architecture was historically not virtualizable, due to sensitive instructions that could not be trapped.

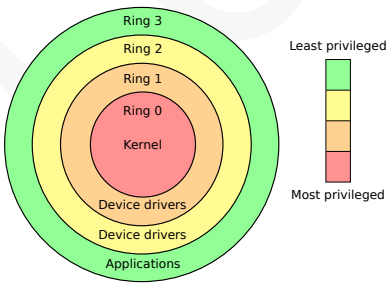


Figure 10: Privilege Rings

- Ring 0: Typically hardware interactions
- Ring 1: Typically device drivers
- Specific gates between rings (not ad hoc)
- Allows to ensure for example that spyware cannot turn on web cam or recording device etc.

Virtualization Strategy

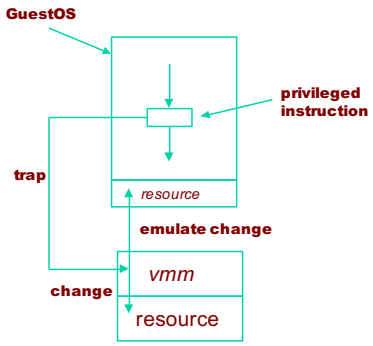


Figure 11: Virtualization Strategy

- De-privileging
  - VMM emulates the effect on system / hardware resources of privileged instructions whose execution traps into the VMM (aka trap-and-emulate)
  - Typically achieved by running Guest OS at a lower hardware priority level than the VMM
  - Problematic on some architectures where privileged instructions do not trap when executed at de-privileged level
- Primary / shadow structures
  - VMM maintains “shadow” copies of critical structures whose “primary” versions are manipulated by the Guest OS, e.g. memory page tables
  - Primary copies needed to insure correct versions are visible to Guest OS
- Memory traces
  - Controlling access to memory so that the shadow and primary structure remain coherent
  - Common strategy: write-protect primary copies so that update operations cause page faults which can be caught, interpreted, and addressed

Full virtualisation vs Para-virtualisation

Full virtualisation

- allow an unmodified guest OS to run in isolation by simulating full hardware (e.g. VMware)
  - Guest OS has no idea it is not on physical machine
- Advantages
  - Guest is unaware it is executing within a VM
  - Guest OS need not be modified
  - No hardware or OS assistance required
  - Can run legacy OS
- Disadvantages
  - Can be less efficient

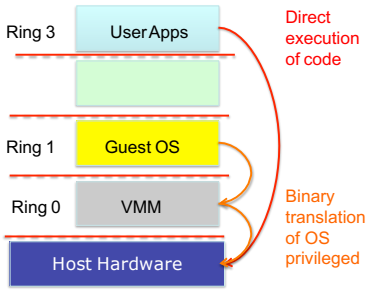


Figure 12: Full Virtualization

- User / kernel split typically
  - VMM run in Ring 0
  - Apps run in Ring 3
- Virtualization (Guest OS) uses extra rings
- VMM traps privileged instructions and translates to hardware specific instructions

Para-virtualization

- VMM / Hypervisor exposes special interface to guest OS for better performance. Requires a modified / hypervisor-aware Guest OS (e.g. Xen)
  - Can optimize systems to use this interface since not all instructions need to be trapped / dealt with
- Advantages
  - Lower virtualisation overheads, so better performance, e.g. Xen
- Disadvantages
  - Need to modify guest OS (cannot run arbitrary OS)
  - Less portable
  - Less compatibility

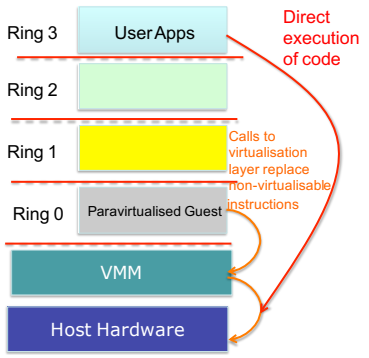


Figure 13: Para-Virtualization

Hardware-assisted Virtualization vs Binary Translation

Hardware-assisted Virtualization

- Hardware provides architectural support for running a Hypervisor (e.g. KVM)
  - New processors typically have this
  - Requires that all sensitive instructions trappable
- Advantages
  - Good performance
  - Easier to implement
  - Advanced implementation supports hardware assisted DMA, memory virtualization
- Disadvantages
  - Needs hardware support

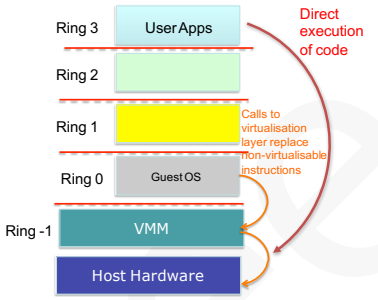


Figure 14: Hardware-assisted Virtualization

- New Ring -1 supported Page tables, virtual memory management, DMA for high speed reads etc

Binary Translation

- Trap and execute occurs by scanning guest instruction stream and replacing sensitive instructions with emulated code (e.g. VMware)
  - Do not need hardware support, but can be much harder to achieve.
    * Rarely ever 1:1 mapping between instruction sets
- Advantages
  - Guest OS need not be modified
  - No hardware or OS assistance required
  - Can run legacy OS
- Disadvantages
  - Overheads
  - Complicated
  - Need to replace instructions “on-the-fly”
  - Library support to help this, e.g. vCUDA

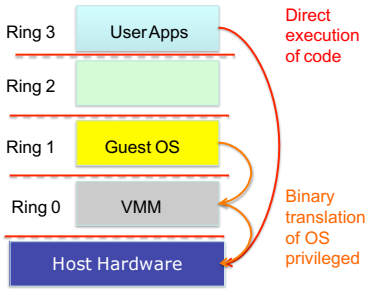


Figure 15: Binary Translation

Bare Metal Hypervisor vs Hosted Virtualization

- Bare Metal Hypervisor
  - VMM runs directly on actual hardware
    - Boots up and runs on actual physical machine
    - VMM has to support device drivers, all hardware management
- Hosted Virtualization
  - VMM runs on top of another operating system

Operating System Level Virtualization

- Lightweight VMs
- Instead of whole-system virtualization, the OS creates mini-containers
  - A subset of the OS is often good enough for many use cases
    - Cannot use for running Windows on Linux etc, but often not needed
  - Akin to an advanced version of “chroot”
    - operation that changes apparent root directory for current running process and subprocesses. Program run in such a modified environment cannot access files and commands outside that environmental directory tree. Aka a chroot jail
- Examples
  - Docker, LXC, OpenVZ, FreeBSD Jails etc
- Advantages
  - Lightweight
  - Many more VMs on same hardware
  - Can be used to package applications and all OS dependencies into container
- Disadvantages
  - Can only run apps designed for the same OS
  - Cannot host a different guest OS
  - Can only use native file systems
  - Uses same resources as other containers

Shadow Page Table

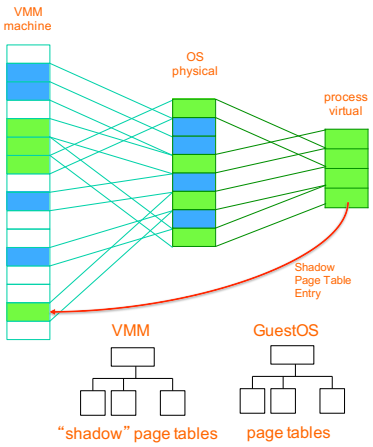


Figure 16: Shadow Page Table

- Conventionally page tables store the logical page number → physical page number mappings
- VMM maintains shadow page tables in lock-step with the page tables
- Adds additional management overhead
- Hardware performs guest → physical and physical → machine translation

Live Migration

Clark et al. Live migration of virtual machines. 2005

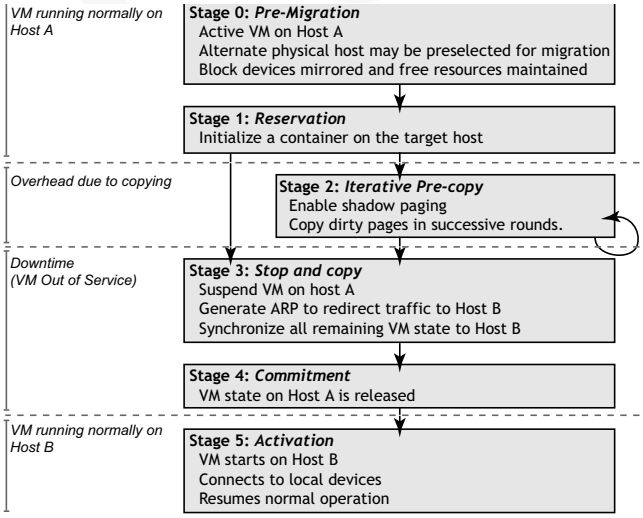


Figure 17: Migration timeline

- Stage 0: Pre-Migration
  - We begin with an active VM on physical host A. To speed any future migration, a target host may be preselected where the resources required to receive migration will be guaranteed.
- Stage 1: Reservation

  - A request is issued to migrate an OS from host A to host B. We initially confirm that the necessary resources are available on B and reserve a VM container of that size. Failure to secure resources here means that the VM simply continues to run on A unaffected.

- Stage 2: Iterative Pre-Copy
  - During the first iteration, all pages are transferred from A to B. Subsequent iterations copy only those pages dirtied during the previous transfer phase.
- Stage 3: Stop-and-Copy
  - We suspend the running OS instance at A and redirect its network traffic to B. As described earlier, CPU state and any remaining inconsistent memory pages are then transferred. At the end of this stage there is a consistent suspended copy of the VM at both A and B. The copy at A is still considered to be primary and is resumed in case of failure.
- Stage 4: Commitment
  - Host B indicates to A that it has successfully received a consistent OS image. Host A acknowledges this message as commitment of the migration transaction: host A may now discard the original VM, and host B becomes the primary host.
- Stage 5: Activation
  - The migrated VM on B is now activated. Post-migration code runs to reattach device drivers to the new machine and advertise moved IP addresses.

Lecture 9.2 OpenStack

OpenStack Component

Compute Service	Nova	Elastic Compute Cloud (EC2)
Image Service	Glance	Amazon Machine Image
Block Storage Service	Cinder	Elastic Block Store
Object Storage Service	Swift	Simple Storage Service (S3)
Security Management	Keystone	Identity and Access Management
Orchestration Service	Heat	CloudFormation
Network Service	Neutron	Networking
Monitoring Service	Ceilometer	CloudWatch
Database Service	Trove	
Dashboard Service	Horizon	Management Console
Search Service	Searchlight	
Security API	Barbican	
Data Processing	Sahara	Elastic MapReduce (EMR)

Lecture 9.3 Containerization

Virtualization vs Containerization

- The many advantages of virtualization, such as application containment and horizontal scalability, come at a cost: resources. The guest OS and binaries can give rise to duplications between VMs wasting server processors, memory and disk space and limiting the number of VMs each server can support.
- Containerization allows virtual instances to share a single host OS (and associated drivers, binaries, libraries) to reduce these wasted resources since each container only holds the application and related binaries. The rest are shared among the containers.

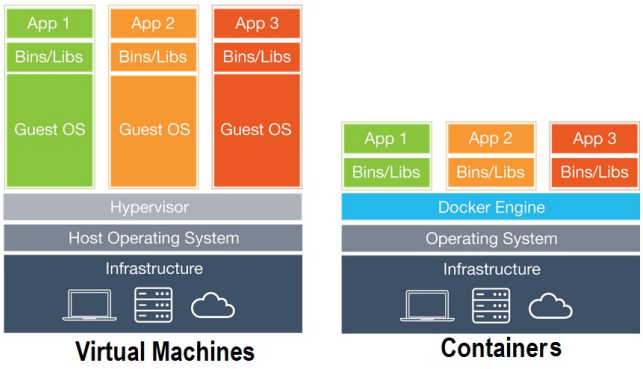


Figure 18: Virtualization vs Containerization

1. Guest OS
   - run on virtual hardware; have their own OS kernels
   - share same OS kernel
2. Communication
   - through Ethernet devices
   - IPC mechanisms (pipes, sockets)
3. Security
   - depends on the Hypervisor
   - requires close scrutiny
4. Performance
   - small overhead incurs when instructions are translated from guest to host OS
   - near native performance
5. Isolation
   - file systems and libraries are not shared between guest and host OS
   - file systems can be shared and libraries are shared
6. Startup time
   - slow (minutes)
   - fast (a few seconds)
7. Storage
   - large size
   - small size (most is re-use)

Are Containers better than VMs?

- the size of the task on hand
- the life span of the application
- security concerns
- host operation system

Docker

Docker Nomenclature

- Container
  - a process that behaves like an independent machine, with its own operating system, file system, network interfaces and applications.
- Image
  - a blueprint for a container, a container is an instance of an image.
- Dockerfile
  - the recipe to create an image. Every instruction in the recipe is a layer that is stored independently, so that only changed layers need to be re-run or transferred, the rest will be cached.
- Docker registry
  - a repository of Docker images. The most import one is the Docker Hub.
- Docker Compose
  - Compose is a tool for defining and running multi-containers Docker applications. With Compose you can create and start all services from a pre-defined YAML configuration file.

Manage Data in Docker

- By default all files created inside a container are stored on a writable container layer.
  - Data does not persist when a container is no longer running.
  - It is difficult to move the data out of a container.
  - It is also difficult to move the data into a container.
- Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops.
  - Docker volumes (managed by Docker `/var/lib/docker/volume/`)
  - Bind mounts (managed by user)

沒揀擇冷或暖甜和酸 晴天雨季缺或圓

今日

Lecture 12 Security and Clouds

Terminology

- single sign-on
  - Single sign-on (SSO) is a property of access control of multiple related, yet independent, software systems. With this property, a user logs in with a single ID and password to gain access to a connected system or systems without using different usernames or passwords, or in some configurations seamlessly sign on at each system. W
- federated identity
  - A federated identity in information technology is the means of linking a person's electronic identity and attributes, stored across multiple distinct identity management systems. Federated identity is related to single sign-on (SSO), in which a user's single authentication ticket, or token, is trusted across multiple IT systems or even organizations. W
- identity provider
  - An identity provider (abbreviated IdP) is a system entity that creates, maintains, and manages identity information for principals while providing authentication services to relying party applications within a federation or distributed network. An identity provider is "a trusted provider that lets you use single sign-on (SSO) to access other websites." W

Technical Challenges

- Authentication
- Authorization
- Audit / Accounting
- Confidentiality
- Privacy
- Fabric management
- Trust

Public Key Cryptography

- Also called Asymmetric Cryptography
- Two distinct keys
  - Private key
  - Public Key
- Two keys complementary, but essential that cannot find out value of private key from public key
  - With private keys can digitally sign messages, documents, ... and validate them with associated public keys (check whether changed, useful for non-repudiation)
- Public key cryptography simplifies key management
  - The longer keys are left in storage, more likelihood of their being compromised
    * Instead use public key for short time and then discard

    * Public keys can be freely distributed
  - Only private key need to be kept long term and kept securely
- Mechanism connecting public key to user with corresponding private key is Public Key Certificate
  - Public key certificate contains public key and identifies the user with the corresponding private key
  - Public key certificates issued by trusted *Certification Authority*

Public Key Infrastructure

A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. It is required for activities where rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred. W

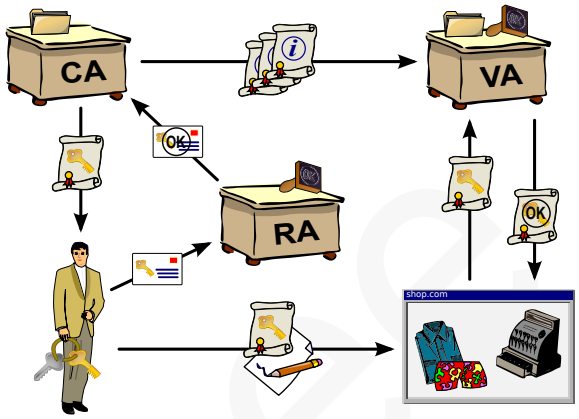


Figure 19: Public Key Infrastructure W

A certificate authority stores, issues and signs the digital certificates. A registration authority verifies the identity of entities requesting their digital certificates to be stored at the CA.

Certification Authority

- Responsibilities
  - Policy and procedures
    * how-tos, dos and don'ts of using certificate
    * processes that should be followed by users, organizations, service providers and consequence for violating them
  - Issuing certificates
    * often need to delegate to local *Registration Authority*
    * prove who you are, e.g. with passport, student card
  - Revoking certificates
    * Certificate Revocation List (CRL) for expired / compromised certificates
  - Storing and archiving
    * keeping track of existing certificates, various other information

Cloud Security Challenges

- Single sign-on
  - the grid model needed
  - currently not solved for cloud-based IaaS
  - onus is on non-cloud developers to define / support this
- Auditing
  - logging, intrusion detection, auditing of security in external computer facilities
- Deletion and encryption
  - data deletion with no direct hard disk
  - scale of data
- Liability
- Licensing
  - many license models
  - challenges with the cloud delivery model
- Workflows
  - Many workflow tools for combing SoA services / data flows
  - Many workflow models
  - Serious challenges of defining, enforcing, sharing, enacting
  - Security-oriented workflows
- The ever-changing technical / legal landscape

我非你杯茶 也可盡情地喝吧
別遺忘有人在為你聲沙

浮誇