Notes for Cluster and Cloud Computing(COMP90024)

http://blog.yzzhan.info/post/clustercomputing/

宝典在手 万事无忧

正式闭典。祝有缘人得高分。

网络一线牵

珍惜这段缘

试卷

2013-2015 答案

https://github.com/AlanChaw/COMP90024-Cloud_Computing-Review/blob/master/Past%20exams%20and%20answers/2013%20Exam%20Answer.md

说明

蓝色字体 -- wyy补充

橙色字体 -- zcz补充

Information Session

Cloud Characteristics:

- **On-demand self-service**: A consumer can provision computing capabilities as needed without requiring human interaction with each service provider.
- **Networked access:** Capabilities are available over the network and accessed through standard mechanisms that promote use by hetergeneous client platforms.
- **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.

- **Rapid elasticity:** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly upon demand.
- **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 service.

Cloud Computing Flavours:

- Compute Clouds:
  - Amazon Elastic Compute Cloud
  - Azure
- Data Clouds:
  - Amazon Simple Storage Service
  - Google docs
  - iCloud
  - Dropbox
- Application Clouds:
  - App Store
  - Virtual Image Factories
  - Community-specific
- Public/Private/Hybrid/Mobile/Health Clouds

History

Distributed System

- once had detailed standards
- then had open distributed processing with slightly less rigorous compliance demands
- The first focus is on Transparency and Heterogeneity of computer-computer interactions. (Finding -> Binding -> checking -> invoking) in heterogeneous environment.

Grid Computing

- Grid Computing transfer computer-computer focus to organization-organization focus.
- Grid computing is distinguished from conventional high-performance computing systems such as cluster computing in that grid computers have

each node set to perform a different task/application. Grid computers also tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers.
- Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes. Grids are often constructed with general-purpose grid middleware software libraries. Grid sizes can be quite large.

Hard of compute Grid

- Information System: What resources are available
- Monitoring and Discovery Systems: What status of those resources
- Job scheduling/resource brokering: Please run these {jobs}
- Virtual organization support
- Security

Domain Driver

Challenges happen in multiple perspectives in research domains. - Big data - Big compute - Big distribution - Big collaboration - Big security

The solution to these challenges focus on `Computing Scaling`, `Network Scaling`.

Computing Scaling

Challenge of computing scaling

- Vertical Computational Scaling (Faster processors)
- Horizontal Computational Scaling (More processors): Easy to add more, but hard to design and develop.

General Analysis

Computing Scaling could be divided into: - Vertical Computational Scaling: - Have faster processors - Limits of fundamental physics/matter(nanoMOS) - Horizontal Computational Scaling: - Have more processors: Easy to add more processors but hard to design, develop, test, debug and delpoy. - HTC is far more important than HPC

Multiple meaning of “add more”: - Single machine multiple cores - Typical laptop/PC/server these days - 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(DANGER of delays/lost messages)

Mathematical Analysis

Question: If n processors(cores) are thrown at a problem how much faster will it go?

Amdahl’s Law

Terminologes:

- Basic terms
    - σ: The time that costs by unparallelable part.
    - π: The time that costs by parallelable part.
    - N: The number of processors
- Derived terms
    - $T(1)=\sigma+\pi$
    - T(1)=σ+π: Time for serial computation without enhanced by parallelism.

- $T(N)=\sigma+$
- π
- N
- T(N)=σ+πN: Time for N parallel computations with max enhanced by parallelism.
- S(N)=T(1)/T(N): Time proportion between no parallelism applied and max parallelism applied.
- $\pi/\sigma=$
- $1-\alpha$
- α
- π/σ=1−αα: Time proportion between cost on parallelable part and unparallelable part.
- α
- α: The proportion of unparallelable part.
- $S=T(1)/T(N)=$
- 1
- $\alpha+(1-\alpha)/N$
- ≈
- 1
- α
- S=T(1)/T(N)=1α+(1−α)/N≈1α: Speed up proportion.

Equation:

$S=1/\alpha$

S=1/α. For example, 95% of paralleled program with infinite numbers of cores will lead to 20 times faster than unparalleled program

Gustafson-Barsis’s Law

Different Terminologies:

- Basic terms:
  - σ: The time that costs by unparallelable part.
  - π: The time that costs by parallelable part.
  - N: The number of processors

- Derived terms
    - $T(1)=\sigma+N\pi$
    - T(1)=σ+Nπ: Time for serial computation without enhanced by parallelism.
    - $T(N)=\sigma+\pi$
    - T(N)=σ+π: Time for N parallel computations with max enhanced by parallelism.
    - α
    - α: Fraction of running time sequential program spends on parallel parts.

Equation:

$$S(N)=\alpha+N(1-\alpha)$$

S(N)=α+N(1−α) Speed up S using N processes is given as a linear formula dependent on the number of processes and the fraction of time to run sequential parts.

Example: N=32, 并行后时间128s, 不可并行部分运行12s

$$\alpha=\frac{12}{128}=0.09375$$
$$S(32)=\alpha+32(1-\alpha)\approx 29$$

Comparison

Amdahl’s Law suggests that with limited task, speed up could not be too fast.

Gustafson-Barsis’s Law suggests that with enough processors and remaining tasks, speed up will always meet the requirement.

Amdahl’s law: 并行处理器的数量在达到充分大时，已经不能有效改善总体的处理性能。

Barisis’s law : 处理器越大，计算量增加越大，计算精度越高

Correctness in parallelisation requires synchronisation. Synchronization and atomic operations causes loss of performance and communication latency.

Further parallel improvement

Basic Architecture in a simplest level

A computer comprises: - CPU for executing programs - Memory that stores/executing programs and related data - I/O systems - Permanent Storage for read/writing data into out of memory

Important issue is to balance the components especially for HPC systems.

To deal with data, here are four different kinds of combination.

	Single Instruction	**Multi Instruction**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- Single Instruction, Single Data Stream (SISD):
    - Single control unit (CU/CPU) fetches single Instruction Stream from memory.
    - Sequential computer which exploits no parallelism in either the instruction or data streams
- Multiple Instruction, Single Data stream (MISD):
    - Parallel computing architecture where many functional units (PU/CPU) perform different operations on the same data
    - Example: fault tolerant computer architectures, multiple error checking on the same date source
- Single Instruction, Multiple Data stream (SIMD):
    - focus is on data level parallelism, only a single process (instruction) at a given moment(Concurrency)
    - multiple processing elements that perform the same operation on multiple data points simultaneously

    - Example: to improve performance of multimedia use such as for image processing
- Multiple Instruction, Multiple Data stream (MIMD)
    - Number of processors that function asynchronously and independently.
    - Machines can be shared memory or distributed memory categories.
    - Example: HPC.

Approaches for parallelism in different level

- Explicit vs Implicit parallelism
    - Explicit Parallelism: programmer is responsible for parallelism effort.
    - Implicit Parallelism: Compiler is responsible for identifying parallelism and scheduling of calculations and the placement of data.
- Hardware: Multiple cores that can process data and perform computational tasks in parallel. To address the issue that CPU not doing anything whilst waiting for caching. Many chips have mixture cache L1 for single core, L2 for pair cores and L3 shared with all cores.
- Operating System:
    - Compute parallelism
    - Processes
    - Threads
    - Data parallelism
    - Caching
- Software/Application: Programming language supports a range of parallelisation/concurrency features.

Data Parallelism Approaches

- Distributed Data/Distributed File System -> CAP(Consistency, Availability, Partition Tolerance)
- To guarantee the data/transaction is reliable, ACID(Atomicity, Consistency, Integrity, Durability) need to be satisfied.

Erroneous Assumptions of Distributed Systems

- The network is reliable
    - Reliable means that there is guarantee that data can send and arrive successfully over the network.

    - Reliable means that lower layers in the networking stack protect me from these issues
- Latency is zero
    - Latency means that the time cost by transmission. Latency becomes more and more significant if distance become further.
- Bandwidth is infinite
    - Bandwidth means that amount of data could be sent in a unit of time.
- The network is secure
    - SQL injections
    - Man in the middle attacks
    - Masquerade
    - Trojans/Viruses
- Topology doesn’t change
    - Topology means that the route taken from source to destination. Routine selected will decide:
    - Latency
    - IP
    - Service
    - Routine is not fixed unless specific routing protocols selected.
- There is one administrator
    - Administrator: There should be an adiminstrator for every module.
- Transport cost is zero
- The network is homogeneous
    - There are multiple approaches to design parallel or distributed systems
    - No single algorithm
    - No single technical solution
    - Eco-system of approaches explored over time and many open research questions/challenges
- Time is ubiquitous
    - Protocols: NTP synchronizes participating computers to within a few milliseconds of Coordinated Universal Time(UTC).

Design of parallel Program

The flow of parallel program design could be divided into four key points.

- Partitioning
    - Primary idea: decomposition of computational activities and data into smaller tasks
    - Strategies:

    - Master-worker/Slave Model: Master decomposes the problem into samll tasks, distributes to workers and gathers partial results to produce the final result.
    - SPMD: common exploited model(Mapreduce). The main idea is that each process executes the same piece of code, but on different parts of the data.
    - Pipeline: Suitable for applications involving multiple stages of execution, typically operate on large number of datasets.
    - 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.(eg: merge sort)
    - Speculation: Used when it is quite difficult to achieve parallelism through the previous paradigms.
- Communication
    - informaton stream and coordination among tasks that are created in the partitioning stage.
- Agglomeration
    - Tasks and communication created in above stages are evaluated for performance and implementation cost
    - Tasks may be grouped into larger tasks to improve communication
    - Individual communications can be bundled
- Mapping/Scheduling
    - Assign tasks to processors such that job completion time is minimized and resource utilization is maximized.

Parallel System, Distributed Computing and HPC/HTC

HTC (high throughput computing)

HTC jobs generally involve running multiple independent instances of the software on multiple processors, at the same time. Serial systems are suitable for these requirements.

HTC is worthwhile when one needs to:

- run many jobs that are typically similar but not highly parallel;
- run the same program with varying inputs;
- run jobs that do not communicate with each other;

- execute on physically distributed resources using grid-enabled technologies;
- make use of many computing resources over long periods of time to accomplish a computational task.

HPC (high performance computing)

HPC jobs generally involve running a single instance of parallel software over many processors. Results at various instances throughout the computation are communicated among the processors, requiring a parallel environment.

HPC: use when one needs to:

- run jobs where rapid communication of intermediate results is required to perform the computations;
- make intense use of large amounts of computing resources in relatively short time periods.

Linux Environment and Modules
Environment modules provide for the dynamic modification of the user's environment via module files.
命令
module help / avail / whatis <modulefile> / display <modulefile> /load <modulefile>/ unload <modulefile>/ switch <modulefile1><modulefile2>/ purge

Batch Systems and Work Managers

The Portable Batch System is a utility software that performs job scheduling by assigning unattended background tasks expressed as batch jobs among the available resources.

Slurm

A. 命令大全：

1. 检查队列
   squeue | less 或者 showq -p cloud | less
2. 提交任务
   sbatch [jobscript]
3. 检查工作状态
   squeue -j [jobid] 或者 scontrol show job [jobid]
4. 删除任务

scancel [jobid]

5. 共享内存多线程 多核
   SBATCH --cpus-per-task = 8
6. 分布式内存 多节点
   SBATCH --nodes=2
   SBATCH --ntasks-per-node =4
7. 数组 apply the same task across multiple datasets :
   SBATCH --array = 1- 10 myapp ${SLURM_ARRAY_TASK_ID}.csv
8. 依赖条件：
   after/ afterok/afternotok/before/beforeok/beforenotok
   eg: #SBATCH --dependency = afterok:myfirstjobid mysecondjob

B. Interactive job

For real-time interaction, with resource requests made on the command line, an interactive job is called. This puts the user on to a compute node.
交互式任务是一种特殊的队列任务，在该模式下，用户可以直接登录到计算节点，此后 所有的操作都在这个节点上进行。这个功能主要是方便用户在服务器上调试程序，以便能够实时看到程序的输出。

when to do that?

user wants to run a large script, or wants to test or debug a job.
# sinteractive --nodes=1 --ntasks-per-node=2

C. X-Windows Forwarding

交互通过图形显示
需要补充

D. Shared Memory Parallel Programming

One form of this is multithreading, whereby a master thread forks a number of sub-threads and divides tasks between them. The threads will then run concurrently and are then joined at a subsequent point to resume normal serial application.

Implementation --- OpenMP(Open Multi-Processing)

It's an application program interface that includes directives for multi-threaded, shared memory parallel programming.
pros: it's an easier form of parallel programming.
limited: it is limited to a single system unit(no distributed memory) and is thread-based rather than using message passing.

E. Distributed Memory Paraller Programming

MPI(Message Passing Interface) is one of the most well popular standards along with a popular implementation as openMPI.
The core principle is that many processors should be able cooperate to solve a problem by passing messages to each through a common communications network.

MPI代码示例
MPI_COMM_WORLD.Rank()
MPI_COMM_WORLD.Size()
MPI.Init()
MPI.Finalize()
MPI_Status()
MPI_Request()
MPI_Barrier()
MPI_Wtime()
MPI_Reduce & MPI_Allreduce
MPI_Bcast
MPI_Scatter
MPI_Gather
有待补充

Cloud Computing

Concepts & Development

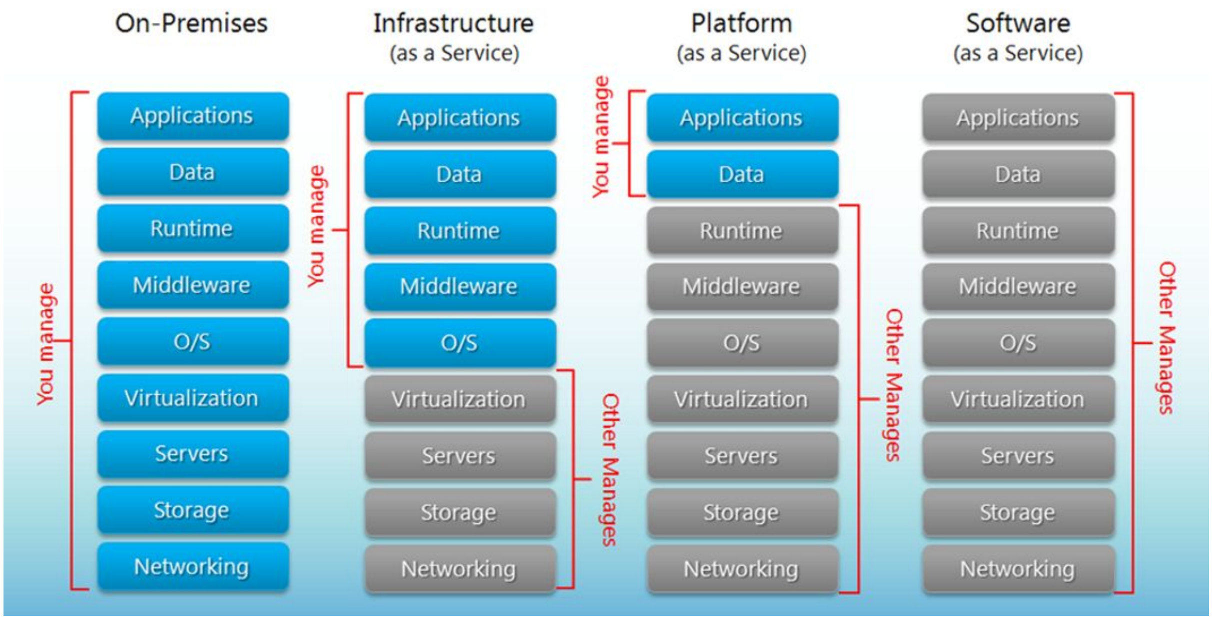
Before cloud computing, the **main issue** is that capacity and utilization could not tightly meet.

- Even if peak load can be correctly anticipated, without elasticity we waste resources during nopeak times.
- Underprovisioning 1: potential revenue from users not serverd
- Underprovisioning 2: some users desert the site permanently after experiencing poor service; this attrition and possible negative press result in a permanent loss of a portion of the revenue stream.

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

Common Cloud Model could be divided into three aspects:

- Deployment Models:
    - Private
    - Pros: 1. Control 2. Consolidation of resources 3. Easier to secure 4. More trust
    - Cons: 1. Not relevance to core business 2. Staff/Management overheads 3. Hardware obsolescence 4. Over/under utilisation challenges.
    - Community
    - Public
    - Pros: 1. Utility computing 2. Can focus on core business 3. Cost-effective 4. Right-sizing 5.Democratisation of computing
    - Cons: 1.Security 2. Loss of control 3. Possible lock-in 4. Dependency of Cloud provider continued existence
    - Hybrid
    - Pros: 1.Cloud-bursting
    - Cons: Difficult to 1. move data.resources when needed 2. decide what data can go to public cloud 3. public cloud compliant with PCI-DSS
- Delivery Models: Separation of Responsibilities



x-as-a-Service: - Testing-as-a-Service - Management/Governance-as-a-Service - Software-as-a-Service

- Examples:
    - Gmail
    - On-live

    - Microsoft Office 365
- Process-as-a-Service
- Information-as-a-Service
- Database-as-a-Service
- Storage-as-a-Service
- Infrastructure-as-a-Service
    - Examples:
    - Amazon Web Services
    - Oracle Public Cloud
    - Rackspace Cloud
    - NeCTAR
- Integration-as-a-Service
- Security-as-a-Service
- Platform-as-a-Service
    - Examples:
    - Google App Engine
    - Microsoft Azure
    - Amazon Elastic MapReduce

Auto-Deployment —- Ansible

Reasons for auto-deployment: - Esay to forget what software you installed, and what steps you took to configure the system - Manual process is error-prone, can be non-repeatable - Snapshots are monolithic - provide no record of what has changed.

Automation provides: - A record of what you did - Codifes knowledge about the system - Makes process repeatable - Makes it programmable – “Infrastructure as Code”

Configuration management(CM) tools > Configuration management refers to the process of systematically hadnling 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.

Ansible

Ansible is an automation tool for configuring and managing computers.

Pros: - Easy to learn - Minimal requirements - Idempotent(repeatable) - Extensible - Supports push or pull - Rolling updates - Inventory management - Ansible Vault for encrypted data

Unimelb/Nectar Introduction of modules & steps of launching an instance

1. Lauching a new VM:
   - copy pub key
   - select key pair
   - select security group p
   - availability zone

   解释**地域**（**Region**）和**可用区**（**AZ：Available Zone**）：
   地域就是物理意义上的不同地方的机房，这个不同地方，一般来说距离较远，机房之间用光纤直连的成本较高。
   同一个地域之内又分成多个**可用区**，简单点理解，可以认为就是同城不同机房，云计算服务商会从底层的机房电力/网络等层面仔细设计来保障一个可用区出现故障的时候不会影响到另外一个可用区
2. Connecting to VM via SSH
   - priviate key : ssh -i <private-key> <username> @ <hostname>
3. Create a volume
   - must be in the same availability zone as the instance
4. Attach a volume
   - check the device name: sudo fdisk -l
   - create the mounting point: sudo mkdir /mnt/demo
   - format the volume: sudo mkfs.ext4 /dev/vdb
   - mkfs = make file system
   - ext4 = type of file system
   - mount the volume: sudo /dev/vdb /mnt/demo
   - check the result: df -h
5. Installing the application
   - sudo apt-get install vim
6. create snapshots
   - snapshot for an instance
   - snapshot for a volume
7. Restore a snapshot
   - create an instance from an instance snapshot
   - create a volume from a volume snapshot

8. setting up security groups
    - Security groups act as a virtual firewall that controls the traffic for one or more instances
    - It contains a set of security rules
    - Default security group only allows SSH

ReST and Docker(Containalization)

Web Service

Basic Idea – Service-oriented Architecture

When an architecture is completely contained within the same machine. components can communicate directly. However, when components are distributed such a direct approach typically can not be used.

Service(combination and commonality) are often used to form a Service-oriented Architecture(SoA).

Service-oriented Architecture(SoA) Core Ideas:

- A set of externally facing services that a business wants to provide to their customers or partners/collaborators.
- An architectural pattern based on service providers, one or more brokers, and service requestors based on agreed service descriptions.
- A set of architectural principles, patterns and criteria that support modularity, encapsulation, loose coupling, separation of concerns, reuse and composability.
- A programming model complete with standards, tools and technologies that supports development and support of services(could be many flavours of service)
- A middleware solution optimized for service assembly, orchestration, monitoring, and management. Can include tools and approaches that combine services together.

Service-oriented Architecture(SoA) Design Principle:

- Standardized service contract: Services adhere to a communications agreement as defined collectively by one or more service-description documents.
- Service loose coupling: Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
- Service abstraction: Beyond descriptions in the service contract, services hide logic from the outside world.
- Service autonomy: Services have control over the logic they encapsulate.
- Service statelessness: Services minimize resource consumption by deferring the management of state information when necessary.
- Service discoverability: Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- Service composability: Services are effective composition participants, regardless of the size and complexityh of the composition.
- Service Granularity: A design consideration to provide optimal scope at the right granular level of the business functionality in a service operation.
- Service normalization
- Service optimization
- Service relevance
- Service encapsulation
- Service location transparency

SoA for Web

Web services implement SoA with two main flavours

- SOAP-based web services (Simple Object Access Protocol - 简单对象访问协议, 数据交换的协议规范, 交换带结构信息)
- ReST-based web services (Representational State Transfer - 表现层状态转移, 资源在网络中以某种表现形式进行状态转移)
- (Both flavours to call services over HTTP)

and many other flavous - Geospatial Services(WFS, WMS, WPS) - Health services(HL7) - SDMX(Statistical Data Markup exchange)

SOAP/WS

SOAP(Simple Object Access Protocol) is built upon Remote Procedure Call paradigm, a language independent function call that spans another system.

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.

SOAP 提供了自己的协议并专注于公开某些应用逻辑(不是数据)为服务。SOAP 公开的是**操作**。SOAP 专注于访问通过不同的接口实现了某些业务逻辑的命名操作。

- Can use HTTP and other protocols
- Build up on remote procedure call (a language independent function call that spans another system)
- Covers all aspects of a remote services including discovery, description, request/response.
- WS-* (security) goes into SOAP headers for additional functionality.
  RESTful/WS and ROA Architecture
  ReST(Representation State Transfer) is intended to evoke an image of how a well-designed Web app behaves. A network for web pages, where the user progresses through an application by selecting links and resulting in next page then being transferred to the user and rendered for their use.
- REST leverages less bandwidth, making it more suitable for internet usage.

Steps For

Client⇋Resource

Client⇋Resource, Client sends “http://amazon.com/product/123" and Resource return Product.html:

1. Client requests Resource through Identifier(URL)
2. Server/proxy sends representation of Resource
3. This puts the client in a certain state.
4. Representation contains URLS allowing navigation.

5. Client follows URL to fetch another resource.
6. This transitions client into yet another state.
7. Representational State Transfer

A ROA(Resource-Oriented Architecture) is a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web.

A resource is anything that's important enough to be referenced as a thing in itself.

- PUT should be used when target resource URL is known by the client.
- POST should be used when target resource URL is server generated

ReST - Uniform Interface

- Identification of Resources: All important resources are identified by one(uniform) resource identifier mechanism(HTTP URL)
- Manipulation of Resources through representations: Each resource can have one or more representations. Such as application/xml, application/json, text/html. Clients and Servers negotiate to select representation.
- Self-descriptive messages: Requests and resources contain not only data but additional headers describing how the content should be handled. Such as if it should be cached, authentication requirements, etc. Access methods mean the same for all resources
- Keywords: HTTP, GET, HEAD, OPTIONS, PUT, POST, DELETE, CONNECTION, TRACE, PATCH.

HTTP Methods can be

- **safe**: Do not change, repeating a call is equivalent to not making a call at all

比如 ： *GET,OPTION,HEAD*

- **Idempotent**: Effect of repeating a call is equivalent to making a single call. HTTP方法的幂等性是指一次和多次请求某一个资源应该具有同样的副作用

比如：*PUT,DELETE*

- **Neither:**

比如：*POST*

- PUT should be used when target resource URL is known by the client.
- POST should be used when target resource URL is server generated

PUT和POST区别 什么时候用哪个合适？
举一个简单的例子，假如有一个博客系统提供一个Web API，模式是这样
http://superblogging/blogs/post/{blog-name}，很简单，将{blog-name}替换为我们的blog名字，往这个URI发送一个HTTP PUT或者POST请求，HTTP的body部分就是博文，这是一个很简单的REST API例子。
我们应该用PUT方法还是POST方法？**取决于这个REST服务的行为是否是idempotent的**，假如我们发送两个http://superblogging/blogs/post/Sample请求，服务器端是什么样的行为？如果产生了两个博客帖子，那就说明这个服务不是idempotent的，因为多次使用产生了副作用了嘛；如果后一个请求把第一个请求覆盖掉了，那这个服务就是idempotent的。前一种情况，应该使用POST方法，后一种情况，应该使用PUT方法。
　　也许你会觉得这个两个方法的差别没什么大不了的，用错了也不会有什么问题，但是你的服务一放到internet上，如果不遵从HTTP协议的规范，就可能给自己带来麻烦。比如，没准Google Crawler也会访问你的服务，如果让一个不是indempotent的服务可以用indempotent的方法访问，那么你服务器的状态可能就会被Crawler修改，这是不应该发生的。
总结：更新用Put,所有属性一起更新，这是幂等的，用POST就不操心，是不是幂等了

HTTP status code

200:OK 201:Created 202: Accepted 203: Non-Authoritative 300: Multiple Choices
301: Moved Permanently
400: Bad Request 401: Unauthorized 402: Payment Required 403: Forbidden
404: not found 405: Method Not Allowed 406: Not Acceptable 407: Proxy Auth Required
408: Request Timeout 409: Conflict
500: Internal Server Error 501: Not Implemented 502: Bad Gateway
503: Server Unavailbale 504: Gateway Timeout 505: Version Not Supported

Rest Principle （更多官方解释https://restfulapi.net/hateoas/）

1. Addressability （可寻址能力）
2. Uniform Interface (上面有提及具体3点)
3. Resources and Representations instead of RPC
4. HATEOAS （Hypermedia as the Engine of Application State ）

HATEOAS--Hypermedia as the Engine of Application State
-Resource representations contain links to identified resources
-Resources and state can be used by navigating links
- links make interconnected resources navigable
- without navigation, identifying new resources is server-specific
-Restful applications navigate instead of calling
- representations contain information about possible traversals

- application navigates to the next resource depending on link semantics
- navigation can be delegated since all links use identifiers

RPC-orientated systems need to expost the available functions

- functions are essential for interacting with a service
- introspection or interface descriptions make functions discoverable

ReST 2.0

Motivation: Everything as a service(EaaS) paradigm, Vast number of entities and services, Link services together to create workflows and mashups.

Extend the API(Application Programming Interface) notation to facilitate the sharing and usage of services among developers, testers, and in some cases end users.

Comparison between SOA and ROA

- An application built with a Service Oriented Architecture is more a ‘Facade’, e.g. it combines or composes its outgoing functionality based on functionality that is in the services it uses ‘behind the screens’ (possibly over the network). E.g. its core processing consists of calling external services, supplying them with parameters, and combining the results with possibly some extra processing or algorithms for the user.
- An application built with a Resource Oriented Architecture does more of its processing internally (e.g. as opposed to calling external components) but uses external resources as input. E.g. its core processing consists of retrieving static resources and then doing more calculating internally.

Big Data and Related Technologies

DBMS design in Big Data Environment

Four “Vs” relates to Big data:

- **Volume**: volume is one of most important criteria.

- **Velocity:** the frequency at which new data is being brought into system and analytics performed.
- **Variety:** the variability and complexity of data schema. The more complex the data schemas you have, the higher the probability of them changing along the way, adding more complexity.
- **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.

Why Big Data need special database rather than Relational DBMSs

- **RDBMS finds it challenging to handle such huge data volumes.** To address this, RDBMS added more central processing units (or CPUs) or more memory to the database management system to scale up vertically.

- **the majority of the data comes in a semi-structured or unstructured format from social media, audio, video, texts, and emails.** However, the second problem related to unstructured data is outside the purview of RDBMS because relational databases just can't categorize unstructured data. They're designed and structured to accommodate structured data such as weblog sensor and financial data.

- **"big data" is generated at a very high velocity.** RDBMS lacks in high velocity because it's designed for steady data retention rather than rapid growth.

DBMSs for Distributed Environment

- A **key-value store** is a DBMS that allows the retrieval of a chunk of data given a key: fast but crude. (Redis, PostgreSQL,Hstore)

- A **BigTable DBMS** stores data in columns grouped into column families, with rows potentially containing different columns of the same family.(Apache Cassandra, Apache Accumulo)

- A **Document-oriented DBMS** stores data as structured documents, usually expressed as XML or JSON. (Apache CouchDB, MongoDB)

CouchDB Cluster Architecture

- Nodes (几个节点), shards (每个表切几片), replica (重复几次, CouchDB里的n)
- All nodes answer requests (read or write) at the same time
- Sharding (splitting of data across nodes) is done on every node
- When a node does not contain a doc when the request comes, the node requests it from another node and returns to the client
- Nodes can be added / remove easily, the data re-balanced automatically
- Primary node fails, a sub-node is chosed as primary node automatically

Clustered NoSQL DBMS Comparison

Clusters needs to

- Distribute the computing load over multiple computers. (eg. for availability) - Store multiple copies of data. (eg. to achieve redundancy)

Sharding

Sharding is the partitioning of a database “horizontally”(ie. the database rows or documents) are partitioned into subsets that are stored on different servers. Every subset of rows is called a shard.

- Number of shards:
    - larger than the numebr of replica
    - the max number of nodes is equal to the number of shards
- Advanage of shards:
    - Improve the performance through the distribution of computing load across nodes.
    - Make it easier to move data files around(eg. when adding new nodes to cluster)
- Strategy of shards:
    - Hash sharding: distribute evenly across the cluster.
    - Range sharding: similar rows are stored on the same node.

Replication

Replication is the action of storing the same row(or document) on different nodes to make the database fault-tolerant.

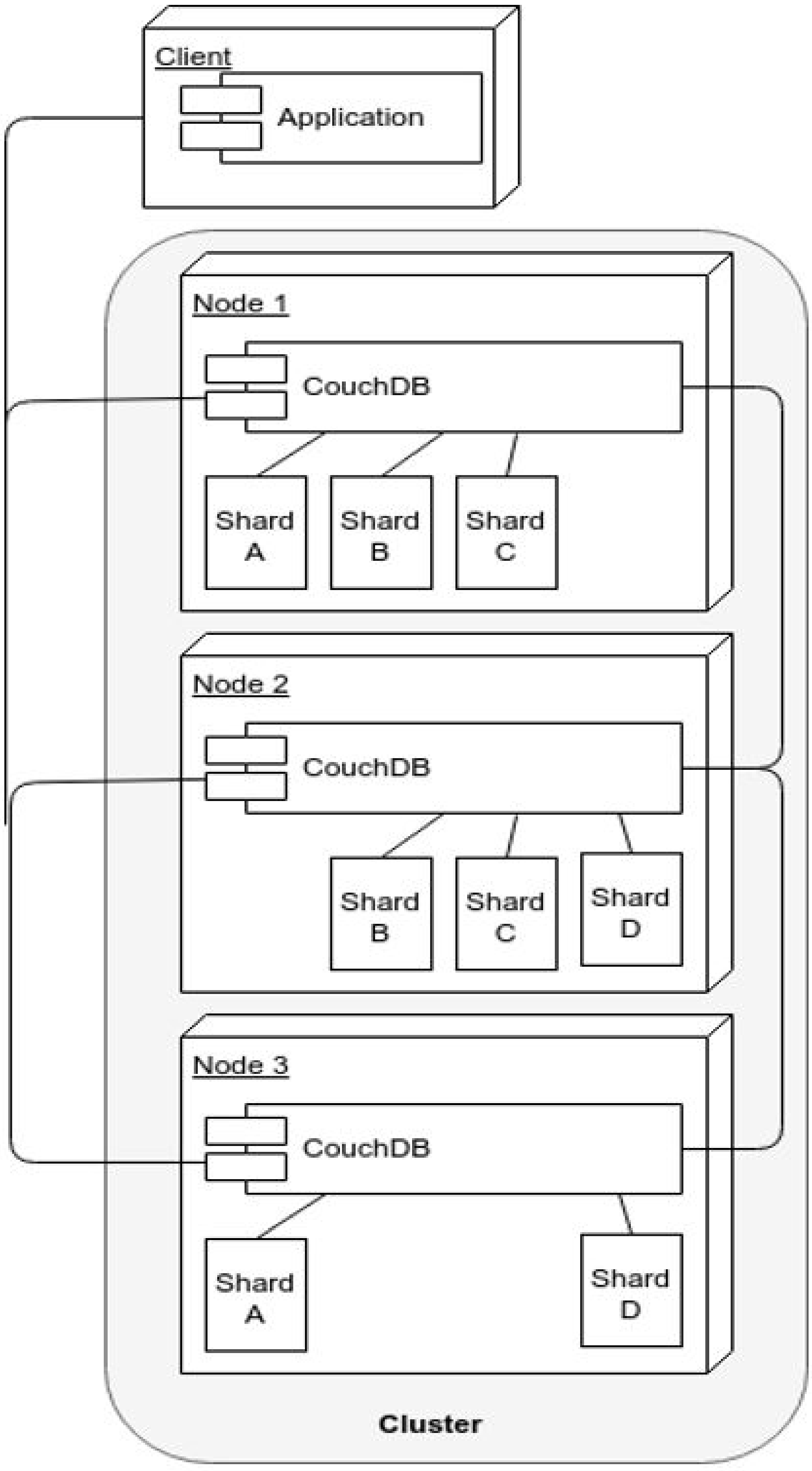
Combined replication and sharding aim to maximize the availability while maintaining a minimum level of data safety.

Partition

- A partition is a grouping of logically related rows in the same shard (for instance, all the tweets of the same user)
- Partitioning **improves performance by restricting queries to a single shard**
- To be effective, **partitions have to be relatively small** (certainly **smaller than a shard**)
- A database has to **be declared “partitioned” during its creation**
- Partitions are a new feature of CouchDB 3.x

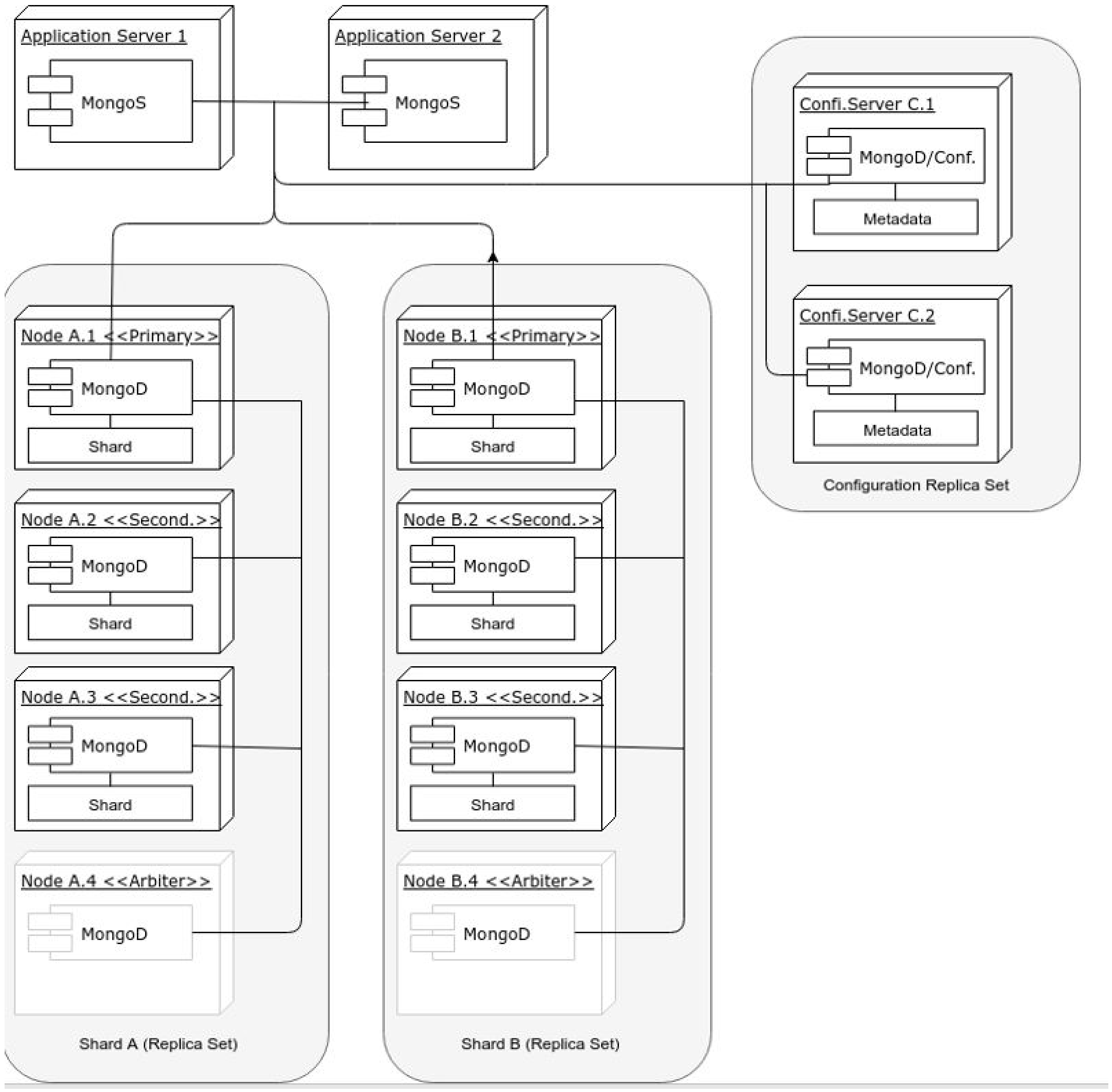
eg: n is the number of replicas(how many times the same data item is repeated across the cluster)
q is the number of shards
n*q is the total number of shard files distributed in the different nodes of the cluster
下图中例子举例
n=2(replica = 2 也就是每个分片被复制了2次)
q=4 一共划分了4片
n*q = 2*4 = 8 为整个集群的分片文件总数

CouchDB Cluster Architecture



CouchDB sharding process: - Sharding is done on every node(One node contains multiple shards) - All nodes answer requests(read or write) at the same time - Nodes can be added/removed easily, and their shards are re-balanced automatically upon addition/deletion of nodes. - If node request don't have respective node, node will request another node for document and return it to client.

MongoDB Cluster Architecture



A mongoDB sharded cluster consists of three components:

- shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.

- mongos: The mongos acts as a query router, providing an interface between client application and the sharded cluster.

- config server: Config servers store metadata and configuration settings for the cluster.

Sharding process:

- Sharding is done at the replica set level(One node contains one shard).

- Write requests is based on primary node in a replica set, Read requests depends on configuration, be answered by every node.

- Updates flow only from primary to secondary.

- Read request by default bases on primary node, it can be changed by modifying the configuration.

mongodb分片和复制架构具体
https://blog.csdn.net/baidu_36095053/article/details/80192239

MongoDB vs CouchDB Clusters

- MongoDB cluster is considerably more complex.
- MongoDB cluster is less available(only primary nodes can talk to client)
- 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 CouchDB, losing access to one quarter of data.

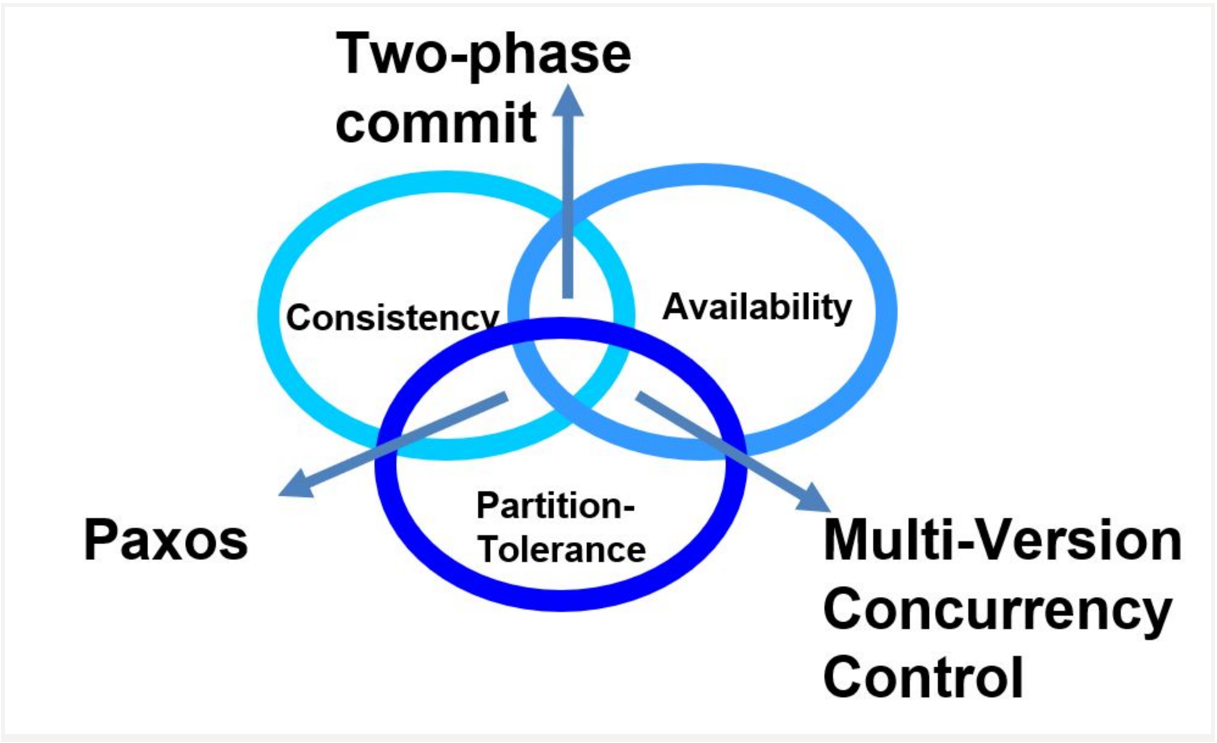
- ○ in MongoDB, losing access to half the data.(Actually there are 10 nodes.)

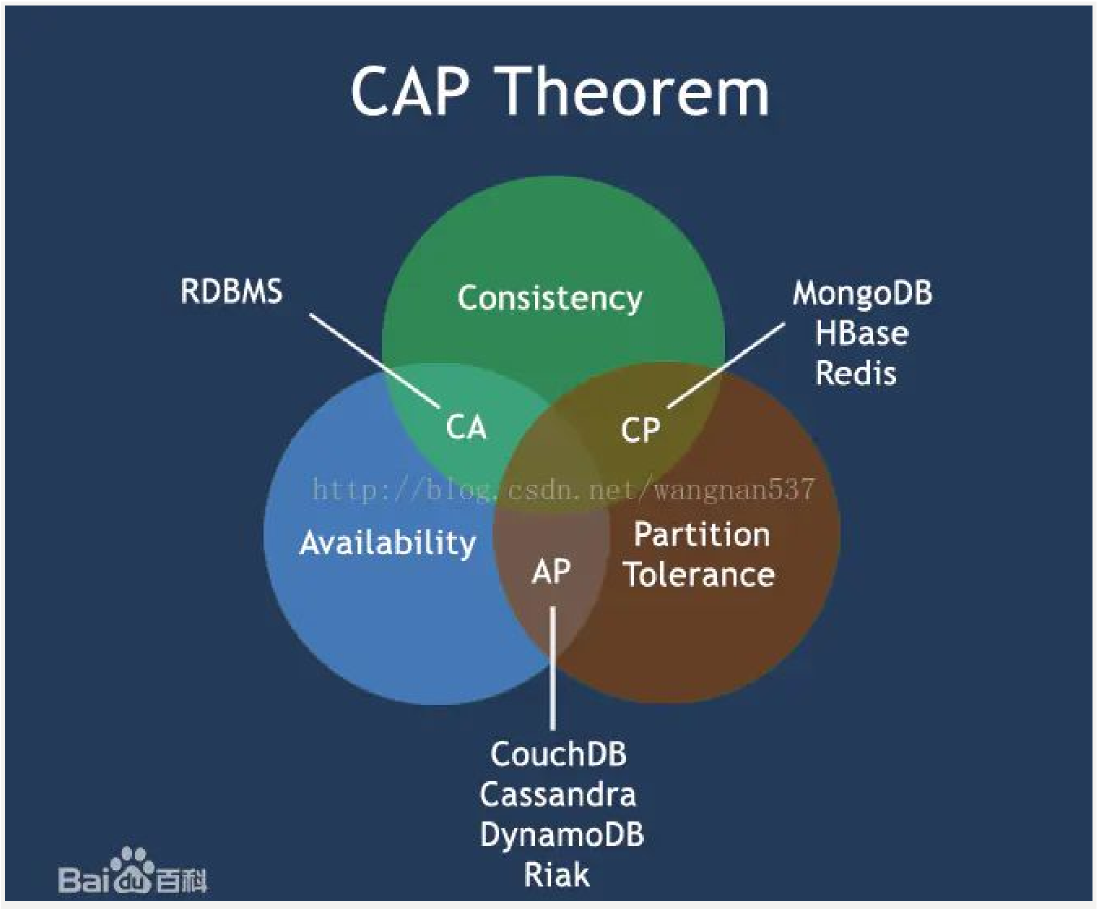
CAP Theorem 更详细解释https://www.jianshu.com/p/88cb4ef57d91

Consisteny: every client receiving an answer receivers the same answer from all nodes in the cluster.

Aailability: 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 rest of the cluster.





Consistency and Availability: Two phase commit

A two-phase commit is a standardized protocol that ensures that a database commit is implementing in the situation where a commit operation must be broken into two separate parts.

关注一致性和可用性，它需要非常严格的全体一致的协议，比如“两阶段提交”（2PC）。CA 系统不能容忍网络错误或节点错误，一旦出现这样的问题，整个系统就会拒绝写请求，因为它并不知道对面的那个结点是否挂掉了，还是只是网络问题。唯一安全的做法就是把自己变成只读的。

The two-phase commit is implemented as follows:

- Phase 1: Each server that needs to commit data writes its data records to the log. If a server is unsuccessful, it responds with a failure message. If successful, the server replies with an OK message.

- Phase 2: This phase begins after all participants respond OK. Then, the coordinator sends a signal to each server with commit instructions. After committing, each writes the commit as part of its log record for reference and sends the coordinator a message that its commit has been successfully implemented. If a server fails, the coordinator sends instructions to all servers to roll back the transaction. After the servers roll back, each sends feedback that this has been completed.

Two-Phase Commit **enforce consistency** and **reduce availability**. It is a good solution when the cluster is co-located, less good when it is distributed.

Consistency and Partition-Tolerance: Paxos

CP (consistency + partition tolerance)：关注一致性和分区容忍性。它关注的是系统里大多数人的一致性协议，比如：Paxos 算法 (Quorum 类的算法)。这样的系统只需要保证大多数结点数据一致，而少数的结点会在没有同步到最新版本的数据时变成不可用的状态。这样能够提供一部分的可用性

https://my.oschina.net/u/3676895/blog/2222607
https://matt33.com/2018/07/08/distribute-system-consistency-protocol/

In Paxos, every node is either a proposer or an accepter: - a proposer proposes a value (with a timestamp) - an accepter can accept or refuse it

Paxos clusters can recover from partitions and maintain consistency, but the smaller part of a partition will not send responses, hence the **availability is compromised.**

Availability and Partition-Tolerance: MVCC
这样的系统关心可用性和分区容忍性。因此，这样的系统不能达成一致性，需要给出数据冲突，给出数据冲突就需要维护数据版本。Dynamo 就是这样的系统。
In MVCC, concurrent updates are possible without distributed locks, since the updates will have different revision number; the transaction that completes last will get a higher revision number, hence will be considered as the current value.

MVCC可能存在的问题--conflict
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)

CouchDB uses **MVCC** and **MongoDB** uses a **mix** of two-phase commit(for replicating data from primary to secondary nodes) and Paxos-like(to elect a primary node in a replica-set)

MapReduce Algorithm

- MAP: distributes data across machines
- REDUCE: hierarchically summarizes them until the result is obtained.

CouchDB 相关代码和知识点

1. adding and deleting a database
    - curl -X PUT “http://localhost:5984/exampledb”
    - curl -X DELETE “http://localhost:5984/exampledb”
2. listing all databases of an instance
    - curl -X GET “http://localhost:5984/_all_dbs”
3. insert a document
    - curl -X POST “http://localhost:5984/exampledb” --header “Content -Type:application/json” --data ‘{“type”:”account”,”holder”:”alice”,”initialbalance”:1000}’
4. retrieve a document
    - curl -X GET “http://localhost:5984/exampledb/c34ds3jkjjalkjdklsadjklajdl”
5. set the ID of a document as “charlie”
    - curl -X PUT “http://localhost:5984/exampledb/charlie” --header “Content -Type:application/json” --data ‘{“type”:”account”,”holder”:”alice”,”initialbalance”:1000}’
6. Error 409 when updating documents -- conflicts:document update conflicts
   MVCC保证可用性机制 MVCC relies on monmotonically increasing revision numbers and the preservation of old object versions to ensure availability(when an object is updated, its versions can still be read)
   The way to avoid conflicts **in MVCC (**避免冲突方法) is to state which revision the update refers to.
7. 有待补充

Big Data Analytics

Introduction

Analytics Example

- Full-text searching (similar to the Google search engine)
- Aggregation of data (e.g. summing up the number of hits by web-page, day, month, etc.)
- Clustering (e.g. grouping customer into classes based on their spending habits)
- Sentiment analysis (e.g. deciding whether a tweet on a given topic express a positive or negative sentiment)
- Recommendations (suggesting additional products to a client during checkout based on the choices made by other clients who bought a similar product)

Challenges of Big Data Analytics

- Reading and writing distributed dataset
- Preserving data in the presence of failing data nodes
- Supporting the execution of MapReduce tasks
- Being fault-tolerant
- Coordinating the execution of tasks across a cluster.

Apache Hadoop

Hadoop

- an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built fro commodity hardware.

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

Hadoop Distributed File System(HDFS)

HDFS break large file into smaller blocks. However, HDFS blocks are much larger than blocks used by an ordinary file system. **Reasons:**

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

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

Hadoop Resource Manager(YARN) — MapReduce Task Manager

YARN deals with executing MapReduce jobs on a cluster. - Central Resource Manager (on the master) - Node Manager (on slave machines)

When a MapReduce job is scheduled for execution on a Hadoop cluster, YARN starts Application Master that negotiates resources with the Resource Manager and starts Containers on the slave nodes.

Apache Spark

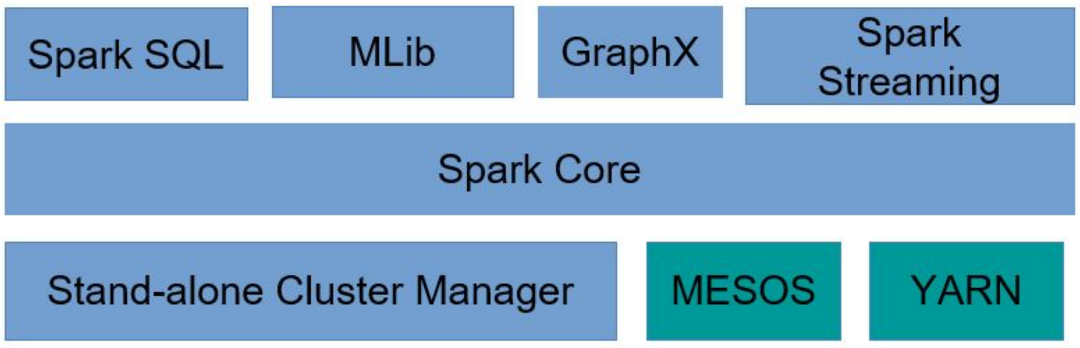
Spark

A fast and general engine for large-scale data processing.

Hadoop MapReduce works well towards performing relatively simple jobs on large datasets. Complex jobs requires strong incentive for caching data in memory and in having finer-grained control on the execuion of jobs.

Spark can operate within the Hadoop architecture, using YARN and Zookeeper to manage computing resources, and storing data on HDFS.

One of the strong points of Spark is the tightly-coupled nature of its main components:



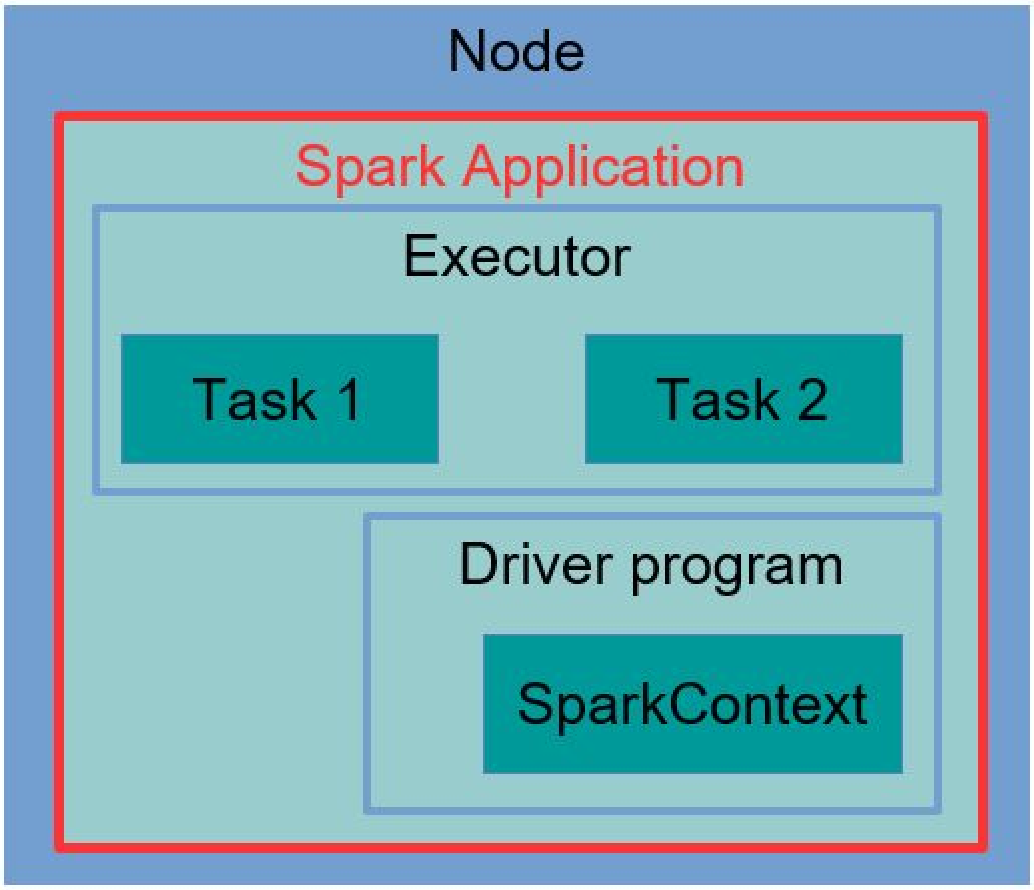
Spark Runtime Architecture

Included Components:

- Job: the data processing that has to be performed on a dataset.
- Task: a single operation on a dataset
- Executors: the processes in which tasks are executed
- Cluster Manager: the process assigning tasks to executors
- Driver program: the main logic of the application
- Spark application: Driver pro0gram + Executors
- Spark Context: the general configuration of the job

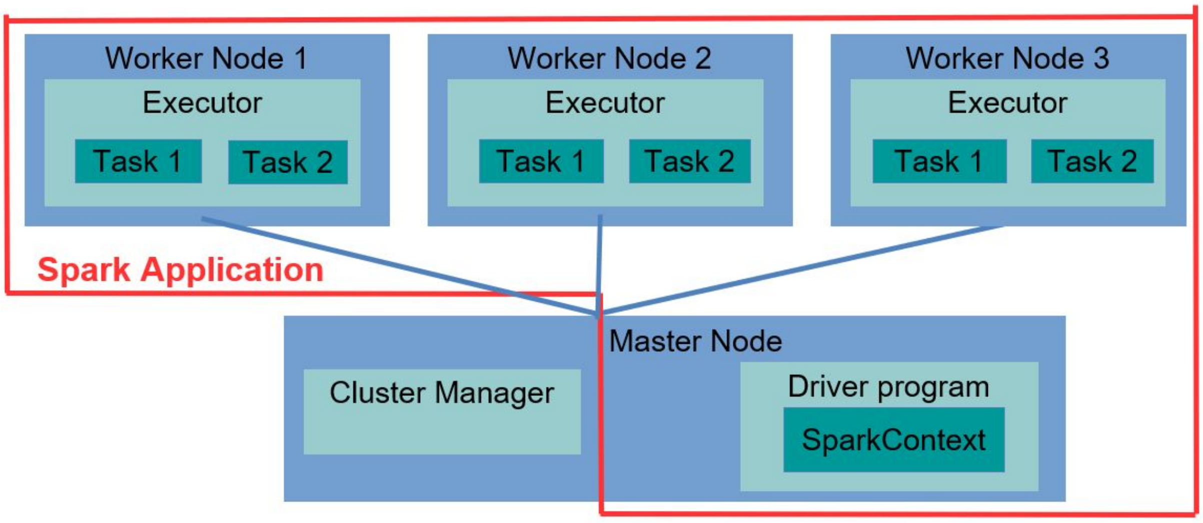
Local Mode

In local mode, every Spark component runs within the same JVM. However, the Spark application can still run in parallel, as there may be more than one executor active.



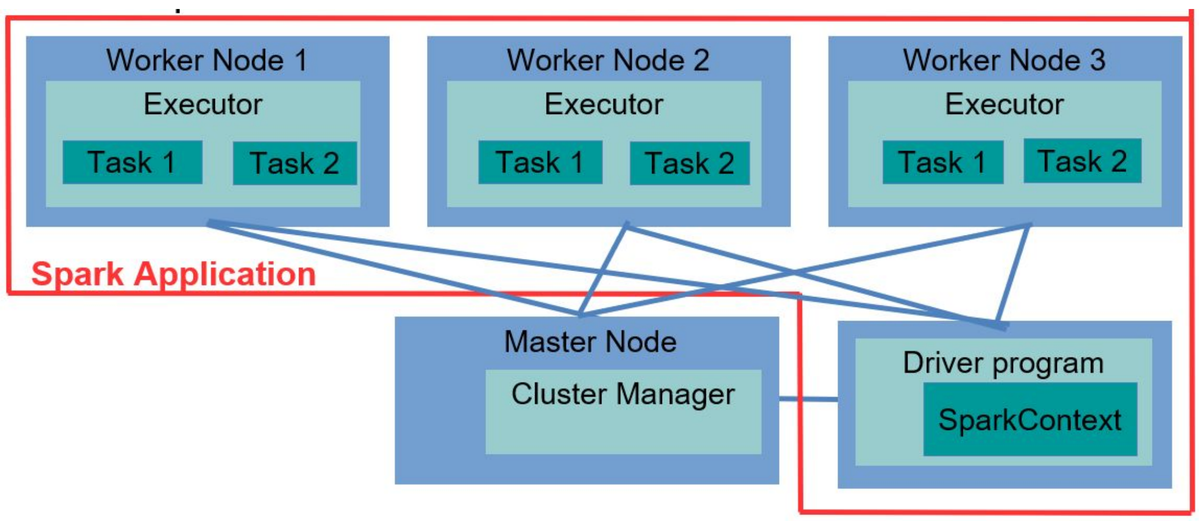
Cluster Mode

In cluster mode, every component is executed on the cluster. Job can run autonomously. This is the common way of running non-interactive Spark jobs.



Client Mode

In client mode, the driver program talks directly to the executors on the worker nodes. Therefore, the machine hosting the driver program has to be connected to the cluster until job completion. Client mode must be used when the applications are interactive.



Spark Context

The deployment mode is set in the *Spark Context*, which is **Spark Context** also used to set the configuration of a Spark application, including the cluster it connects to in cluster mode.

Spark Contexts can also be used to tune the execution by setting the memory, or the number of executors to use.

Example see Lecture 9 Page 29

Resilient Distributed Dataset(RDD)

- **Resilient:** data are stored redundantly
- **Distributed:** data are split into chunks, and these chunks are sent to different nodes
- **Dataset**: a dataset is just a collection of objects, hence very generic

Properties of RDD:

- immutable - transient - lazily-evaluated: Declared transformation will no be executed immediately. Until the action found and executed, the transformation will start executing.

Transformation is executed only after action is executed.

```
/*
 *  Example of Transformations of RDDs
*/

//selects elements from an RDD
rdd.filter(lambda);

//returns an RDD without duplicated elements
rdd.distinct();

//merges two RDDs
rdd.union(otherRdd);
```

```
//returns elements common to both
rdd.intersection(otherRdd)

//removes elements of otherRdd
rdd.substract(otherRdd)

//returns the Cartesian product of both RDDs
rdd.cartesian(otherRdd)

rdd.map(lambda)
rdd.flatMap(lambda)
rdd.reduceByKey(lambda)
rdd.join(otherRdd)
```

```
/*
 *  Example of Action of RDDs
*/

//returns all elements in an RDD
rdd.collect();

//returns the number of elements in an RDD
rdd.count();

//applies the function to all elements repeatedly, resulting
in one result(say, the sum of all elements)
rdd.reduce(lambda)

//applies lambda to all elements of an RDD
rdd.foreach(lambda)

//save files
sf.saveAsTextFile("./counts");
```

Cloud Underpinning and Other Things

Virtualization (lecture 8.1-1)

Virtual Machine Monitor/Hypervisor

The virtualisation layer between the underlying hardware the virtual machines and guest operating systems it supports.

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.

User space⇌Kernel Space

User space⇌Kernel Space by context switch.

Inside the virtual machine, there are Virtual Network Device, VHD(Virtual Hard disk), VMDK(Virtual Machinie Disk), qcow2(QEMU Copy on Write)

Motivation

- Server Consolidation: to improve utilization and reduce energy consumption
- Personal virtual machines can be created on demand
- Security/Isolation
- Hardware independency

Classification of Instructions

- Privileged Instructions: Instructions that trap if the processor is in user mode and do not trap in kernel mode

  特权指令：系统中有一些操作和管理关键系统资源的指令，这些指令只有在最高特权级上能够正确运行。如果在非最高特权级上运行，特权指令会引发一个异常，处理器会陷入到最高特权级，交由系统软件处理了。在不同的运行级别上，指令的执行效果不同，而且并

不是每个特权指令都会引发异常，它有可能被直接忽略。硬件区分特权指令和普通指令的目的，举个例子，就像操作系统提供系统调用一样。

- Sensitive Instructions: Instructions whose behaviour depends on the mode or configuration of the hardware.(different behaviour in different mode)

  操作特权资源的指令，包括修改虚拟机的运行模式或者下面物理机的状态；读写时钟、中断等寄存器；访问存储保护系统、地址重定位系统及所有的I/O指令。

- Innocuous Instructions: instructions that are neither Privileged nor Sensitive

Theorem(Popek and Goldberg)

For any conventional third generation computer, a virtual machine monitor （VMM） may be constructed if the set of sensitive instructions for that computer is a subset of the set of of privileged instructions.

根据Popek和Goldberg的理论，如果指令集支持虚拟化就必须满足**所有的敏感指令都是特权指令**。这样，当Guest OS运行在非最高特权级时，执行任意特权指令都能产生trap。该条件保证了任何影响VMM或VM正确运行的指令在VM上执行时都能被VMM捕获并将控制权转移到VMM上，从而保证了虚拟机环境的等价性和资源可控制性，保证虚拟机正确运行。

具体参考http://leonstudio.org/p/126

Typical Virtualisation Strategy(lecture8.1-1 p12 有待补充)
链接：https://blog.csdn.net/defeattroy/article/details/8802101
VMM needs to support:

1. De-privileging
2. Primary/shadow structures
3. Memory traces

虚拟化场景下，要求将GuestOS内核的特权解除，从原来的0降低到1或者3。这部分特权指令在GuestOS中发生的时候，就会产生Trap，被VMM捕获，从而由VMM完成。这就是虚拟的本质方法，特权解除和陷入模拟(Privilege deprivileging/Trap-and-Emulation)。虚拟化场景中敏感指令必须被VMM捕获并完成。对于一般 RISC 处理器，如 MIPS，PowerPC 以及 SPARC，敏感指令肯定是特权指令，但是x86 例外，但是x86绝大多数的敏感指令是特权指令，但是由部分敏感指令不是特权指令，执行这些指令的时候不会自动trap被VMM捕获。

Aspect of VMMs https://zhuanlan.51cto.com/art/201703/536043.htm

Full virtualisation

Intercept every system-level calls and try to process and deal with it.

- User Apps directly executes code with host hardware
- Privileged instruction by Guest OS is translated to hardware specific instructions.

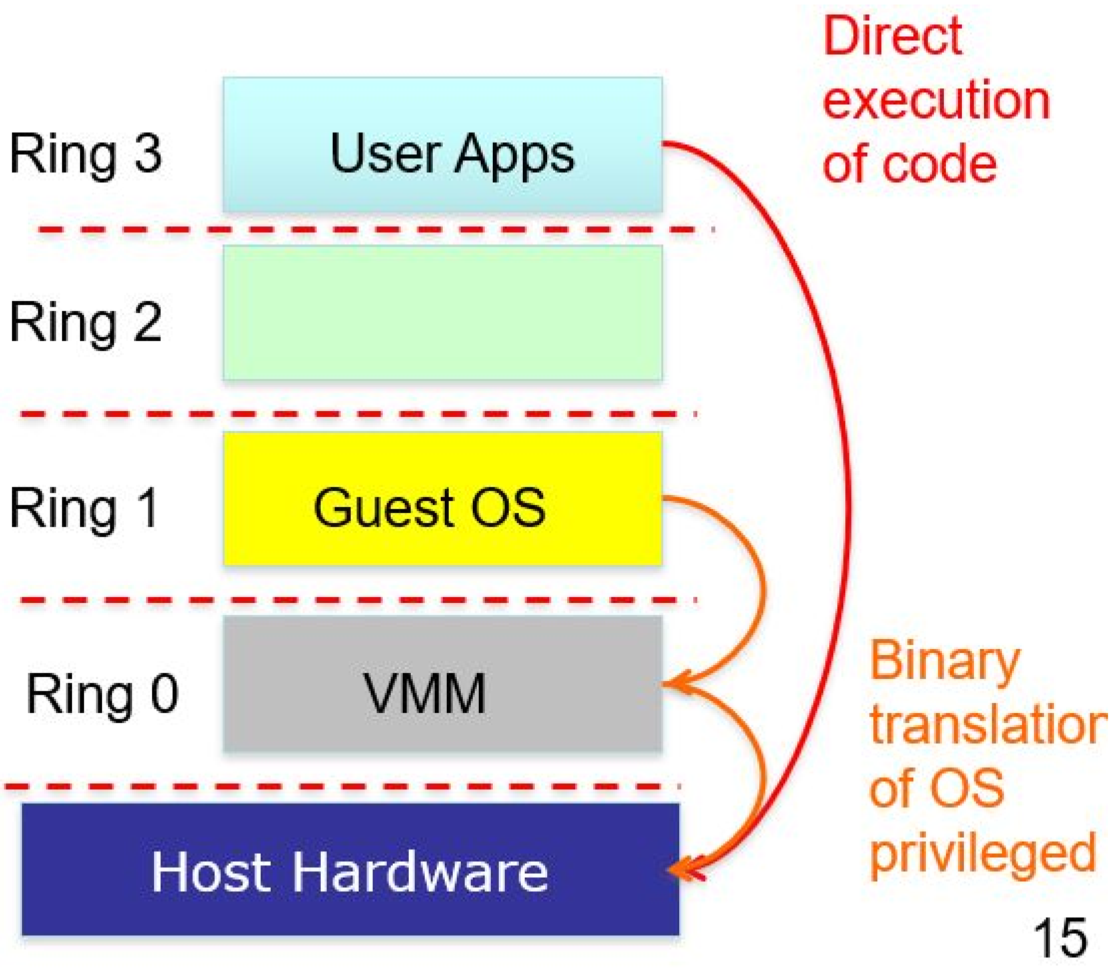
. Guest OS uses extra rings, VMM traps privileged instructions and translates to

hardware specific instructions.

Pros: - Guest is unware it is executing within a VM - Guest OS need not be modified

- No hardware or OS assistance required - Can run legacy OS

Cons: - can be less efficient



Para-virtualisation

provide some available interface for sensitive calls. Pros: - Lower virtualisation overheads - Performance

Cons: - Need to modify guest OS - Less portable - Less compatibility

Hardware-assisted virtualisation

Hardware provides architectural support for running a Hypervisor - New processors typically have this - Requires that all sensitive instructions trappable

Pros: - Good performance - Easier to implement - Advanced implementation supports hardware assisted DMA, memory virtualization

Cons: - Needs hardware support

Binary Translation

Trap and execute occurs by scanning guest instruction stream and replacing sensitive instruction with emulated code

Pros: - Guest OS need not be modified - No hardware or OS assistance required - Can run legacy OS

Cons: - Overhead - Complicated - Need to replace instructions “on-the-fly”

Bare Metal Hyperviosr

VMM runs directly on actual hardware - Boots up and runs on actual physical machine - VMM has to support device driver

Hosted Virtualization

VMM runs on top of another operating system

Operating System Level Virtualisation

It’s a lightweight VMs. Instead of whole-system virtualisation, the OS creates mini-containers - A subset of the OS is often good enough for many use cases - Similar to an advanced version of “chroot”

Pros: - Lightweight - Many more VMs on same hardware - Can be used to package applications and all OS dependencies into container. - Uses same resources as other containers

Cons: - 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

Memory Virtualisation

- Conventional case, page tables store the logical page number and physical page number mappings
- In VMM case, VMM maintains shadow page tables in lock-step with the page tables. Additional management overhead is added.
- Hardware performs guest->physical and physical -> machine translation

Live Migration from vitualisation perspective 有待补充(L8.1 P25)

OpenStack Components

Services: 有待补充每个服务的具体内容

- Compute Service(code-named Nova)
- Image Service(code-named Glance)
- Block Service(code-named Cinder)
- Object Service(code-named Swift)
- Security Service(code-named Keystone)
- Orchestration Service(code-named Heat)
- Network Service(code-named Neutron)
- Container Service(code-named Zun)
- Database Service(code-named Trove)
- Dashboard Service(code-named Horizon)
- Search Service(code-named Searchlight)

Nova

- Manages the lifecycle of compute instances in an OpenStack environment
- Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand - Virtualisation agnostic

HEAT **--**Orchestration Service

Serverless(Function as a Service—FaaS)

“无服务器”架构试图帮助开发者摆脱运行后端应用程序所需的服务器设备的设置和管理工作。这项技术的目标并不是为了实现真正意义上的“无服务器”，而是指由第三方云计算供应商负责后端基础结构的维护，以服务的方式为开发者提供所需功能，例如数据库、消息，以及身份验证等。简单地说，这个架构的就是要让开发人员关注代码的运行而不需要管理任何的基础设施。程序代码被部署在诸如AWS Lambda这样的平台之上，通过事件驱动的方法去触发对函数的调用。很明显，这是一种完全针对程序员的架构技术。其技术特点包括了事件驱动的调用方式，以及有一定限制的程序运行方式，例如AWS Lambda的函数的运行时间默认为3秒到5分钟。从这种架构技术出现的两年多时间来看，这个技术已经有了非常广泛的应用，例如移动应用的后端和物联网应用等。简而言之，无服务器架构的出现不是为了取代传统的应用。然而，从具有高度灵活性的使用模式及事件驱动的特点出发，开发人员／架构师应该重视这个新的计算范例，它可以帮助我们达到减少部署、提高扩展性并减少代码后面的基础设施的维护负担。

FaaS is also know as Serverless computing. The idea behind Serverless/Faas is to develop software applications without bothering with the infrastructure.

It’s Server-unseen rather than Server-less. A FaaS service allows functions to be added, removed, updated, executed, and auto-scaled. FaaS is an extreme form of microservice architecture.

Pros: - Simpler deployment - Reduced computing costs - Reduced application complexity due to loosely-coupled architecture

A function that does not modify the state of the system is said to be **side-effect free.** A function that changes the system somehow is not side-effect free(eg. a function that writes to the file system the thumbnail of an image)

Side-effect free functions can be run in parallel. and are guaranteed to return the same output from the same input. However, side-effect function is inevitable in complex system.

FaaS Application

- Functions are triggered by events
- Functions can call each other
- Functions and events can be combined to build software applications
- For instance: a function can be triggered every hour (say, to compress log files), or every time disk space on a volume is scarce (to remove old log files), or when a pull-request is closed in GitHub, or when a message is stored in a queue
- Combining event-driven scenarios and functions resembles how User Interface software is built: user actions trigger the execution of pieces of code

Stateful/Stateless Function

Stateful Function

is one whose output changes in relation to internally stored information(its input cannot entirely predict its output).

Stateless Function

is one that does not store information internally.

Synchronous/Asynchronous Function

Synchronous Function:

- **By default functons in FaaS are synchronous**, hence they return their result immediately.

Asynchronous Function:

- Asynchronous functions return a code that informs the client that the execution has started, and then trigger an event when the execution completes.
- there may be functions that take longer to return a result, hence they incur timeouts and lock connections with clients in the process, hence it is better to transform them into *asynchronous functions*
- In more complex cases a *publish/subscribe pattern* involving a queuing system can be used to deal with asynchronous functions

Difference between proprietary FaaS services and open-source FaaS framework

- the latter can be deployed on your cluster, peered into, disassembled, and improved by you.

OpenFaaS

- OpenFaaS is an open-source framework that uses Docker containers to deliver FaaS functionality
- every function in OpenFaaS is a Docker container, ensuring loose coupling between functions(Functions can be written in different languages and mixed freely)
- OpenFaaS can use either Docker Swarm or Kubernetes to manage cluster of nodes on which functions run
- By using Docker containers as functions, OpenFaaS allow to freely mix different languages and environments at the cost of decreased performance, as containers are inherently heavier than threads. However, by using a bit of finesse(策略), a container with a single executable, can weight only a few MBs.
- calling a function in OpenFaaS is done via POST HTTP request:
  curl -XPOST “http://0.0.0.0:8080/function/wcmp” --data “hjkgftyfuy” (we can use either GET or POST methods, but POST allows data of arbitrary size to be passed in the body)
- Auto-scalability

OpenFaaS can add more Docker containers when a function is called more often, and remove containers when the function is called less often.
The scaling-up and down of functions can be tied to memory or CPU utilization as wellt

- 有待补充tutorial等代码示例

Security and Clouds

Authentication

Authentication is the establishment and propagation of a user's identity in the system.

Does not check what user is allowed to do, only that we know who they are.

Masquerading danger

Public Key Infrastructures(PKI)

Certification Authority – Central component of PKI

CA's responsibilities:
- policy and procedures
  Processes that should be followed by users, organisations, service providers.
- Issuing certificates
  Often need to delegate to local Registration Authority(Prove who you are)
- Revoking certificates
  Certificate Revocation List(CRL) for expired/compromised certificates
- Storing, archiving
  Keeping track of existing certificates, various other information

CA流程步骤 需要补充 lecture 10.1 p16
也可以具体参考 https://www.jianshu.com/p/c65fa3af1c01

- PKI is an arrangement that binds public key with respective identities of entities(like people and organization).

- The binding is established through a process of registration and issurance of certificates at and by a certificate authority
- The PKI role that assures valid and correct registration is called a registration authority(RA). RA is responsible for accepting requests for digital certificates and authenticating the entity making the request.

Authorisation

Authorisation

is concerned with controlling access to resources based on policy.

There are many approaches for authorisation

- Group Based Access Control
- Role Based Access Control
- Identity Based Access Control
- Attribute Based Access Control

Authorization defines what they can do and define and enforce rules. It is realized through Virtual Organization(VO).

- Provide conceptual framework for rules and regulations for resources to be offered/shared between VO members.
- Different domains place greater/lesser emphasis on expression and enforcement of rules and regulations

RBAC - Basic idea is to define: - Roles: roles are often hierachical - Actions: allowed/not allowed for VO members - Resources: comprising VO infrastructure(computer, data) - Policy consists sets of these rules:

$$Role \times Action \times Target$$

Role×Action×Target - eg. Can user with VO role X invoke service Y on resource Z - Policy engines consume this information to make access decisions

Other Cloud Security Challenges

- Single sign-on
    - The grid model needed but not solved for Cloud-based IaaS currently.
- Auditing: logging, instrusion detection, auditing of security in external computer facilities.
    - well established in theory and practice and for local systems.
- Deletion and Encryption
    - Data deletion with no direct hard disk
- Liability
- Licensing
    - Challenges with the Cloud delivery model
- Workflows
- The Ever Changing Techinal/Legal Landscape

Auto-deploying by Ansible

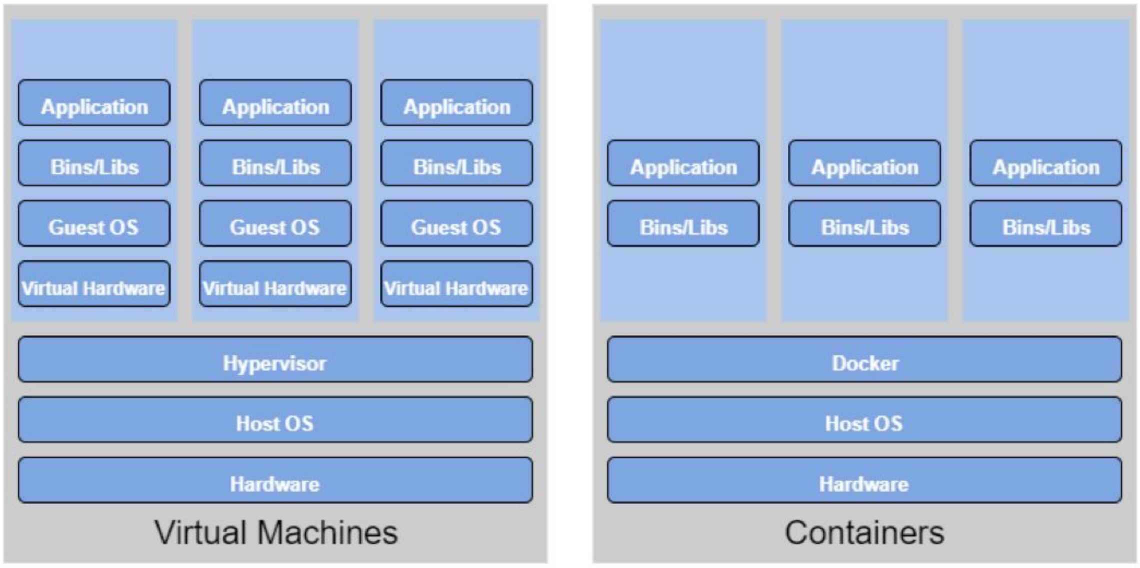
Docker

Background Introduction

Virtualization vs. Containerization

The advantages of virtualization, such as application containment and horizontal scalability come at a cost: resources.

The containerization allows virtual instances to share a single host OS to reduce wasted resources since each container only holds the application and related binaries. The rest are shared among the containers.



Container

Container is similar to concept of resouce isolation and allocation as a virtual machine.

Without bunding the entire hardware environment and full OS.

Container Orchestration Tools -- Kubernetes and Docker SWARM & Openshift

Container orchestration technologies provides a framework for integrating and managing container **at scale.**

- Features: - Networking - Scaling - Service discovery and load balancing - Health check and self-healing - Security - Rolling updates

- Goals: - Simplify container management processes - Help to manage availability and scaling of containers

Introduction to Docker

Docker is by far the most successful containerization technology. It uses resource isolation features of the Linux kernel to allow independent “containers” to run within a single Linux instance.

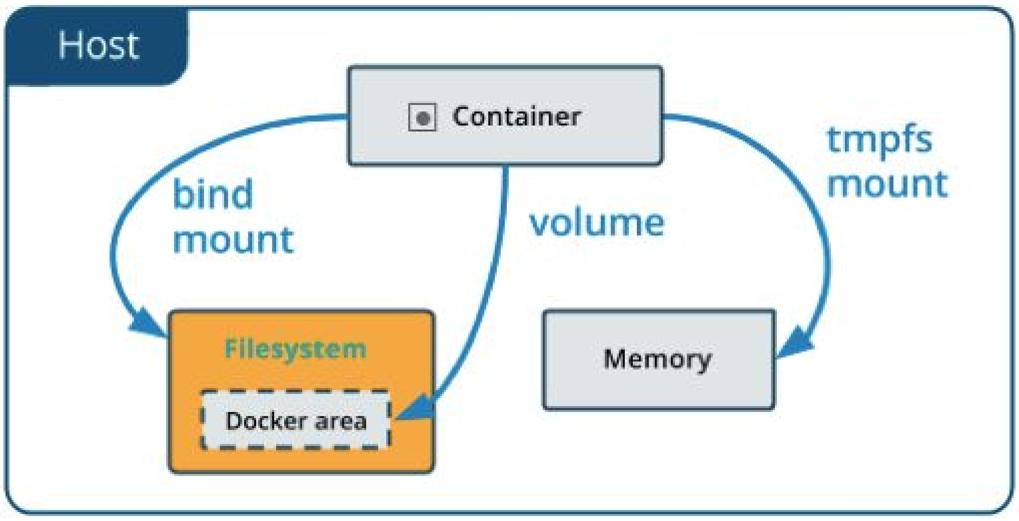
Concepts

- Container:a process that behaves like an independent machine, it is a runtime instance of a docker image.
- Image: a blueprint for a container.
- Dockerfile: the recipe to create an image
- Registry: a hosted service containing repositories of images. (Docker Hub)[https://hub.docker.com]
- Tag: a label applied to a Docker image in a repository.
- Docker Compose: Compose is a tool for defining and running multi-containers Docker applications.
- Docker SWARM: a standalone native clutering / orchestration tool for Docker.

Manage Data in Docker
By default, data inside a Docker container won’t be persisted when a container is no longer exist. Docker provides two options for containers to store files on 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, anywhere on the file system)



Networking

- Network mode “host”: every container uses the host network stack(share the same IP address). Ports cannot be shared across container(Linux only)
- Network mode “bridge”: container can re-use the same port, as they have different IP addresses, and expose a port of their own that belongs to the hosts, allowing the container to be somewhat visible from the outside.

Extra

Normal Web Service requires multiple dependencies such as PHP, MySQL, Redis, Node, etc. Usually we will install the dependencies manually, which however introduces several problems.

- Different services require the same software with different versions.
- Different services in a single environment will modify the file at the same time, such as config of system, nginx
- The same service needs to be deploy manually, which greatly waste the resources.

The following solution is to pack the whole system, which introduces the virtualization technology. It saves the resources but there are lots of other problems.

- Packed virtual machine file including image is pretty large. - Packed virtual machine file - The process of pack could not be automatic.

Docker 命令

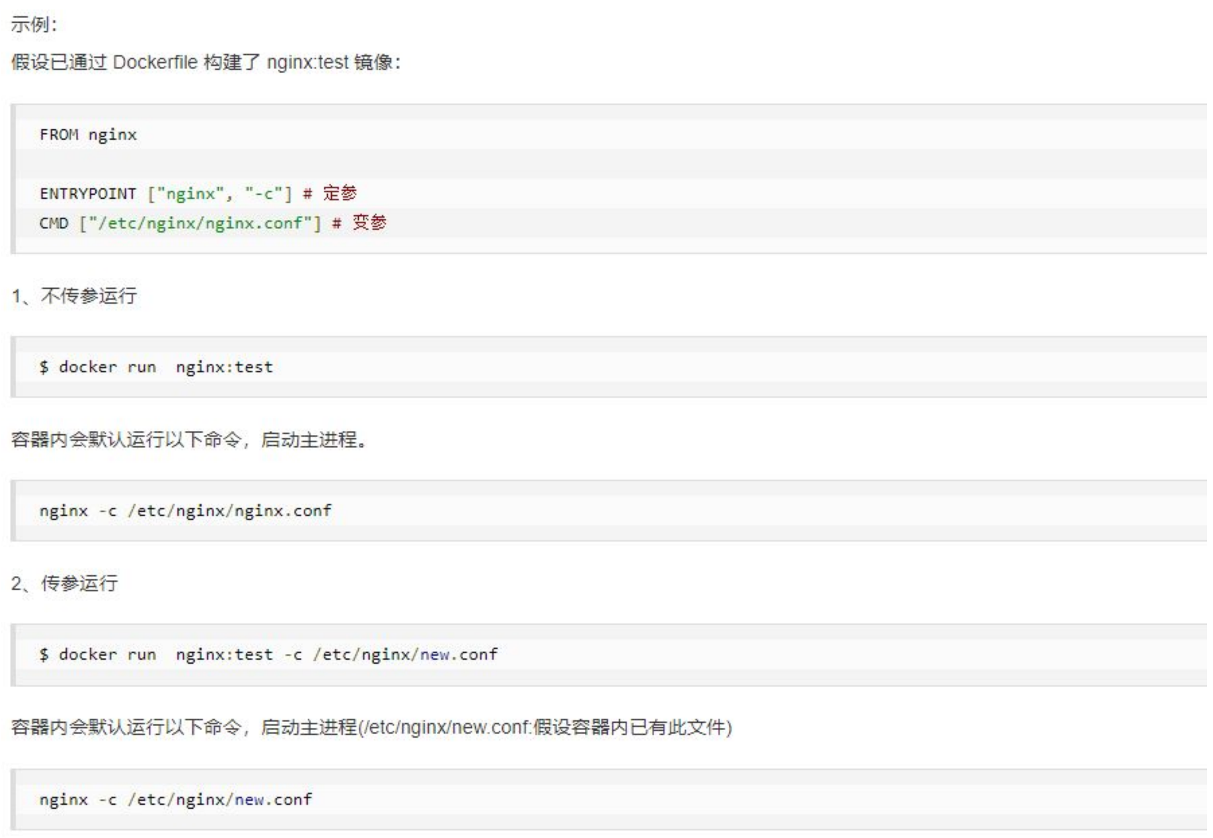
1. login to a public docker registry
   docker login -u username
2. login to a private docker registry
   docker login -u username https://dadada.com
3. logout
   docker logout
4. pull a docker image
   docker pull name:tag
5. list all images
   docker images
6. tag an image
   docker tag <source_image> <target_image>
   docker tag nginx abc_nginx:1.2
7. push an image to registry hub
   docker push abc_nginx:1.2
8. run a docker container
   - create a container,then start the container
     docker create --name nginx -p 8080:80 nginx
     docker start nginx
   - run a container
     docker run --name nginx -p 8080:80 -d nginx
9. list docker containers
   - list running containers
     docker ps
   - list all containers
     docker ps -a
10. stop/restart/remove a docker
    docker restart nginx
    docker stop nginx
    - remove a non-running container
      docker rm nginx
    - remove a running container
      docker rm -f nginx
11. running a shell within a docker
    docker exec -it -w /usr/share/nginx/html/ nginx sh

12. manage data in docker

- **create a volume**
  docker volume create --name htdocs
- **start a container with a volume attached**
  docker run --name nginx-volume -p 8080:80 -v htdocs:/usr/share/nginx/html -d nginx
- **start a container with bind mount attached**
  docker run --name nginx-bind -p 8081:80 -v $(pwd)/htdocs:/usr/share/nginx/html -d nginx
- **Question: with named volume, the content of the container was there,but with bind mount the directory was empty. WHY?**
- **Answer**: A new named volume's contents can be pre-populated by a container.
  当容器外的对应目录是空的，volume会先将容器内的内容拷贝到容器外目录，而mount会将外部的目录覆盖容器内部目录

13. Dockerfile

- FROM
- ENV
- WORKDIR
- COPY
- RUN
- ENTRYPOINT
  ENTRYPOINT gets executed when the container starts. CMD specifies arguments that will be fed to the ENTRYPOINT.



示例：

假设已通过 Dockerfile 构建了 nginx:test 镜像：

```
FROM nginx

ENTRYPOINT ["nginx", "-c"] # 定参
CMD ["/etc/nginx/nginx.conf"] # 变参
```

1、不传参运行

```
$ docker run  nginx:test
```

容器内会默认运行以下命令，启动主进程。

```
nginx -c /etc/nginx/nginx.conf
```

2、传参运行

```
$ docker run  nginx:test -c /etc/nginx/new.conf
```

容器内会默认运行以下命令，启动主进程(/etc/nginx/new.conf:假设容器内已有此文件)

```
nginx -c /etc/nginx/new.conf
```

- CMD

14. create an image
    - docker build -t demo2
    - create a container from image
      docker run --name demo2 -p 8080:80 -d demo2
15. docker compose
    - start the containers
      docker-compose up -d
    - stop the containers
      docker-compose stop
    - remove the containers
      docker-compose down
16. docker swarm -- docker orchestration tool
    - docker machine
      docker-machine create manager
      docker-machine create worker1
      docker-machine create worker2
      docker-machine ls
      docker-machine ssh manager
    - create a docker swarm
      docker swarm init --advertise-addr 192.168.99.100
      docker-machine ssh manager docker swarm join-token manager

- join a docker swarm
  docker-machine ssh worker1 docker swarm join --token aaaaa 192.168.99.100:2377
  docker-machine ssh manager docker node ls 查看当前集群里的所有节点
- create a service
  docker-machine ssh manager docker service create --replicas 3 -p 8083:80 --name nginx nginx:alpine
    - list a service
      docker-machine ssh manager docker service ls
    - check a service
      docker-machine ssh manager docker service ps
    - what if one of the containers stop?
    - It’s self-healing

- scale up /down
  docker-machine ssh manager docker service scale nginx=6
  docker-machine ssh manager docker service scale nginx=2
- rolling update
  docker-machine ssh manager docker service update --image abc:1
nginx