CCC Review

Outline

1. **Lecture 1**

   – Information Session & How we got here (Distributed Systems, Grid…)
   - Richard Sinnott
   - NO workshops

2. **Lecture 2**

   Domain Drivers – tour of some big data projects
   - Richard Sinnott
   - Workshop/demo on driving AURIN (needed for assignment 2)

3. **Lecture 3**

   Parallel Systems, Distributed Computing and HPC/HTC
   - Richard Sinnott
   - Workshop on Git (Farzad)

4. **Lecture 4**

   HPC @ UniMelb and Practicalities of HPC/HTC
   - Richard Sinnott, Lev Lafayette & Farzad Khodadadi
   - Linux / HPC practicalities and welcome to Spartan!!!
   - More using SPARTAN & using mpi4py on SPARTAN workshop (Lev/Farzad)

5. **Lecture 5**

   Cloud Computing – Programming Clouds: Getting to grips with the UniMelb Research Cloud!
   - Richard Sinnott & Farzad Khodadadi & Yao Pan
   - Introduction to Cloud Computing
   - Getting to grips with OpenStack/UniMelb Research Cloud
   - Workshop on Scripting the Cloud (Introduction to Ansible demonstration) (Yao Pan)

6. **Lecture 6**

   ReST, Twitter (Needed for Assignment II) & Docker
   - Richard Sinnott, Farzad Khodadadi & Yao Pan
   - Web services and Representational State Transfer (ReST)
   - Examples of coding/demonstrating ReST and Twitter (Farzad)
   - Introduction to Containers (Yao Pan)
   - Workshop on Demonstration of Docker/Docker SWARM (Yao Pan)

7. **Lecture 7**

   Big Data and Related Technologies
   - Luca Morandini (Data Architect, AURIN)
   - Big Data V-challenges, CAP Theorem and noSQL technologies
   - Workshop on CouchDB via Docker (Luca)

8. **Lecture 8**

   Cloud Underpinnings and Other Things
   - Richard Sinnott & Farzad Khodadadi & Luca Morandini
   - Virtualisation background (Rich)
   - Compare and Contrast AWS with NeCTAR (Farzad)
   - Workshop on serverless architectures and demonstration of openFaaS (Luca)

9. **Lecture 9 Big Data Analytics**

   Big Data Analytics
   - Luca Morandini
   - Big Data Technologies – Hadoop, HDFS, Spark, …
   - Workshop on Hadoop/Spark cluster on Cloud (Luca)

- **Part 1:** Introduction to big data analytics
  - Types of analysis performed
  - Distributed computing on big data
- **Part 2:** Apache Hadoop
  - The Hadoop ecosystem
  - Hadoop Distributed File System
  - Hadoop architecture
  - Programming with Hadoop
- **Part 3:** Apache Spark
  - Why Spark
  - Spark Architecture
  - Programming in Spark
- **Part 4:** Apache Spark Workshop

Lecture1

1. **Cloud computing**

   It is a variety of different types of computing concepts that involve a large number of computers that are connected through a real-time communication network (typically the Internet). cloud computing is a synonym for distributed computing over a network and means the ability to run a program on many connected computers at the same time.
   是一种口语化的表达，用于描述各种不同类型的计算概念，这些概念涉及通过实时通信网络（通常是互联网）连接的大量计算机。云计算是一个行话术语，没有一个公认的非明确的科学或技术定义。在科学上，云计算是网络上分布式计算的同义词，意味着能够在许多连接的计算机上同时运行一个程序。该术语的流行可归因于其在营销中被用于销售托管服务，即在远程位置上运行客户端服务器软件的应用服务供应。

   cloud computing allows companies to avoid upfront infrastructure costs, and focus on projects that differentiate their businesses instead of on infrastructure.

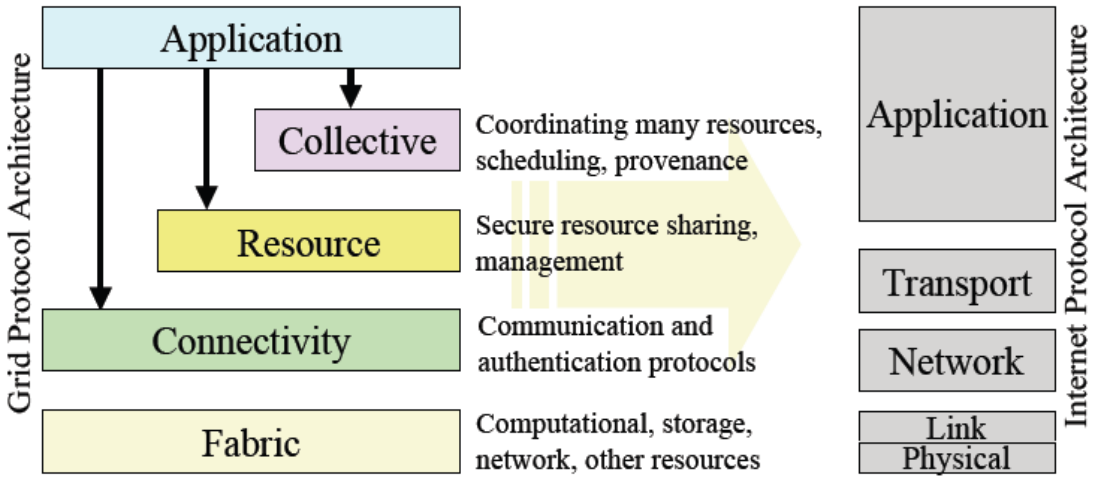
   cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and enables IT to more rapidly adjust resources to meet fluctuating and unpredictable business demand.
   支持者称，云计算使企业能够避免前期的基础设施成本，并专注于使其业务与众不同的项目，而不是基础设施。支持者还声称，云计算使企业能够更快地启动和运行其应用程序，提高可管理性和减少维护，并使 IT 部门能够更迅速地调整资源，以满足波动和不可预测的业务需求。云提供商通常采用 "按需付费 "的模式。

2. **Cloud Characterist**
   a. **On-demand self-service.** A consumer can provision computing capabilities as needed without requiring human interaction with each service provider.
   b. **Networked access**. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous client platforms.
   c. **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.
   d. **Rapid elasticity.** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly upon demand.
   e. **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.
3. **Cloud flavors**
   a. Compute clouds: Amazon Elastic Compute Cloud, Azure
   b. Data clouds: Amazon Simple Storage Service, Google docs
   c. Application clouds: App store,Virtual image factories
   d. Private, public, hybrid, mobile, health, … clouds
4. **Grid computing**

   这种计算模式是利用互联网把分散在不同地理位置的电脑组织成一个“虚拟的超级计算机”，其中每一台参与计

算的计算机就是一个“节点”，而整个计算是由成千上万个“节点”组成的“一张网格”，

a. 网格计算主要是聚合分布资源，支持虚拟组织，满足高端服务。云计算的资源相对集中，主要是以数据中心的形式提供底层资源服务。它通过虚拟技术形成独立的云，云是由许多资源构成的庞大计算池。
b. 网格计算的服务形式是执行作业，当接收到网格高性能调度系统分配给的任务后，在一个阶段内完成作业，产生数据返给用户；而云计算支持持久服务，用户可以利用云计算作为部分 IT 基础设施，实现业务的托管或外包



Compute grid needs:

a. Information Systems: What resources are available, Servers, CPUs, memory, storage, queues.
b. Monitoring and Discovery Systems: What is the status of those resources
c. Job scheduling/resource brokering

Lecture 2 Domain Drivers

1. **Why we need cluster or cloud computing**
   a. Big Data – and the hype!
   b. Big Compute
   c. Big Distribution: Distributed, completely heterogeneous data
   d. Big Collaboration
   e. Big Security
   f. Data is messy
2. 医药企业 储存量 安全性
3. **Swarm**
   a. Anonymous
   b. No leader
   c. Team size flexibility
      Dockerized AWS/NeCTAR
      Kubernetes/Docker SWARM
   d. Arbitrary contributions
4. **Swarm clouds**
   Developed on openStack (NeCTAR)
   - Scripted solution using Docker & Kubernetes

   Deployed to AWS (US)
   - $1000+ / month for basic use
5. **Benefit of script solution**
   a. Developed/test/trialled on free Cloud (NeCTAR)

b. Deployed to AWS when ready
c. Not possible to do if would have used AWS Elastic Container Solution for Kubernetes

6. **Script vs snapshot**
   Script harder but can see how it implement
   Snapshot easier but can see the implement procedure
7. **AURIN**
   Distributed, (completely!) heterogeneous datasets
   Data interrogation services
   Security (unit level data, health data, commercial data!)
   Online analysis tools
   Collaboration!!!
8. **Domain-driven design**
   Domain-driven design (DDD) is an approach to software development for complex needs by connecting the implementation to an evolving model.
   领域驱动设计是一种通过将实现连接到持续进化的模型来满足复杂需求的软件开发方法.

Lecture 3 Distributed and Parallel Computing Systems

1. **Compute scaling**
   a. Vertical Computational Scaling (Faster processors): Limits of fundamental physics
   b. Horizontal Computational Scaling (More processors): Easy to add more, but hard to design and develop.
      a) Single machine multiple cores
         Typical laptop/PC/server these days
      b) Loosely coupled collection/cluster of machines
         Pooling/sharing of resources
      c) Tightly coupled cluster of machines
         Typical HPC/HTC set-up (SPARTAN, NCI, …)
         Many servers in same rack/server room (often with fast message passing interconnects)
      d) Widely distributed clusters of machines
         UK NGS, EGEE, … distributed systems more generally
2. **Amdahl's Law**
   类似极限的思想。假设忽略通信开销。当一个程序有 95%可并行，5%不可并行，那就算这 95%通过并行在一瞬间执行完，剩下的 5%也还是要用那么多时间，所以理论上最大能加速 20 倍.
   Over simplify the problem
3. **Gustafson-Barsis's Law**
   α = \frac{不可并行部分所用的时间}{并行后所用的总时间}

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

   他虽然不能解决线性部分的限制， 但是在给定时间内，更多的资源可以更大规模的问题
4. **Computer Architecture**
   a. CPU for executing programs
   b. Memory that stores/executing programs and related data
   c. I/O systems (keyboards, networks, …)
   d. Permanent storage for read/writing data into out of memory
   e. Of key importance (especially for HPC systems!) is the balance of all of these
5. **Flynn's Taxonomy**
   a. SISD (Single instruction, Single Data stream

Single control unit (CU/CPU) fetches single Instruction Stream from memory.

b. **SIMD (Single instruction, Multiple Data streams)**
multiple processing elements that perform the same operation on multiple data points simultaneously. (Image processing) 现在的电脑常用

c. MISD (Multiple instruction, Single Data stream)
Parallel computing architecture where many functional units (PU/CPU) perform different operations on the same data. (Not common)

d. MIMD (Multiple instruction, Multiple Data streams)
number of processors that function asynchronously and independently **(HPC)**
At any time, different processors may be executing different instructions on different pieces of data machines can be shared memory or distributed memory categories

6. Approaches for Parallelism

a. **Explicit(明确的)** vs Implicit(含蓄的) parallelism
1) Implicit: Supported by parallel languages and parallelizing compliers. Hard to do.
2) Explicit: The programmer is responsible for most of the parallelization effor. Consider SPARTAN.自己规定怎么并行处理

b. Hardware
1) Parallelisation 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.
共享一套计算单元，同一个 cache
2) Multi-core:
Multiple cores that can process data and (in principle!!!) perform computational tasks in parallel. Typically share same cache, but issue of cache read/write performance and cache coherence.
不同的计算单元，同一个 cache
3) 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.
共享内存
4) Non-Uniform Memory Access (NUMA)
Non-uniform memory access (NUMA) 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.
不共享内存

c. Operating System
d. Software / Applications
e. Some or all of those

7. MPI (message passing interface)

MPI_Init	:initiate MPI computation
MPI_Finalize	:terminate computation
MPI_COMM_SIZE	:determine number of processors
MPI_COMM_RANK	:determine my process identifier
MPI_SEND	:send a message
MPI_RECV	:receive a message

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
```

```
comm.Barrier() # synchronization here
```

8. (HT)Condor

A specialized workload management system for compute-intensive jobs developed at University of Wisconsin

a. Offers job queueing mechanisms, scheduling policies, priority schemes, resource monitoring/management
b. User submits jobs to Condor and it chooses when and where to run the jobs, monitors their progress, and informs the user upon completion
c. Allows to harvest “free” (?) CPU power from otherwise idle desktop workstations
d. No need for shared file system across machines

9. **Error Assumption of Distributed system (challenge)**

The network is reliable
Latency is zero
Bandwidth is infinite
The network is secure
Topology doesn't change
There is one administrator
Transport cost is zero
The network is homogeneous
Time is ubiquitous

10. **Challenges with Distribution**
    a. Single point failure.
    b. General assumptions that typically do not hold in the real world.
    c. Dependene analysis is hard for core that uses pointers, recursion, ...;
    d. Loops can have unknown number of iterations
    e. Access to global resources (e.g. Shared variables)
11. **Design stage of parallel program**
    a. Partitioning
       Decomposition of computational activities and data into smaller tasks
    b. Communication
       Flow of information and coordination among tasks that are created in the partitioning stage
    c. Agglomeration
       asks and communication structure created in the above stages are evaluated for performance and implementation cost
       Tasks may be grouped into larger tasks to improve communication
       individual communications can be bundled
    d. Mapping / Scheduling
       Assigning tasks to processors such that job completion time is minimized and resource utilization is maximized.
12. **Parallelisation Paradigms(范例)**
    a. **Master Worker/Slave Model**
       Masterdecomposesthe problem into small tasks, distributes to workers and gathers partial results to produce the final result
    b. **Single-Program, Multiple-Data (SPMD)**
       Each process executes the same piece of code, but on different parts of the data. Data is typically split among the available processors.
    c. **Data Pipelining**
       Suitable for applications involving multiple stages of execution, that typically operate on large number of data sets.
    d. **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.
    e. Speculative(投机的) Parallelism

Lecture 4 Spartan HPC System

1. **HPC**

   High-performance computing (HPC) is any computer system whose architecture allows for above average performance. A system that is one of the most powerful in the world, but is poorly designed, could be a "supercomputer".

2. **Job schedular**
3. **Clustering computing**

   Clustered computing is when two or more computers serve a single resource.

4. **Use spartan**

   Logging in

   Submitting and Running jobs with scripts

5. **Shared Memory vs. Distributed Memory**
   a. Multi-threads: Shared Memory

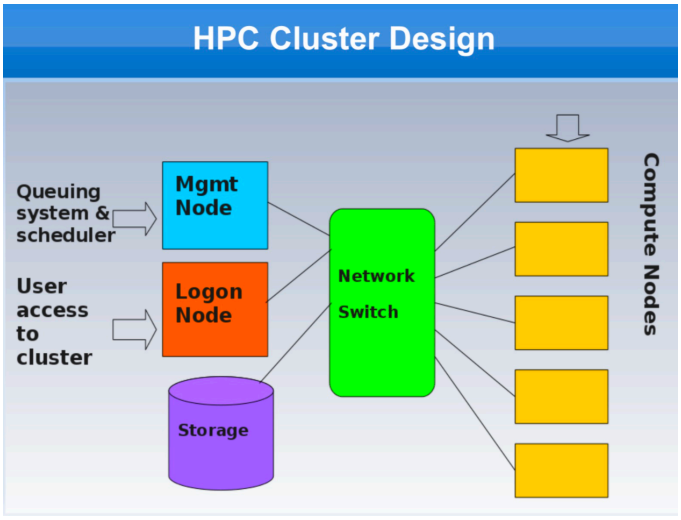
      There is no doubt that OpenMP is an easier form of parallel programming, however it is limited to a single system unit (no distributed memory) and is thread-based rather than using message passing

   b. MPI: Distrubuted Memory Parallel Programming

      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.

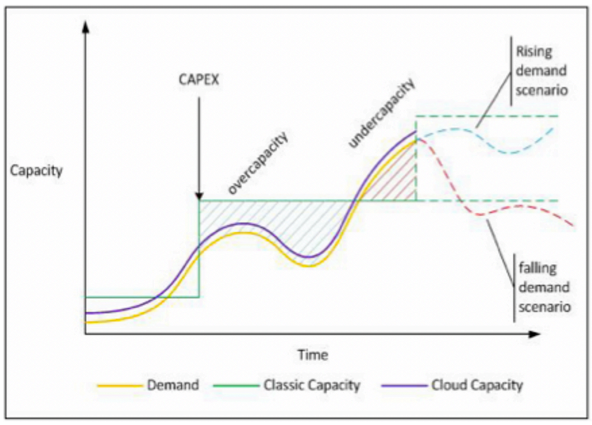
      The programmer is responsible for identifying opportunities for parallelism and implementing algorithms for parallelisation using MPI.

6. **HPC Cluster Design:**



Week 5 Cloud Computing, NeCTAR, Ansible, Git

1. **Cloud vs classic capacity**



2. Deployment Model (Private cloud vs Public cloud)

a. Private

Control, secure, consolidation of resources 整合资源

Utility challange, management overhead(超支) 不懂底层技术知识

b. Public

Utility(随用随取) computing, Can focus on core business, cost-effective

Security proble, loss of control, possible lock-in

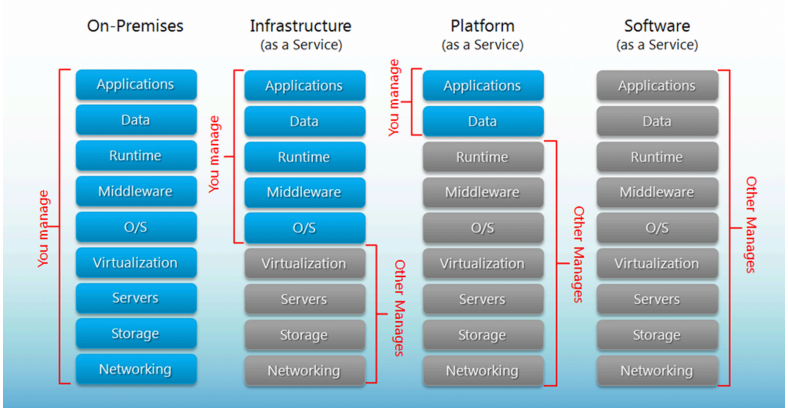
c. Community

d. Hybrid (杂交)

use privae cloud, but burst into public cloud when needed

how to move data?, how to decide which data to be public?

3. Delivery model



a. Software as a Service (**SaaS**)

Gmail, office 365...

b. Platform as a Service (**PaaS**)

Google App Engine, Amazon Elastic MapReduce

c. Infrastructure(底部结构) as a Service (**IaaS**) (primary focus of this course)

AWS, Orical Public Cloud, Nectar...

4. NeCTAR Melbourne research cloud

Based on OpenStack

Open source cloud technology (more later lecture)

Many associated/underpinning services

Compute Service (code-named Nova)

Image Service (code-named Glance)

Block Storage Service (code named Cinder)

Object Storage Service (code-named Swift)

Security Management (code-named Keystone)

Orchestration Service (code-named Heat)

Network Service (code-named Neutron)

Metering Service (code-named Ceilometer)

5. **Automation (Ansible, specifically)**

   Deploying complex cloud systems requires a lot of moving parts

   Automation provides a record of what you did

   Codifies knowledge about the system

   Makes provess repeatable and programmable

6. **Ansible**

   Easy to learn: Playbooks in YAML, templates in Jinja2, sequential execution.

   Minimal requirements: Single command to install, uses ssh to connect to target machine

   Repeatable

   Extensible 模块化

   Supports push or pull

   Rolling updates 不间断更新方式

   Intentory management (存货管理？) 可以放一些参数

   有自己的加密方式，保存信息

   用 yaml 语言，variable 放参数，Intentory 放 ip 地址，使用时需要 nectar 的 openRC 文件

7. **Git**

   Distributed (decentralized) version-control system.

   Keep tracking changes

   Revert to a specific checking point

   Work with people

8. **NeCTAR demo**

   a. **Dashboard**

      Overview (Resource allocation)

      Instances (Create, terminate and configure instances)

      Volumes (Create, terminate, attach and backup)

      Images (Create image, list image)

      Access & Security (Security groups, key pairs, API access)

      Object store (Store data as an object)

   b. **Launching a new VM**

      Flavor (defines the compute, memory, and storage capacity)

      Ephemeral disk (not persistence storage!!!)

      Create key pair

      Copy pub key (cloud.pub)

      Select key pair

      Select security group

      Availability zone

   c. **Connecting to VM via SSH**

   d. **Create a Volume**

      Must be in the same availability zone as the instance

   e. **Attach a volume**

      Check the device name: sudo fdisk -l

         sudo = runs commands with security privileges of another user

      Create the mounting point: sudo mkdir /mnt/demo

      Format the volume: sudo mkfs.ext4 /dev/vdb

      Mount the volume: sudo /dev/vdb /mnt/demo

      Check the result: df -h

   f. **Installing an application**

sudo apt-get install vim

g. **Create snapshots**

Snapshot for an instance

Snapshot for a volume

h. **Restore a snapshot**

Create an instance from an instance snapshot

Create a volume from a volume snapshot

i. **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 (from anywhere)

Create a security group

Week 6 Web Services, ReST Services and Twitter demo

1. **Restful service**

Instead of sending a request and getting some information back just like the SOAP dose, restful way will give the client the information they want together with a bunch of links to other resources that the client may also need and interact with. Each time the client click the link, the state of the client will be changed and reach other resources, the way that lead the client to navigate is called representational state transfer (ReST).

2. **ROA resource-oriented architecture**

A ROA 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. Anything could be resource and you don't care about the service but the resource, the thing is much more important.

3. **Put vs Post**
   a. PUT should be used when target resource url is known by the client
   b. POST should be used when target resource URL is server generated, create a new one
4. **Key step of restful**

The key thing is to make the resources could be navigate, so it can be called restful.

5. **ROA procedure**
   a. Figure out the data set
   b. Split the data set into resources and for each kind of resource
   c. Name the resources with URIs
   d. Expose a subset of the uniform interface
   e. Design the representation(s) accepted from the client
   f. Design the representation(s) served to the client
   g. Integrate this resource into existing resources, using hypermedia links and forms
6. **Rest vs soap**

One of the reasons that the restful way is better than the SOAP is that all service are driven by limit number of options(put, get, post, delete, etc.), so users could easily use these services without spending a lot of time figure out what the service is and how to use it.

7. **Uniform Interface**
   a. Identification of Resources

      All important resources are identified by one (uniform) resource identifier mechanism (e.g. HTTP URL)

      所有重要资源都通过一种（统一）资源标识符机制（例如 HTTP URL）进行标识
   b. Manipulation of Resources through representations

      Each resource can have one or more representations. Such as application/xml, application/json, text/html, etc. Clients

and servers negotiate to select representation.
通过表述来操纵资源
每个资源可以具有一个或多个表示形式。 例如 application / xml，application / json，text / html 等。客户端和服务器协商选择表示形式。

c. Self-descriptive messages
Requests and responses 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 (actions) mean the same for all resources (universal semantics: put, get, post, etc.)
请求和响应不仅包含数据，还包含描述应如何处理内容的其他标头。例如是否应该缓存，身份验证要求等。所有资源的访问方法（动作）均相同（通用语义：put，get，post 等）。

d. HATEOAS - Hyper Media as the Engine of Application State
Let user can navigate instead of just give the simple information

e. Resource representations contain links to identified resources

f. RESTful applications navigate instead of calling

8. HTTP methods (Safe, Idempotent)

a. Safe methods
Do not change repeating a call is equivalent to not making a call at all.

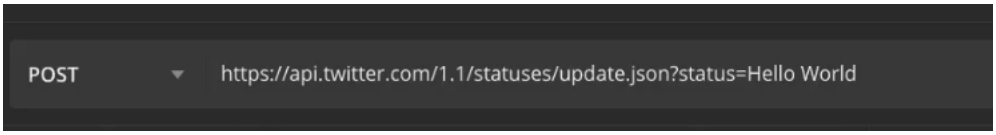
b. Idempotent methods
Effect of repeating a call is equivalent to making a single call

c. GET, OPTIONS, HEAD - Safe

d. PUT, DELETE - Idempotent

e. POST - Neither safe nor idempoten

9. Twitter search api




```
twurl authorize --consumer-key key       \
                --consumer-secret secret
```

```
twurl -X POST /1.1/statuses/destroy/1234567890.json
```

```
curl -XGET http://localhost:5000/todo/api/v1.0/tasks --basic --user miguel:python
```

10. Build a restful api

```
#!flask/bin/python
from flask import Flask, jsonify, abort, request, make_response, url_for
from flask_httpauth import HTTPBasicAuth

```

A server listening at a specific path,
The api to create a new task

```python
@app.route('/todo/api/v1.0/tasks', methods = ['POST'])
@auth.login_required
def create_task():
    if not request.json or not 'title' in request.json:
        abort(400)
    task = {
        'id': tasks[-1]['id'] + 1,
        'title': request.json['title'],
        'description': request.json.get('description', ""),
        'done': False
    }
    tasks.append(task)
    return jsonify( { 'task': make_public_task(task) } ), 201
```

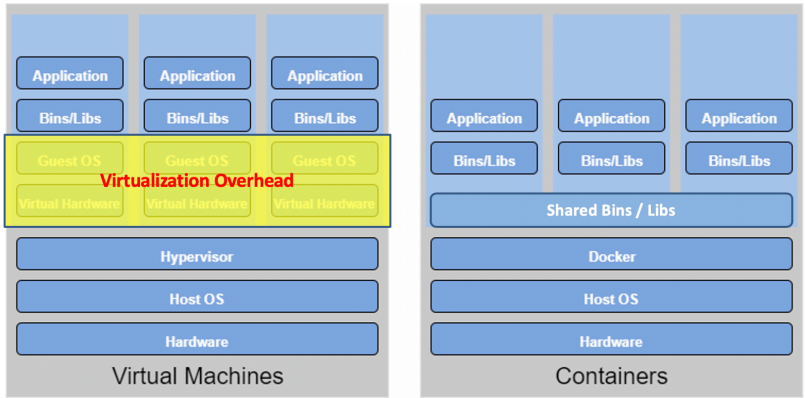
Update a task

```python
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods = ['PUT'])
@auth.login_required
def update_task(task_id):
    task = list(filter(lambda t: t['id'] == task_id, tasks))
    if len(task) == 0:
        abort(404)
    if not request.json:
        abort(400)
    if 'title' in request.json and type(request.json['title']) != str:
        abort(400)
    if 'description' in request.json and type(request.json['description']) is not str:
        abort(400)
    if 'done' in request.json and type(request.json['done']) is not bool:
        abort(400)
    task[0]['title'] = request.json.get('title', task[0]['title'])
    task[0]['description'] = request.json.get('description', task[0]['description'])
    task[0]['done'] = request.json.get('done', task[0]['done'])
    return jsonify( { 'task': make_public_task(task[0]) } )
```

11. Virtualization vs containerization

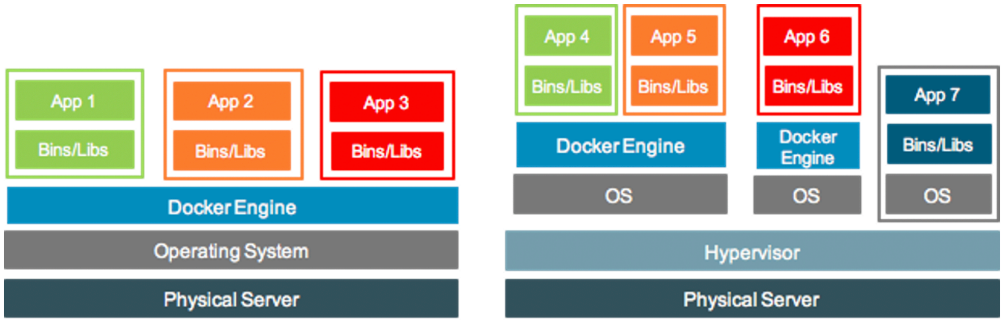
a. The many advantages of virtualization, such as application containment and horizontal scalability, come at a cost: resources. The guest OS and binaries can give rise to duplications between VMs wasting server processors, memory and disk space and limiting the number of VMs each server can support.

b. 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.



Parameter	Virtual Machines	Containers
Guest OS	Run on virtual HW, have their own OS kernels	Share same OS kernel
Communication	Through Ethernet devices	IPC mechanisms (pipes, sockets)
Security	Depends on the Hypervisor	Requires close scrutiny
Performance	Small overhead incurs when instructions are translated from guest to host OS	Near native performance
Isolation	File systems and libraries are not shared between guest and host OS	File systems can be shared, and libraries are
Startup time	Slow (minutes)	Fast (a few seconds)
Storage	Large size	Small size (most is re-use)

c. When deploying applications on the cloud, the base computation unit is a Virtual Machine. Usually docker containers are deployed on top of VMs. Docker container and VM can co-exist.



12. Container Orchestration

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

b. **Features**:

Networking

Scaling

Service discovery and load balancing

Health check and self-healing

Security

Rolling updates

c. **Goals**:

Simplify container management processes

Help to manage availability and scaling of containers

d. Kubernetes/ Docker SWARM / Docker Compose

13. Docker

a. Container: a process that behaves like an independent machine, it is a runtime instance of a docker image.

b. Image: a blueprint for a container.

c. Dockerfile: the recipe to create an image.

d. Registry: a hosted service containing repositories of images. E.g., the Docker Hub (https://hub.docker.com)

e. Repository: is a sets of Docker images.

f. Tag: a label applied to a Docker image in a repository.

g. Docker Compose: Compose is a tool for defining and running multi-containers Docker applications.

h. Docker SWARM: a standalone native clustering / orchestration tool for Docker.

14. Manage data on docker

a. By default, data inside a Docker container won't be persisted when a container is no longer exist.
b. You can copy data in and out of a container.
c. Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops.
d. Docker volumes (Managed by Docker, /var/lib/docker/volume/)
e. Bind mounts (Managed by user, any where on the file system)

15. Docker network

a. Use the host network stack, different port
b. Use the bridge, re-use the same port as they have different ip address, to be visible from the outside

16. Docker use

a. Log in

Login to a public Docker Registry, i.e. Docker Hub.
Syntax: docker login [OPTIONS] [SERVER]
E.g.: docker login --username=foo

b. Remove container

Remove a non-running container
Syntax: docker rm [OPTIONS] CONTAINER [CONTAINER ...]
E.g.: docker rm nginx

Remove a running container
docker rm -f nginx

c. Manage data in docker

- Create a volume
  Syntax: docker volume create [OPTIONS] [VOLUME]
  E.g.: docker volume create --name htdocs

- Start a container with a **volume** attached
  *E.g.: docker run --name nginx-volume -p 8080:80 *
  -v htdocs:/usr/share/nginx/html -d nginx

- Start a container with **bind mount** attached
  *E.g.: docker run --name nginx-bind -p 8081:80 *
  -v $(pwd)/htdocs:/usr/share/nginx/html -d nginx

d. Create image

- Create an image
  Syntax: docker build [OPTIONS] PATH
  E.g.: docker build -t demo2 .

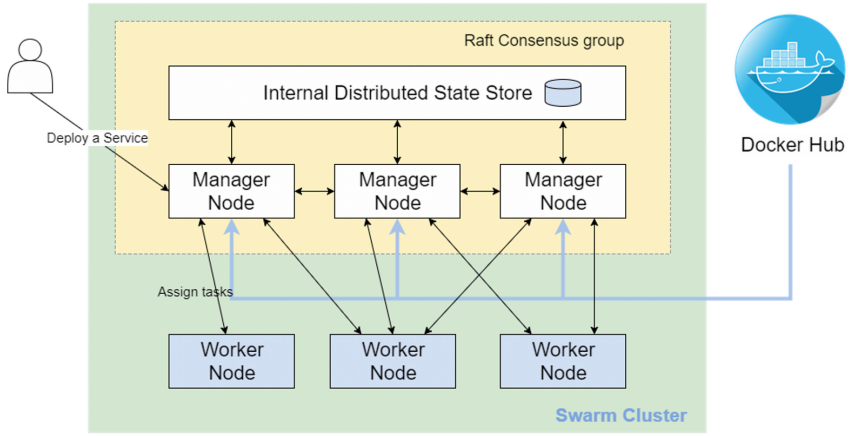
- Create a container from the image
  *docker run --name demo2 -e WELCOME_STRING="COMP90024" *
  -p 8081:80 -d demo2

e. Docker compose

- Start the containers
  Syntax: docker-compose up [OPTIONS]
  E.g.: docker-compose up -d
- Stop the containers
  Syntax: docker-compose stop [OPTIONS] [SERVICE …]
  E.g.: docker-compose stop
- Remove the containers
  Syntax: docker-compose down [OPTIONS]
  E.g.: docker-compose down

f. Docker swarm

1) **Raft consensus group** consists of internal distributed state store and all manager nodes.
2) **Internal Distributed State Store** is a built-in key-value store of Docker Swarm mode.
3) **Manager Node** conducts orchestration and management tasks. Docker Swarm mode allows multiple manager nodes in a cluster. However, only one of the manager nodes can be selected as a leader.
4) **Worker Node** receives and executes tasks directly from the manager node
5) **Node Availability**: In Docker Swarm mode, all nodes with ACTIVE availability can be assigned new tasks, even the manager node can assign itself new tasks (unless it is in DRAIN mode).
6) **Service** consists of one or more replica tasks which are specified by users when first creating the service.
7) **Task**: A task in Docker Swarm mode refers to the combination of a single docker container and commands of how it will be run.
8) Manager Node 负责执行编排和管理任务。 Docker Swarm 模式允许集群中有多个管理器节点。 但是，只能选择一个管理器节点作为领导者。
9) 工作节点直接从管理节点接收并执行任务
10) 节点可用性：在 Docker Swarm 模式下，可以为所有具有 ACTIVE 可用性的节点分配新任务，即使管理器节点也可以为其分配新任务（除非它处于 DRAIN 模式）。
11) 服务由一个或多个副本任务组成，这些任务由用户在首次创建服务时指定。
12) 任务：Docker Swarm 模式下的任务是指单个 Docker 容器及其运行方式命令的组合。



g. Docker machine – docker swarm

Docker Machine lets you install Docker Engine on virtual hosts and manage the hosts with docker-machine commands. You can use Docker Machine to create Docker hosts on a local computer, across a network, in a data centre, or on cloud providers.

Syntax: docker-machine create [OPTIONS] [ARG…]
docker-machine create manager
docker-machine create worker1
docker-machine create worker2

Syntax: docker swarm init [OPTIONS]
docker swarm init --advertise-addr 192.168.99.100

Add a manager

docker-machine ssh manager docker swarm join-token manager

Worker join swarm

Syntax: docker swarm join [OPTIONS]
docker-machine ssh worker1 docker docker swarm join --token SWMTKN-1-2uj8tpltiekyk1e5n4dcugcokjo8a2cfuvgby9s8ru8jwgg6q2-f3xw8888sfw0ayvgrcuv4efam 192.168.99.100:2377

h. Docker service

- Create a service
  Syntax: docker service create [OPTIONS] IMAGE [COMMAND]
  docker-machine ssh manager docker service create --replicas 3 -p 8083:80 --name nginx nginx:alpine

- List a service
  Syntax: docker service ls [OPTIONS]
  docker-machine ssh manager docker service ls

- Check a service
  Syntax: docker service ps [OPTIONS] SERVICE [SERVICE …]
  docker-machine ssh manager docker service ps nginx

- Scale up / down
  Syntax: docker service scale SERVICE=REPLICAS
  docker-machine ssh manager docker service scale nginx=6

  docker-machine ssh manager docker service scale nginx=2

- Rolling update
  Syntax: docker service update [OPTIONS] SERVICE
  docker-machine ssh manager docker service update --image alwynpan/comp90024:demo1 nginx

Week 7 Big Data and CouchDB

1. **Features of Big Data**
   a. **Volume**: yes, volume (Giga, Tera, Peta, …) is a criteria, but not the only one 数据量大
   b. **Velocity**: the frequency at which new data is being brought into the system and analytics performed 数据更新快
   c. **Variety**: the variability and complexity of data schema. The more complex the data schema(s) you have, the higher the probability of them changing along the way, adding more complexity.数据类型很多
   d. **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.数据越多数据的真实性就越低
2. **Why not relational db? NoSQL**
   a. 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.
   b. Therefore, it makes sense to use DBMSs that are built upon data models that are not relational (relational model: tables and relationships amongst tables).
   c. While there is nothing preventing SQL to be used in distributed environments, alternative query languages have been used for distributed databases, hence they are sometimes called NoSQL DBMSs

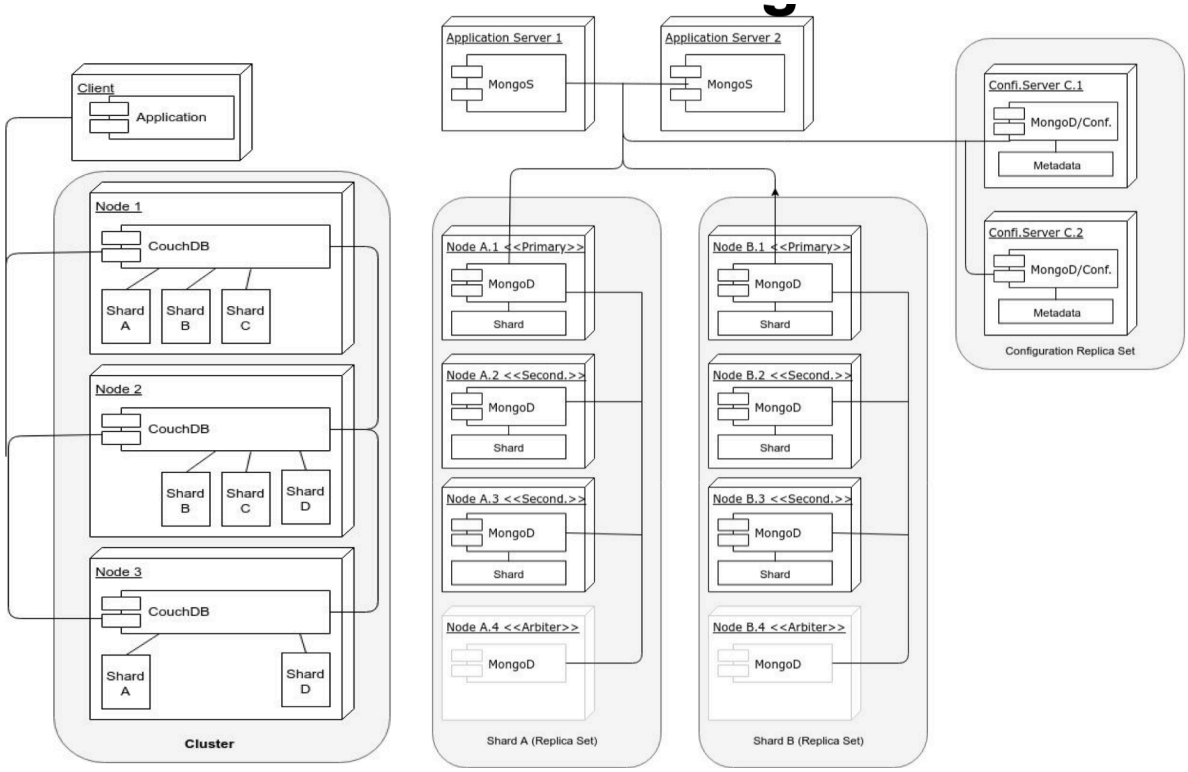
虽然关系型 DBMS 在确保一致性方面非常出色，但它们依赖于规范化的数据模型，而在大数据的世界里（想想 Veracity 和 Variety），这些数据模型已经不能被视为理所当然。因此，使用建立在非关系型数据模型（关系型模型：表和表之间的关系）上的 DBMS 是有意义的。在分布式环境中使用 SQL 是没有任何障碍的，但分布式数据库已经使用了其他的查询语言，因此有时被称为 NoSQL DBMS。

   d. While Relational DBMSs are extremely good for ensuring consistency and availability, the normalization that lies at the heart of a relational database model implies fine-grained data, which are less conducive to partition-tolerance than coarse-grained data.

虽然关系型 DBMS 在保证一致性和可用性方面有极好的表现，但关系型数据库模型的核心是规范化，意味着细粒度的数据，与粗粒度的数据相比，更不利于分区容忍度。

3. **Database for distributed environment 都是 Nosql**
   a. **A key-value** store is a DBMS that allows the retrieval of a chunk of data given a key: fast, but crude (e.g. Redis, PostgreSQL Hstore, Berkeley DB)
   b. **A BigTable** DBMS stores data in columns grouped into column families, with rows potentially containing different columns of the same family (e.g. Apache Cassandra, Apache Accumulo)
   c. **A Document-oriented** DBMS stores data as structured documents, usually expressed as XML or JSON (e.g. Apache CouchDB, MongoDB)
4. **Distributed database** are run over clusters which is a set of computers. The cluster need to distribute the computing workload to achieve the availability, and it will save multiple copy of data to achieve redundancy.
5. **CouchDB cluster architecture**

   Nodes (几个节点), shards (每个表切几片), replica (重复几次, CouchDB 里的 n)
   a. All nodes answer requests (read or write) at the same time
   b. Sharding (splitting of data across nodes) is done on every node
   c. When a node does not contain a document (say, a document of Shard A is requested to Node 2), the node requests it from another node (say, Node 1) and returns it to the client
   d. Nodes can be added/removed easily, and their shards are re-balanced automatically upon addition/deletion of nodes
   e. Primary node fails, a sub-node is chosed as primary node automatically



6. **MongoDB cluster architecture**

   MongoDB 和对比详见 lecture 7，估计不考
   a. Sharding (splitting of data) is done at the replica set level, hence it involves more than one cluster (a shard is on top of a replica set)
   b. ● 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
   c. (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)

7. **CouchDB vs MongoDB**
   a. MongoDB clusters are considerably more complex than CouchDB ones
   b. ● MongoDB clusters are less available, as - by default - only primary nodes can talk to clients for read operations, (and exclusively so for write operations)
   c. ● MongoDB software routers (MongoS) must be embedded in application servers, while any HTTP client can connect to CouchDB
   d. ● Losing two nodes out of three in the CouchDB architecture shown, means losing access to one quarter of data
8. **CAP- Consistency, Availability, Partition-Tolerance**
   a. Consistency: every client receiving an answer receives the same answer from all nodes in the cluster
   b. Availability: every client receives an answer from any node in the cluster
   c. Partition-tolerance: the cluster keeps on operating when one or more nodes cannot communicate with the rest of the cluster
9. **Consistency and Avalibility: Two phase commit (两阶段提交)**
   所有子节点都提交以后再完成事物，否则回滚。
   reduced availability, but enforced consistency 锁住的时候没有办法访问
   Good when clister is co-located(同地协作), less good when it is distributed
10. **Consistency and Partition-Tolerance: Paxos Algorithm**
    针对基于消息传递的分布式模型的系统，为了保证一致性和容错性
    每个节点是一个 proposer 或 accepter，也可以同时都是
    proposer 提出一个提案（proposal），带有 timestamp 或 id
    accepter 可以接受也可以拒绝，未经批准的叫提案（proposal），批准了的叫决议（value）
    若提案获得多数派（majority）的 acceptor 的接受，则该提案被批准（chosen）
    Without enough accept, the smaller part of a partition (the part that is not in the quorum) will not send responses, hence the availability is compromised
11. **Availability and Partition-tolerance: Multi-version Concurrency Control (MVCC) (多版本并发控制)**
    MVCC 意图解决读写锁造成的多个、长时间的读操作饿死写操作问题。每个事务读到的数据项都是一个历史快照（snapshot)并依赖于实现的隔离级别。写操作不覆盖已有数据项，而是创建一个新的版本，直至所在操作提交时才变为可见。快照孤立使得事物看到它启动时的数据状态（from wiki)
    MVCC 可以确保每个事务(T)通常不必“读等待”数据库对象(P)。这通过对象有多个版本，每个版本有创建时间戳与废止时间戳（WTS)做到的. (CouchDB 用的这个，当版本号冲突的时候会报错）
12. **Blockchain 区块链**
    Lecture 7 P18
    Using MVCC, peer-to-peer nodes, each containing a copy of the entire database,
13. **CAP for couchDB and mangoDB**
    While CouchDB uses MVCC, 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)
14. **Why document-orienterd for big data**
    While Relational DBMSs are extremely good for ensuring consistency and availability, the normalization that lies at the heart of a relational database model implies fine-grained data, which are less conducive to partition-tolerance than coarse-grained data.
    虽然关系 DBMS 对于确保一致性和可用性非常好，但是关系数据库模型的核心是规范化，它意味着细粒度的数据，与粗粒度的数据相比，它不利于分区容忍。
15. **Sharding**
    a. Sharding is the partitioning 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.
    b. Usually the number of shards is larger than the number of replicas, and the number of nodes is larger than the number of replicas (usually set to 3) 通常，分片的数量大于副本的数量而节点的数量大于副本的数量（通常设置为 3）
    c. The main advantage of a sharded database lies in the improvement of performance through the distribution of computing load across nodes.

16. **Replication and Sharding**
    a. Replication is the action of storing the same row (or document) on different nodes to make the database fault-tolerant.
    b. Replication and sharding can be combined with the objective of maximizing availability while maintaining a minimum level of data safety.
    c. n(replica) * q(shard) is the total number of shard files distributed in the different nodes of the cluster
17. **partition**
    a. A partition is a grouping of logically related rows in the same shard (for instance, all the tweets of the same user)
    分区是同一分片（例如，同一用户的所有推文）中与逻辑相关的行的分组
    b. ● Partitioning improves performance by restricting queries to a single shard
    分区通过将查询限制为单个分片来提高性能
    c. ● To be effective, partitions have to be relatively small (certainly smaller than a shard)
    要有效，分区必须相对较小（一定小于碎片
18. **Why couchDB in this class**
    Lecture 7 p28
    a. Is open-source, hence you can peruse the source code and see how things work
    b. ● It has MapReduce queries, hence you can understand how this programming paradigm works
    c. ● It is easy to setup a cluster
    d. ● It has sharding, replication, and partitions
    e. ● The HTTP API makes it easy to interact with it
19. **CouchDB features**
    a. Doc-oriented DMBS
    b. HTTP ReST API
    c. Web-based admin interface
    d. Web-ready
    e. Supports MapReduce (with JS)
    f. Supports Mango queries (JSON)
    g. Supports replication, sharding, clusters
20. CouchDB Views: Views are fast and store aggregated data, but are inflexible and use a lot of storage
    CouchDB 实际上用了一种 lazy 的方法来实现 view，当 MapReduce 的 view 被提交的时候 view 不会被立即生成，只有当第一次使用的时候才会计算生成，通常需要很长时间
21. **CouchDB**

Adding and deleting a database is done through a HTTP call:

```
curl -X PUT "http://localhost:5984/exampledb"
curl -X DELETE "http://localhost:5984/exampledb"
```

Listing all databases of an instance is even simpler

```
curl -X GET "http://localhost:5984/_all_dbs"
```

...every response's body is a JSON object:

```
["exampledb", "twitter", "instagram"]
```

• To insert a document:

```
curl -X POST "http://localhost:5984/exampledb" --header
  "Content-Type:application/json" --data '{"type": "account", "holder":
  "Alice", "initialbalance": 1000}'
```

Response: 201 (202 if less than the prescribed write operations were successfully performed)

```
{"ok":true,"id":"c43bcff2cdbb577d8ab2933cdc0011f8","rev":"1-b8a039a8143b474b
3601b389081a9eec"}
```

• To retrieve a document:

```
curl -X GET "http://localhost:5984/exampledb/c43bcff2cdbb577d8ab2933cdc0011f8"
```

Response: 200

```
{"_id":"c43bcff2cd577d8ab2933cdc0011f8","_rev":"1-b8a039a8143b474b3601b38908
1a9eec","type":"account","holder":"Alice","initialbalance":1000}
```

22. **_id & _rev**

- `_id`: is the ID of a single document which can be set during the document load; by default it is generated by CouchDB and guaranteed to be unique
- `_rev`: revision number of a document. It is guaranteed to be increasing per-document, i.e. every database instance will pick up the same revision as the current version of the document

CouchDB cluster 上即使有版本号更新，也依然会冲突，比如两个机器同时更新版本号

23. What Happens When a Conflict Happens on a Cluster of CouchDB Nodes?

a. When the revision number is not sent when documents are updated a 409 is raised in a single-node database, but something similar may happen on a clustered database even if the revision number is sent

b. ● When a cluster is partitioned and two nodes receive two different updates of the same document, two different revisions are added. However, only one of these is returned as the current revision (the "winning" revision is computed deterministically, hence guaranteed to be the same on any node of the cluster). At any rate, the "losing" revision is still stored in the database, and can be used to solve the conflict.

c. ● To help in the merging of conflicting revisions, CouchDB can return all the conflicts in a database

d. 如果在文档更新时未发送修订号，则在单节点数据库中引发 409，但是即使发送了修订号，在群集数据库上也可能会发生类似的情况

e. 对集群进行分区并且两个节点收到同一文档的两个不同更新时，将添加两个不同的修订版。 但是，仅返回其中一个作为当前修订版（"获胜"修订版是确定性计算的，因此可以保证在群集的任何节点上都相同）。 无论如何，"丢失"的修订版仍存储在数据库中，可用于解决冲突。

f. 为了帮助合并冲突的修订版，CouchDB 可以返回数据库中的所有冲突

24. couchDB 操作

a. CouchDB 的删除和彻底清除是不一样的

b. 大量的文档可以被同时操作（删除，更新等）使用 Bulk 来完成 _bulk_docs

c. Attachment，向文档添加一或多个附件，可以传递二进制数据或不需要解析的 json 文件

d. 可以通过设置删除之前存储的修订版本数量的限制来定制压缩，例如 一天中的压缩时间

25. CouchDB query

a. MapReduce Views: results of MapReduce processes that are written as B-tree indexes to disk and become part of the database

b. ● Mango Queries: queries expressed in JSON, following the MongoDB queries syntax (Mango queries can also use B-tree indexes to speed-up computations)

c. ● Full-text search: queries that can search form specific works or portions of words(via the Closueau plugin, running in a separate JVM instance)

26. CounchDB Views

Lecture7 p45

a. File: {"_id": "c43bcff2cdbb577d8ab2933cdc18f402", "name": "Chris White", "type": "transaction", "amount": 100}
Map: function(doc) {emit([doc.name], doc.amount);}
Reduce: function(keys, values, rereduce) {return sum(values);}

b. Result:

```
["Alice Smith"],      500
["Bob Dole"],         1000
["Charlie Black"],    1500
["Chris White"],      100
```

Note: results are ordered by key ascending

```
function(doc) {
 if (doc.type === "text" ) {
   var words= doc.contents.split(/[\s\.,]+/);
   for (var i in words)  {
     if (words[i].length > 1) {
       emit([words[i].substring(0,1),words[i]],
       1);
    …
["a", "ad"], 1
["a", "adipisicing"], 1
["a", "aliqua"], 1
```

c.

d. View use

key	限定结果中只包含键为该参数值的记录。
startkey	限定结果中只包含键大于或等于该参数值的记录。
endkey	限定结果中只包含键小于或等于该参数值的记录。
limit	限定结果中包含的记录的数目。
descending	指定结果中记录是否按照降序排列。
skip	指定结果中需要跳过的记录数目。
group	指定是否对键进行分组。
reduce	指定reduce=false可以只返回 Map 方法的运行结果。

e. Group level
group_level 1 就是只按照 key 的第一个字段 group，2 是按照前两个字段 group

f. List and Show Functions 以 json 以外的形式展示

g. Update and Validate Functions 更新和检测权限

27. CouchDB replication

a. curl -H 'Content-Type: application/json' -X POST http://localhost:5984/_replicate -d ' {"source": "http://myserver:5984/foo", "target": "bar", "create_target": true, "continuous": true}'

b. Note the **continuous** attribute is set to true; if this were false, the database instance would be replicated only once

c. To cancel a replication, just issue the same exact JSON, but with an additional **cancel** attribute set to true

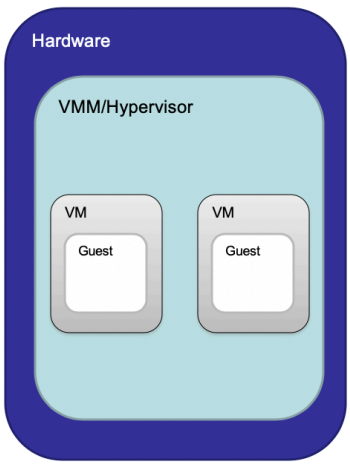
28. CouchDB Partition

curl -X PUT "http://localhost:5984/testpart?partitioned=true" --user '<username>:<password>'

29. CouchDB cluster

Lecture 8.1 Virtualisation

1. The structure of virtual machine



a. **Virtual Machine Monitor/Hypervisor**: The virtualisation 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

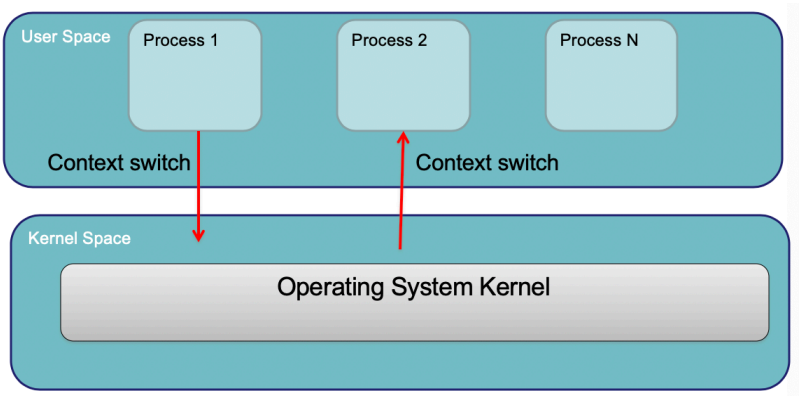
Appears as though in control of system resources

虚拟化层，在硬件和 VM 之间。VM 的环境应该和实际的物理机一样，只是性能上有所减弱。

b. **Virtual Machine**: A representation of a real machine using hardware/software that can host a guest operating system

c. **Guest Operating System**: An operating system that runs in a virtual machine environment that would otherwise run directly on a separate physical system

2. Kernel mode vs user mode



a. Processes run in user mode (low privileged)

b. OS Kernel runs in kernel mode (privileged)

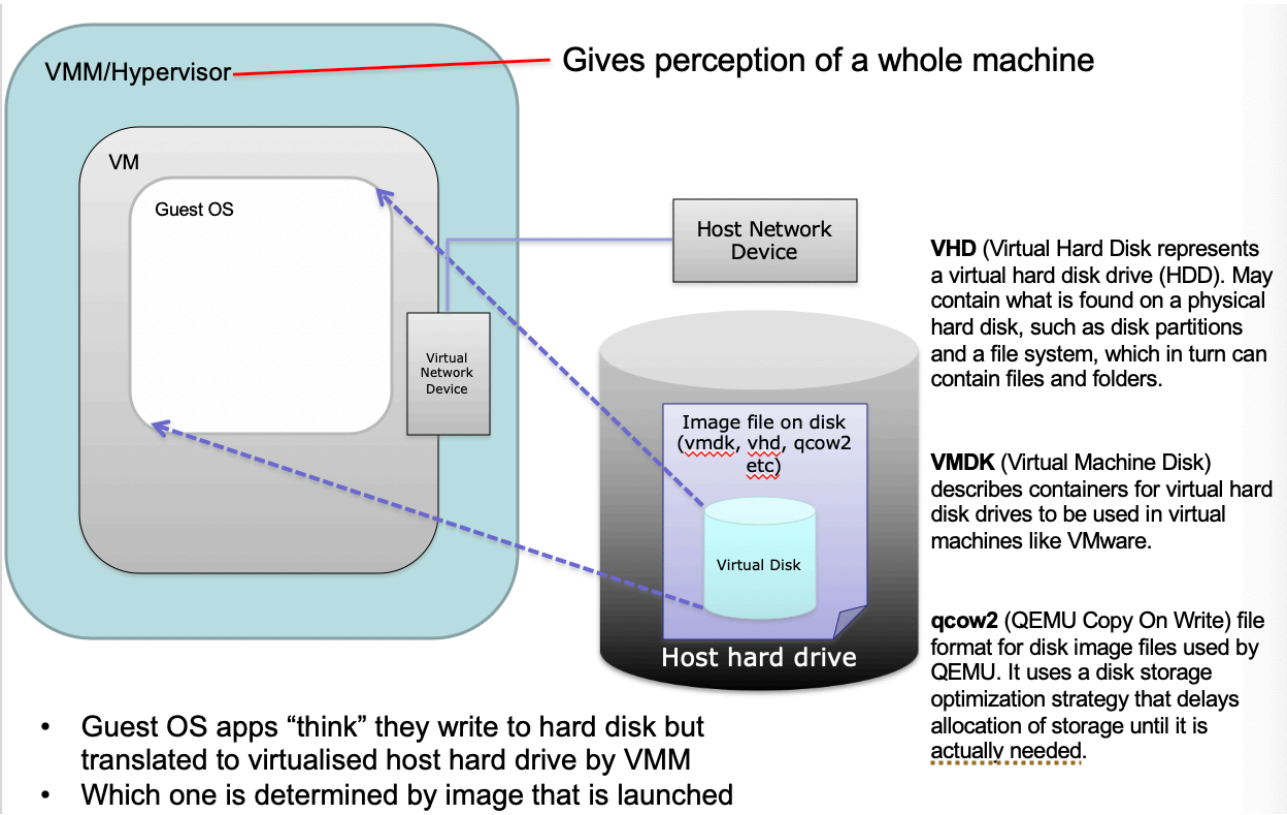
c. 大部分的 application 都是在 user space 上运行的，而某些有底层需求的 call 需要在 kernel space 进行，而 VMM/hypervisor 需要发现这些 call 并且处理这些请求。

d. 每一个 VM 有一个 user space，开多个 VM 是，可能需要发现并处理多个 VM 的 call

e. OS typically virtualises memory, CPU, disk etc giving appearance of complete access to CPU/memory/disk to application processes

3. What happens in a VM

VHD, VMDK, qcow2



4. Motivation of VM 为什么使用虚拟机

a. Server Consolidation 服务器整合

Increased utilization 提高利用率 高性能，单个使用的时候没法完整利用

Reduced energy consumption

e.g. one physical server and 8 virtual server doing 8 projects

b. Personal virtual machines can be created on demand

No hardware purchase needed

Public cloud computing

c. Security/Isolation

Share a single machine with multiple users

d. Hardware independence

Relocate to different hardware 硬件迁移，搬家

5. Instructions classification

a. Privileged Instructions: 特权指示，只能在 kernel 中执行，不能在 user space 执行

instructions that trap if the processor is in user mode and do not trap in kernel mode

b. Sensitive Instructions: 敏感指示，在 kernel mode 和 user 中执行的是不一样的

instructions whose behaviour depends on the mode or configuration of the hardware

Different behaviours depending on whether in user or kernel mode

e.g. POPF interrupt (for interrupt flag handling)

从 user 发送到 kernel 的 sensitive 指示是 priviledged instruction

c. Innocuous Instructions: instructions that are neither privileged nor sensitive

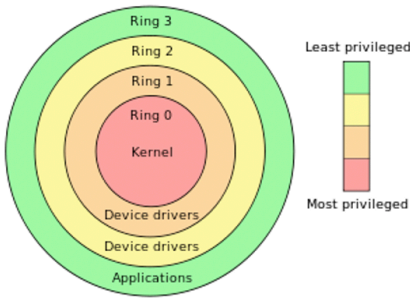
Read data, add numbers etc

6. Theorem- Popek and Goldberg virtualization requirements

- i.e. have to be trappable

Example of 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 can't turn on web cam or recording device etc



a. 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

b. Properties

Fidelity (等价性): 一个运行于 VMM 下的程序，其行为应与直接运行于等价物理机上的同程序的行为完全一致.

Performance (效率性): 机器指令中经常使用的那一部分应在没有 VMM 干预下执行

Safety (安全性): VMM 对虚拟资源进行完全控制

7. Typical Virtualisation Strategy

a. De-privileging

处理虚拟机发出的 privileged instructions

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 GuestOS 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

b. Primary/shadow structures

维护 shadow page table，把 primary page table 给 guest OS 看

VMM maintains "shadow" copies of critical structures whose "primary" versions are manipulated by the GuestOS, e.g. memory page tables

Primary copies needed to insure correct versions are visible to GuestOS

c. 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

Someones app/code doesn't crash the server you are using!!!

8. Aspects of VMMs

a. Full virtualization

allow an unmodified guest OS to run in isolation by simulating full hardware

Lets say if you want to use a full windows or ubuntu, this is the robust way, like VMWare. Every single call goes down to the server will be trapped and managed. One of the management measure is call Binary Translation

Advs: Guest is unaware it is executing within a VM, Guest OS need not be modified, no hardware or OS assistance required

Disadvs: But can be less efficient

b. Para-virtualization

VMM/Hypervisor exposes special interface to guest OS for better performance. Requires a modified Guest OS

Some interface would be open, but not every os system could be in use, and for those calls that won't change the system state could be directly goes to the hardware from the user application

不改变系统状态的 call 可以直接进入 hardware

Advs: Lower virtualisation overheads, so better performance

Disadvs: Need to modify guest OS, less portable (便携性差), less compatibility (兼容性差)

c. Hardware-assisted virtualisation

Hardware provides architectural support for running a VMM

Requires that all sensitive instructions trappable

New Ring -1 supported: Page tables, virtual memory mgt, direct memory access for high speed reads etc

Such things could be done in real hardware

Advs: Good performance, eaisier to implement

Disadvs: Needs hardware support

d. Binary Translation

先 trap 后替换 sensitive call

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

Advs: Guest OS need not to be modified, no HW or OS assistance required

Disadvs: Overheads, complicated, need to replace instructions "on-the-fly", library support to help this

e. Bare Metal Hypervisor

VMMs runs directly on actual hardware. 直接部署在主机硬件上，以管理硬件和 guest machine。

f. Host Virtualisation

VMMs runs on top of another operating system. 作为软件层部署在主机操作系统上，现在常用的 VMware Player 和 VirtualBox 就是这种类型

g. OS level Virtualisation

h. Lightweight VMs

Creates mini containers, Docker

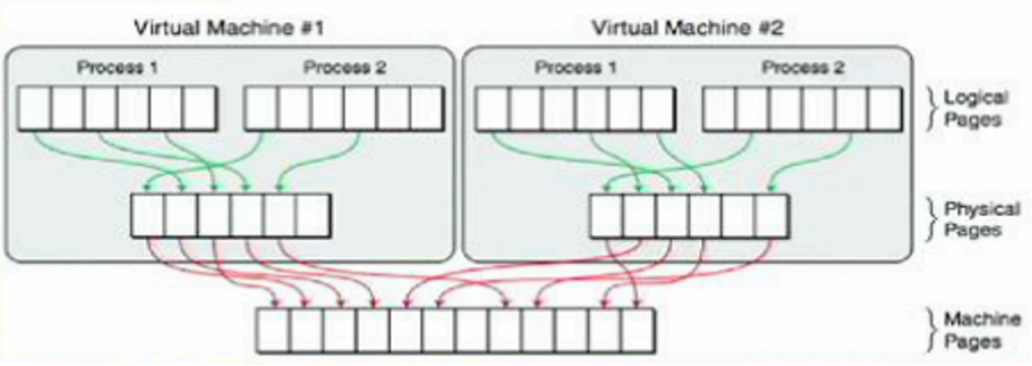
Advs: Lightweight, many more VMs on same hardware, less deployment time in seconds.

Disadvs: Can only run apps designed for the same OS, cannot host a different guest OS, only use native file systems

9. Memory Virtualisation

Conventionally(照常地) page tables store the logical page number -> physical page numebr mappings

Shadow page tables (见前边)



10. **Shadow page table**

    Hardware performs guest -> physical and physical -> machine translation

11. **Live migration of VM 将虚拟机转移位置**

    a. Stage 0: Pre-Migration - Active VM on Host A

    b. Stage 1: Reservation - Initializa a container on the host B

    c. Stage 2: Iterative Pre-copy - Enable shadow paging, copy dirty pages in successive rounds

    d. Stage 3: Stop and copy - Suspend VM on host A, generate ARP to redirect traffic to host B, sync all remaining VM state to host B

    e. Stage 4: Commitment - VM state on host A is released

    f. Stage 5: Activation - VM starts on Host B, connects to local devices, resumes normal operation

    没有被调用的部分慢慢转移，检测到被调用的就先转移别的。

Lecture 8.2 OpenStack & Comparing and Contrasting AWS with NeCTAR Cloud

1. **NeCTAR vs AWS**

   UniMelb/NeCTAR research cloud is open source cloud platform: open stack

   AWS is mainstream cloud platform

2. **OpenStack**

   a. Offers free and open-source software platform for cloud computing for (mostly) IaaS

   b. Consists of interrelated components (services) that control / support compute, storage, and networking resources

   c. Often used through web-based dashboards, through command-line tools, or programmatically through ReSTful APIs

3. **OpenStack components**

   a. Compute Service (code-named **Nova**)

      1) Manages the lifecycle of compute instances in an OpenStack environment

      2) Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand

      3) Virtualisation agnostic 不可知 could use all kinds of virtualization methods

      4) API

         Nova-api - accepts/responds to end user API calls; supports openStack Compute & EC2 & admin APIs

      5) Compute Core

         Nova-compute - Daemon that creates/terminates VMs through hypervisor APIs

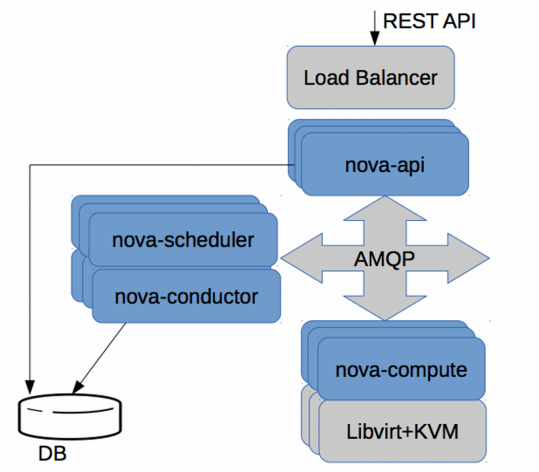
         Nova-scheduler - schedules VM instance requests from queue and determines which server host to run

         Nova-conductor - Mediates interactions between compute services and other components, e.g. image database

      6) Networking

         Nova-network - Accepts network tasks from queue and manipulates network, e.g. changing IPtable rules

7) Build a instance



b. Image Service (code-named Glance)
c. Block Storage Service (code named Cinder)
d. Object Storage Service (code-named Swift)
e. Security Management (code-named **Keystone**)
   a) Provides an authentication and authorization* service for OpenStack services (track users / permission)
   b) But keystone do little authorization, mostly identify the user
   c) Provides a catalog of endpoints for all OpenStack services
f. Orchestration Service (code-named **Heat**)
   1) Template-driven service to manage lifecycle of applications deployed on OpenStack. 模板驱动的服务，以管理部署在 Openstack 上的应用程序的生命周期。
   2) Stack: Another name for the template and procedure behind creating infrastructure and the required resources from the template file. 指从模板文件中创建基础设施和所需资源背后的模板和程序。
   3) Can be integrated with automation tools such as Ansible.
g. Network Service (code-named Neutron) could be used to set security group
h. Container Service (code-named Zun)
i. Database service (code-named Trove)
j. Dashboard service (code-named Horizon)
k. Search service (code-named Searchlight)
l. Data processing service(sahara) :specify Hadoop version, cluster topology, machine learning staff

4. The way to build a instance
   a. Use the identity service to check the identity
   b. Log into the dashboard
   c. Use the image service, choose an image like ubuntu
   d. Use the storage service to choose the volume size
   e. Use the network service to choose the security group
5. Deal with the flow

   Lets say if 300 wants to build their instance at the same time, the open stack use a queuing system.

   Typically asynchronous queuing systems used (AMQP)
   a. Whoever request a compute resources
   b. They are given some time
   c. The request comes in
   d. The request deployed
   e. Then take the next request
6. Heat details:
   a. heat_template_version: allows to specify which version of Heat, the template was written for (optional) 允许指定

模板是为哪个版本的 Heat 编写的

b. Description: describes the intent of the template to a human audience (optional) 向人类受众描述模板的意图
c. Parameters: the arguments that the user might be required to provide (optional) 用户可能需要提供的参数
d. Resources: the specifications of resources that are to be created (mandatory) 要创建的资源规格(**必填**)
e. Outputs: any expected values that are to be returned once the template has been processed (optional) 模板处理后要返回的任何预期值

7. Creating stack in NeCTAR
   a. Create the template file according to your requirements
   b. Provide environment details (name of key file, image id, etc)
   c. Select a name for your stack and confirm the parameters
   d. Make sure rollback checkbox is marked, so if anything goes wrong, all partially created resources get dumped too
8. Use the stack could easily delete all the settings and all the staffs created (volumes or security groups, etc.) all at once.

Lecture 8.3 serverless, Function as a service (FaaS)

1. **What is serverless**
   a. FaaS is also know as Serverless computing (more catchy, but less precise)
   b. The idea behind Serverless/FaaS is to develop software applications without bothering with the infrastructure (especially scaling-up and down as load increases or decreases)
   c. Therefore, it is more Server-unseen than Server-less
   d. A FaaS service allows functions to be added, removed, updated, executed, and auto-scaled
   e. FaaS is, in a way, an extreme form of microservice architecture
2. **Why FaaS?**
   a. Simpler deployment (the service provider takes care of the infrastructure)
   b. ● Reduced computing costs (only the time during which functions are executed is billed)
   c. ● Reduced application complexity due to loosely-coupled architecture
   每个 function 用什么语言写都无所谓，很方便。
3. FaaS application
   a. Functions are triggered by events
   b. ● Functions can call each other
   c. ● Functions and events can be combined to build software applications
   d. Combining event-driven scenarios and functions resembles how User Interface software is built: user actions trigger the execution of pieces of code
4. Side-effect Free Functions
   a. A function that does not modify the state of the system is said to be side-effect free (for instance, a function that takes an image and returns a thumbnail of that image)
   b. ● A function that changes the system somehow is not side-effect free (for instance, a function that writes to the file system the thumbnail of an image)
   c. ● Side-effect free functions can be run in parallel, and are guaranteed to return the same output given the same input
   d. ● Side-effects, however, are almost inevitable(必然发生的) in a relatively complex system. Therefore consideration must be given on how to make functions with side effects run in parallel, as typically required in FaaS environments.
5. **Stateful/Stateless Functions**
   a. A subset of functions with side-effects is composed of stateful functions
   b. ● A stateful function is one whose output changes in relation to internally stored information (hence its input cannot entirely predict its output), e.g. a function that adds items to a “shopping cart” and retains that information internally
   c. ● Conversely, a stateless function is one that does not store information internally, e.g. adding an item to a “shopping cart” stored in a DBMS service and not internally would make the function above stateless, but not side-effect free.

有副作用的函数子集是由有状态的函数组成的

有状态的函数是指其输出根据内部存储的信息而变化的函数（因此其输入不能完全预测其输出），例如，一个在"购物车 "中添加项目并在内部保留该信息的函数。
相反，无状态函数是指不在内部存储信息的函数，例如，将一个项目添加到存储在 DBMS 服务中的 "购物车 " 中，而不是在内部添加，就会使上述函数成为无状态函数，但并非无副作用。这一点在 FaaS 服务中很重要，因为同一个函数有多个实例，不能保证同一个用户会调用同一个函数实例两次。

6. **Synchronous/Asynchronous Functions**
   a. By default functions in FaaS are synchronous, hence they return their result immediately (or almost so)
   b. ● However, 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
   c. ● Asynchronous functions return a code that informs the client that the execution has started, and then trigger an event when the execution completes

默认情况下，FaaS 中的函数是同步的，因此它们会立即返回结果（或几乎如此
* 但是，可能有一些函数需要更长的时间来返回结果，因此它们会产生超时，并在此过程中锁定与客户端的连接，因此最好将它们转化为异步函数。
异步函数返回一个代码，通知客户端执行已经开始，然后在执行完成时触发一个事件。

7. **OpenFaaS**
   a. **OpenFaaS is an open-source framework that uses Docker containers to deliver FaaS functionality**
   b. **● Every function in OpenFaaS is a Docker container, ensuring loose coupling between functions (functions can be written in different languages and mixed freely)**
      每一个都开了一个新的 container
   c. ● Functions are passed a request as an object in the language of choice and return a response as an object
   d. ● OpenFaaS can use either Docker Swarm or Kubernetes to manage cluster of nodes on which functions run
8. 通过 post 请求来调用方法
9. **OpenFaaS can add more docker containers when a function is called more often and reduce the number of containers when the function is called less often.**
10. Open FaaS is a vary fast and rapid data processing
11. Build and deployment of OpenFaaS on Docker
    a. Install and start OpenFaaS

```
git clone https://github.com/openfaas/faas
docker swarm init
cd faas
./deploy_stack.sh
```

    b. Check all Docker services have started
       6 services should be seen

```
docker service ls
```

    c. Command-Line Interface installation:
       The username and password should be generated through this step
       Linux:

```
curl -sSL https://cli.openfaas.com | sudo sh
```

       MacOS:

```
brew install faas-cli # or curl -sSL https://cli.openfaas.com | sudo sh
```

    d. Login to the cluster

```
faas-cli login -u admin -p <password>
```

e. Use brower to see the UI

f. Create function from templete

```
faas-cli template pull # Grab all the tamplates in different languages
faas-cli new wcmp --lang=node # Creates a new function in Node.js
cd wcmp # Goes into the newly created function (only package.json and handler.js there)
```

g. Build and Deployment of wcmp function

```
faas-cli build --image=wcmpimage --lang=node --handler=./ --name=wcmp # Build image
faas-cli deploy --image=wcmpimage --name=wcmp # Deploy function to local OpenFaaS instance
curl -XPOST "http://0.0.0.0:8080/function/wcmp" --data 'italy' # Call the function
```

12. Change open saas rules

In the yaml file you can change the cpu load, number of the function processing per seconds

In the other shell window, the number of replicas can be observed to raise, and then fall.+ By default the scaling-up (and down) function counts the number of successful (HTTP stastus of 200) function invocations per function replica in 10 seconds and scale when there are more than 5 of them. (Every time the number of new function replicas is 20% of the maximum allowed).

在另一个 shell 窗口中，可以观察到复制数的提高，然后下降.+默认情况下，scaling-up(和 down)函数在 10 秒内计算每个函数复制的成功(HTTP stastus 为 200)函数调用次数，当有超过 5 个函数时，就会进行缩放。每当新的函数副本数量为最大允许量的 20%)。

```
groups:
- name: prometheus/alert.rules
  rules:
  - alert: service_down
    expr: up == 0
  - alert: APIHighInvocationRate
    expr: sum(rate(gateway_function_invocation_total{code="200"}[10s])) BY (function_name) > 5
    for: 5s
    labels:
      service: gateway
      severity: major
    annotations:
      description: High invocation total on {{ $labels.function_name }}
      summary: High invocation total on {{ $labels.function_name }}
```

Lecture 9 Big Data Analytics, Hadoop, Spark

1. Challenge of big data analysis

a. Reading and writing distributed datasets

b. ● Preserving data in the presence of failing data nodes

c. ● Supporting the execution of MapReduce tasks

d. ● Being fault-tolerant (a few failing compute nodes may slow down the processing, but not stop it)

e. ● Coordinating the execution of tasks across a cluster 协调

2. Apache hadoop

a. when it comes to big data, the majority of applications are built on top of an open-source framework: Apache Hadoop

b. Apache Hadoop started as a way to distribute files over a cluster and execute MapReduce tasks, but many tools have now been built on that foundation to add further functionality. Apache Hadoop 最初是作为一种在集群上分发文件和执行 MapReduce 任务的方式，但现在已经有许多工具在此基础上增加了更多的功能。

3. Hadoop Distributed File System (HDFS)

储存的单位块比较大，meta data 储存的很少，

a. High fault tolerance, larger blocks

b. Reduced need for memory to store information about where the blocks are (metadata)

c. ● More efficient use of the network (with a large block, a reduced number network connections needs to be kept

open)

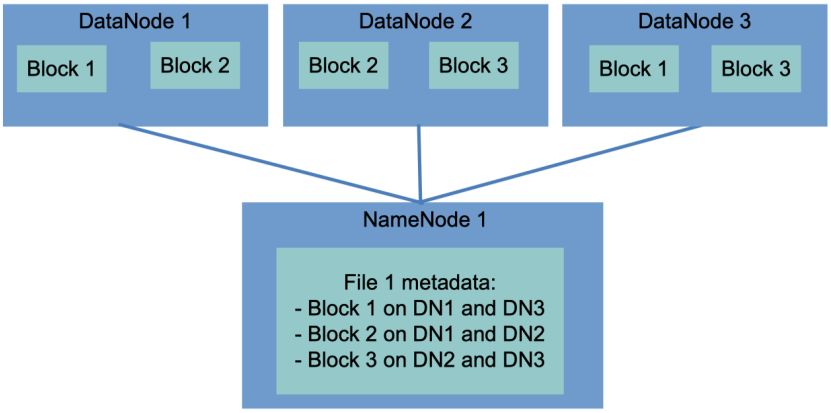
d. ● Reduced need for seek operations on big files

e. ● Efficient when most data of a block have to be processed

4. **HDFS Architecture**

Data nodes and name nodes

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



5. **The Hadoop Resource Manager (YARN)**

就是 map reduce 里面的 manager

a. The other main component of Hadoop is the MapReduce task manager, YARN (Yet Another Resource Negotiator)

b. ● YARN deals with executing MapReduce jobs on a cluster. It is composed of a central Resource Manager (on the master) and many Node Managers that reside on slave machines.

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

6. **Why spark?**

Hadoop 在大数据集上统一执行简单的小任务， spark 执行精细的需要更多管理的任务（机器学习）

a. While Hadoop MapReduce works well, it is geared towards performing relatively simple jobs on large datasets.

b. ● 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.

c. ● Apache Spark was designed to reduce the latency inherent in the Hadoop approach for the execution of MapReduce jobs.

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

7. **Spark programing**

a. Get data in and out

- The simplest way to get data into Spark is reading them from a CSV file from the local file system:

```
csv = sc.textFile("file.csv")
```

- With a small change, Spark can be made to read from HDFS file systems (or Amazon S3):

```
csv = sc.textFile("hdfs://file.csv")
csv= sc.textFile("s3://myBucket/myFile.csv")
```

Can use java library to parse json file, or load serialised Java objects

- SQL queries can also be used to extract data:

```
df = sqlContext.sql("SELECT * FROM table")
```

Relational and nosql database could also be a data source

b. Spark shell

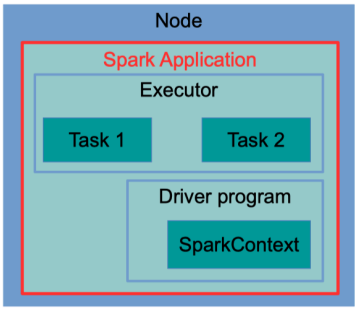
The Spark Shell allows to send commands to the cluster interactively in either Scala or Python.

While the shell can be extremely useful, it prevents Spark from deploying all of its optimizations, leading to poor performance.

8. **Spark application**
   a. Job: the data processing that has to be performed on a dataset ○
   b. Task: a single operation on a dataset ○
   c. Executors: the processes in which tasks are executed ○
   d. Cluster Manager: the process assigning tasks to executors ○
   e. Driver program: the main logic of the application ○
   f. Spark application: Driver program + Executors ○
   g. Spark Context: the general configuration of the job

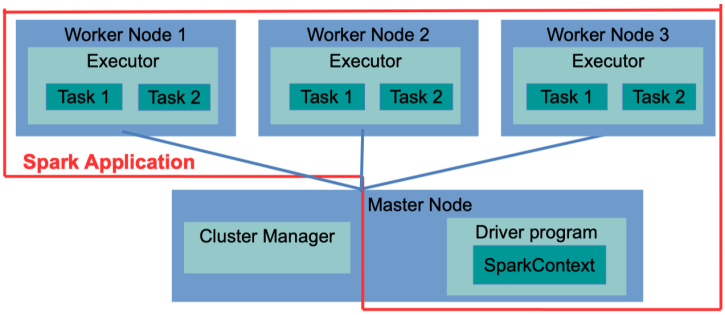
9. **Spark Runtime Architecture**
   a. Local mode

   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. (Local mode is good when developing/debugging)



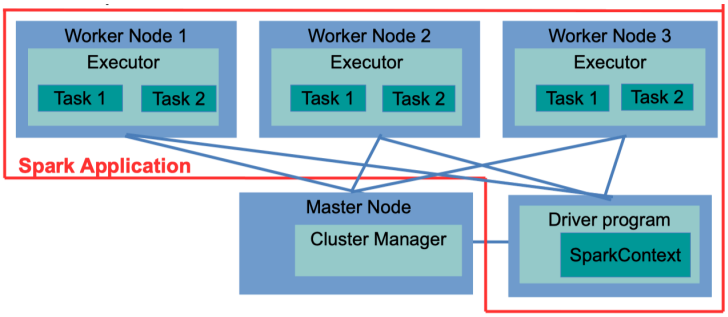
   b. Cluster mode

   In cluster mode, every component, including the driver program, is executed on the cluster; hence, upon launching, the job can run autonomously. This is the common way of running non-interactive Spark job



   c. 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,**



10. **Spark contents**

设置部署模式，配置，连接的集群。设置内存大小和执行器数量

   a. The deployment mode is set in the Spark Context, which is also used to set the configuration of a Spark application,

including the cluster it connects to in cluster mode.

b. ● For instance, this hard-coded Spark Context directs the execution to run locally, using 2 threads (usually, it is set to the number of cores):
sc = new SparkContext(new SparkConf().setMaster("local[2]"));

c. ● This other hard-coded line directs the execution to a remote cluster:
sc = new SparkContext(new SparkConf() .setMaster("spark://192.168.1.12:6066"));

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

11. Use shell or script to submit a job to spark, the script should contain or the information, such as the master, the deploy mode. The application jar should be available from all the nodes in the cluster.
Uber-JARs can be assembled by Maven with the Shade plugin.

12. Resilient Distributed Datasets (RDDs)

a. Resilient Distributed Datasets (RDDs) are the way data are stored in Spark during computation, and understanding them is crucial to writing programs in Spark:

b. ● Resilient (data are stored redundantly, hence a failing node would not affect their integrity)

c. ● Distributed (data are split into chunks, and these chunks are sent to different nodes)

d. ● Dataset (a dataset is just a collection of objects, hence very generic)

13. Properties of RDDs

a. RDDs are immutable, once defined, they cannot be changed (this greatly simplifies parallel computations on them, and is consistent with the functional programming paradigm)

b. ● RDDs are transient, they are meant to be used only once, then discarded (but they can be cached, if it improves performance)

c. ● 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)

14. rdd action vs rdd transformation

a. Transformation:转移位置，分组，合并

b. Action：得到一个输出

c. 详细见 lecture 9 p40

d. since RDDs are immutable, the result of the final transformation is cached, not the input RDD. In other words, when this statement is executed

15. Spark Jobs, Tasks, and Stages

a. A Job is the overall processing that Spark is directed to perform by a driver program

b. ● A **Task** is a single transformation operating on a single partition of data on a single node

c. ● A **Stage** is a set of tasks operating on a single partition

d. ● A **Job** is composed of more than one stage when data are to be transferred across nodes (shuffling)

e. ● The fewer the number of stages, the faster the computation (shuffling data across the cluster is slow)

16. 用 docker 集群做 spark

1. Build sparker 的 docker 镜像
2. Docker-compose 设置集群
3. 指定 spark 的 master
4. 执行 task（word count）
5. 执行过程中可以在 browser 查看执行状态，有几个 worker，执行的什么任务，是否完成
6. 可以用 docker-compose stop, docker-compose start 结束和重启集群

Lecture 10 Security and Clouds

1. **Digit security**

   Resources – access control – privilege(u don't wanna brute force ways to be successful) – user communitites

2. **Challenge of security**

   a. Grids and Clouds (IaaS) allow users to compile codes that do stuff on physical/virtual machines

      It is hard to stop someone do some bad stuff.

      Some of the application needs security technology

   b. Should try to develop generic security solutions

      To avoid each area build their own which is incompatible.

   c. Clouds allow scenarios that stretch inter-organisational security, but what if new resources added or users have changed. Or the organization decide to change their policy.

3. **Technical challenge of security**

   a. Authentication 认证

   Means you prove your identity. 并不检查用户被允许做什么，只是让我们知道（可以检查！）他们是谁

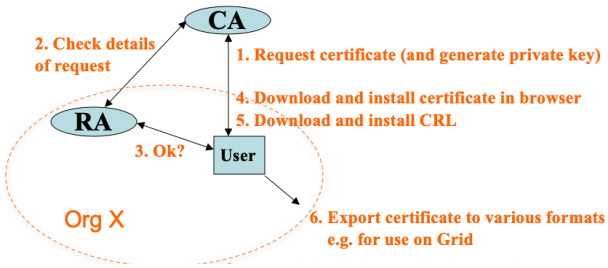
   Public Key Infrastructures (PKI) underpins MANY systems

   Public key & private key

   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

   Central component of PKI is **Certification Authority (CA)** they will check your identity

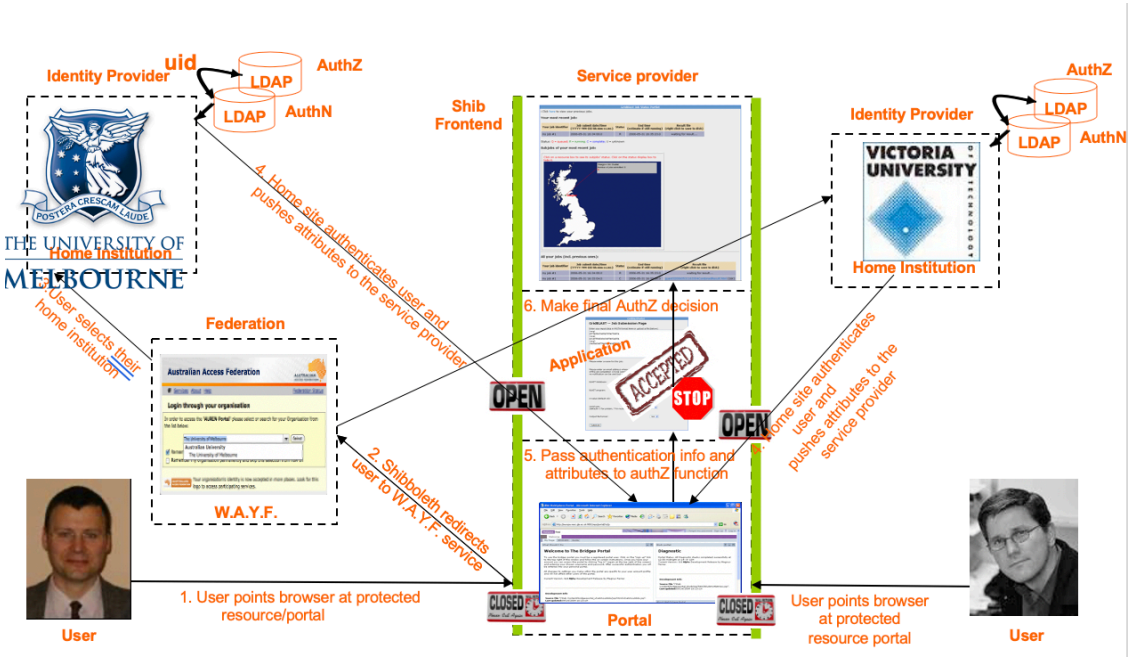


   b. Authorization 授权

   验证了身份，但是还要检查你有没有权限来做什么事

   RBAC 一个常用的理念：roles applicable to specific collaboration，actions allowed/not allowed for VO members，resources comprising VO infrastructure (computers)

   身份 A 是否对资源 B 做处理 C，管理政策对用户透明



   c. Audit/accounting 审计/会计

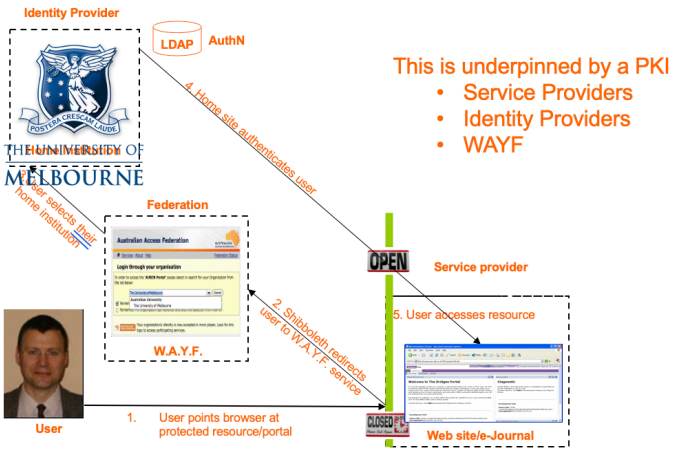
d. Confidentiality 保密性
e. Privacy 隐私权
f. Fabric management 织物管理
g. Trust

4. PKI and Cloud

IaaS – key pair!

Prove your identity and get into the cloud

Need for single sign on



不管去哪个网站，都由你自己的机构来查验你的身份，机构之间相互信任彼此的查验结果。

当用户登录时，就可以获取所有系统的访问权限，不用对每个单一系统都逐一登录

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 any of several related systems.

5. Other cloud security challenges

a. Single sign-on
   The Grid model (and Shib model!) needed
   Currently not solved for Cloud-based IaaS
b. Auditing 审计
   记录，入侵检测，有很好的验证，但在云环境不成熟
c. Deletion (and encryption!!!) 确保每一个都删除了，成本很高，实际上还不如添加一个新的
   1) Data deletion with no direct hard disk
      Many tools and utilities don't work!
   2) Scale of data
d. Licensing
e. Workflow
   defining, enforcing, sharing, enacting, security-oriented workflows

Sample exam

A.) Describe some of the current challenges associated with large-scale distributed systems. [4]
B.) Cloud computing solves some of these issues but not all. Explain. [4]
C.) What are *availability zones* in NeCTAR and what restrictions do they impose on NeCTAR Cloud-based application developers? [2]

A. 1. Data heterogeneity: 数据形式不一样，整体处理需要预处理，数据不规范，花费
   2. hardware issues: 硬件差距

3. scalability：越大，网络延迟，网络失败，
4. fault tolerance:
5. **result in software stacks:**

   Many diverse faults can happen with distributed systems, e.g. server failures or partial failures, network outages, overloading of components etc. There is no simple solution to this that has been widely adopted/accepted. Each system tends to develop its own technical solution, e.g. using queuing or having back-ups/failovers of system for failures. **This can result in complex software stacks and recipes that have to be cooked to address specific needs/demands.**

B. 1. Scalability, pay by need,
   2. 部署方便，自动部署
   3. 有工具 load balancing

C. Volume 和 instance 必须在同一个 avai zone， data center 是指地址

A.) In the context of distributed databases, explain the concepts of:

1. *Consistency* [1]
2. *Availability* [1]

B.) Give an example of a database technology that supports *Availability* in the presence of a (network) partition. [1]

C.) In the context of CouchDB clusters what is the meaning of:

1. *Replica number* [1]
2. *Number of shards* [1]
3. *Read quorum* [1]
4. *Write quorum* [1]

D.) Describe the three different Apache SPARK runtime modes:

1. *Local* [1]
2. *Cluster* [1]
3. *Client* [1]

B 问题，MVCC 解决 availability，

C 问题 Read quorum 读数据的时候多看几个 node 中储存的数据是否一致，都是这个再返回这个值

A.) Explain Amdahl's law and discuss the challenges of its practical implementation. [2]

B.) The actual performance as experienced by users of shared-access HPC facilities such as the Edward cluster at the University of Melbourne can vary – where here performance can be considered as the throughput of jobs, i.e. from the time of first job submission to the time of last job completion. Explain why this can happen. [2]

C.) Explain how the Edward cluster has been set up to minimize this. [2]

D.) Explain what users can do to optimize their throughput (use) of the Edward cluster. [2]

E.) Describe some of the challenges with application benchmarking on HPC facilities. [2]

B 问题 分区 cloud physical gpgpu 各干各的

C 问题 先用小分的测试代码正确性 估算时间 合理选择 node memory