Summary CCP

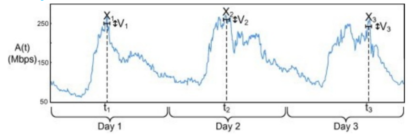
René Bernhardsgrütter, 13.05.2014/17.06.2014
asset = Kapital, Vermögen, Gewinn;

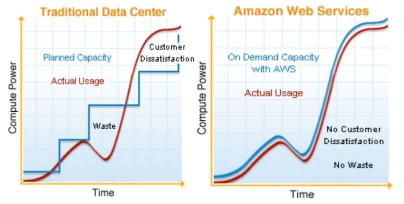
Why CC?

Business

Capital Expenses (CAPEX): a cost which cannot be deducted in the year in which it is paid or incurred and must be capitalized. Something that creates a future benefit. E.g. acquiring fixed assets, hardware, research.
Operational Expenses (OPEX): Ongoing cost for running a product, business, or system (day-by-day). E.g. license fees, telephone, cooling.
Total Cost of Ownership (TCO): CAPEX+OPEX = TCO
Return o. Inv. (ROI): $\frac{\text{(Gain from Investment - Cost of Investment)}}{\text{Cost of Investment}}$
Net Present Value (NPV): Helps access the suitability of an investment, considers the lifetime of the investment, and allows to find the investment cost as seen in the present.
Uptime: $MonthlyUptime\% = \frac{Minutes\_in\_Month - Minutes_{RegionUnavailable}}{Minutes\_in\_month} x100$

Busy Hour





Considerations

How much ICT: Knowlege, skill, ability, capability, capacity, HW, Infrastructure, Platform, App, ...
Consume ICT when needed? Now, tomorrow, in 1 year.
Grow with demands? Start-up to enterprise, ...
Shrink with my demands? Seasons, financial crisis, ...
Pay for actual use? Per day, hour, KB, Mb/s, ...

CC Definitions

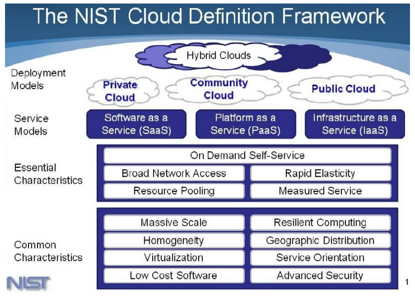
1. On-demand self-service
2. Broad network access
3. Resource pooling (multi-tenancy)
4. Rapid elasticity
5. Measured service (pay-as-you-go)

Service Models

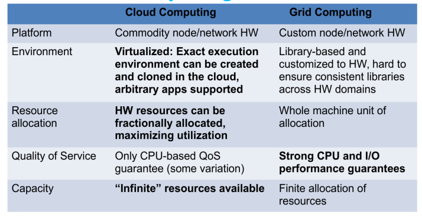
Software as a Service (SaaS): Apps on a cloud infrastr.
Platform as a Service (PaaS): deploy onto the cloud infrastructure consumer-created or acquired applications.
Infrastructure as a Service (IaaS): provision processing, storage, networks, and other fundamental computing resources.

CC Types

Private cloud: Use by single orga. for many consumers.
Community cloud: Use by specific community.
Public cloud: provisioned for open use by public.
Hybrid cloud: two or more distinct cloud infrastructures.

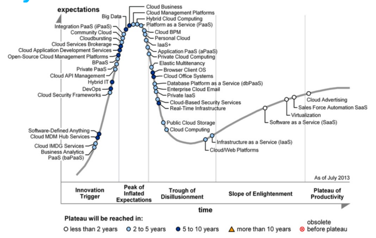


Cloud vs Grid Computing



	Cloud Computing	Grid Computing
Platform	Commodity node/network HW	Custom node/network HW
Environment	**Virtualized: Exact execution environment can be created and cloned in the cloud, arbitrary apps supported**	Library-based and customized to HW, hard to ensure consistent libraries across HW domains
Resource allocation	**HW resources can be fractionally allocated, maximizing utilization**	Whole machine unit of allocation
Quality of Service	Only CPU-based QoS guarantee (some variation)	**Strong CPU and I/O performance guarantees**
Capacity	**"Infinite" resources available**	Finite allocation of resources

Hype Cycle



Evaluation

Fundamental Evaluation Criteria (ordered)
- Security
- Compliance
- TOC
- Service Level Agreements
- Disaster Recovery
- Performance
- Openness
- Controll
- Transparency
- Technology
- Location

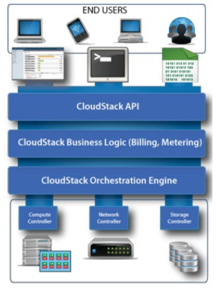
Advantages
- Lower costs
- Unlimited capacity
- Better performance
- Focus on core expertise
- Latest technology
- Technology indpendence
- Universal access
- Data reliability
- Digital compatibility
- Improved collaboration

versus

- **Disadvantages**
  - Provider dependence
  - No network no cloud
  - Outages
  - Feature limitation
  - Lock-in
  - Data and service security
  - Compliance issues
  - Transparency
  - Hidden CAPEX for cloud application development

CloudSigma

- No fixed Server sizes.
- Open O/S – KVM full vert, any x86 OS, full access.
- Open Networking – 'virtual wire' private/custom public networks.



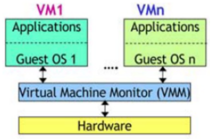
- Tiered Storage Solutions: All SSD Compute Storage, at least 4x the performance of spinning disk. Zadara HDD for lower performance SLA, same availability SLA, same features.
- Burst (every 5 min) and Subscription (3, 6, 12, 24 months) Utility Billing models.

Infrastructure as a Server (IaaS)

Basic svc to virtualize: **Compute, Storage, Networking**.

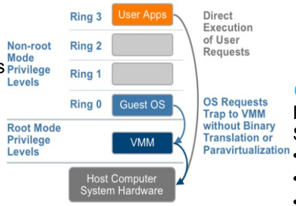
Compute Virtualization

Taxonomy: Full-, Para-, OS-level virtualization



Full

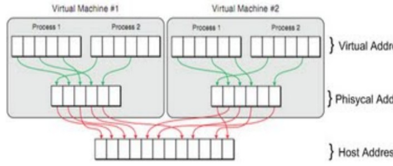
- Complete simulation of the underlying hardware.
- Many instructions run on physical CPU.
- Unmodified Operating System supported.
- X86 difficult, OS needs direct access to hardware (Ring 0), Virtualizing privileged instructions is complicated. Solution: Trap and virtualize the execution of certain sensitive, non-virtualizable instructions (Guest is not aware being virtualized).



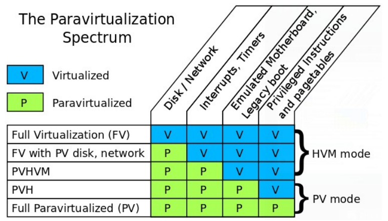
Ring 3
User Apps
Direct Execution of User Requests
Ring 2
Non-root Mode Privilege Levels
Ring 1
Ring 0
Guest OS
OS Requests Trap to VMM without Binary Translation or Paravirtualization
Root Mode Privilege Levels
VMM
Host Computer System Hardware

Binary translation is very complicated and slow!

X86 Software Memory Virtualization:
- VMM maintains Shadow Page Table
- Consists of two types of memory address translations: VA-PA, VM Page Table of virtual machine's OS. PA-HA, managed by VMM.
- On guest OS requests access to VA, intercepted and MMU instructed to access Shadow Page Table directly.



Para-Virtualization



The Paravirtualization Spectrum

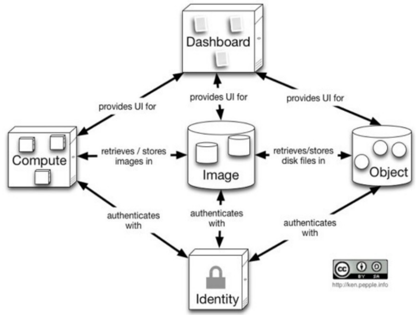
🟦 V Virtualized
🟩 P Paravirtualized

	Disk / Network	Interrupts, Timers	Emulated Motherboard, Legacy boot	Privileged Instructions and pagetables	
Full Virtualization (FV)	V	V	V	V	HVM mode
FV with PV disk, network	P	V	V	V	HVM mode
PVHVM	P	P	V	V	HVM mode
PVH	P	P	P	V	PV mode
Full Paravirtualized (PV)	P	P	P	P	PV mode

Storage Virtualization

Separation of logical storage from physical storage. Concept of Logical Unit Identifier (LUN) Host-, storage-, or network-based.

OpenStack



Compute: Nova

Provides virtual servers on demand (KVM, Xen, Vmware) Schedules networking and block storage.
- Server - VM instance in the compute system.
- Flavor - Hardware configs for a server.
- Image - pre-built OS.
- Reboot - Either soft or hard reboot of a server.
- Rebuild - Remove all server data, reset imate.
- Resize - Convert an existing server to a different flavor, in essence, scaling the server up or down.

CRUD: Create (POST), Retrieve (GET), Update (PUT), Delete (DELETE)

Image: Glance

VM image registration and storage

Object Storage: Swift

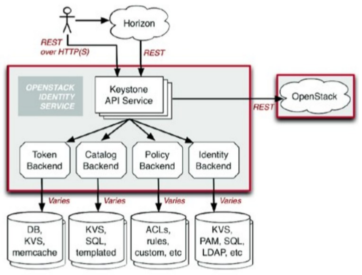
Store and retrieve data. The objects (data) are stored in *buckets* (containers). Eventually consistent design.

Networking: Neutron

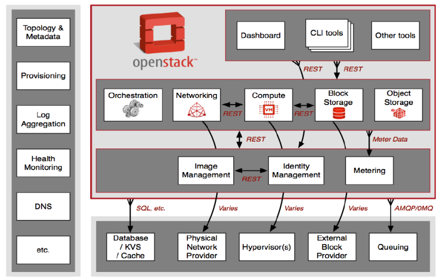
Abstraction to describe **virtual networks, subnet, ports**.
Virtual Network: An isolated L2 segment (like a VLAN)
Virtual Subnet: A block of v4 or v6 IP addresses and associated configuration state.
Virtual Port: End-point for attaching a single device, such as the NIC of a virtual server, to a virtual network.
Neutron Agents can be plugged in to control SDN. This plugin architecture allows different technologies (Linux VLANs, OpenFlow, etc.).
RYU is one of those external solutions.

Identity: Keystone

Auhtentication and authorization (to OpenStack services).



Dashboard: Horizon

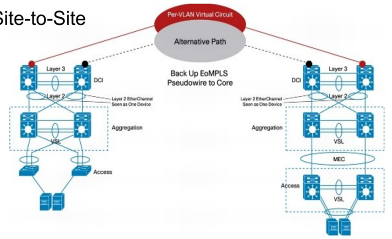


Data Centers

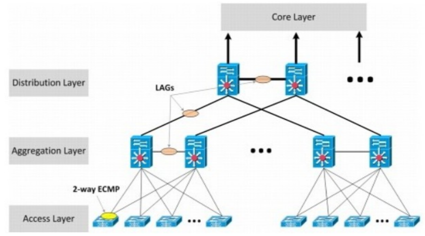
Dedicated Server: Server + Infra
Server Hosting: Server+OS, but probably virtualized
Rack: Top-of-rack-switch..servers..storage..powershelf.

DC-DC

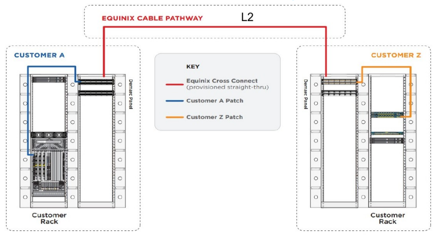
Site-to-Site



DC-Internal



Rack-to-Rack Cross Connect Wiring



DC Requirements

Support different stakeholders (providers, customers): Internal and external view, providers, customer, administrator
Robust and redundant: Expect that a system, where components will fail
Modular and heterogeneous: Switches/Hardware from different vendors, with different link speed, with different protocols, with different ...
Simple in the scaling characteristic: It should be easy to scale up as well to scale down, no complex and mostly automated configurations.
Flexible in the topology: Topology must be flexible to support all sorts of connectivity services, Rack-To-Rack, Rack-to-Internet, Rack-to-Enterprise, Rack-to-Site, Site-to-Site, ...
Efficient+effective: Traffic eng. within site/between sites.
Isolate tenants: A DC should be able to host and isolate different tenants.

Tenants require a flexible network

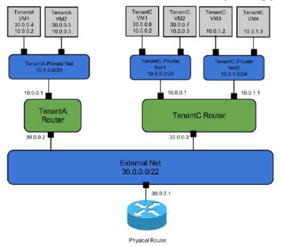
- Self-Service interface
- On-demand instantiation and release
- Scale according to demand
- Quality of Service support

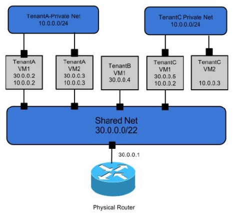
Standards

Ethernet: Standard Layer-2 technology supporting flat networks only and no Quality of Service
VLAN: Virtual LAN's enable isolation of different Networks at Layer-2 (DC-internal or external via VXLAN)
MPLS: Connection oriented mechanism to forward packets by a pre-signalized path trough a network
GRE: Generic Routing Encapsulation is a protocol encapsulates other protocols in a tunnel

Cloud Networking

Flat Networking: In private cloud, enterprise or campus. Services have to communicate with each other, but tenants are different. E.g. moodle.zhaw.ch.
Mixed Flat and Private Networking: Enterprise,campus.
Per-tenant Routers with Private Networks: In *public clouds* hosting *multiple tenants* or for tenants with *multi-network requirements*, when tenants need *control over own network topology*.





Software Defined Networking

SDN decouples control logic of a network device into a controller. Separation of *Control* and *Data* plane.

The SDN controller holds the network logic and SDN enabled devices are plain packet forwarding devices. Best protocol at the moment: OpenFlow.

RYU

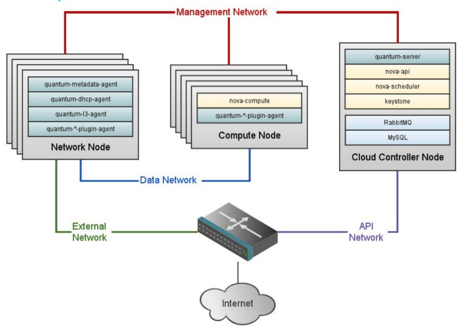
Ryu is a component-based SDN framework. Supports different "Southbound" protocols like **OVSDB**, **OpenFlow**, and SNMP. Controls bridge on every compute node.

OpenFlow for discovery and OpenFlow events (OFPPortConfig, OFPHello).
OVSDB for advanced features like GRE tunnels.

OpenvSwitch

An *intelligent bridge* to apply OpenFlow and OVSDB with Native support for isolation, live migration and QoS.
OVSDB: **OpenvSwitch**-only protocol, over JSON-RPC.

With OpenStack



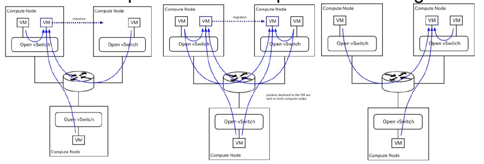
Quantum server manages network associations and does so via RYU REST interface. RYU talks to OpenvSwitch directly over OpenFlow and OVSDB **via management network**.

RYU reacts on events sent by OpenStack

Create/Delete VM Instance: RYU creates quantum port.
Compute node boots up: RYU detects bridge and performs basic init.
Network creation/delete: RYU handles the key-value pair sent by quantum and isolates, with VLAN or GRE.

Live Migration

RYU is aware of the MAC addresses inside the FlowTable of OpenvSwitch => duplicts traffic in migration:



OpenFlow

A vSwitch/"datapath" consists of 1 or more Flow Tables, 1 Group Table, a channel to the controller.

Ports

Physical port: one-to-one mapping to phyiscal interfaces of the datapath.
Local port: port that can be defined by the switch using non-openflow method (tunnels or aggregation groups)
Logical port: reserved OpenFlow ports: **ALL** (port, representing all ports on the device), **CONTROLLER** (port, representing the connected controller), **IN_PORT** (port where the packet arrived, either physical or local), **ANY** (Special case for wildcard ports in OFP commands).

Tables

Pipeling via OpenFlow Tables:
OpenFlow only: Packets sent along pipelining process.
OpenFlow hybrid: Devices support an attribute to decide, if a packet is forwarded along the OFP-pipeline or not.

Matching

The match-fields can be Layer 2 or Layer 3.

Group Table

A mechanism, to implement e.g. broadcasts.

Standards

A "standard" is only a specification. To be a "standard" it requires adoption, the beginning of adoption comes with reference implementations.

Reason for standards

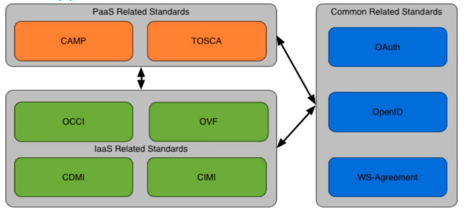
Conformance: Define what a system does (not).
Interoperability: How and what to communicate.
Integration: Describes how a system brings together different concepts and technologies.
Portability Operations: Ensure a common process view and mapping between systems and processes.

Standard Defining Organization (SDO)

Industry- or sector-based standards organizations that develop and publish industry specific standards.

Role:Set scope of work, provide forum, IPR protection, process to ease editing and collaboration, and provide mechanisms to reach consensus.

Could Applications



Types of Standards

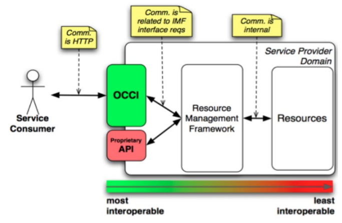
Industry standards are defined and driven out or the product and interest of the industry. Most of theses standards are developed in closed groups and require one voting mechanism or another. E.g. OCCI, CIMI.
De-facto standards are those defined by a (single) entity and are adapted by the community so quickly/widely that the gain is huge popularity. E.g. Amazon EC2.
Dejure standards are those driven from government

related entities. Community driven, industry or defacto standards can become dejure standards if a government officially embraces one.E.g. HTML, PDF.
Community standards are those driven by a mostly open community of people who by a certain process reach agreement on a set of features a standard should have. Most community standards are open and driven by individual people for the common good. E.g. OpenID.

Open Standards

Organisation + Specification + Implementation + Adoption = OpenStandard
E.g. OCCI, OpenStack

OCCI as Open Standard Example



Protocol and API for Mgmt of Cloud Service Resources. **OCCI is an API and Protocol**. It sits on the boundary of a Service Provider and Service Consumer. It does not assume anything about behind the boundary.

Components

Core: common model elements.
Infrastructure: models infrastructure elements (compute, storage, network) – extension of Core.
RESTful HTTP Rendering: how to serialise the model “on-the-wire”, mapping to REST (HTTP verbs).

Concept		Description
Category		Way of describing type(s) incl. their attributes, actions and relationships for entities
	Kind	Type of a resource
	Mixin	Way to extend a Kind (new behavior); Mechanism for Tagging, (Resource/OS) Templating
	Action	A „push the button operation“
Entity		Something that is instantiated
	Resource	A resource with its attributes, links and applicable actions - an „instance“, Created from Category
	Link	Exposes relationship between resources

Query Interface (QI)	A way of discovering Capabilities
CRUD	Create, Retrieve, Update and Delete ops on QI and Entities

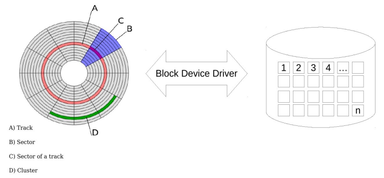
Where Do Standards Matter?

Not needed factivities related to **Start-ups, Prototyping**.
Needed for interop., **large IT orgs, government**.

Storage

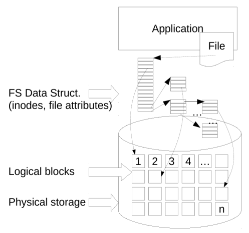
Types of Storage

Block



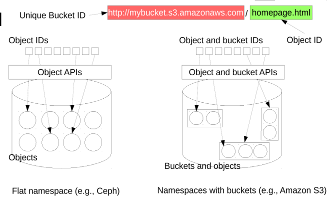
Data on logical blocks over a complex physical layout.

File



Data as file hierarchies by FS data structures that map files to logical blocks. Unix uses the concept of inodes, which store file metadata (attributes) and pointers to data.

Object



Data as objects (data, metadata, identifiers, ...) and offered to client applications via APIs. Devices that store objects are typically referred to as **Object Storage Devices (OSDs)**.

Storage Area Network (SAN)

Dedicated **network to access** storage at **block level**. Basic requirements are **capacity and performance**.
“Noisy neighbor” effect: SAN performance is affected by network peak loads caused by separate users.

QoS factors

Bandwidth: data throughput on the system.
Latency: delay in executing read/write operations.
Queue depth: # of operations waiting to be processed.

Distributed Filesystems (DFS)

Goals: high performance, utility storage, transparency (access, location, concurrency, failure, replication, migration), heterogeneity, scalability.
Trade-off with respect to a local file system: Longer delays, unreliability, unpredictability, security.

DFS Architecture

Client server: simple.
Cluster-based: high-performance and fast-recovery.
Peer-to-peer: Symmetric systems usually deploying Distributed Hash Tables (DHTs) as lookup mechanisms.

Cloud Storage

Features

No single point of failure
High Availability: Availability of the cloud storage system must be increased with data replication and application of redundant erasure codes.
Data distribution (striping): When more throughput needed, replication of data to different locations for parallel offer of file/object.
Data replication: Required according to user policies (e.g., store replicas on different physical devices) to recover from failures.
Geo-replication: Allows to migrate frequently accessed chunks of data closer to user for better performance.
Data Placement: Cloud storage systems may implement different data placement policies aimed at using available storage resources in efficient ways (e.g., balancing data across physical disks). This also includes redistributing existing data in case of drastic changes in the system (e.g., a storage node fails).

Cinder

Persistent **block-level** storage for OpenStack compute instances. The lifecycle of Cinder volumes is independent of that of instances (like USB-HDD). Snapshots possible.

Components

cinder-api: WSGI app that authenticates and routes requests throughout the Block Storage system.
cinder-scheduler: Schedules/routes requests to the appropriate volume service. Depending upon your configuration this may be simple round-robin scheduling to the running volume services, or it can be more sophisticated through the use of the Filter Scheduler. The Filter Scheduler is the default in Grizzly and enables filter on things like Capacity, Availability Zone, Volume Types and Capabilities as well as custom filters.
cinder-volume: Manages Block Storage devices, specifically the back-end devices themselves.
cinder-backup: Provides a means to back up a Cinder Volume to SWIFT.

Benefits

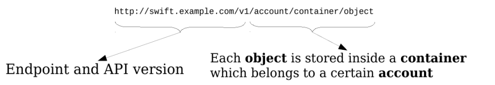
- Instances can be terminated and restarted while persistently storing data on lifecycle-independent Cinder volumes (as opposed to ephemeral storage).
- Misbehaving instances can be replaced and new ones can be attached to the same storage with a simple reconfiguration through the Dashboard.
- Multiple storage backends available, from DAS to fully-featured DFSs (e.g., Ceph).
- Cinder volumes can be used as boot disks for cloud instances.

Swift (Object Storage)

- Swift is a highly available, distributed, eventually consistent object/blob storage system.
- Horizontally scalable, e.g., by adding new storage servers. Built-in replication and data distribution tools.
- Simple object storage design. Main storage concept are objects (units of storage containing actual data) and containers (collections of objects).

Principles

- Every Swift object has a URI and associated metadata.
- Swift allows the configuration of **zones**: Logical concept to reflect physical storage system. Mappable to physical drives, servers, racks, switches, entire datacenters.
- Replica in different zones (default: 3)
- REST-ful HTTP APIs.
- Data location inside the cluster is transparent.
- Data cluster can be expanded with no downtimes.
- Nodes can be removed/swapped with no downtimes.



http://swift.example.com/v1/account/container/object
Endpoint and API version
Each object is stored inside a container which belongs to a certain account

`GET http://swift.example.com/v1/account/`	Lists all containers of an account
`PUT http://swift.example.com/v1/account/container_id`	Creates a new container
`GET http://swift.example.com/v1/account/container_id/`	Lists all objects of a container
`PUT http://swift.example.com/v1/account/container_id/object_id`	Creates a new object

Access

Swift-Details

Proxy Server: Handles all the API requests coming from clients and implements some failure recovery policies.
Ring: Maintains the mapping between the name of entities and their physical locations. The ring controls how data is distributed and moved across the cluster.

Performance

Capacity is a collective result of the effectivity and efficiency of available resources and a **quantification of System Performance**.

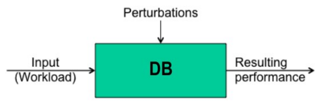
System Performance Evaluation

Defines several *Activities*, in different phases of the System Life-Cycle.
Life-Cycles: Design, Impl., Deployment, Provisioning, Operation, Disposal.

Activities

- Define the system under test
- Setting performance objectives
- Define performance characteristics and models
- Performance analysis of development code
- Tests of software builds
- Benchmarking software releases
- PoC testing in the target environment
- Configuration optimization of the production environment
- Monitoring and analysis of issues

System under Test



Approaches based on mathematical models, empirical data (emulation or systems in use).
Workload: For a DB => number of queries issued per time by a client.
Perturbations: Parallel processes in a multi-tasking system on a single-Core-CPU.
Metric: DB Response Time, here defined as the total time the DB needs to reply to a set of queries.
Resulting Performance: The DB Response Time is poor because the system that hosts the DB is under heavy load by parallel processes.

Terminology

IOPS: Input/Output Operations per Second.
Throughput: Data Rate (quantity of Data per Time), Operation Rate (quantity of Operations/Transactions per Time). Throughput is the *Total Amount of Work* that can be performed by a System per Unit of Time.
Response Time: Time for an operation to complete.
Latency: Time an operation spends waiting to be svced.
Time-based Utilization: How busy a resource is over a period of time is (= resource usage).
Capacity-based Utilization: E.g. a disk is able to deliver a certain amount of throughput. That proportion is called the utilization. This implies that a disk at 100% utilization cannot accept any more work. **Time-bsd!=Capacity-bsd**, then 100% utilization only means it is busy 100% of the

time. 100% busy != 100% capacity.
Saturation: Degree to which a resource has queued work it cannot service. Saturation begins to occur at 100% utilization (capacity-based).
Bottleneck: A resource that limits overall system performance (limiting factor).
Cache: Fast storage element that buffers certain elements to accelerate Input/Output operations.

Statistics

Observation-based: Application request takes 10ms, of that, 9ms is disk I/O, suggest to cache I/O in memory, with expected RAM latency around 10µs.
Experiment-based:Application transaction latency is ~10ms, increase the application thread count to allow more concurrency, Application transaction latency averages 2ms.

QoS vs QoE

Quality of Service: Measures performance and quality of service based on quantitative, **objective**, tech. metrics.
Quality of Experience: Measures a customer's experience with a service, subjective rating.

High QoS != high QoE => Get high QoE and QoS ratings!

Caching

hit ratio: hits/total accesses (hits + misses)
cache miss rate: misses/s
runtime: hit rate x hit latency + miss rate x miss latency

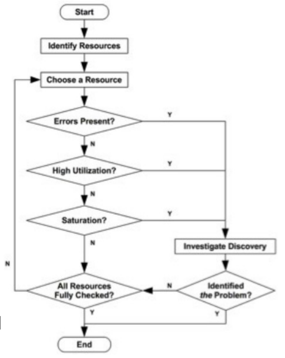
Cache States:
Cold: Empty or filled with unwanted data. Hit ratio is 0.
Warm: Has useful data but hit ration < 99 %.
Hot: Has requested data, high hit ratio > 99 %.

Perspectives

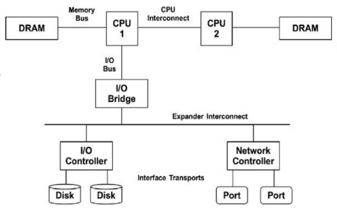
Resource analysis : CPUs, Memory, Disks, buses... Focus on utilization, to identify which resources are at their limit. Metrics of IOPS, Throughput, Utilization, Saturation.
Workload analysis: Performance of the a[[. Targets include Requests (workload applied), Latency (Response time) and Completion (Were there any errors?). Metrics of Latency, Throughput (transactions per second).

USE Method

Resource: all physical server functional components (CPUs, busses, ...).
Utilization: for a set time interval, the percentage of time that the resource was busy servicing work. While busy, the resource may still be able to accept more work; the degree to which it cannot do so is identified by saturation. (Time-based utilization).



Saturation: the degree to which the resource has extra work that it can't service, often waiting on a queue.
Errors: the count of error events.



Benchmarking

Tests performance in controlled env., to compare choices and performnc limits.
Reasons: Comp. diff. sys/compnnts for tuning, Troublesh., etc..

Principles

Repeatbl, observbl, portbl, easily to check, realistic.

Types

Micro-Benchma.: Art. workloads to test part. operation.
Simulation: Simulate customer application workload.
Replay: Testing performance with actual operations.

Methodologies

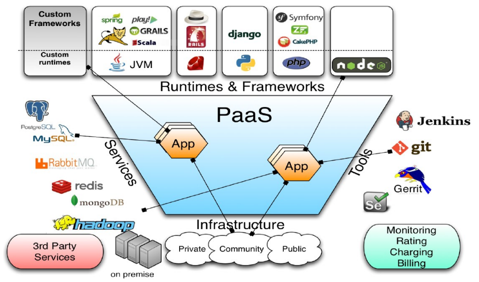
Active Benchmarking: Analyze performance while a benchm. is running using performance monitoring tools.
Passive Benchmarking: Run performance benchmark without analyzing the workload in real time.

USE in Benchmarking

USE helps in Active Benchmarking to identify resources used by the simulated workload and actively analyze utilization during a benchmark run.

Platform as a Service (PaaS)

PaaS is all about writing code, building data and consuming services ... nothing else. All the developer cares about is their code.



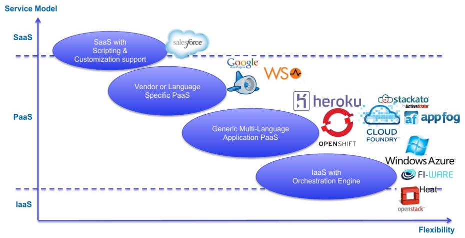
Advantages

Software Engineer / Dev: Focus on Code and Dev Process. Easy, fast deployment & scalability. Application monitoring, migration, ...
System Engineer / Administrator: Focus on administration & monitoring. HW, OS & base Services. Less support in deploying, migrating & monitoring apps.
Architects / Management: Higher Flexibility (scalability, variety of svc). Agile deploymnt. Demand cost (OPEX).

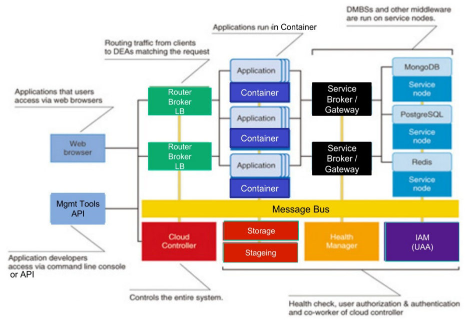
Disadvantages

Less flexible than IaaS: Only provided limited Runtimes & Services available.
Dependency on Provider: Lock-In.
Change of Software / System Architecture: To take advantage of all the positive aspects, the Design of the System Architecture must be changed.

Classification



Generic PaaS Architecture



DMBSs and other middleware are run on service nodes.
Applications run in Container
Routing traffic from clients to DEAs matching the request
Applications that users access via web browsers
Web browser
Router Broker LB
Router Broker LB
Application
Container
Application
Container
Application
Container
Service Broker / Gateway
Service Broker / Gateway
MongoDB
Service node
PostgreSQL
Service node
Redis
Service node
Mgmt Tools API
Message Bus
Cloud Controller
Storage
Stageing
Health Manager
IAM (UAA)
Application developers access via command line console or API
Controls the entire system.
Health check, user authorization & authentication and co-worker of cloud controller

Message bus: Central communication system, used by all components for internal communication. Publish / subscribe based. Protects "itself" at all cost: If it hangs, the whole system breaks.
Router: Routes in-traffic (URL/public IP). Provides Load Balancing. Often SSL-Endpoint.
Application Container: Droplet Execution Agent DEA.
Cloud Controller: Manages the life-cycle of applications. Provides containers. Binds svc to applications.
Health manager: Monitors the state of the applications.
Service Broker: Provisioning of service instances (setup, credentials, ...). Bind/Unbind service to application.

PaaSD - App Design/Depl.

Cloud-Native Application: An application that has been (re)designed or (re)architected to operate in a cloud env.

Goals

- No single point of failure -> redundant components
- Distributed state -> E.g. containers. No central DB. (message bus).
- Self healing -> health manager.
- Horizontal scalable -> add VMs if required (cont./srv).

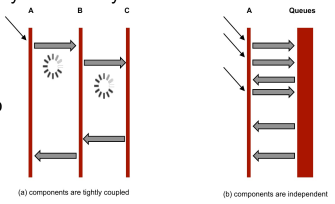
Principles

- Loose coupling (to replace components individually)
- Event-driven
- Asynchronous
- Non-blocking
- Idempotent (no side effects when called multiple times)
- Eventually consistent (not always) (e.g. health manager is periodically checking and restore desired condition)
- Language-independent communication (RESTfull API, Libraries to access message bus)
- Dynamically discoverable components (E.g. Components are announcing themselves on the message-bus)
- No inter-component dependencies (Launch in any order (wait until dependencies are resolved, Scale up and down independently)
- Monitor using defined end points (e.g. HTTP)

General Guidelines

- Consider scalability from the beginning
- Work out parallelisms & distribute workload
- Make an efficient use of elasticity
- Design for failure (design-time recovery)
- Keep data and computational resources close
- Application components must be decoupled and communicate asynchronously

Loosely Coupled Components =>



A
B
C
A
Queues
(a) components are tightly coupled
(b) components are independent

Stateless Design

If components do not keep any internal state, no information is lost if they stop/fail, but stateless components rely on ext. storage.

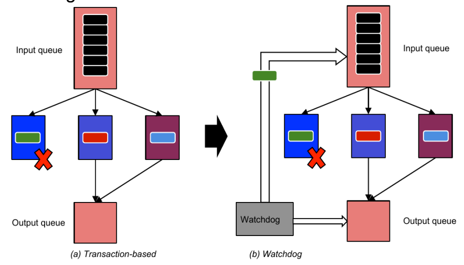
Load Balancer

Resources must be reserved and released according to instantaneous needs, but this might affect pricing.

Availability

A well-designed cloud application must be able to offer high-availability on top of a low-availability platform.

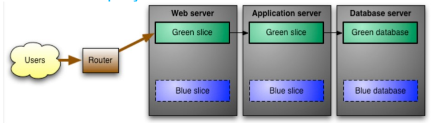
Watchdog-Pattern:



Zero-Downtime Deployment

Key to zero-downtime release is decoupling the various parts of release so they can happen independently as far as possible.

Green/Blue Deployment

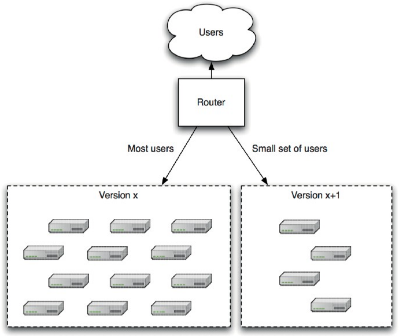


- Two identical environments (blue & green)
- Green is running productively
- Deploy new release to blue environment, let it warm up
  → smoke tests to check it works
  → move to the new version by changing the router configuration
- Blue becomes productive. Green is used for the next release.
- If something goes wrong, change router back to green env.

When DB schema changes:
V1: Put application into read-only mode before switch, copy green DB, restore into blue DB, perform migration, change to read-write mode.
V2: Design the application, so that you can migrate the DB independently of the upgrade process.

Canary releasing

Benefits: Rolling back is easy, A/B testing can be done by routing some users to new version and some to the old. You can check if the application meets capacity requirements gradually.
Disadvantages: Harder to manage in smaller installations, Imposes further constraints on DB schema upgrades, Limited to a few versions of the application in production.



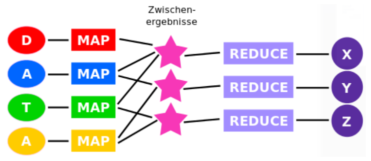
Data processing in the Cloud

Map-Reduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
An abstraction for performing computations on data:
- automatic parallelization of computations
- large-scale data distribution
- simple, yet powerful interface
- user-transparent fault tolerance commodity hardware

Common Operations on Data

"Map" step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.
"Reduce" step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.



Procedure

- Eingabedaten (D, A, T, A) auf Reihe von Map-Prozessen verteilt (bunte Rechtecke), welche die vom Nutzer bereitgestellte Map-Funktion berechnen.
- Map-Prozesse werden idealerweise parallel ausgeführt.
- Jede Map-Instanz legt Zwischenergebnisse ab.
- Von jeder Map-Instanz fließen Daten in eventuell verschiedene Zwischenergebnisspeicher.
- Sind alle Zwischenergebnisse berechnet, ist die Map-Phase beendet und die Reduce-Phase beginnt.
- Für jeden Satz an Zwischenergebnissen berechnet jeweils genau ein Reduce-Prozess (violette Rechtecke) die vom Nutzer bereitgestellte Reduce-Funktion und damit die Ausgabedaten (violette Kreise X, Y und Z).
- Reduce-Prozesse werden auch parallel ausgeführt.

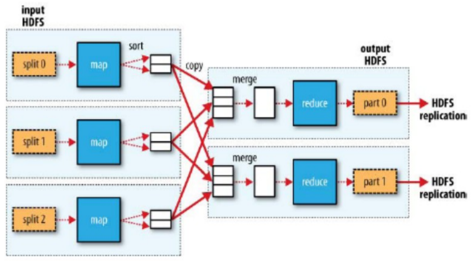
Hadoop

HDFS

Distributed File System, splits data into chunks of fixed size (default: 64MB). Large files, sequential reads.
Server Types: Namenode to keep the metadata, Datanode to store data.
Failures: Handled through chunk level replication (3x).
Pattern: Write-once-ready-many.

Hadoop MapReduce Framework

Job: program that executes map and reduce processing across a data set.
Task: an execution of a Mapper or a Reducer on a slice of data also called, Task-In-Progress (TIP).
Task Attempt: particular instance of an attempt to execute a task on a machine.


input HDFS
split 0
map
sort
copy
split 1
map
split 2
map
merge
reduce
part 0
output HDFS
HDFS replication
merge
reduce
part 1
HDFS replication