Computing Infrastructure

[Sofia Martellozzo]

Academic Year 2021-2022

Contents

1 Computing Infrastructure

1.1 Introduction

Definition 1. Computing Infrastructure: *Technological infrastructure that provides hardware and software for computation to other systems and services.*

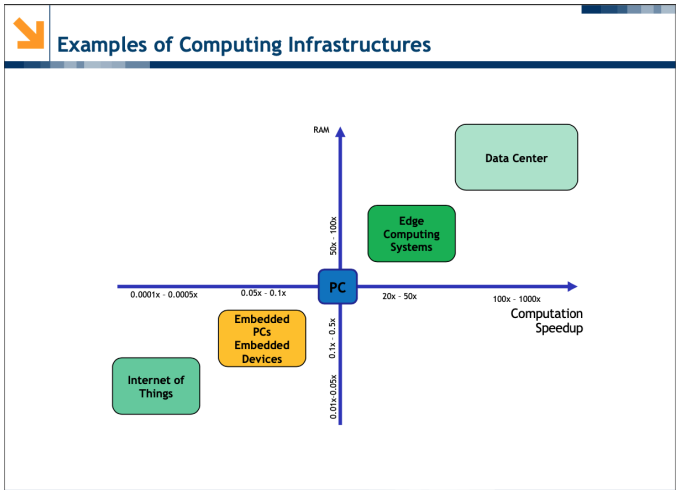


Figure 1: Computing Infrastructure

Advantages:

- Lower IT costs
- High performance
- Instant software updates
- "Unlimited" storage capacity
- Increased data reliability
- Universal document access
- Device Independence

Disadvantages:

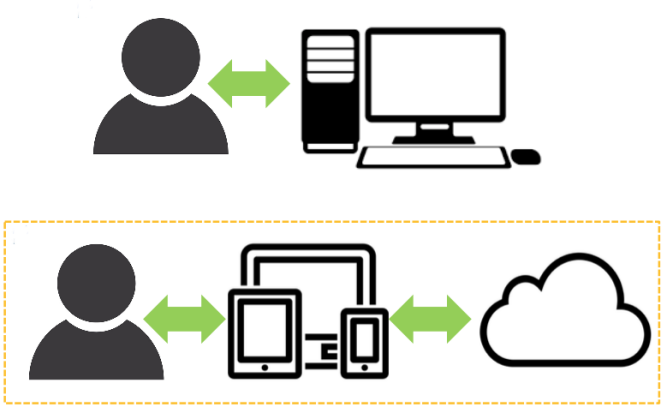
- Require a constant Internet connection
- Do not work well with low-speed connections
- Hardware Features might be limited
- Privacy and security issues
- High Power Consumption (1% overall worldwide total energy consumption due to datacenters)
- Latency in making decision

Amortized Cost	Component	Sub-Components
~45%	Servers	CPU, memory, disk
~25%	Infrastructure	UPS, cooling, power distribution
~15%	Power draw	Electrical utility costs
~15%	Network	Switches, links, transit

2 Data WareHouse

2.1 Introduction

In the last few decades, computing and storage have moved from PC- like clients to smaller, often mobile, devices, combined with large internet services.
Traditional enterprises are also shifting to Cloud computing.



User experience improvements

- Ease of management (no configuration or backups needed)
- Ubiquity of access

Advantages to vendors

- Software-as-a-service allows faster application development (easier to make changes and improvements)
- Improvements and fixes in the software are easier inside their data centers (instead of updating many millions of clients with peculiar hardware and software configurations)
- The hardware deployment is restricted to a few well-tested configurations.

Server-side computing allows

- Faster introduction of new hardware devices (e.g., HW accelerators or new hardware platforms)
- Many application services can run at a low cost per user.

Some workloads require so much computing capability that they are a more natural fit in datacenter (and not in client-side computing).
A couple of examples (Search services (web, images, and so on), Machine and Deep Learning (GPT-3).

2.2 From Data Centers to Warehouse-scale computers

Data centers = is a place in which there are many servers
Whareouse = is a type of Data Center, it works as a computer.
The trends toward server-side computing and widespread internet services created a new class of computing systems:

Definition 2. warehouse-scale computers (WSCs) *The massive scale of the software infrastructure, data repositories, and hardware platform.*

- *is an internet service (= service provided by inyternet)*
- *may consist of tens or more individual programs (= not a single program, a collection of them that together create/provide the service)*
- *such programs interact to implement complex end-user services such as email, search, maps or machine learning.*

Data centers are buildings where multiple servers and communication units are co-located because of their common environmental requirements and physical security needs, and for ease of maintenance.
In **Traditional Data Center**: typically host a large number of relatively small- or medium-sized applications, each applications is running on a dedicated hardware infrastructure that is de-coupled and protected from other systems in the same facility, applications tend not to communicate each other. Those data centers host hardware and software for multiple organizational units or even different companies.

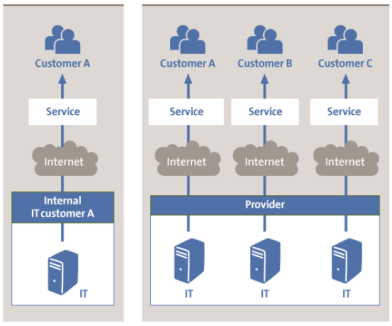


Figure 2: Traditional Data Center

WSCs belong to a single organization, use a relatively homogeneous hardware and system software platform (=¿ easier to manage and cheaper, but has limitations on functionalities), and share a common systems management layer (such as Google, Facebook, Alibaba, Amazon, Dropbox...).
(you have 1 services that you want to provide to a 'huge' amount of customers)
Run a smaller number of very large applications (or internet services).
The common resource management infrastructure allows significant deployment flexibility.
The requirements of:

- homogeneity
- single-organization control
- cost efficiency

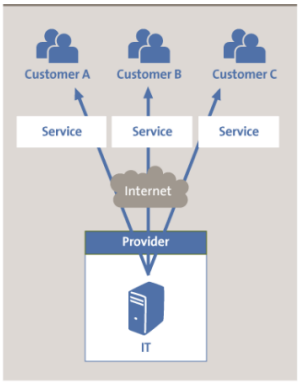


Figure 3: Warehouse-Scale Computers

Initially designed for online data-intensive web workloads, WSCs also now power public clouds computing systems (e.g., Amazon, Google, Microsoft). Such public clouds do run many small applications, like a traditional data center. (All of these applications rely on Virtual Machines (or Containers), and they access large, common services for block or database storage, load balancing, and so on, fitting very well with the WSC model).
These are not just a collection of servers:
The software running on these systems executes on clusters of hundreds to thousands of individual servers (far beyond

a single machine or a single rack)
+
The machine is itself this large cluster or aggregation of servers and needs to be considered as a single computing unit. (=> scale up in terms of performance)
Several data-centers: Multiple Data Center located far apart (placed near point of interest) =¿ becomes important also the PRIVACY: data of a country must remain in it.
Multiple data centers are (often) replicas of the same service (to reduce user **latency** and improve serving **throughput**).
A request is typically fully processed within one data center.
Availability: Services provided through WSCs must guarantee high availability, typically aiming for at least 99.99% uptime (i.e., one-hour downtime per year). Achieving such fault-free operation is difficult when a large collection of hardware and system software is involved.
WSC workloads must be designed to gracefully tolerate large numbers of component faults with little or no impact on service level performance and availability.

2.3 Architectural Overview of WSCs

Hardware implementation of WSCs might differ significantly each other; However, the architectural organization of these systems is relatively stable.

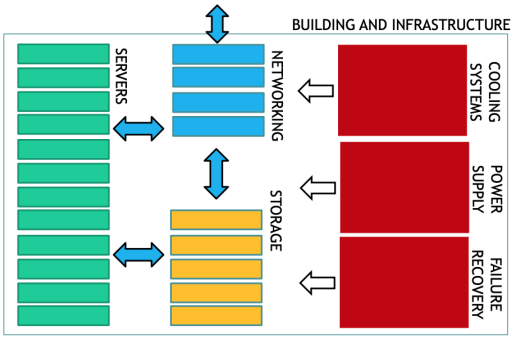


Figure 4: Warehouse-Scale Computers Overview

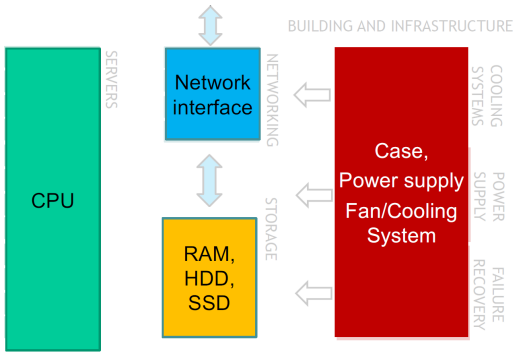


Figure 5

- **Servers**: the main processing equipment
- **Storage**: how and where to store the information
- **Networking**: providing internal and external connections
- **Building and Infrastructure**:WSC has other important components related to power delivery, cooling, and building infrastructure that also need to be considered

3 Server

3.1 Overview

Servers hosted in individual shelves are the basic building blocks of WSCs. They are interconnected by hierarchies of networks, and supported by the shared power and cooling infrastructure. They are stored in shelves called wraks, that are organized along corridors. All servers are connected to each other in the same wrak, but also (on a higher level) communicate through different wrack. They have 3 type of shape:

- Rack (1U or more)
- Blade enclosure format
- Tower

They may differ in:

- Number and type of CPUs.
- Available RAM Locally attached disks (HDD, SSD or not installed).
- Other special purpose devices (like GPUs, DSPs and coprocessors).

and are usually built in a tray or blade enclosure format, housing the motherboard, chipset, additional plug-in components.

3.1.1 The Motherboard

The motherboard provides sockets and plug-in slots to install CPUs, memory modules (DIMMs), local storage (such as Flash SSDs or HDDs), and network interface cards (NICs) to satisfy the range of resource requirements.

3.1.2 Chipset and additional components

- Number and type of CPUs:
  - From 1 to 8 CPU socket
  - Intel Xeon Family, AMD EPYC, etc..
- Available RAM
  - From 2 to 192 DIMM Slots
- Locally attached disks:
  - From 1 to 24 Drive Bays
  - HDD or SSD (see specific lecture)
  - SAS (higher performance but more expensive) or SATA (for entry level servers, usually cheaper)
- Other special purpose devices:
  - From 1 to 20 GPUs per node, or TPUs
  - NVIDIA Pascal, Volta, etc..
- Form factor:
  - Form 1U to 10U
  - Tower

We distinguish between rack, tower and blade.

3.1.3 Rack

Racks are special shelves that accommodate all the IT equipment and allow their interconnection. The rack are used to store many **Rack server**.
IT equipment must conform to specific sizes to fit into the rack shelves. Them has different hight, but same SIZE, that is **standardize** (so make them easier to design): server racks are measured in rack units "U's".
The advantages of using these racks is that it allows designers to stack up (one on top of the others) other electronic devices along with the servers.

A Rack is not only a physical structure:

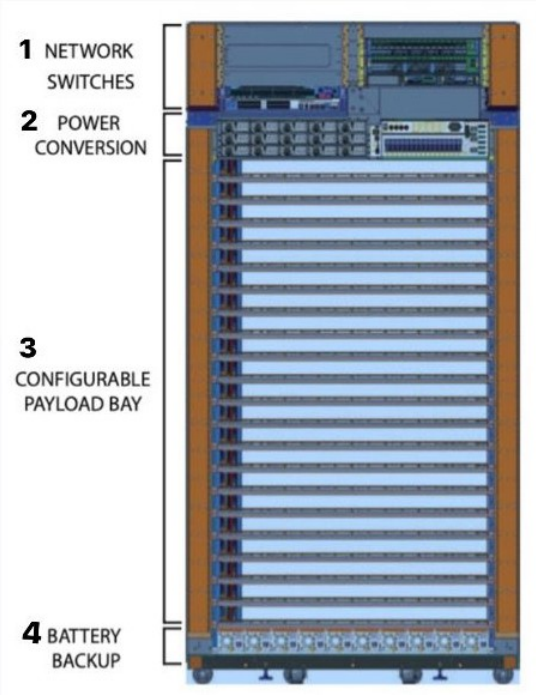


Figure 6: RACK Servers

(3) The rack is the shelf that holds tens of servers together.
(2-4) It handles shared power infrastructure, including power delivery, battery backup, and power conversion.
(2) There are 2 types of electronic delivery: it can provide/automatically select the server (can swich ON and OFF remotely the servers).
(4) It is used in case of problems with energy (power failure):

- just swich off if a power failure occur, them cannot supply all the power demand (execution)
- it provides 15 minutes of power, that is the time to swich on the power supply generator (source)

(3) The width and depth of racks vary across WSCs: some are classic 19-in wide, 48-in deep racks, while others can be wider or shallower.
(1) It is often convenient to connect the network cables at the top of the rack, such a rack-level switch is appropriately called a Top of Rack (TOR) switch.

Rack servers:
It is designed to be positioned in a bay, by vertically stacking servers one over the another along with other devices

Pros

- **Failure containment**: very little effort to identify, remove, and replace a malfunctioning server with another.
- **Simplified cable management**: easy and efficient to organize cables.
- **Cost-effective**: Computing power and efficiency at relatively lower costs.

Cons

- **Power usage**: Needs of additional cooling systems due to their high overall component density, thus consuming more power.
- **Maintenance**: Since multiple devices are placed in racks together, maintaining them gets considerably tough with the increasing number of racks.

3.1.4 Tower

It is the simplest but is not usually adopted. **Tower Servers** look and feel a lot like traditional tower PCs. Like everything they have their Pros and Cons.

Pros

- **Scalability and ease of upgrade**: customized and upgraded based on necessity.
- **Cost-effective**: Tower servers are probably the cheapest of all kinds of servers
- **Cools easily**: Since a tower server has a low overall component density, it cools down easily.

Cons

- **Consumes a lot of space**: These servers are difficult to manage physically.
- **Provides a basic level of performance**: a tower server is ideal for small businesses that have a limited number of clients.
- **Complicated cable management**: devices aren't easily routed together

3.1.5 Blade

Blade servers are the latest and the most advanced type of servers in the market. They can be termed as hybrid rack servers (like orizontal rack server but placed vertically), in which servers are placed inside blade enclosures, forming a blade system. The biggest advantage of blade servers is that these servers are the smallest types of servers available at this time and are great for conserving space. A blade system also meets the IEEE standard for rack units and each rack is measured in the units of "U's".

Pros

- **Load balancing and failover**: Thanks to its much simpler and slimmer infrastructure, load balancing among the servers and failover management tends to be much simpler.
- **Centralized management**: In a blade server, you can connect all the blades through a single interface, making the maintenance and monitoring easy.
- **Cabling**: Blade servers don't involve the cumbersome tasks of setting up cabling. Although you still might have to deal with the cabling, it is near to negligible when compared to tower and rack servers.
- **Size and form-factor**: They are the smallest and the most compact servers, requiring minimal physical space.

Cons

- **Expensive configuration**: Although upgrading the blade server is easy to handle and manage, the initial configuration or the setup might require heavy efforts in complex environments.
- **HVAC**: Blade servers are very powerful and come with high component density. Therefore, special accommodations have to be arranged for these servers in order to ensure they don't get overheated. Heating, ventilation, and air conditioning systems must be managed well in the case of blade servers.

3.2 Data-center architecture

The IT equipment is stored into corridors and organized into racks (the goal is to maximize the number of racks =¿ max number of servers).
Corridors where servers are located are split into *cold aisle*, where the front panels of the equipment is reachable, and *warm aisle* where the back connections are located.
cooling system: Cold air flows from the front (cool aside), cools down the equipment, and leave the room from the back (warm aide). There is an additional roof in order to do not waste cold air on the Back side (cold part of the corridor).

3.3 Hardware accelerators

Hardware accelerator are accurate particular applications that support specific high operation (of ML) with a lot of data. Deep learning models began to appear and be widely adopted, enabling specialized hardware to power a broad spectrum of machine learning solutions. To satisfy the growing compute needs for deep learning, WSCs deploy specialized accelerator hardwares:

- GPU
- TPU
- FPGA

3.3.1 Graphical Processing Units (GPU)

Data-parallel computations: the same program is executed on many data elements in parallel. The scientific codes are mapped onto the matrix operations. High level languages (such as CUDA, OpenCL, OPENACC, OPENMP, SYCL) are required. Up to 1000x faster than CPU.

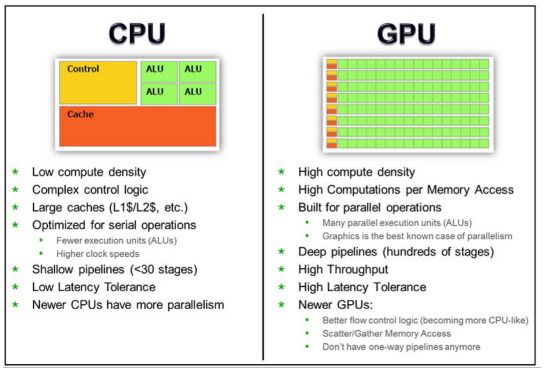


Figure 7: CPU vs GPU

The performance of such a synchronous system is limited by the slowest learner and slowest messages through the network. Since the communication phase is in the critical path, a high performance network can enable fast reconciliation of parameters across learners.
GPUs within the rack: PCI AND NVlink GPUs are configured with a CPU host connected to a PCIe-attached accelerator tray with multiple GPUs.
GPUs within the tray are connected using high-bandwidth interconnects such as NVlink.
NVLINK evolution and NVSwitch In the A100 GPU, each NVLink lane supports a data rate of 50x 4 Gbit/s in each direction. The total number of NVLink lanes increases from six lanes in the V100 GPU to 12 lanes in the A100 GPU, now yielding 600 GB/s total

3.3.2 Tensor Processing Unit (TPU)

While suited to ML, GPUs are still relatively general purpose devices. In recent years, designers further specialized them to ML-specific hardware: Custom-built integrated circuit developed specifically for machine learning and tailored for TensorFlow.

A Tensor is an n-dimensional matrix. This is the basic unit of operation in with TensorFlow.
TPUs are used for training and inference:

- TPUv1 is an inference-focused accelerator connected to the host CPU through PCIe links.
- Differently, TPUv2 and TPV3 focus training and inference

Each Tensor core has an array for matrix computations (MXU) and a connection to high bandwidth memory (HBM) to store parameters and intermediate values during computation. **TPUv2**
8 GiB of HBM for each TPU core, one MXU for each TPU core, 4 chips, 2 cores per chip.

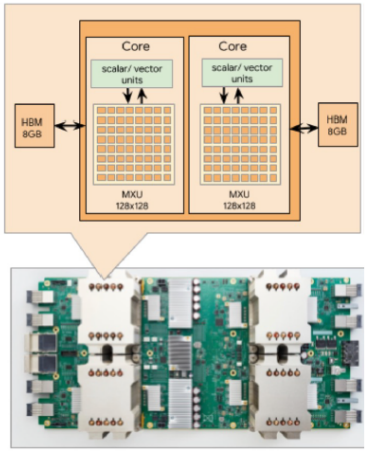


Figure 8: TPUv2 - 4 chips, 2 core per chip

In a rack multiple TPUv2 accelerator boards are connected through a custom high-bandwidth network to provide 11.5 petaflops of ML compute.
The high bandwidth network enables fast parameter reconciliation with well-controlled tail latencies.
Up to 512 total TPU cores and 4 TB of total memory in a TPU Pod (64 units).
TPUv3 is the first **liquid-cooled accelerator** in Google's data center (here there is no fresh air to cool it down, used fresh liquid). 2.5x faster than TPUv2. Such supercomputing-class computational power supports new ML capabilities (e.g., AutoML), and rapid neural architecture search. The v3 TPU Pod provides a maximum configuration of 256 devices for a total 2048 TPU v3 cores, 100 petaflops and 32 TB of TPU memory.
TPUv4 announced June 2021, used to support Google services (not yet available as a cloud service).One v4 TPU pod includes 4096 devices: About 2.7x faster than TPUv3 and same computing capacity as 10 millions of laptops.

3.3.3 Field-Programmable Gate Array (FPGA)

Array of logic gates that can be programmed ("configured") in the field, i.e.,by the user of the device as opposed to the people who designed it.
Array of carefully designed, and interconnected digital subcircuits, that efficiently implement common functions offering very high levels of flexibility. The digital subcircuits are called configurable logic blocks (CLBs).
VHDL and Verilog are hardware description languages (HDLs), that allow to "describe" hardware. HDL code is more like a schematic that uses text to introduce components and create interconnections.
Microsoft deployed FPGAs inside its Datacenters.

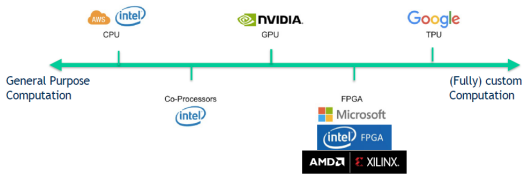


Figure 9: Overview

3.3.4 Advantages and Disadvantages

CPU

- **Advantages**:Easy to be programmed and support any programming framework.
  Fast design space exploration and run your applications.
- **Disadvantages**: Most suited for simple AI models that do not take long to train and for small models with small training set.

GPU

- **Advantages**:Ideal for applications in which data need to be processed in parallel like the pixels of images or videos.
- **Disadvantages**: Programmed in languages like CUDA and OpenCL and therefore provide limited flexibility compared to CPUs.

TPU

- **Advantages**:Very fast at performing dense vector and matrix computations and are specialized on running very fast program based on Tensorflow.
- **Disadvantages**: For applications and models based on the TensorFlow. Lower flexibility compared to CPUs and GPUs.

FPGA

- **Advantages**:Higher performance, lower cost and lower power consumption compared to other options like CPUs and GPU.
- **Disadvantages**: Programmed using OpenCL and High-level Synthesis (HLS).
  Limited flexibility compared to other platforms.

4 Storage

Nowadays machines generate data at an unprecedented rate

- Disks and Flash SSDs are the building blocks of today's WSC storage systems.
- These devices are connected to the data-center network and managed by sophisticated distributed systems

Examples:

- Direct Attached Storage (DAS)
- Network Attached Storage (NAS)
- Storage Area Networks (SAN)
- RAID controllers

4.1 Hard Disk Drives

A hard disk drive (HDD) is a data storage using rotating disks (platters) coated with magnetic material.
Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially.
An HDD consists of one or more rigid ("hard") rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.
The **Sector** is where you read and/or write; The organization of the data is in order to minimize the movement of the head (that increase the seek time): place the data that are related near! If the head continues to go forward and back increase the Latency!

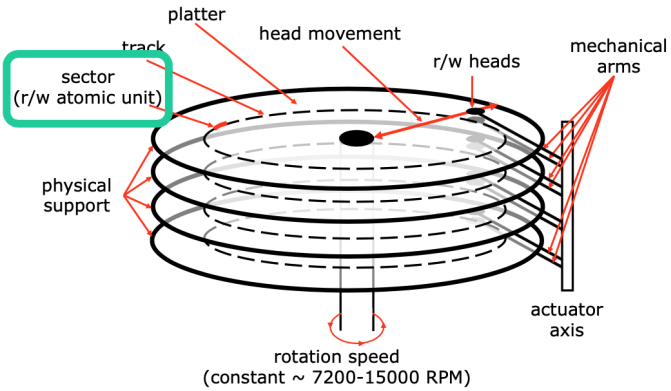


Figure 10: Hard Disk

4.1.1 Read Write heads, basic characteristic

- float on a film of air (tens of nanometers) above the platters
- one head for each magnetic platter
- cylinder: set of tracks with the same radius
- **seek time**: time required to reach the track that contains the data, 3÷14 ms

4.1.2 Other characteristics

- Diameter: about 9 cm (3,5÷2.5 in) - two surfaces
- Rotation speed: 7200÷15000 RPM round per minute
- Track density: 16,000 TPI (Track Per Inch)
- Sectors: 512 Byte (usually), but might be different (are numbered sequentially, have a header and an error correction code)
- Heads: can be parked close to the center or to the outer diameter (mobile drives)
- Disk buffer cache: embedded memory in a hard disk drive that has the function of a buffer between the disk and the computer (several MB)

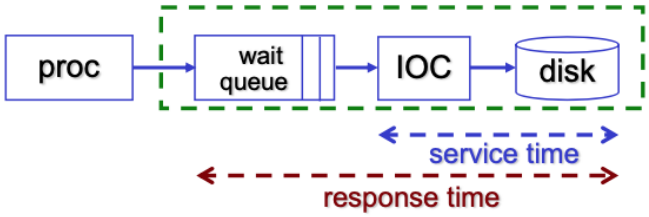


Figure 11: HDD service and response time

Service Time
s_{disk}: seek time + rotational latency + data transfer time + controller overhead

- **seek** time: head movement time (≈ ms), is a function of the number of cylinders traversed
- **latency** time: time to wait for the sector (≈ ms, $\frac{1}{2}$ round)
- **transfer** time: is a function of rotation speed, storing density, cylinder position (≈ MB/sec)
- **controller overhead**: buffer management (data transfer) and interrupt sending time

no queue, average time to serve a single I/O request.
Response Time
service time + queue time, average time to serve an I/O request in working conditions

4.2 Solid-state Storage Device

No mechanical or moving parts like HDD
Built out of transistors (like memory and processors)
Retain information despite power loss unlike typical RAM
A controller is included in the device with one or more solid state memory components
It uses traditional hard disk drive (HDD) interfaces (protocol and physical connectors) and form factors
Higher performance than HDD
It stores Bit, based on Transistors (which compose them):

- Single-level cell (SLC) : single bit per cell
- Multi-level cell (MLC) cell : two bits per cell
- Triple-level cell (TLC) : three bits per cell
- QLC, PLC...

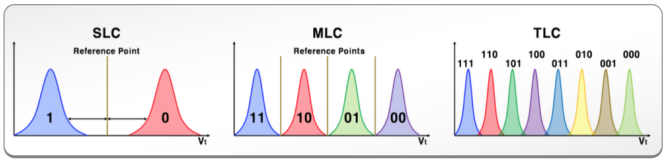


Figure 12: SSD

4.2.1 Internal Organization

NAND flash is organized into Pages and Blocks:
A **page** contains multiple logical block (e.g. 512B-4KB) addresses (LBAs), a **block** typically consists of multiple pages (e.g., 64) with a total capacity of around 128-256KB; Block/Page terminology in the SSD context can clash with previous use.
Blocks (or Erase Block): smallest unit that can be erased. It consists of multiple pages and can be cleaned using the ERASE command.
Pages : smallest unit that can be read/written. It is a sub-unit of an erase block and consists of the number of bytes which can be read/written in a single operations through the READ or PROGRAM commands.
Pages can be in three states:

- Dirty (or INVALID): they contain data, but this data is no longer in use (or never used)

- Empty (or EREASED): they do not contain data
- In use (or VALID): the page contains data that can be actually read

Only empty pages can be written
Only dirty pages can be erased, but this must be done at the block level (all the pages in the block must be dirty or empty)
It is meaningful to read only pages in the "in use" ("valid") state
If no empty page exists, some dirty page must be erased:

- If no block containing just dirty or empty pages exists, then special procedures should be followed to gather empty pages over the disk
- To erase the value in flash memory the original voltage must be reset to neutral before a new voltage can be applied, known as write amplification

Remark: we can write and read a single page of data from a SSD but we have to delete an entire block to release it

WRITE AMPLIFICATION: the actual amount of information physically written to the storage media is a multiple of the logical amount intended to be written.

Wear out: breakdown of the oxide layer within the floating-gate transistors of NAND flash memory.
The erasing process hits the flash cell with a relatively large charge of electrical energy.
Each time a block is erased:

- the large electrical charge actually degrades the silicon material
- after enough write-erase cycles, the electrical properties of the flash cell begin to break down and the cell becomes unreliable

Flash Translation Layer = layer that matches the file system and the SSD.

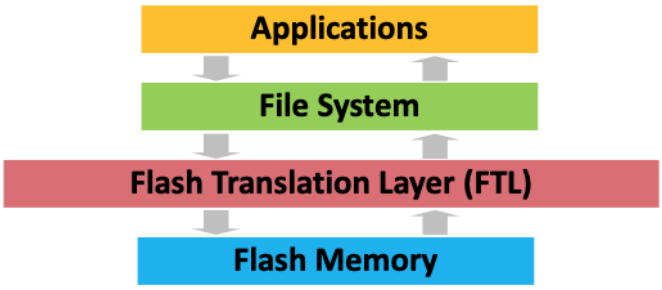


Figure 13: Flash Translation Layer

Direct mapping between Logical to Physical pages is not feasible
FTL is an SSD component that make the SSD "look as HDD":

1. Data Allocation and Address translation
   - Efficient to reduce Write Amplification effects
   - Program pages within an erased block in order (from low to high pages): Log-Structured FTL
2. Garbage collection: Reuse of pages with old data (Dirty/Invalid)
3. Wear leveling: FTL should try to spread writes across the blocks of the flash ensuring that all of the blocks of the device wear out at roughly the same time

Garbage collection
When an existing page is updated à old data becomes obsolete!
Old version of data are called garbage and (sooner or later) garbage pages must be reclaimed for new writes to take place.
Garbage Collection is the process of finding garbage blocks and reclaiming them: Simple process for fully garbage blocks or more complex for partial cases. I.e. Basic steps:

1. Find a suitable partial block

2. Copy non-garbage pages
3. Reclaim (erase) the entire block for writing

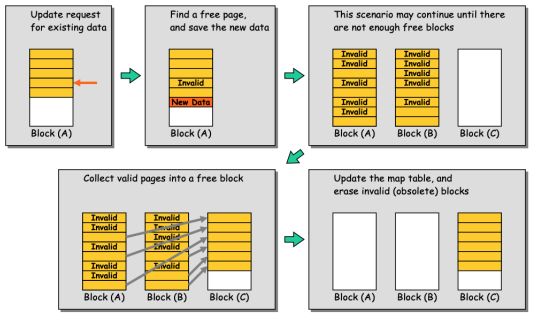


Figure 14: Garbage collection (Example)

Garbage collection is expensive: Require reading and rewriting of live data and the ideal garbage collection is reclamation of a block that consists of only dead pages.
The **8cost** of Garbage Collection depends on the amount of data blocks that have to be migrated.
Solutions to alleviate the problem:

- Overprovision the device by adding extra flash capacity: Cleaning can be delayed
- Run the Garbage Collection in the background using less busy periods for the disk

When performing background GC the SSD assumes to know which pages are invalid
Problem: most file systems don't actually delete data (E.g., on Linux, the "delete" function is unlink() and it removes the file meta-data, but not the file itself)

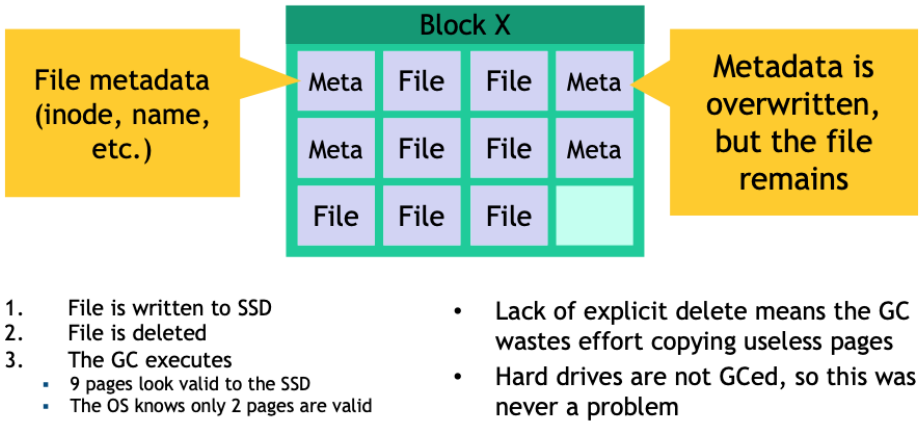


Figure 15: Garbage collection (Example)

New SATA command TRIM (SCSI – UNMAP): The OS tells the SSD that specific LBAs are invalid and may be GCed.
OS support for TRIM (Win 7, OSX Snow Leopard, Linux 2.6.33, Android 4.3).

The size of page-level mapping table is too large: With a 1TB SSD with a 4byte entry per 4KB page, 1GB of is needed for mapping.
Some approaches to reduce the costs of mapping:

- Block-based mapping: Coarser grain approach.
  Mapping at block granularity, to reduce the size of a mapping table. Small write problem: the FTL must read a large amount of live data from the old block and copy them into a new one.

- Hybrid mapping : Multiple tables.
  FTL maintains two tables:
  - Log blocks: page mapped
  - Data blocks: block-mapped

  When looking for a particular logical block, the FTL will consult the page mapping table and block mapping table in order

- Page mapping plus caching : Exploiting Data Locality.
  Basic idea is to cache the active part of the page-mapped FTL, f a given workload only accesses a small set of pages, the translations of those pages will be stored in the FTL memory. High performance without high memory cost if the cache can contain the necessary working set. Cache miss overhead. exists

Wear Leveling
Erase/Write cycle is limited in Flash memory:
- Skewness in the EW cycles shortens the life of the SSD
- All blocks should wear out at roughly the same time
Log-Structured approach and garbage collection helps in spreading writes. However, a block may consist of cold data

- the FTL must periodically read all the live data out of such blocks and re-write it elsewhere
- Wear leveling increases the write amplification of the SSD and decreases performance (Simple Policy: Each Flash Block has EW cycle counter, Maintain —Max(EW cycle) – Min(EW cycle)—< e)

4.2.2 SSD summary

- They cost more than the conventional HDD
- Flash memory can be written only a limited number of times (wear):
  have a shorter lifetime
  error correcting codes
  over-provisioning (add some spare capacity)
- Different read/write speed
  Write amplification
- Write performance degrades of one order of magnitude after the first writing
- Often the controller become the real bottleneck to the transfer rate
- SSD are not affected by data-locality and must not be defragmented (actually, defragmentation may damage the disks)
- Flash Translation Layer is one of the key components
  - Data Allocation
  - Address Translation
  - Garbage Collection
  - Wear Leveling

4.3 HDD vs SSD

Unrecoverable Bit Error Ratio (UBER): A metric for the rate of occurrence of data errors, equal to the number of data errors per bits read
Endurance rating : Terabytes Written (TBW is the total amount of data that can be written into an SSD before it is likely to fail). The number of terabytes that may be written to the SSD while still meeting the requirements

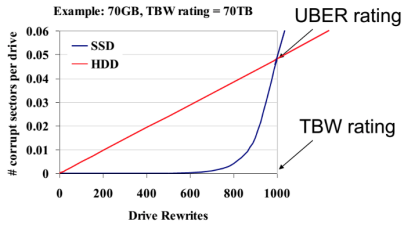


Figure 16: Garbage collection (Example)

Memory cells can accept data recording between 3,000 and 100,000 during its lifetime: once the limit value is exceeded, the cell "forgets" any new data

A typical TBW for a 250 GB SSD is between 60 and 150 Terabytes of data written to the drive; This means that in order to overcome, for example, a TBW of 70 Terabytes, a user should write 190 GB every day for a year or fill his SSD on a daily basis for two thirds with new files for a whole year

It is difficult to comment on the duration of SSDs: Dell, in an old study (2011), spoke of an estimated duration between three months and ten years explaining, however, that there are so many factors (temperature and workload) that may depend on the life of an SSD that is very difficult to make predictions.

4.4 Hybrid solution

HDD + SSD

- Some large storage servers use SSD as a cache for several HDD. Some mainboards of the latest generation have the same feature: they combine a small SSD with a large HDD to have a faster disk.
- Some HDD manufacturers produce Solid State Hybrid Disks (SSHD) that combine a small SDD with a large HDD in a single unit.

4.5 Storage system

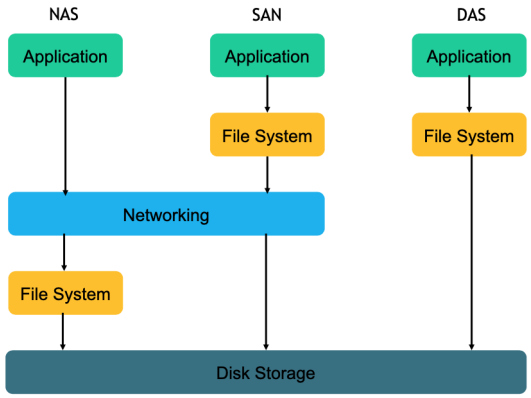


Figure 17: NAS SAN DAS

Definition 3. *A* **Direct Attached Storage** *(DAS) is a storage system directly attached to a server or workstation. They are visible as disks/volumes by the client OS.*

Definition 4. *A* **Network Attached Storage** *(NAS) is a computer connected to a network that provides only file-based data storage services (e.g., FTP, Network File System and SAMBA) to other devices on the network and is visible as File Server to the client OS*

Definition 5. Storage Area Networks *(SAN) are remote storage units that are connected to a server using a specific networking technology (e.g., Fiber Channel) and are visible as disks/volumes by the client OS Direct Attached Storage (DAS) is a storage system directly*

4.5.1 DAS

DAS is a storage system directly attached to a server or workstation
The term is used to differentiate non-networked storage from SAN and NAS (that will be described later).
Main features:

- limited scalability (for ex. if I want 100 server connected, I have to multiply x100 the capacity)
- complex management
- to read files in other machines, the "file sharing" protocol of the OS must be used (on application laver, if I have one Windows and one Linux they must be able to communicate)

Internal and external:

- DAS does not necessary mean "internal drives" (it could have an external hard disk)
- All the external disks, connected with a point-to-point protocol to a PC can be considered as DAS

4.5.2 NAS

A NAS unit is a computer connected to a network that provides only file-based data storage services to other devices on the network.
NAS systems contain one or more hard disks, often organized into logical redundant storage containers or RAID.
Provide file-access services to the hosts connected to a TCP/IP network though Networked File Systems/SAMBA.
Each NAS element has its own IP address.
Good scalability (incrementing the devices in each NAS element or incrementing the number of NAS elements).

NAS vs DAS
The key differences between direct-attached storage (DAS) and NAS are:

- DAS is simply an extension of an existing server and is not necessarily networked
- NAS is designed as an easy and self-contained solution for sharing files over the network

Comparing NAS with local (non-networked) DAS, the *performance* of NAS depends mainly on the speed of and congestion on the network

4.5.3 SAN

Storage Area Networks, are remote storage units that are connected to a PC/server using a specific networking technology.
SANs have a special network devoted to the accesses to storage devices.
Two distinct networks (one TCP/IP and one dedicated network, e.g., Fber Channer).
High scalability (simply increasing the storage devices connected to the SAN network).

NAS vs SAN

- NAS provides both storage and a file system
- This is often contrasted with SAN which provides only block-based storage and leaves file system concerns on the "client" side
- One way to loosely conceptualize the difference between a NAS and a SAN is that:
  - NAS appears to the client OS (operating system) as a file server (the client can map network drives to shares on that server)
  - a disk available through a SAN still appears to the client OS as a disk: it will be visible in the disks and volumes management utilities (along with client's local disks), and available to be formatted with a file system
- NAS is used for low-volume access to a large amount of storage by many users
- SAN is the solution for petabytes (1012) of storage and multiple, simultaneous access to files, such as streaming audio/video

	Application Domain	Advantages	Disadvantages
DAS	• Budget constraints • Simple storage solutions	• Easy setup • Low cost • High performance	• Limited accessibility • Limited scalability • No central management and backup
NAS	• File storage and sharing • Big Data	• Scalability • Greater accessibility • Performance	• Increased LAN traffic • Performance limitations • Security and reliability
SAN	• DBMS • Virtualized environments	• Improved performance • Greater scalability • Improved availability	• Costs • Complex setup and maintenance

Figure 18: NAS vs SAN vs DAS

5 Networking

Communication equipment allows network interconnections among the devices.
They can be:

- Hubs
- Routers
- DNS or DHCP servers
- Load balancers
- Technology switches
- Firewalls

The performance of servers increases over time, the demand for inter-server bandwidth naturally increases as well!
We can double the aggregate compute capacity or the aggregate storage simply by doubling the number of compute or storage elements, but how?
Networking has no straightforward horizontal scaling solution.
Doubling leaf bandwidth is easy: with twice as many servers, we'll have twice as many network ports and thus twice as much bandwidth.

But if we assume that every server needs to talk to every other server, we need to deal with **bisection bandwidth**.

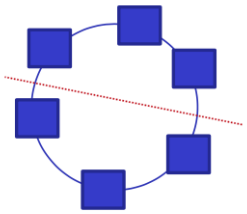


Figure 19: bisection bandwidth

Definition 6. *The bandwidth across the narrowest line that equally divides the cluster into two parts. Characterizes network capacity since randomly communicating processors must send data across the "middle" of the network*

5.1 Architecture

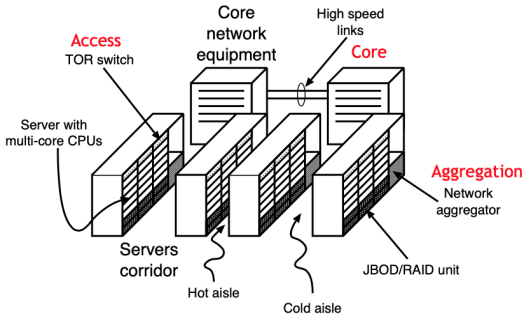


Figure 20: Data-center network architectures

Switches at the access layer can be put into two positions:

- *Top-Of-the-Rack* (TOR): **Access** switches are put at the top of each rack. The number of cables is limited. The number of ports per switch is also limited (lower costs). However, the scalability is also limited.
- *End-Of-the-Line* (EOL): Switches are positioned one per corridor, at the end of a line of rack. **Aggregation** switches must have a larger number of ports, and longer cables are required (higher costs). However, the system can scale to have a larger number of machines.

Three layer architecture configures the network in three different layers:

- core
- aggregation
- access

Bandwidth can be increased by increasing the switches at the core and aggregation layers, and by using routing protocols such as Equal Cost Multiple Path (ECMP) that equally shares the traffic among different routes.

This solution is very simple, but can be very expensive in large data-centers since:
• Upper layers require faster network equipments. (For example: 1 GB Ethernet at the access layer, 10 GB Ethernet at the aggregation layer, 25 GB Optical connections at the core layer)
• The cost in term of acquisition and energy consumption can be very high
Another benefit of SANs: a way to tackle network scalability, offload some traffic to a special-purpose network connecting servers to storage units

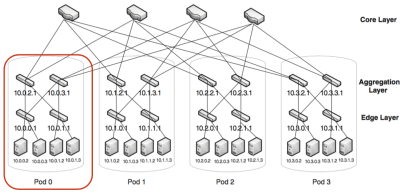


Figure 21: Fat-tree topologies

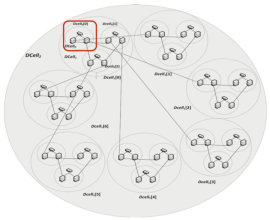


Figure 22: D-Cell topology

Fat-tree topologies use a larger number of slower speed switches and connections. In particular, nodes are divided into pods characterized by the same number of nodes and switches.

D-Cell topology, defines the network in recursive way. Cells are organized in levels. Switches connects nodes at the lower level.Some nodes belong to different cells: they perform routing among them to create a higher level cell.

5.1.1 High Performance Clusters

High-Performance Computing (HPC) supercomputer clusters often have a much lower ratio of computation to network bandwidth.

Applications (e.g., weather simulations) distribute their data across RAM in all nodes, and nodes need to update neighboring nodes after performing relatively few floating-point computations.
Traditional HPC systems have used proprietary interconnects with leading-edge link bandwidths, much lower latencies, and some form of a global address space (where the network is integrated with CPU caches and virtual addresses)

High **throughputs** but much more expensive solutions.

5.2 Network to support Virtualization

Connection endpoints (i.e., IP address/port combinations) can move from one physical machine to another
Typical networking hardware as well as network management software don't anticipate such moves and in fact often explicitly assume that they're not possible
(All machines in a given rack have IP addresses in a common subnet, which simplifies administration and minimizes the number of required forwarding table entries routing tables)
Solution: the cluster manager that decides the placement of computations also updates the network state through programmable Network control planes (Software Defined Networks)

5.3 The interplay of storage and networking technology

The success of WSC distributed storage systems can be partially attributed to the evolution of data center networking fabrics:*disk locality is no longer relevant in intra-data center computations.*
This observation enables:

- simplifications in the design of distributed disk-based storage systems
- utilization improvements

since any disk byte in a WSC facility can, in principle, be utilized by any task regardless of their relative locality

5.4 Balanced design

Computer architects are trained to solve the problem of finding the right combination of performance and capacity from the various building blocks that make up a WSC

The right building blocks are apparent only when one considers the entire WSC system.
It is important to characterize the kinds of workloads that will execute on the system with respect to their consumption of various resources.
Keeping in mind three important considerations:

1. Smart programmers may be able to restructure their algorithms to better match a more inexpensive design alternative
2. The most cost-efficient and balanced configuration for the hardware may be a match with the combined resource requirements of multiple workloads
3. Fungible resources tend to be more efficiently used

System balance: **Storage Hierarchy**

Example

We assume a system with 5,000 servers, each with 256 GB of DRAM, one 4 TB SSD, and eight 10 TB disk drives.
Each group of 40 servers is connected through a 40-Gbps link to a rack-level switch (TOR)
Each rack has an additional 10-Gbps uplink bandwidth per machine for connecting the rack to the cluster-level switch (AGGREGATION)
Network latency numbers assume a TCP/IP transport, and networking bandwidth values assume that each server is using its fair share of the available cluster-level bandwidth.
For disks, typical commodity disk drive (SA T A) latencies and transfer rates are considered.

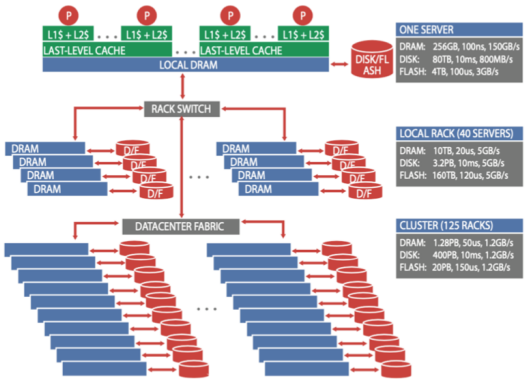


Figure 23: Storage Hierarchy

A large application that requires servers in many racks to operate must deal effectively with large discrepancies in latency, bandwidth, and capacity.

These discrepancies are much larger than those seen on a single machine, making it more difficult to program a WSC:

- A key challenge for architects of WSCs is to smooth out these discrepancies in a cost-efficient manner
- A key challenge for software architects is to build SW infrastructure and services that hide this complexity

Three specific comments about SSDs:

1. Much faster than HDDs
2. Demand a high bandwidth
3. In the worst case, writes to flash can be several orders of magnitude slower than reads

6 Building and Infrastructures

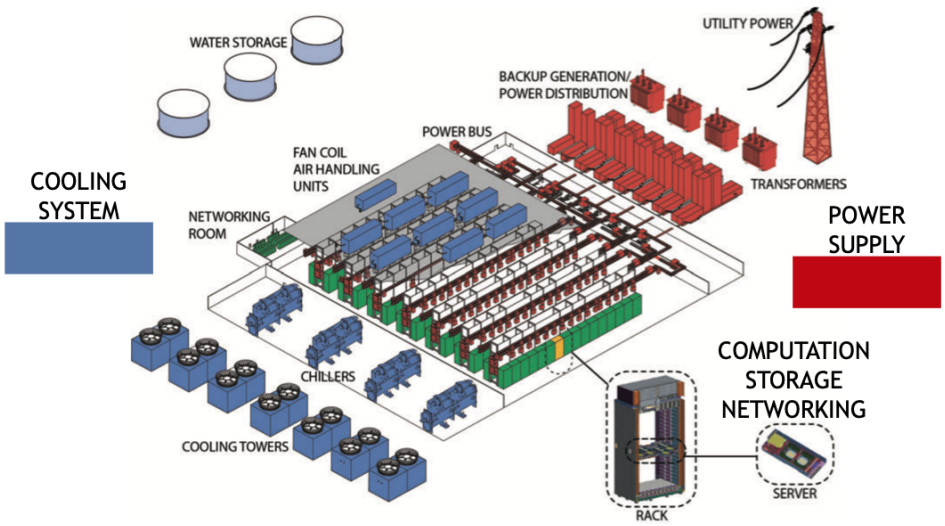


Figure 24: Main components of a typical data center

WSC has not just computation, storage and networking; it has other important components related to power delivery, cooling, and building infrastructure that also need to be considered.

6.1 Power System

In order to protect against power failure, battery and diesel generators are used to backup the external supply.

The UPS typically combines three functions in one system:

1. contains a transfer switch that chooses the active power input (either utility power or generator power)
2. contains some form of energy storage (electrical, chemical, or mechanical) to bridge the time between the utility failure and the availability of generator power
3. conditions the incoming power feed, removing voltage spikes or sags, or harmonic distortions in the AC feed

6.2 Cooling System

IT equipment generates a lot of heat: the cooling system is usually a very expensive component of the datacenter, and it is composed by coolers, heat-exchangers and cold water tanks.

6.2.1 Open-Loop

The simplest topology is fresh air cooling (or air economization)— essentially, opening the windows. This is a single «open-loop» system.

Definition 7. *Free cooling, i.e., open-loop, refers to the use of cold outside air to either help the production of chilled water or directly cool servers. It is not completely free in the sense of zero cost, but it involves very low-energy costs compared to chillers.*

6.2.2 Closed-Loop

Definition 8. *Closed-loop systems come in many forms, the most common being the air circuit on the data center floor.*

The goal is to isolate and remove heat from the servers and transport it to a heat exchanger.
Cold air flows to the servers, heats up, and eventually reaches a heat exchanger to cool it down again for the next cycle through the servers.

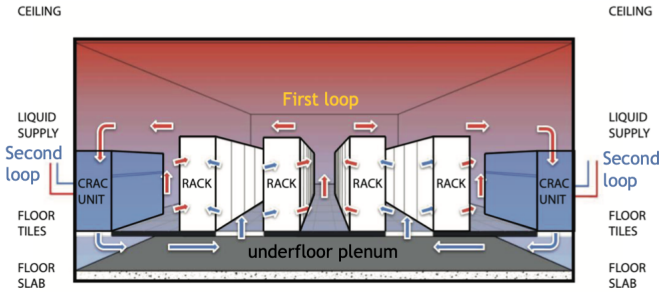


Figure 25: Closed-loop with two loop

Closed-loop with two loops

- The airflow through the underfloor plenum, the racks, and back to the CRAC (a 1960s term for computer room air conditioning) defines the primary air circuit, i.e., the **first loop**.
- The **second loop** (the liquid supply inside the CRACs units) leads directly from the CRAC to external heat exchangers (typically placed on the building roof) that discharge the heat to the environment.

A three-loop system commonly used in large-scale data center

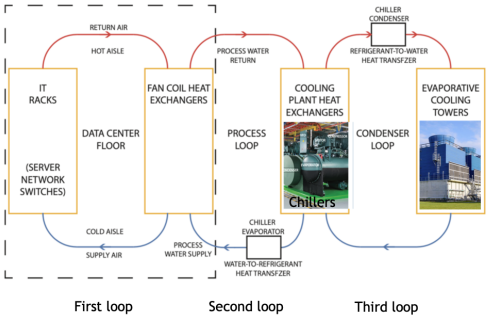


Figure 26: Closed-loop with three loop

A water-cooled **chiller** can be thought of as a water-cooled air conditioner.

Cooling towers cool a water stream by evaporating a portion of it into the atmosphere. They do not work as well in very cold climates because they need additional mechanisms to prevent ice formation

6.2.3 Comparison

Each topology presents tradeoffs in complexity, efficiency, and cost:

- Fresh air cooling can be very efficient but does not work in all climates, requires filtering of airborne particulates, and can introduce complex control problems.
- Two-loop systems are easy to implement, relatively inexpensive to construct, and offer isolation from external contamination, but typically have lower operational efficiency.
- A three-loop system is the most expensive to construct and has moderately complex controls, but offers contaminant protection and good efficiency.

6.2.4 In-rack cooling

In-rack cooler adds an air-to-water heat exchanger at the back of a rack so the hot air exiting the servers immediately flows over coils cooled by water, essentially reducing the path between server exhaust and CRAC input

6.2.5 In-row cooling

In-row cooling works like in-rack cooling except the cooling coils are not in the rack, but adjacent to the rack.

6.2.6 Liquid cooling

We can directly cool server components using cold plates, i.e., local liquid-cooled heat sinks:

- Impractical to cool all compute components with cold plates.
- Components with the highest power dissipation are targeted for liquid cooling while other components are air-cooled.

The liquid circulating through the heat sinks transports the heat to a liquid-to-air or liquid-to-liquid heat exchanger that can be placed close to the tray or rack, or be part of the data center building (such as a cooling tower).

Container-based

Container-based data centers go one step beyond in-row cooling by placing the server racks inside a container (typically 6 to 12 mt long) and integrating heat exchange and power distribution into the container as well.

6.3 Power consumption

Data-center power consumption is an issue, since it can reach several MWs.
Cooling usually requires about half the energy required by the IT equipment (servers + network + disks).
Energy transformation creates also a large amount of energy wasted for running a datacenter.

- DCs consume 3% of global electricity supply (416.2 TWh ¿ UK's 300 TWh).
- DCs produce 2% of total greenhouse gas emissions (same as worldwide air traffic pre-pandemic).
- DCs produce as much CO2 as The Netherlands or Argentina.

Amortized Cost	Component	Sub-Components
~45%	Servers	CPU, memory, disk
~25%	Infrastructure	UPS, cooling, power distribution
~15%	Power draw	Electrical utility costs
~15%	Network	Switches, links, transit

Figure 27: power consumption

Definition 9. Power usage effectiveness *(PUE) is the ratio of the total amount of energy used by a DC facility to the energy delivered to the computing equipment*

$\text{PUE} = \frac{TotalFacilityPower}{ITEquipmentpower}$

Total facility power = covers IT systems (servers, network, storage) + other equipment (cooling, UPS, switch gear, generators, lights, fans, etc.)
Data Center infrastructure Efficiency (DCiE): PUE inverse

6.4 Tiers

Data-center availability is defined by in four different tier level. Each one has its own requirements.

Tier Level	Requirements
1	• Single non-redundant distribution path serving the IT equipment • Non-redundant capacity components • Basic site infrastructure with expected availability of 99.671%
2	• Meets or exceeds all Tier 1 requirements • Redundant site infrastructure capacity components with expected availability of 99.741%
3	• Meets or exceeds all Tier 2 requirements • Multiple independent distribution paths serving the IT equipment • All IT equipment must be dual-powered and fully compatible with the topology of a site's architecture • Concurrently maintainable site infrastructure with expected availability of 99.982%
4	• Meets or exceeds all Tier 3 requirements • All cooling equipment is independently dual-powered, including chillers and heating, ventilating and air-conditioning (HVAC) systems • Fault-tolerant site infrastructure with electrical power storage and distribution facilities with expected availability of 99.995%

Figure 28: data-center tiers

7 Virtualization

7.1 Cloud Computing

Definition 10. *A coherent, large-scale, publicly accessible collection of computing, storage, and networking resources. Available via Web service calls through the Internet. Short- or long-term access on a pay-per-use basis*

How is Cloud implemented? Virtualization

Hardware resources (CPU, RAM, ecc...) are partitioned and shared among multiple *virtual machines* (VMs).
The *virtual machine monitor* (VMM) governs the access to the physical resources among running VMs.
In this way are provided performance, isolation and security.

7.2 Virtual Machines

Definition 11. *A* **Machine** *is an execution environment capable of running a program.*

What's the difference between a physical machine and a virtual machine?

Start from the **computer architecture**:

- Sets of instructions characterized by the levels at which are considered
- OS creates new instructions for programs to access devices/hw
- the set of instructions that a program can use might be structured at different levels
  (We usually program the software part, exploiting the hardware.)

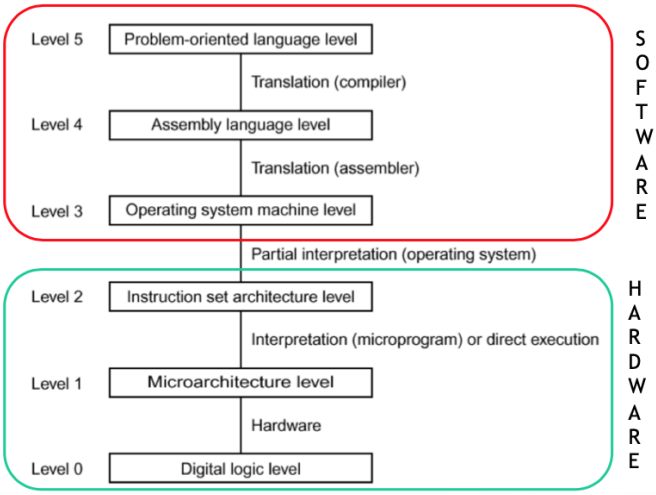


Figure 29: machine levels

The ISA (Instruction Set Architecture) corresponds to **Level 2** in the layered execution model.
ISA marks the division between hardware and software.

- User ISA: aspects of the ISA that are visible to an application program (sum, multiplication,..., logical operations, branches, etc...). When application interacts with the HW, User ISA is used.
- System ISA: aspects visible to supervisor software (i.e., the OS) which is responsible for managing hardware resources (e.g., hide the complexity of CPUs, define how app access memory, communicate with the HW). When the OS interacts with the HW (Drivers, MM, Sched.), System ISA is used.

The ABI (Application binary interface) corresponds to **Level 3** in the layered execution model.
Provides a program (via the OS): the hardware resources and services available in a system.

- User ISA: aspects of the ISA that are visible to an application program (sum, multiplication,..., logical operations, branches, etc...).
- System Calls: calls that allow programs to interact with shared hardware resources indirectly by OS

Definition 12. *A* **Virtual Machine** *(VM) is a logical abstraction able to provide a virtualized execution environment.*

More specifically, a VM:

- (provides) Identical Software behavior

- (consists in a) Combination of physical machine and virtualizing software
- (may appear as) Different resources than physical machine
- (may result in) Different level of performance

Its **tasks** are:
• To map virtual resources or states to corresponding physical ones
• To use physical machine instructions/calls to execute the virtual ones.

Two **types** of Virtual Machines:

1. System VMs
2. Process VMs

7.2.1 System VMs

The VMM supports the levels 0-2 of the architecture.

Provide a *complete system environment* that can support an operating system (potentially with many user processes)
It provides *operating system* running in it access to underlying hardware resources (networking, I/O, a GUI).

The VM supports the operating system as long as the system environment is alive.
Virtualizing software placed between hardware and software (**emulates the ISA interface** seen by software)

The virtualization software is called VMM (**Virtual Machine Monitor**)

The VMM can provide its functionality either working **directly** on the hardware, or running **on another OS**.

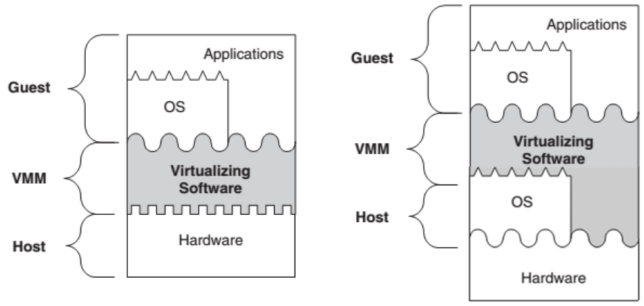


Figure 30: System Virtual Machine

7.2.2 Process VMs

The runtime software supports the levels 0-3 of the architecture

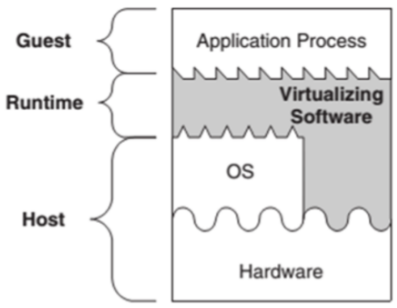


Figure 31: Process Virtual Machine

Able to support an individual process
The virtualizing software is placed at the ABI interface, on top of the OS/hardware combination.
The virtualizing software emulates both user-level instructions and operating system calls.
The virtualization software is usually called **Runtime Software**.

Definition 13. Host *the underlying platform supporting the environment/system*

Definition 14. Guest *the software that runs in the VM environment as the guest.*

7.2.3 Process and System VMs

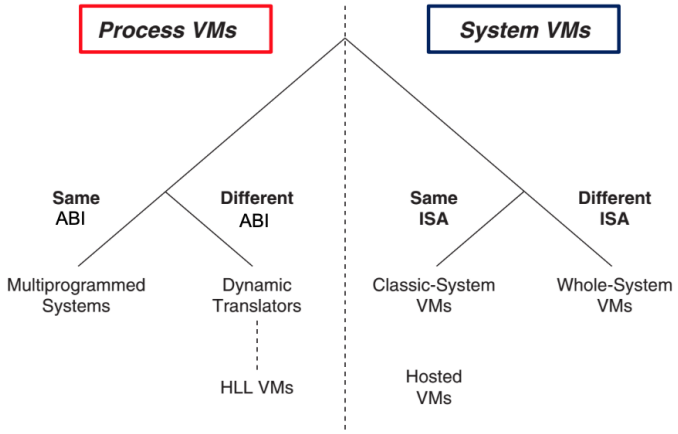


Figure 32: Different types of Virtualization

Multiprogrammed systems

Same ABI / OS:
- VM formed by OS call interface + user ISA
- Common approach of all modern OS for multi user support (Task/Process Manager)
- Each user process is given:
  - the illusion of having a complete machine to itself.
  - its own address space and is given access to a file structure
- OS timeshares and manages HW resources to permit this
- **Arguable is a real VM**

Different ISA / ABI

Definition 15. Emulation *refers to those software technologies developed to allow an application (or OS) to run in an environment different from that originally intended.*

It is required when the VMs have a different ISA / ABI from the architecture where they are running on.
Example: Obsolete hardware emulators or other architectures emulators.

An **emulator** reads all the bytes included in the memory of system it is going to reproduce.
Interpretation:

- Interpreter program fetches, decodes, emulate the execution of individual source instruction
- Can be a slow process
- E.g., the Space Invaders (The system was based on the Intel 8080 CPU, with 8KB ROM, 1KB RAM, 7KB VRAM, plus a shift-register to speed up scrolling.)

High-Level Language VM (HLL VMs)

Goal: to have an isolated execution environment for each application (or for different instances of the same application).
VM Task:

- Translates application byte code to OS-specific executable
- Minimize HW/OS-specific feature for platform independence

Applications run normally but: Sandboxed, Can "migrate", Are not conflicting one another and Are not "installed" in a strict sense
Example: Java Virtual Machine (Java can work on every architecture for which an interpreter, called the Java Runtime Environment (JRE), exists)

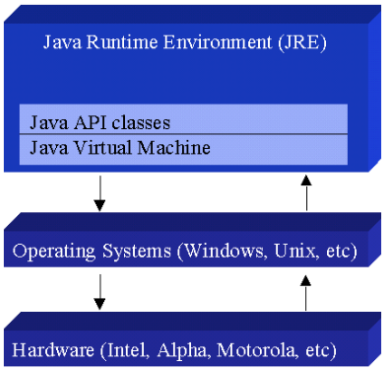


Figure 33: HLL VMs

Same ISA

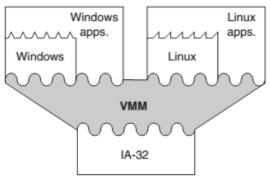


Figure 34: Classic system VMs

The VMM is on bare hardware, and virtual machines fit on top.
The VMM can intercept guest OS's interaction with hardware resources.
The most efficient VM architecture (HW executes the same instructions of VMs)
Two different OSs on the same on the same HW

Hosted VM

virtualizing software is on top of an existing host operating system.

Whole-system VMs

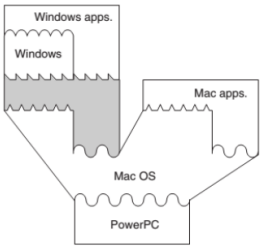


Figure 35: Whole system VMs

Virtualize all software: ISAs are different so both application and OS code require emulation, e.g., via binary translation. (no native execution possible)
Usually implement the VMM and guest software on top of a conventional host OS running on the hardware.
the VM software must emulate the entire hardware environment and all the guest ISA operations to equivalent OS call to the Host.
Example: MS Word and Windows running on a Power PC (not x86 family)

7.3 Implementation

Given a typical layered architecture of a system by *adding layers* between execution stack layers.
Depending on where the new layer is placed, we obtain different types of Virtualization.

7.3.1 Hardware-level virtualization

Virtualization layer is placed between hardware and OS.
The interface seen by OS and application might be different from the physical one

7.3.2 Application-level virtualization

A virtualization layer is placed between the OS and some applications (E.g.: JVM (Java Virtual Machine))
Provides the same interface to the applications.
Applications run in their environment, independently from OS

7.3.3 System-level virtualization

The virtualization layer provides the interface of a physical machine to a secondary OS and a set of application running in it, allowing them to run on top of an existing OS.
Placed between the system's OS and other OS (E.g.: VMware Wks/Player, VirtualBox)
Enable several OSs to run on a single HW

7.4 Properties

- **Partitioning**:
  - Execution of multiple OSs on a single physical machine
  - Partitioning of resources between the different VMs
- **Isolation**:
  - Fault tolerance and security (as at the hardware level)
  - Advanced resource control to guarantee performance (managed by the hypervisor)
- **Encapsulation**:
  - The entire state of a VM can be saved in a file (e.g., freeze and restart the execution)
  - Because a VM is a file, can be copied/moved as a file
- **HW-independence**:
  - Provisioning/migration of a given VM on a given physical server .

7.5 Virtual Machine Manager (VMM)

Definition 16. *An application that:*

- *manages the virtual machines*
- *mediates access to the hardware resources on the physical host system*
- *intercepts and handles any privileged or protected instructions issued by the virtual machines*

This type of virtualization typically runs virtual machines whose operating system, libraries, and utilities have been compiled for the same type of processor and instruction set as the physical machine on which the virtual systems are running. **Crucial to support Cloud Computing**

Three terms are used to identify the same thing, some authors gives slightly different meanings:

- *Virtual Machine Monitor*: the virtualization software
- *Hypervisor*: a virtualization software that runs directly on the hardware
- *Virtual Machine Manager*: a VMM or Hypervisor that is also used to create, configure and maintain virtualized resources. It provides a user-friendly interface to the underlying virtualization software

Hyperversor types

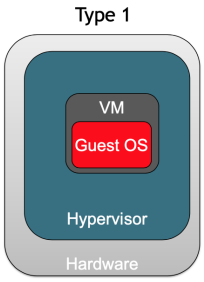


Figure 36: hyp type1

Bare-metal or type 1 hypervisors: akes direct control of the Hardware

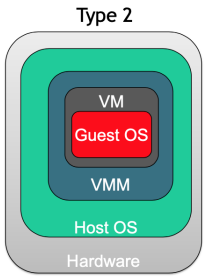


Figure 37: hyp type2

Hosted or type 2 hypervisors (VMM): reside within a host operating system and leverage code of the host OS (E.g. CPU scheduling, memory management)

7.5.1 Type 1 hypervisor

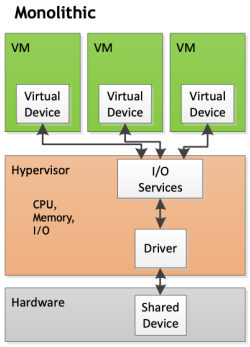


Figure 38: monolithic

Device drivers run within the hypervisor

- Better performance
- Better performance isolation

Can run only on hardware for which the hypervisor has drivers

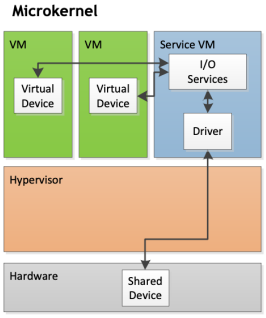


Figure 39: microkernel

Device drivers run within a service VM

- Smaller hypervisor
- Leverages driver ecosystem of an existing OS
- Can use 3rd party driver (even if not always easy, recompiling might be required)

Hacks

Sometimes Type I Hypervisor can be used to provide hacks to some OSs. Two well-known examples on PC hardware:

1. Windows 7 genuine without an official license: the Hypervisor mimics vendor specific hardware on different machines to allow the installation of pre-authorized S.O. copies (e.g., Pre-activated version of Windows on HP machines);
2. Mac OS on not-Apple hardware: the Hypervisor emulates the EFI (Extended Firmware Interface) of an Apple hardware on a standard PC.

7.5.2 Type 2 hypervisor

They are characterized by at least two OSs running on the same hardware:

- The Host OS controls the hardware of the system
- The Guest OS is the one the runs in the VM.

The VMM runs in the Host OS, while applications run in the Guest OS.

There is a Host OS that manages the hardware where the VMM runs.
+ More flexible in terms of underlying hardware.
+ Simpler to manage and configure (VMM can use the Host OS to provide GUI, not only BIOS).
- Special care must be taken to avoid conflict between Host OS and Guest OS (e.g., Virtual Memory).
- The Host OS might consume a non-negligible set of physical resources (e.g., 1 core for the Host OS).

7.6 Techniques

Different ways to implement (system level / same ISA) virtualization:

7.6.1 Paravirtualization

Guest OS and VMM collaborate:

- VMM present to VMs an interface similar but not identical to that of the underlying hardware
- Goal: reduce guest's executions of tasks too expensive for the virtualized environment ("hooks" allow the guest(s) and host to request and acknowledge tasks which would otherwise be executed in the virtual domain, where execution performance is worse).

Pros

- Simpler VMM
- High Performance

Cons

- Modified Guest OS (Cannot be used with traditional OSs, e.g., available on some Linux Releases)

7.6.2 Full Virtualization

Provides a complete simulation of the underlying hardware.

- the full instruction set
- input/output operations
- Interrupts
- memory access

Some protected instructions are trapped and handled by Hypervisor

Pros

- Running unmodified OS

Cons

- Hypervisor mediation (to allow the guest(s) and host to request and acknowledge tasks which would otherwise be executed in the virtual domain)
- Not on every architecture (requires some hardware support)

Para VS Full-virtualization

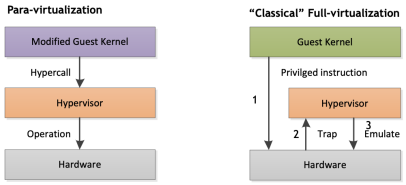


Figure 40: para vs full virtualization

7.7 Containers

Definition 17. *Containers are pre-configured packages, with everything you need to execute the code (code, libraries, variables and configurations) in the target machine.*

The main advantage of containers is that their behavior is predictable, repeatable and immutable:
when I create a "master" container and duplicate it on another server, I know exactly how it will be executed.There are no unexpected errors when moving it to a new machine or between environments.

Example: a container for a website,
• We are not required to export/import the development/test/production environments,
• We create a container that contains the site and bring it to the destination environment

7.7.1 Containers and Virtual Machine

VM provides hardware virtualization, while containers provide virtualization at the operating system level. The main difference is that the containers share the host system kernel with other containers.

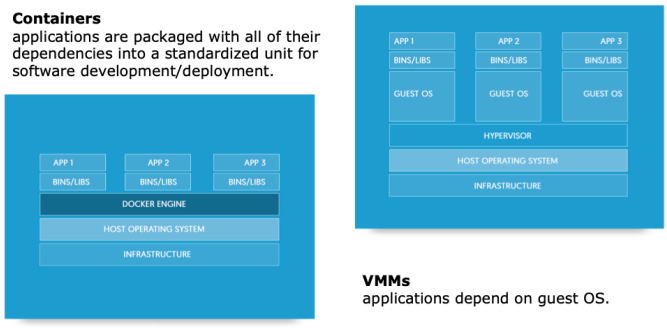


Figure 41: container vs VMMs

youtube video

7.7.2 Characteristics

- Flexible: even the most complex applications can be containerized;
- Light: the containers exploit and share the host kernel;
- Interchangeable: updates and updates can be distributed on the fly;
- Portable: you can create locally, deploy in the Cloud and run anywhere;
- Scalable: it is possible to automatically increase and distribute replicas of the container;
- Stackable: containers can be stacked vertically and on the fly.

Containers ease the deployment of applications and increase the scalability but they also impose a modular application development where the modules are independent and uncoupled.

7.7.3 Usage

- Helping make your local development and build workflow faster, more efficient, and more lightweight.
- Running stand-alone services and applications consistently across multiple environments.
- Using Container to create isolated instances to run tests.
- Building and testing complex applications and architectures on a local host prior to deployment into a production environment.
- Building a multi-user Platform-as-a-Service (PAAS) infrastructure.
- Providing lightweight stand-alone sandbox environments for developing, testing, and teaching technologies, such as the Unix shell or a programming language.
- Software as a Service applications.

7.7.4 Software for "Containers"

Figure 42: docker

- Docker aims to create and distribute software inside containers.
- Open source platform of tools that helps to "build, ship and run any app, anywhere"
- Docker is based on a DockerFile (a textual file which contains all the commands that a user can call on the command line to assemble an image). DockerFiles define a compilation process which, if sent to the "Docker build" command, will produce an immutable docker image (a sort of snapshot of the application).
- It is possible to run the docker where the docker daemon is running: e.g., a laptop, a server, in the cloud or a Raspberry PI
- Regardless of where the image is running, the docker will behave in the same way.
- Docker Swarm is the container orchestration tool developed by Docker closely integrated within the Docker ecosystem

- Kubernetes is a Google open source project
- Recommended for the management of medium and large clusters and complex applications;
- Extensive and customizable solution that allows you to coordinate large-scale node clusters in production more efficiently:
  - Run containers on many heterogeneous machines;
  - Increase or decrease the performance by proportionally modifying (adding or even removing) the containers
  - Share the load between containers;
  - Start new containers on different machines if a machine fails

Figure 43: kubernetes

youtube video

7.8 Theoretical Background: Popek and Goldberg

Assumptions:

- The hardware consists of a processor and a uniformly addressable memory (access time to a memory location is independent of which processor makes the request)
- The processor can operate in system mode (both system and user instructions) or user mode (only user instructions)
- A subset of the instruction set is available only in the system mode
- Memory addressing is done relative to the contents of a relocation register (i.e., a fixed value is added to a virtual address to arrive at a physical memory address).

- No I/O is considered (as I/O could be done in memory).

The machine being virtualized is a 4-tuple:
S = < E, M, P, R >

- Executable Storage **E** (conventional addressed memory of size q), i.e., the RAM
- Mode of Operation **M**, i.e., User or System
- Program Counter **P**
- Memory Relocation Bound Register **R** is a pair (l, b)
  - Physical location l of the virtual memory space
  - Bound b, the size/extension of the virtual memory space

If the address a accessed by a program falls outside the bounds indicated by R, a *memory trap* occurs.A context switch is performed:

- The current state of the machine is saved in location 0 (Current state is represented by values of (M,P,R))
- Copies the contents of location 0 of another process into M,P, and R. (Context switch)

Instructions are then classified on the basis of their behavior as a function of the state S of the machine.

Classifies the instructions of an ISA into 3 different (and possibly overlapping) groups based on their behavior:

1. **Privileged** instructions: Those that trap if the processor is in user mode and do not trap if it is in system mode.
   E.g.:
   - Load PSW (LPSW): load processor status word (PSW) if in system mode, from processor memory.
     Contains the P bit: specifies if the CPU is in user or system mode
     Critical for the control of the system!
   - Set CPU timer (SPT): replaces the CPU interval timer if in system mode
     If executable in user mode, a user program could change the amount of time dedicated to it before swapping occour.

   To specify instructions that interact with hardware, two categories of special instructions are defined: Could affect the correct functioning of the VMM because of their action. Divided into:

2. **Control sensitive** instructions: attempt to change the configuration of resources in the system.(e.g.: the physical memory assigned to a program)
3. **Behavior sensitive** instructions: those whose behavior or result depends on the configuration of resources (i.e.: the processor's mode affects the outcome)

NOTE: all of the guest OS software is hence forced to execute in user mode!

7.8.1 Requirements

System virtualization requires (requirements a VM must have to perform System-Level virtualization):

- Equivalence: A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
- Resource control: The VMM must be in complete control of the virtualized resources (guest cannot modify them).
- Efficiency: A statistically dominant fraction of machine instructions must be executed without VMM intervention.

Theorem: for any conventional (third-generation) computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions (i.e., all the control/behavior instructions must be privileged)

To build a VMM it is sufficient that all instructions that could affect the correct functioning of the VMM (sensitive instructions) always trap and pass control to the VMM.

- Since the VM is in user mode, the execution of sensitive instructions will be trapped by the VMM
- This guarantees the resource control property.

Non-privileged instructions must instead be executed natively (i.e., efficiently).

- This guarantees the efficiency property.

Popek and Goldberg theorem provides the theoretical background for implementing a Virtual Machine Monitor using the trap-and-emulate virtualization.

VMs always access the virtual resources using sensitive instructions.
This allows to trap the guest system, call the VMM which in turn can emulate the resource.

- This ensures the equivalence property.

7.9 Implementing full virtualization

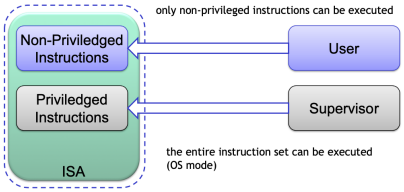


Figure 44: basic computer architecture

The instruction set is divided into (at least) two categories: Non-Priviledged and Priviledged Instruuctions.

A machine operates in two modes: User and Supervisor

7.9.1 workflow

1. The OS starts with the CPU in Supervisor mode.
2. The OS using Supervisor privileged instructions defines the memory bounds where application will run. 87
3. The OS gives the control to the application, changing the CPU to User mode.
4. The application runs normally using its non-privileged instructions.
5. If the application tries to access memory outside the assigned bounds, the CPU traps them and return the control to the OS for stopping the application and prevent compromising the entire system.

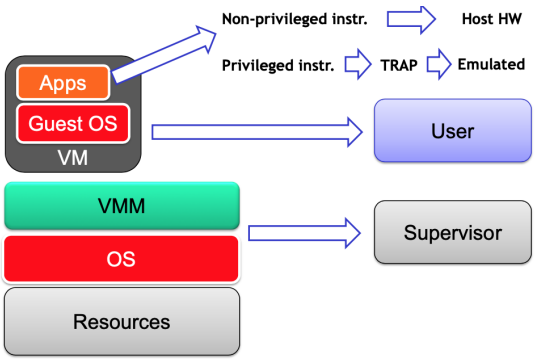


Figure 45: VM environments

7.9.2 Hardware-Assisted

Both AMD and Intel have modified their architectures to support virtualization Intel VT e AMD-V:

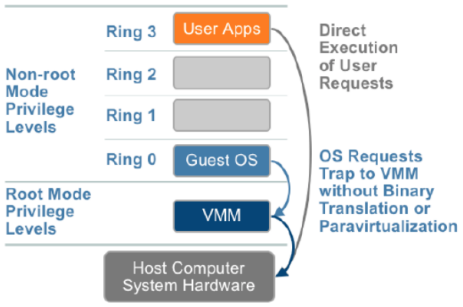


Figure 46: Hardware-Assisted

- a new high privilege ring for the VMM. (VMX root and VMX non-root)
- Hardware transitions from ring 0 to the VMM.
- Processor state is maintained for each guest OS (and the VMM) in separate address spaces

7.9.3 Root operation

Two modes of operation for the processor:

- VMX root operation (VMM)
  - Processor behaves as would have if no VT-x
  - Can use a whole set of instructions called VMX
- VMX non-root operation (VM)
  - Processor limited: critical shared resources are kept under the control of a monitor running in VMX root operation.
  - Limitation extended also to the ring 0!

Both support privilege levels 0-3.
Guest software can run in the rings in which it was originally intended to be run: guest OS believes it is running at level 0

7.9.4 Transitions

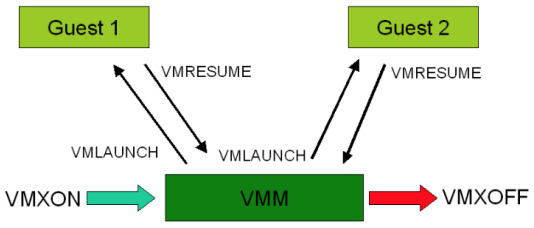


Figure 47: Transitions

VMXON instructions transfer control to VM: state restore from VMCB
Exits occur based on the content of the VMCB

- State saved to VMCB
- Exit codes stored in VMCB to simplify VMM
  (E.g., exit due to I/O provide the port number
  E.g., exit due to page fault provide faulting address and access mode)

8 Cloud Computing

8.1 Migration from Physical to Virtual

Without virtualization:

- Software strongly linked/related with hardware: move/change an application not a easy task
- To isolate failure/crash the classical model is:
  - 1 server
  - 1 operative system (OS)
  - 1 application, with a resulting low CPU utilization (10-15%)
  -
- Low flexibility

With Virtualization:

- Hw-independence: software/hardware no longer strongly related
- High fexibility thanks to pre-built VMs
- OS and applications can be handled as a ≪single entity≫

Consolidation Management: migration from physical to virtual machines, servers are connected one another so it is possible to:

- move Virtual Machines, without interrupting the applications running inside (**scalability**)
- automatically balances the Workloads according to set limits and guarantees (**automatic scalability**)
- Servers and Applications are protected against component and system failure (**high availability**)

Advantages of consolidations

- Different OS can run on the same hardware
- Higher hardware utilization
  - Less hardware is needed (Acquiring costs, Management costs(human resources, power, cooling))
  - Green IT-oriented
- Continue to use legacy software (e.g., software for WIN on Linux machines thanks to VMs)
- Application independent from the hardware

8.2 Cloud Computing

Definition 18. *Cloud computing is a model for enabling*

- *convenient*
- *on-demand*

network access to a shared pool of configurable computing resources, like for example: Networks, Servers, Storage, Applications and Services
that can be rapidly provisioned and released with minimal management effort or service provider interaction.

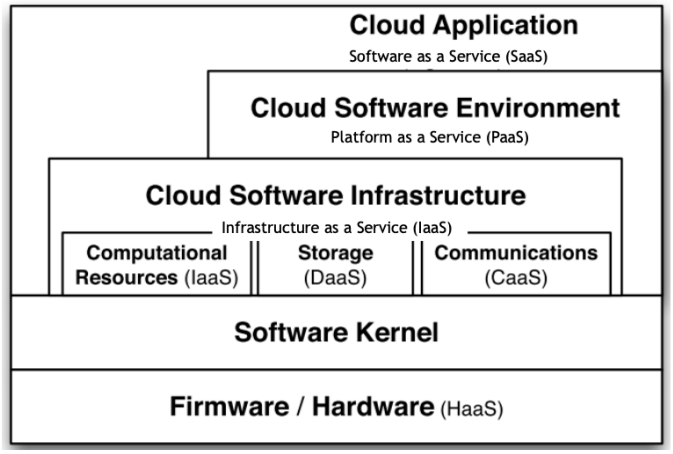


Figure 48: Cloud computing

8.2.1 Cloud Application Layer

SaaS (Software as a Service)
Users access the services provided by this layer through web-portals, and are sometimes required to pay fees to use them.
Cloud applications can be developed on the cloud software environments or infrastructure components.

Example: GMail, Google Docs and related apps (online office), SalesForce.com (CRMaaS)

8.2.2 Cloud Software Environment Layer

PaaS (Platform as a Service)
Users are application developers.
Providers supply developers with a programming-language-level environment with well-defined a API

- Facilitate interaction between environment and apps
- Accelerate the deployment
- Support scalability

Examples in Deep Learning: Amazon SageMaker, Microsoft Azure Machine Learning, Google AI: TensorFlow

8.2.3 Cloud Software Infrastructure Layer

Provides resources to the higher-level layers (i.e., Software and Software Environment) Note that Cloud Apps and Cloud SW might bypass Cloud SW Infrastructure: However, this would reduce

- Simplicity
- Development efforts

Computational Resources

IaaS (Infrastructure or Integration as a Service)

VM's benefits

- Flexibility
- Super-user (root) access to VM for fine granularity settings and customization of installed sw

VM's issues

- Performance interference
- Inability to provide strong guarantees about SLAs

Examples:

- Commercial solutions
  - Amazon Elastic Cloud (EC2): Full virtualization + Based on Xen
  - Windows Azure. (Not just windows-based: it allows also to start VMs for other OSs.)
  - Google Compute Engine. (Same infrastructure as Google.)
  - Rackspace Open Cloud.
  - IBM SmartCloud Enterprise.
  - HP Enterprise Converged Infrastructure.
- Open-source projects
  - Eucalyptus Systems
  - Apache CloudStack
  - Open Stack: The project aims to deliver solutions for all types of clouds (private or public) by being simple to implement, massively scalable, and feature rich.

Storage

DaaS (Data as a Service)
Allows users to store their data at remote disks and access data anytime from any place.
Facilitates cloud applications to scale beyond their limited servers requirements:

- High dependability: availability, reliability, performance (scalability)
- Replication
- Data consistency

Examples: DropBox, iCloud, GoogleDrive.
CEPH is an open source solution.

Communications

CaaS (Communication as a Service)
Communications becomes a vital component in guaranteeing QoS.

CaaS is part of a larger category of services known as software as a service (SaaS), in which vendors offer software products and services over the Internet.
The core concept of CaaS is that accessing these services over the internet is extremely convenient.

Types of CaaS include Voice over Internet Protocol (VoIP) or internet telephone solutions, and video conferencing services.

8.3 Types of Clouds

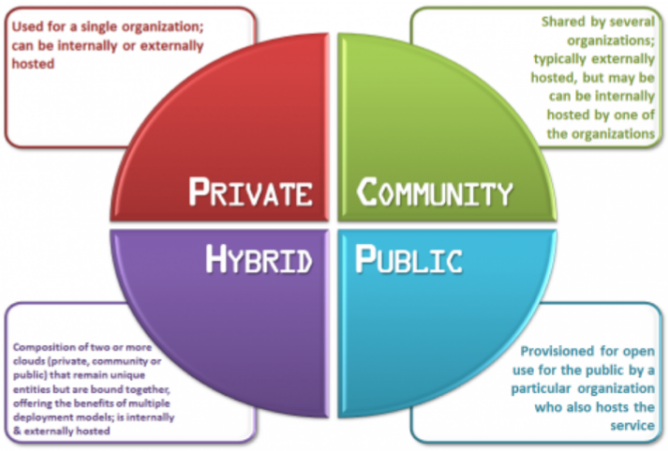


Figure 49: Cloud types

8.3.1 Public

Large scale infrastructure available on a rental basis

- Operating System virtualization (e.g. Xen) provides CPU isolation
- "Roll-your-own" network provisioning provides network isolation
- Locally specific storage abstractions

Fully customer self-service

- Service Level Agreements (SLAs) are advertized
- Requests are accepted and resources granted via web services
- Customers access resources remotely via the Internet

Accountability is e-commerce based

- Web-based transaction
- "Pay-as-you-go" and flat-rate subscription
- Customer service, refunds, etc.

8.3.2 Private

Internally managed data centers
The organization sets up a virtualization environment on its own servers: in its data center or in the data center of a managed service provider
Key benefits:

- you have total control over every aspect of the infrastructure
- you gain advantages of virtualization

Issue: It lacks the freedom from

- capital investment
- Flexibility ("almost infinite" grow of cloud computing)

Useful for companies that have significant existing IT investments

8.3.3 Community

A single cloud managed by several federated organizations.
Combining together several organizations allows economy of scale.
Resources can be shared and used by one organization, while the others are not using them.

Technically similar to private cloud: they share the same software and the same issues but a more complex accounting system is however required.
Hosted locally or externally:

- Typically community clouds shares infrastructures of the participants.
- However they can be hosted by a separate specific organization, or only by a small subset of the partners.

8.3.4 Hybrid

Hybrid clouds are the combination of any of the previous types: usually are companies that holds their private cloud, but that they can be subject to unpredictable peaks of load. In this case, the company rents resources from other types of cloud.
Common interfaces

- To simplify the deployment process, the way in which VMs are started, terminated, address is given and storage is accessed, must be as similar as possible.
- Many standards are being developed in this directions, but none is globally accepted yet.
- Currently, the Amazon EC2 model is the one with more compliant infrastructures.

8.4 Advantages of Cloud Computing

Cloud computing has many positive aspects:

- Lower IT costs
- Improved performance
- Instant software updates
- "Unlimited" storage capacity
- Increased data reliability
- Universal document access
- Device Independence

8.5 Disdvantages of Cloud Computing

However, it can also have some disadvantages:

- Requires a constant Internet connection
- Does not work well with low-speed connections
- Features might be limited
- Can be slow
- Stored data might not be secure
- Stored data can be lost

8.6 Fog/Edge Computing

It stays in the middle between the objects and the cloud: the computation is split. The fog pre-processes the data and, if it is able, it takes a decision, otherwise it sends them to the cloud to exploit its major computation power.

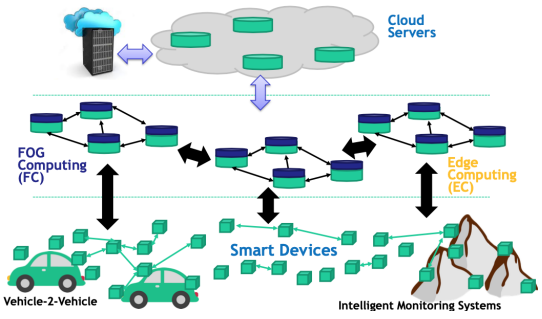


Figure 50: Fog and Edge Computing

9 Machine Learning as a service

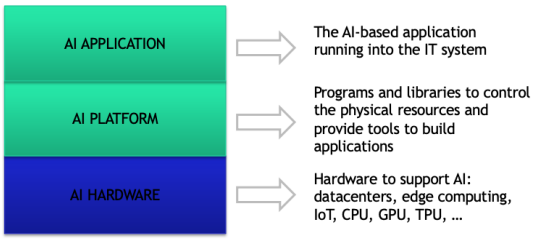


Figure 51: IT perspective for AI

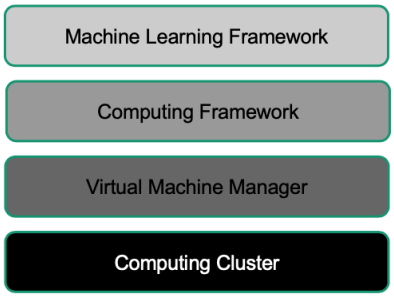


Figure 52: IT architecture for ML/DL

9.1 Computing Cluster

- Servers (parallelelization and scalability are key to find the best model as well as HW accelerators):
  - General Purpose CPU
  - GPU (TPU)
- Storage (multiple storage systems including distributed/parallel file systems):
  - DirectAttachedStorage
  - NAS
  - SAN
- Network (applications are generally non-iterative):
  - Ethernet

9.2 Virtual Machine Manager

- Virtualization is carried out through hypervisor (virtual machines) or containers
- Resources are increased by adding more virtual machines (provided that hardware resources are available)
- User can design personalized software environments and scale instances to the needs

Some examples: VMware, Xen, KVM, HyperX, Kubernetes

9.3 Computing Framework

Computing Frameworks are composed by several modules:

- Cluster Manager
- Data Storage
- Data processing engine
- Graph computation
- Programming Languages (Java, Python, Scala, R)

An application (e.g., a big-data application) operating in a cluster is distributed among different computing (virtual) machines.
Examples:

- Computing Framework: Hadoop, Spark, Flink
- Scheduling: Apache Mesos, Hadoop YARN
- Storage: HDFS, Hbase, Amazon S3, Cassandra, Mongo DB, Spark SQL
- Data Processing Engine: Spark, Map-Reduce

9.4 Machine/Deep Learning Framework

- Machine learning frameworks cover a variety of learning methods for classification, regression, clustering, anomaly detection, and data preparation, and it may or may not include neural network methods.
- Deep learning frameworks cover a variety of neural network topologies with many hidden layers.

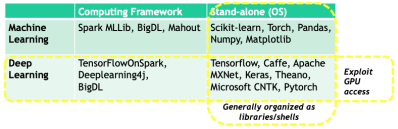


Figure 53: ML/DL frameworks

Cloud comuputing simplifies the access to ML capabilities for

- designing a solution (without requiring a deep knowledge of ML)
- setting up a project (managing demand increases and IT solution)

To support ML in the Cloud companies as Amazon, Microsoft and Google provide

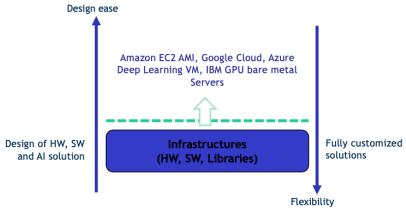


Figure 54: ML Infrastructure as a Service

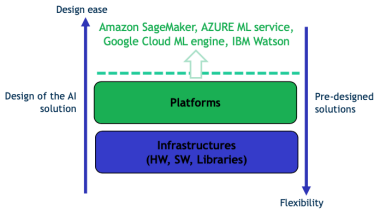


Figure 55: ML Platform as a Service

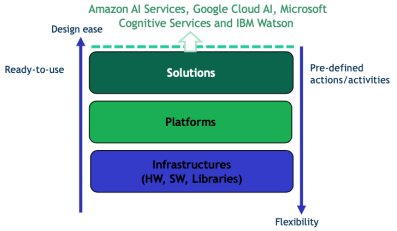


Figure 56: ML software as a Service

Pros

Cloud computing simplifies the access to ML/DL capabilities for

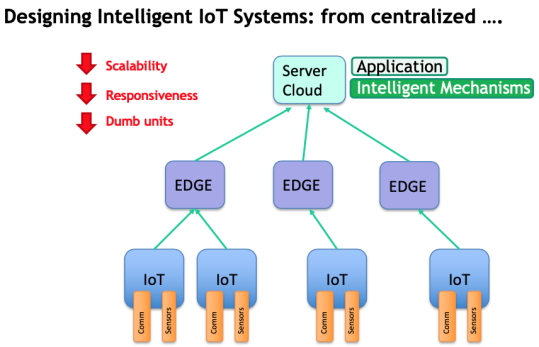
- designing a solution (without requiring a deep knowledge of ML/DL)
- setting up a project (managing demand increases and IT solution)

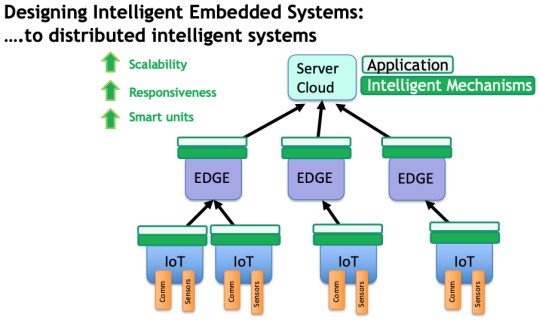
Machine and Deep Learning plaforms for IoT and Edge

- Increase autonomy
- Reduce decision-making latency
- Reduce transmission bandwidth
- Increase energy-efficiency
- Security and Privacy

Cons

- Internet connection
- High Power Consumption
- Privacy and Security
- Latency in making decision





10 RAID

Redundant Arrays of Independent (Inexpensive) Disks

Disk Arrays: proposed in the 1980's [PATTERSON].
Need to increase the **performance**, the **size** and the **reliability** of storage systems.

Several *independent* disks that are considered as a single, large,high-performance logical disk (In contrast with the JBOD (Just a Bunch of Disks) method where each disk is a separate device with a different mount point)
The data are *striped* across several disks accessed in parallel:

- high data transfer rate: large data accesses (heavy I/O op.)
- high I/O rate: small but frequent data accesses (light I/O op.)
- load balancing across the disks

Two orthogonal techniques:

1. **data striping**: to improve performance
2. **redundancy**: to improve reliability

10.1 Data striping

Definition 19. Striping*: data are written sequentially (a vector, a file, a table, ...) in units (stripe unit: bit, byte, blocks) on multiple disks according to a cyclic algorithm (round robin)*

Definition 20. stripe unit*: dimension of the unit of data that are written on a single disk*

Definition 21. stripe width*: number of disks considered by the striping algorithm*

1. multipleindependentI/Orequestswillbeexecutedin parallel by several disks decreasing the queue length (and time) of the disks
2. single multiple-block I/O requests will be executed by multiple disks in parallel increasing of the transfer rate of a single request

10.2 Redundancy

The main **motivation** for the introduction of redundancy: the more physical disks in the array, the larger the size and performance gains; but... the larger the probability of failure of a disk!

The probability of a failure (assuming independent failures) in an array of 100 disks is 100 higher the probability of a failure of a single disk
(if a disk has an Mean Time To Failure (MTTF) of 200,000 hours (23 years) an array of 100 disks will show a MTTF of 2000 hours (3 months))

Definition 22. Redundancy*: error correcting codes (stored on disks different from the ones with the data) are computed to tolerate loss due to disk failures*

Since write operations must update also the redundant information,their performance is worse than the one of the traditional writes.

10.3 Disks

Hard drives are great devices: relatively fast, persistent storage. But them have also limitations:

- How to cope with disk failure? Mechanical parts may break over time and sectors may become silently corrupted
- Capacity is limited: managing files across multiple physical devices is cumbersome (can we make 10x 1 TB drives look like a 10 TB drive?)

Definition 23. RAID*: use multiple disks to create the illusion of a large, faster, more reliable disk*

10.4 Externally

It looks like a single disk:

- RAID is transparent
- Data blocks are read/written as usual
- No need for software to explicitly manage multiple disks or perform error checking/recovery

10.5 Internally

It is a complex computer system:

- Disks managed by a dedicated CPU + software
- RAM and non-volatile memory
- Many different configuration options (RAID levels)

10.6 Levels

- RAID 0 striping only
- RAID 1 mirroring only
  RAID 0+1 (nested levels)
  RAID 1+0 (nested levels)
- RAID 2 bit interleaving (not used)
- RAID 3 byte interleaving - redundancy (parity disk)
- RAID 4 block interleaving - redundancy (parity disk)
- RAID 5 block interleaving - redundancy (parity block distributed)
- RAID 6 greater redundancy (2 failed disks are tolerated)

10.6.1 Level 0

Striping, no redundancy

Data are written on a single logical disk and splitted in several blocks distributed across the disks according to a striping algorithm.

Used where **performance** and **capacity** (rather than *reliability*) are the primary concerns, minimum two drivers required.

lowest cost because it does not employ redundancy (no error correcting codes are computed and stored)
best write performance it does not need to update redundant data and it is parallelized
single disk failure will result in data loss

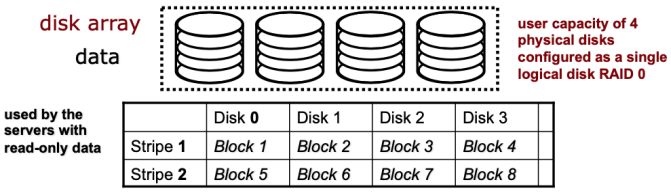
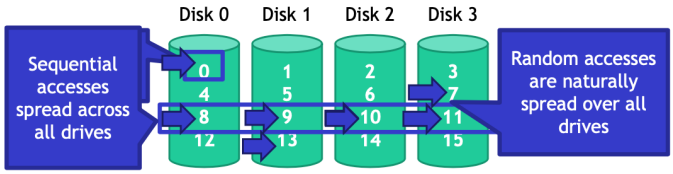


Figure 57: RAID level 0

Striping

Key idea: present an array of disks as a single large disk.
Maximize parallelism by striping data cross all N disks.

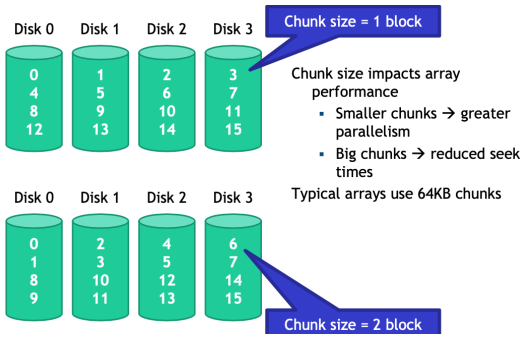


Addressing Blocks

How do you access specific data blocks?
Disk = logical block number % number of disks
Offset = logical block number / number of disks
Example: read block 11

- 11%4=Disk3
- 11 / 4 = Physical Block 2 (starting from 0)

Chunk Sizing



Performance

As usual, we focus on sequential and random workloads

Average seek time	7 ms
Average rotational delay	3 ms
Transfer rate	50 MB/s

- Assume disks in the array have sequential transfer rate S.
  Single large transfer:
  - 10 MB transfer
  - S = transfer size / time to access
  - 10MB/(7ms+3ms+10MB/50MB/s)=47.62MB/s
- Assume disks in the array have random transfer rate R.
  Set of small files:

- 10 KB transfer
- R = transfer size / time to access
- 10KB/(7ms+3ms+10KB/50MB/s)=0.98MB/s

Capacity: N
All space on all drives can be filled with data

Reliability: O
If any drive fails, data is permanently lost
MTTDL = MTTF

Sequential read and write: N*S
Full parallelization across drives

Random read and write: N*R
Full parallelization across all drives

10.6.2 Level 1

Mirroring

Whenever data is written to a disk it is also duplicated (mirrored) to a second disk (there are always **two copies** of the data), minimum 2 disk drives.

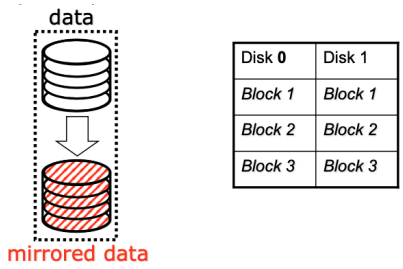


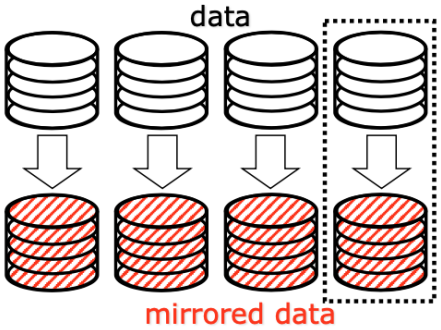
Figure 58: RAID Level 1

high reliability:when a disk fails the second copy is used
read of a data: it can be retrieved from the disk with the shorter queueing, seek, and latency delays
fast writes (no error correcting code should be computed)but still slower than standard disks (due to duplication)
high costs 50 % of the capacity isused

Mirroring

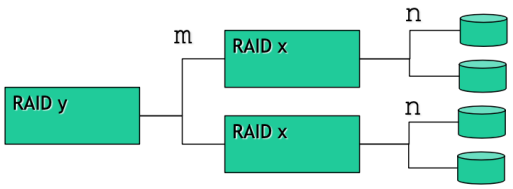
In principle, a RAID 1 can mirror the content over more than one disk.

- This give resiliency to errors even if more than one disk breaks
- It allows with a voting mechanism to identify errors not reported by the disk controller.



In practice this is never used, because the overhead and costs are too high.
RAID 0 offers high performance, but zero error recovery.
Key idea: make two **copies** of all data.

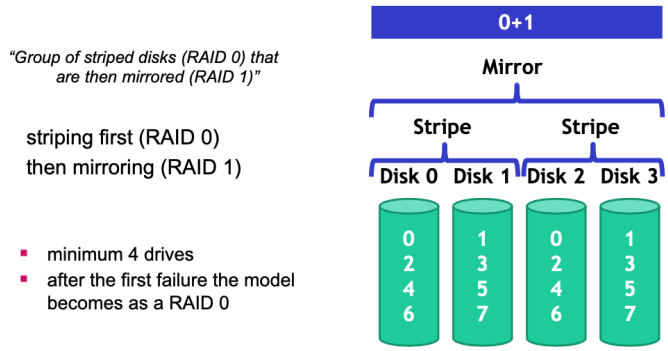
However if several disks are available (always in an *even* number), disks could be coupled.
The total capacity is halved.
Each disk has a mirror; How to organize this? RAID levels can be combined: Raid0+1 Raid1+0



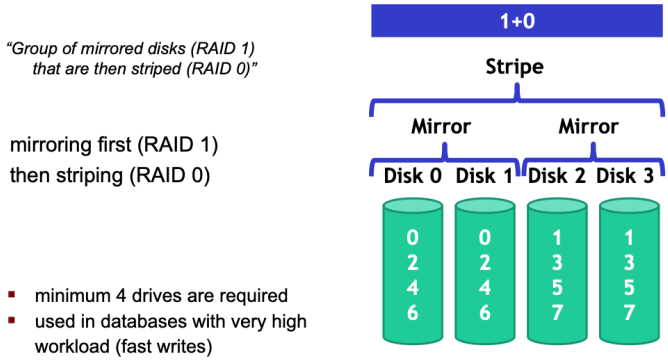
RAID levels can be **combined**
RAID x + y (or xy):

- *n* x *m* disks in total
- Consider m groups of n disks
- Apply RAID x to each group of n disks
- Apply RAID y considering the m groups as single disks

10.6.3 Level 0 + 1



10.6.4 Level 1 + 0



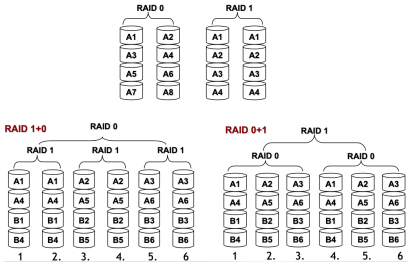


Figure 59: 0+1 and 1+0 organizations

The blocks are the same but are allocated in a different order
Performance on both RAID 10 and RAID 01 are the same.
The **storage capacity** of RAID 10 and RAID 01 is the same.
The main difference is the **fault tolerance** level: for RAID 0+1 is less, for RAID 1+0 is larger.

Capacity: N / 2
Two copies of all data, thus half capacity

Reliability: 1 drive can fail, sometime more
If you are lucky, N / 2 drives can fail without data loss

Sequential write: (N/2)*S
Two copies of all data, thus half throughput

Sequential read: (N/2)*S
Half of the read blocks are wasted, thus halving throughput

Random read: N*R
Best case scenario for RAID 1
Reads can parallelize across all disks

Random write: (N/2)*R
Two copies of all data, thus half throughput

The Consistent Update Problem

Mirrored writes should be atomic (All copies are written, or none are written); However, this is difficult to guarantee (Example: power failure)
Many RAID controllers include a **write-ahead log**

- Battery backed, non-volatile storage of pending writes
- A recovery procedure ensures to recover the out-of-sync mirrored copies

Decreasing the Cost of Reliability

RAID 1 offers highly reliable data storage but, it uses N / 2 of the array capacity;
We can achieve the same level of reliability without wasting so much capacity: use information coding techniques to build light-weight error recovery mechanisms!

10.6.5 Level 4

Party Drive

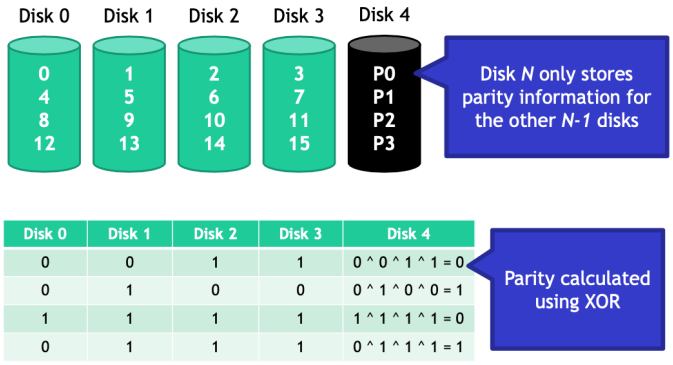
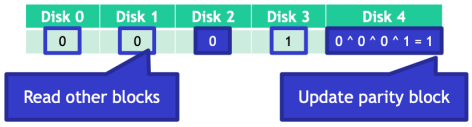


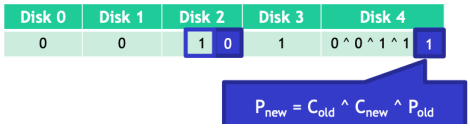
Figure 60: RAID level4

Write

1. Additive parity

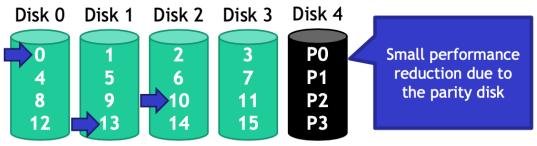


2. Subtractive parity



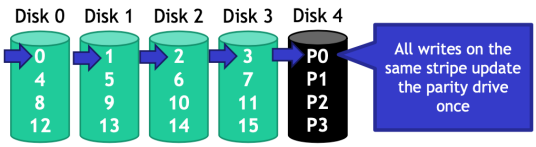
Read

Reads (Serial or Random) are not a problem in RAID4
Parallelization across all non-parity blocks in the stripe



Serial Writes

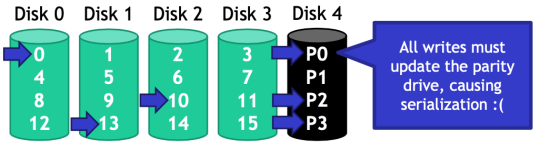
RAID 4 has the same Read and Write write performance.
Parallelization across all non-parity blocks in the stripe.



Random Writes

1. Read the target block and the parity block
2. Use subtraction to calculate the new parity block
3. Write the target block and the parity block

RAID 4 has terrible write performance: Bottlenecked by the parity drive



Capacity: N - 1
Space on the parity drive is lost

Reliability: 1 drive can fail
Massive performance degradation during partial outage

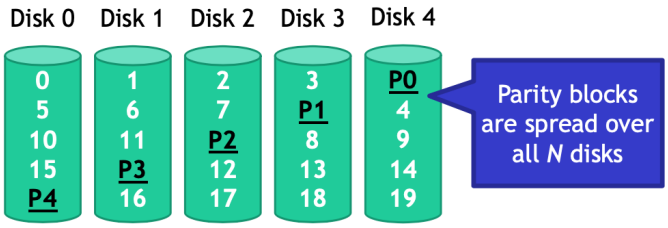
Sequential read and write: (N-1)*S
Parallelization across all non-parity blocks in the stripe

Random read: (N-1)*R
Reads parallelize over all but the parity drive

Random write: R/2
Writes serialize due to the parity drive
Each write requires 1 read and 1 write of the parity drive, thus R / 2

10.6.6 Level 5

Rotating Party



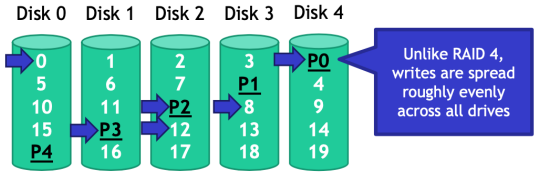
Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	0	1	1	0 ^ 0 ^ 1 ^ 1 = 0
1	0	0	0 ^ 1 ^ 0 ^ 0 = 1	0
1	1	1 ^ 1 ^ 1 ^ 1 = 0	1	1
1	0 ^ 1 ^ 1 ^ 1 = 1	0	1	1

Figure 61: RAID level4

Random Writes

1. Read the target block and the parity block
2. Use subtraction to calculate the new parity block
3. Write the target block and the parity block

Thus, 4 total operations (2 reads, 2 writes); Distributed across all drives



Capacity: N - 1 [same as RAID4]

Reliability: 1 drive can fail [same as RAID4]
Massive performance degradation during partial outage

Sequential read and write: (N-1)*S [same as RAID4]
Parallelization across all non-parity blocks in the stripe

Random read: N*R [vs (N-1)*R]
Reads parallelize over all drives

Random write: N/4 [vs R/2]
Writes parallelize over all drives
Each write requires 2 reads and 2 write, hence N / 4

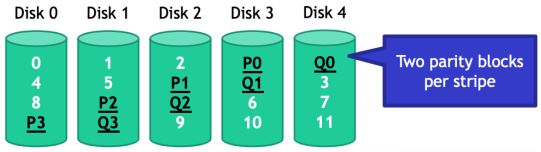
N = number of drives
R = random access speed
S = sequential access speed
D = latency to access a single disk

		RAID 0	RAID 1	RAID 4	RAID 5
	Capacity	*N*	*N / 2*	*N - 1*	*N - 1*
	Reliability	*0*	***1 (maybe N / 2)***	*1*	*1*
Throughput	Sequential Read	***N * S***	*(N / 2) * S*	*(N - 1) * S*	*(N - 1) * S*
	Sequential Write	***N * S***	*(N / 2) * S*	*(N - 1) * S*	*(N - 1) * S*
	Random Read	***N * R***	***N * R***	*(N - 1) * R*	***N * R***
	Random Write	***N * R***	*(N / 2) * R*	*R / 2*	*(N / 4) * R*
Latency	Read	***D***	***D***	***D***	***D***
	Write	***D***	***D***	*2 * D*	*2 * D*

Figure 62: Comparison of levels

10.6.7 Level 6

More fault tolerance with respect RAID5.
2 concurrent failures are tolerated.
Uses Solomon-Reeds codes with two redundancy schemes : (P+Q) distributed and independent.
N + 2 disks required.
High overhead for writes (computation of parities): each write require 6 disk accesses due to the need to update both the P and Q parity blocks (slow writes).
Minimum set of 4 data disks



10.6.8 Level comparison

Best performance and most capacity? -> RAID 0
Greatest error recovery? -> RAID 1 (1+0 or 0+1) or RAID 6
Balance between space, performance, and recoverability? -> RAID 5

RAID level	Capacity	Reliability	R/W performance	Rebuild performance	Suggested applications
0	100%	N/A	Very good	Good	Non critical data
1	50%	Excellent	Very good / good	good	Critical information
5	(n-1)/n	Good	Good/ fair	Poor	Database, transaction based applications
6	(n-2)/n	Excellent	Very good/ poor	Poor	Critical information, w/minimal
1+0	50%	Excellent	Very good/ good	Good	Critical information, w/better performance

10.7 Other Considerations

10.7.1 Many RAID systems include a hot spare

- An idle, unused disk installed in the system
- If a drive fails, the array is immediately rebuilt using the hot spare

10.7.2 RAID can be implemented in hardware or software

- Hardware is faster and more reliable...
- But, migrating a hardware RAID array to a different hardware controller almost never works
- Software arrays are simpler to migrate and cheaper, but have worse performance and weaker reliability (Due to the consistent update problem)

11 Dependability

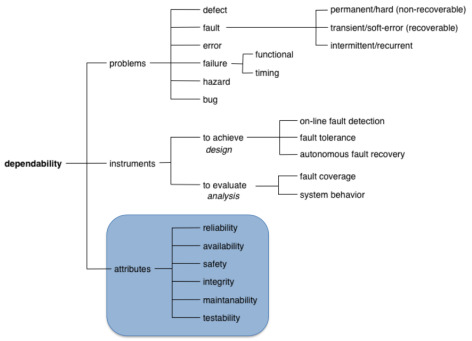


Figure 63: The scenario

11.1 Reliability

Definition 24. Reliability *The ability of a system or component to perform its required functions under stated conditions for a specified period of time*

R(t): probability that the system will operate correctly in a specified operating environment until time t

$$\text{R(t)} = \text{P(not failed during [0, t])}$$
assuming it was operating at time t=0.

t is important
If a system needs to work for slots of ten hours at a time, then ten hours is the reliability target.

1 - R(t): unreliability, also denoted **Q(t)**
R(t) is a non-increasing function varying from 1 to 0 over [0,+∞) $\lim_{x\to\infty} R(t) = 0$.
Often used to characterize systems in which even small periods of incorrect behavior are unacceptable

- Performance requirements
- Timing requirements
- Extreme safety requirements
- Impossibility or difficulty to repair

11.2 Availability

Definition 25. Availability *The degree to which a system or component is operational and accessible when required for use*

Availability = Uptime / (Uptime + Downtime)

A(t): probability that the system will be operational at time t

$$\text{A(t)} = \text{P(not failed at time t)}$$

Literally, readiness for service
Admits the possibility of brief outages
Fundamentally different from reliability

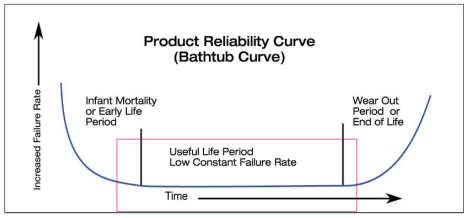
1 - A(t): unavailability
When the system is *not repairable*: A(t) = R(t)
In general (repairable systems): A(t) $\geqslant$ R(t)

MTTF (Mean Time To Failure): mean time before any failure will occur
MTBF (Mean Time Between Failures): mean time between two failures

$$\text{MTBF} = \frac{totaloperatingtime}{numberoffailures}$$

FIT: failures in time
Failure Rate $\lambda = \frac{numberoffailures}{totaloperatingtime}$

- another way of reporting MTBF
- the number of expected failures per one billion hours (10^9) of operation for a device
- MTBF (in h) =109/FIT



- *Infant Mortality*: failures showing up in new systems. Usually this category is present during the testing phases, and not during production phases.
- *Random Failures*: showing up randomly during the entire life of a system.
  - Our main focus
- *Wear Out*: at the end of its life, some components can cause the failure of a system. Pre-emptive mainteinance can reduce the number of this type of failures.

11.3 Reliability & Availability

Two different points of view "reliability: does not break down ..." "availability: even if it breaks down, it is working when needed ..."
Of course they are related: if a system is unavailable it is not delivering the specified system services
It is possible to have systems with low reliability that must be available: system failures can be repaired quickly and do not damage data, low reliability may not be a problem (for example a database management system)

Exploitation of R(t) information is used to compute, for a complex system, its reliability in time, that is the expected lifetime (computation of the MTTF)

Term	Description
Fault	A defect within the system
Error	A deviation from the required operation of the system or subsystem
Failure	The system fails to perform its required function

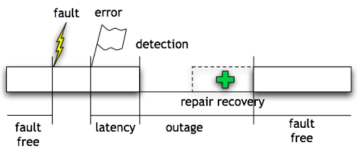


Figure 64: Reliability terminology

An example: a flying drone with an automatic radar-guided landing system
Fault: electromagnetic disturbances interfere with a radar measurement
Error: the radar-guided landing system calculates a wrong trajectory
Failure: the drone crashes to the ground

11.4 Reliability Block Diagrams

An inductive model where a system is divided into *blocks* that represent distinct elements such as components or subsystems.
Every element in the RBD has its own reliability (previously calculated or modelled)
Blocks are then combined together to model all the possible success paths.
RBDs are an approach to compute the reliability of a system starting from the reliability of its components

Figure 65: components in series

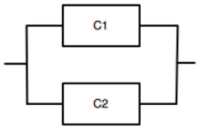
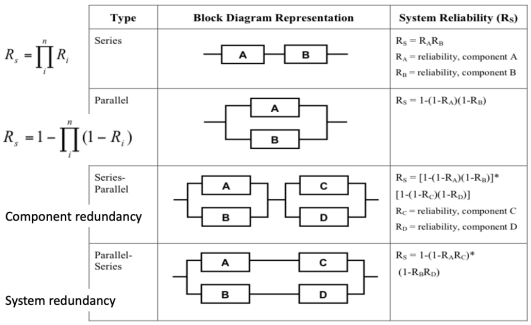


Figure 66: components in parallel

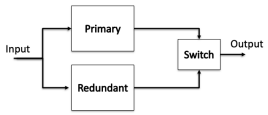
All components must be healthy for the system to work properly

If one component is healthy the system works properly



Type	Block Diagram Representation	System Reliability (R_S)
Series	A B	$R_S = R_A R_B$ R_A = reliability, component A R_B = reliability, component B
Parallel	A B	$R_S = 1-(1-R_A)(1-R_B)$
Series-Parallel	A C B D	$R_S = [1-(1-R_A)(1-R_B)]*$ $[1-(1-R_C)(1-R_D)]$ R_C = reliability, component C R_D = reliability, component D
Parallel-Series	A C B D	$R_S = 1-(1-R_A R_C)*$ $(1-R_B R_D)$

11.5 Standby redundancy



A system may be composed of two parallel replicas:

- The primary replica working all time
- The redundant replica (generally disable) that is activated when the primary replica fails

Obviously we need: A mechanism to determine whether the primary replica is working properly or not (on-line self check) and a dynamic switching mechanism to disable the primary replica and activate the redundant one.

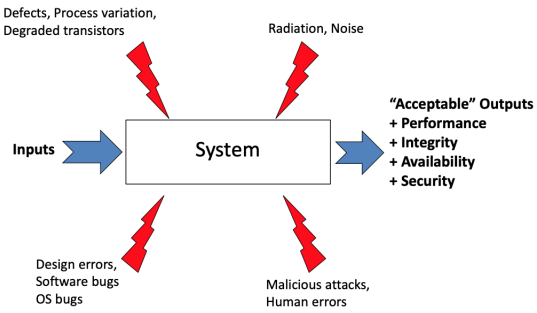
Standby Parallel Model	System Reliability
Equal failure rates, perfect switching	$R_s = e^{-\lambda t}(1+\lambda t)$
Unequal failure rates, perfect switching	$R_s = e^{-\lambda_1 t} + \lambda_1 (e^{-\lambda_1 t} - e^{-\lambda_2 t})/(\lambda_2 - \lambda_1)$
Equal failure rates, imperfect switching	$R_s = e^{-\lambda t}(1+R_{switch}\lambda t)$
Unequal failure rates, imperfect switching	$R_s = e^{-\lambda_1 t} + R_{switch}\lambda_1 (e^{-\lambda_1 t} - e^{-\lambda_2 t})/(\lambda_2 - \lambda_1)$
where R_s = System reliability λ = Failure rate t = Operating time R_{switch} = Switching reliability	

11.6 Dependability

Definition 26. *The ability of a system to perform its functionality while exposing:*

- *Reliability: Continuity of correct service*
- *Availability: Readiness for correct service*
- *Maintainability: Ability for easy maintainance*
- *Safety: Absence of catastrophic consequences*
- *Security: Confidentiality and integrity of data*

A lot of effort is devoted to make sure the implementation matches specifications, fulfills requirements, meets constraints, optimizes selected parameters (performance, energy, ...).
Nevertheless, even if all above aspects are satisfied ... things may go wrong: systems fail



11.6.1 Failure effects

A single system failure may affect a large number of people, it also may have high costs if it impacts economic losses or physical damage. Systems that are not dependable are likely not be used or adopted. Undependable systems may cause information loss with a high consequent recovery cost.

11.6.2 When to think about dependability?

Both at

- design-time:
  - Analyse the system under design
  - Measure dependability properties
  - Modify the design if required

  and at

- runtime:
  - Detect malfunctions
  - Understand causes
  - React

Failures *in development* should be avoided
Failures *in operation* cannot be avoided (things break), they must be dealt with.
Design should take failures into account and guarantee that control and safety are achieved when failures occur.(Effects of such failures should be predictable and deterministic ... not catastrophic)

11.6.3 Where to apply dependability?

Once upon a time dependability has been a relevant aspect only for safety-critical and mission-critical application environments (Space, Nuclear, Avionics) Huge costs, acceptable only when mandatory.

Non-critical critical systems

a failure during operation can have economic and reputation effects (Consumer products)

Mission-critical systems

a failure during operation can have serious or irreversible effects on the mission the system is carrying out
• Satellites
• Automatic Wheather Stations
• Surveillance drones
• Unmanned vehicles

Safety-critical systems

a failure during operation can present a direct threat to human life
• aircraft control systems
• medical instrumentation
• railway signaling
• nuclear reactor control systems

11.6.4 Anatomy of the scenarios

Everything has to work properly for the overall system to be working
the nodes

- computing systems
- sensors and actuators

the communication

- network

the cloud

- data storage
- data manipulation

11.6.5 Failure avoidance paradigm

Conservative design
Design validation
Detailed test (Hardware — Software)
Infant mortality screen
Error avoidance

11.6.6 Failure tolerance paradigm

Error detection / error masking during system operation
On-line monitoring
Diagnostics
Self-recovery & self-repair

11.6.7 Where to work

- technological level:design and manufacture by employing reliable/robust components
  • Highest dependability
  • High cost
  • Bad performance (generally devices from old generation)
- architectural level: integrate normal components using solutions that allow to manage the occurrence of failures
  • High dependability Depending on the
  • High cost adopted solution
  • Reduced performance
  (these 3 depend on the adopted solution)
- software/application level:develop solutions in the algorithms or in the operating systems that mask and recover from the occurrence of failures
  • High dependability Depending on the
  • High cost adopted solution
  • Reduced performance
  (these 3 depend on the adopted solution)

12 Performance

Definition 27. Computer performance:*The total effectiveness of a computer system, including throughput, individual response time and availability. Can be characterized by the amount of useful work accomplished by a computer system or computer network compared to the time and resources used*

12.1 System quality

Common practice: system mostly validated versus *"functional"* requirements rather than versus *quality* ones.
Different (and often not available) *skills* required for quality verification
Short time to market, i.e. quickly available products and infrastructures "seem" to be more attractive nowadays!
Little information related to quality is usually available early in the system lifecycle but its understanding is of great importance from the cost and performance point of view
(During design and system sizing but also during system evolution)

12.1.1 How to evaluate system quality

- Use of intuition and trend extrapolation: unfortunately, those who possess these qualities in sufficient quantity are rare.
  Pro: rapid and flexible
  Con: accuracy
- Experimental evaluation of alternatives: experimentation is always valuable, often required, and sometimes the approach of choice.
  It also is expensive - often prohibitively so
  A further drawback: an experiment is likely to yield accurate knowledge of system behavior under one set of assumptions, but not any insight that would allow generalization
  Pro: excellent accuracy
  Con: laborious and inflexible

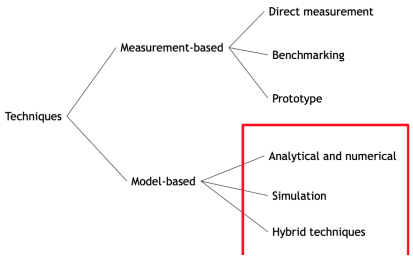
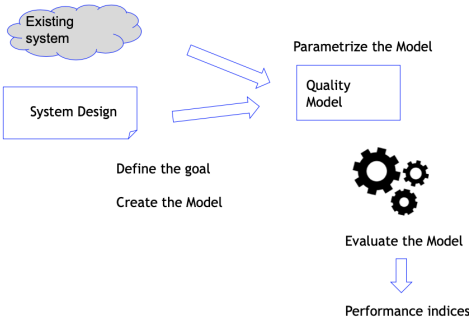


Figure 67: Quality Evaluation techniques

12.2 Model-Based Approach

Definition 28. *an attempt to distill, from the details of the system, exactly those aspects that are essentials to the system behavior*

Systems are complex so make an abstraction of the systems: Models.



Definition 29. Model *A representation of a system that is*

- *simpler than the actual system*
- *captures the essential characteristics*
- *can be evaluated to make predictions*

1. Analytical and Numerical techniquesare based on the application of mathematical techniques, which usually exploit results coming from the theory of probability and stochastic process
    - They are the most efficient and the most precise, but are available only in very limited cases
2. Simulation techniquesare based on the reproduction of traces of the model
    - They are the most general, but might also be the less accurate, especially when considering cases in which rare events can occur
    - The solution time can also become really large when high accuracy is desired
3. Hybrid techniques combine analytical/numerical methods with simulation

12.2.1 Queueing Networks

Queueing theory is the theory behind what happens when you have a lot of jobs, scarce resources, and so long queue and delays.

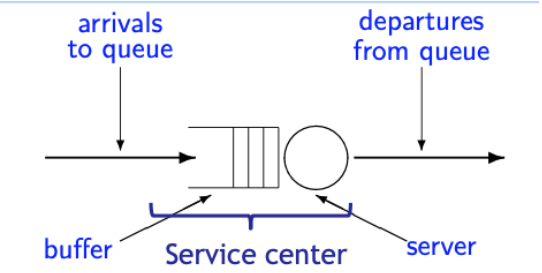
Definition 30. Queueing network modelling *is a particular approach to computer system modelling in which the computer system is represented as a network of queues.A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions*

Queueing theory applies whenever queues come up
Queues in computer systems:

- CPU uses a time-sharing scheduler
- Disk serves a queue of requests waiting to read or write blocks
- A router in a network serves a queue of packets waiting to be routed
- Databases have lock queues, where transactions wait to acquire the lock on a record

Predicting performance e.g. for capacity planning purposes
Queueing theory is built on an area of mathematics called stochastic modelling and analysis
Success of queueing network: low-level details of a system are largely irrelevant to its high-level performance characteristics

Single queue

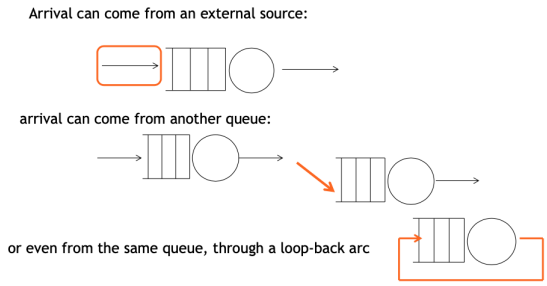


- The basic scenario for a single queue is that customers, who belong to some population arrive at the service facility
- The service facility has one or more servers which can perform the service required by customers Jane Hillston School of Informatics The University of Edinburgh Scotland Performance Modelling — Lecture 5 Queueing Networks
- If a customer cannot gain access to a server it must join a queue, in a buffer, until a server is available
- When service is complete the customer departs, and the server selects the next customer from the buffer according to the service discipline (queueing policy)

Different aspects characterize queuing models:

- **Arrival**

  Arrivals represent jobs entering the system: they specify how fast, how often and which types of jobs does the station service.
  We are interested in the average arrival rate λ (req/s)

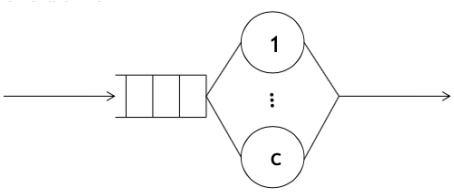


- **Service** The service part represents the time a job spends being served.

  The service time is the time which a server spends satisfying a customer
  As with the inter-arrival time, the important characteristics of this time will be its average duration (advanced: the distribution function)
  If the average duration of a service interaction between a server and a customer is $1/\mu$ then μ is the maximum service rate.

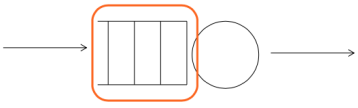
  It can have different number of servers:

  – **a single server**: the service facility only has the capability to serve one customer at a time; waiting customers will stay in the buffer until chosen for service; how the next customer is chosen will depend on the service discipline
  – **an infinite server**: there are always at least as many servers as there are customers, so that each customer can have a dedicated server as soon as it arrives in the facility. There is no queueing, (and no buffer) in such facilities



  Between these two extremes there are multiple server facilities
  These have a fixed number of c servers, each of which can service a customer at any time
  If the number of customers in the facility is less than or equal to c there will no queueing—each customer will have direct access to a server
  If there are more than c customers, the additional customers will have to wait in the buffer

- **Queue** If jobs exceed the capacity of parallel processing of the system, they are forced to wait queueing in a buffer



Customers who cannot receive service immediately must wait in the buffer until a server becomes available
If the buffer has finite capacity there are two alternatives for when the buffer becomes full:

- either, the fact that the facility is full is passed back to the arrival process and arrivals are suspended until the facility has spare capacity, i.e. a customer leaves;
- or, arrivals continue and arriving customers are lost (turned away) until the facility has spare capacity again

If the buffer capacity is so large that it never affects the behaviour of the customers it is assumed to be infinite

When the (one of the) job(s) currently in service leaves the system, one of the job in the queue can enter the now free service center

Service discipline/queuing policy determines which of the job in the queue will be selected to start its service

When more than one customer is waiting for service, we need a rule for selecting which of the waiting customers will be the next one to gain access to a server
The commonly used service disciplines are

- FCFS first-come-first-serve (or FIFO first-in-first-out)
- LCFS last-come-first-serve (or LIFO last-in-first-out)
- RSS random-selection-for-service
- PRI priority, the assignment of different priorities to elements of a population is one way in which classes are formed

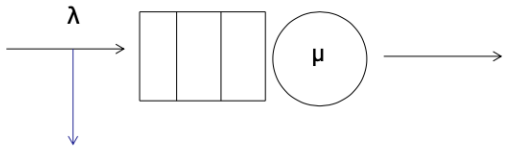
- **Population** The characteristic of the population which we are interested in is usually the size
Clearly, if the size of the population is fixed, at some value N, no more than N customers will ever be requiring service at any time
When the population is finite, the arrival rate of customers will be affected by the number who are already in the service facility (e.g. zero arrivals when all N are all already in the facility)
When the size of the population is so large that there is no perceptible impact on the arrival process, we assume that the population is infinite
Ideally, members of the population are indistinguishable from each other
When this is not the case, we divide the population into classes whose members all exhibit the same behaviour
Different classes differ in one or more characteristics, for example, arrival rate, service demand
Identifying different classes is a workload characterisation task

Example

Consider a wireless access gateway:
• Measurements have shown that packets arrive at a mean rate of 125 packets per second, and are buffered
• The gateway takes 2 milliseconds on average to transmit a packet
• The buffer currently has 13 places, including the place occupied by the packet being transmitted and packets that arrive when the buffer is full are lost
• Goal of the modelling and analysis:

We wish to find out if the buffer capacity which is sufficient to ensure that less than one packet per million gets lost

A single queue center with:
Finite queue capacity=13
FCFS service discipline
Arrival rate $\lambda = 125$ req/s
Service rate $\mu = 1/(2\text{ms}) = 500$ req/s



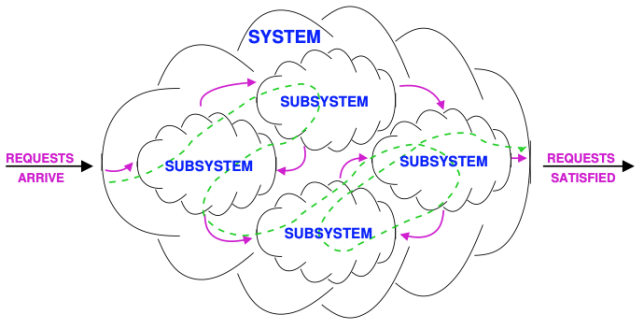
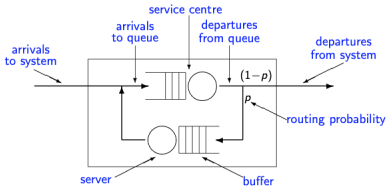


Figure 68: Queueing Networks

For many systems we can adopt a view of the system as a Nectowlloercktionf oqfureuseosurces and devices with customers or jobs ciFrocrumlaatninygsybsetetmwes ewne tchaenmadopt a

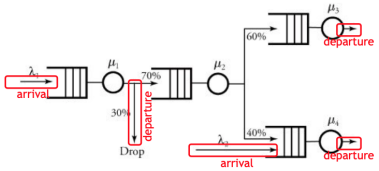
We can associate a service center with each resource in the system and then route customers among the service centres
After service at one service centre a customer may progress to other service centres, following some previously defined pattern of behaviour, corresponding to the customer's requirement

A queueing network can be represented as a graph where nodes represent the service centers k and arcs the possible transitions of users from one service center to another
Nodes and arcs together define the network topology

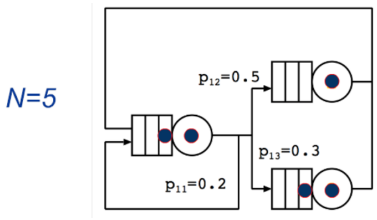


A network may be:

– Open: customers may arrive from, or depart to, some external environment (Open Models are characterized by arrivals and departures from the system)



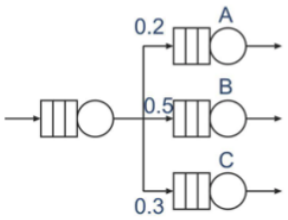
– Closed: a fixed population of customers remain within the system (In closed models we a parameter N that accounts for the fixed population of jobs that continuously circulate inside the system)



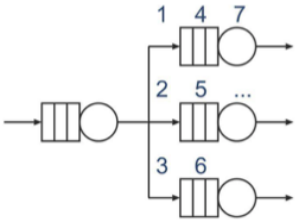
– Mixed: there are classes of customers within the system exhibiting open and closed patterns of behaviour respectively

- **Routing**Whenever a job, after finishing service at a station has several possible alternative routes, an appropriate selection policy must be defined
  The policy that describes how the next destination is selected is called routing
  Routing specification is required only in all the points where jobs exiting a station can have more than one destination
  The main routing algorithms that we will consider are:

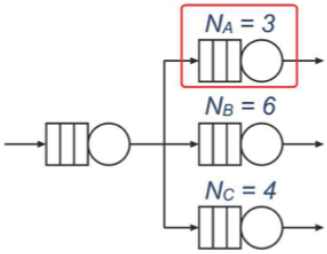
  – **Probabilistic**each path has assigned a probability of being chosen by the job that left the considered station



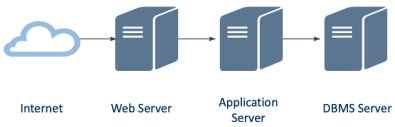
  – **Round robin** the destination chosen by the job rotates among all the possible exits



  – **Join the shortest queue** jobs can query the queue length of the possible destinations, and chose to move to the one with the smallest number of jobs waiting to be served

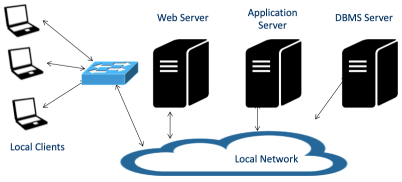


Open Networks: Examples



A client server system, dealing with external arrivals, which is architected with three tiers: the first one includes one web server, the second tier includes one application server and the third one includes a database server
Provide a QN model of the system and evaluate the overall throughput considering that the network delay is negligible with respect to the other devices and two different cases:
1. The only thing we know is that each server should be visited by the application
2. In the second case we know that the application after visiting the web server requires some operations at the application server and then can go back to the web server and leave the system or can require service at the DBMS and then go back to the application server

Closed Networks: Examples

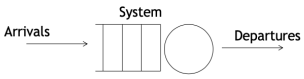


A client server system, with a finite number of customers, which is architected with three tiers: the first one includes one web server, the second tier includes one application server and the third one includes a database server.
Provide a QN model of the system and evaluate the system throughput considering that Network delay is negligible with respect to the other devices. Model the two different cases previously described.

12.3 Operational laws

Definition 31. Operational laws *are simple equations which may be used as an abstract representation or model of the average behaviour of almost any system*

The laws are very general and make almost no assumptions about the behaviour of the random variables characterising the system.
Another advantage of the laws is their simplicity: they can be applied quickly and easily.



Operational laws are based on observable **variables** - values which we could derive from watching a system over a finite period of time
We assume that the system receives **requests** from its environment
Each request generates a **job** or **customer** within the system
When the job has been processed the system responds to the environment with the completion of the corresponding request

If we observed such an abstract system we might measure the following quantities:

- T, the length of **time** we observe the system

- A, the number of request **arrivals** we observe
- C, the number of request **completions** we observe
- B, the total amount of time during which the system is **busy** (B $\leq$ T)
- N, the average **number of jobs** in the system
- λ = A/T , the **arrival** rate
- X = C /T , the **throughput** or completion rate
- U = B/T, the **utilisation**
- S = B/C, the **mean service time** per completed job

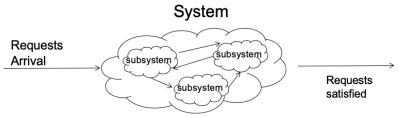
12.3.1 Job flow balanced

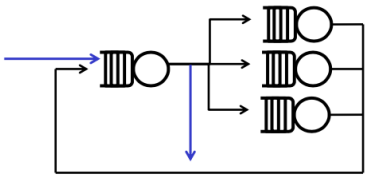
Definition 32. job flow balanced *means that the number of arrivals is equal to the number of completions during an observation period, i.e., A = C*

This is a testable assumption because an analyst can always test whether the assumption holds;It can be strictly satisfied by careful choice of measurement interval.

Note that if the system is job flow balanced the arrival rate will be the same as the completion rate, that is:

$$\lambda = X$$





A system may be regarded as being made up of a number of devices or resources
Each of these may be treated as a system in its own right from the perspective of operational laws
An external request generates a job within the system; this job may then circulate between the resources until all necessary processing has been done; as it arrives at each resource it is treated as a request, generating a job internal to that resource.

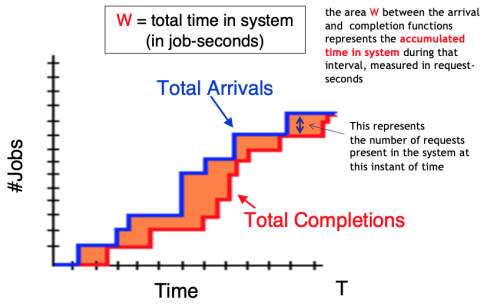
12.3.2 Little Low

$$N = XR$$

Little law can be applied to the entire system as well as to some subsystems

N = average number of requests in the system
W denotes the accumulated time in system (jobs- sec) [if there are 3 requests in the system during a 2 second period, then W is 6 request-seconds]
R the average system residence time per request is: R=W/C

If the system throughput is X requests/sec, and each request remains in the system on average for R seconds, then for each unit of time, we can observe on average XR requests in the system



Residence time corresponds to our conventional notion of **response time**: the period of time from when a user submits a request until that user's response is returned

Back when most processing was done on shared mainframes, think time, Z, was quite literally the length of time that a programmer spent thinking before submitting another job
More generally in interactive systems, jobs spend time in the system not engaged in processing, or waiting for processing: this may be because of interaction with a human user, or may be for some other reason
The think time represents the time between processing being completed and the job becoming available as a request again

12.3.3 Interactive Response Time Law

$$R = N/X - Z$$

The response time in an interactive system is the residence time minus the think time
Note that if the think time is zero, Z=0 and R=N/X,then the interactive response time law simply becomes Little's Law

In an observation interval we can count not only completions external to the system, but also the number of completions at each resource within the system
Note that:

- If Ck > C, resource k is visited several times (on average) during each system level request. This happens when there are loops in the model
- If Ck < C, resource k might not be visited during each system level request. This can happen if there are alternatives (i.e. caching of disks)
- If Ck = C, resource k is visited (on average) exactly once every request

Ck is the number of completions at resource k
We define the visit count, Vk , of the k-th resource to be the ratio of the number of completions at that resource to the number of system completions Vk = Ck/C

12.3.4 Forced Flow Law

The forced flow law captures the relationship between the different components within a system. It states that the throughputs or flows, in all parts of a system must be proportional to one another.

$$X_k = V_k \text{ X}$$

The throughput at the k-th resource is equal to the product of the throughput of the system and the visit count at that resource.

12.3.5 Utilisation Law

If we know the amount of processing each job requires at a resource then we can calculate the utilisation of the resource
Let us assume that each time a job visits the k-th resource the amount of processing, or service time it requires is Sk
Note that service time is not necessarily the same as the response time of the job at that resource: in general a job might have to wait for some time before processing begin
The total amount of service that a system job generates at the k-th resource is called the service demand, D_k:

$$D_k = S_k V_k$$

Average service time S_k accounts for the average time that a job spends in station k when IT IS SERVED

Average service demand D_k accounts for the average time a job spends in station k during its staying in the system. As seen for the visits, depending on the way in which the jobs move in the system, the demand can be less than, greater than or equal to the average service time of station k

The **Response Time** $\widetilde{R_k}$ accounts for the average time spent in station k, when the job enters the corresponding node (i.e., time for the single interaction, e.g. disk request)
The Residence Time R_k accounts instead for the average time spent by a job at station k during the staying in the system: it can be greater or smaller than the response time depending on the number of visits.
Note that there is the same relation between Residence Time and Response Time as the one between Demand and Service Time!

12.3.6 General Response Time Law

One method of computing the mean response time per job in a system is to apply Little's Law to the system as a whole
However, if the mean number of jobs in the system, N, or the system level throughput, X, are not known an alternative method can be used:

$$R_k = N_k/\mathrm{X} = V_k \ \widetilde{R_k}$$