Computing Infrastructures
Theory

Christian Rossi

Academic Year 2023-2024

Abstract

The course topics are:

- Hardware infrastructure of datacenters:
  - Basic components, rack structure, cooling.
  - Hard Disk Drive and Solid State Disks.
  - RAID architectures.
  - Hardware accelerators.
- Software infrastructure of datacenters:
  - Virtualization: basic concepts, technologies, hypervisors and containers.
  - Computing Architecture: Cloud, Edge and Fog Computing.
  - Infrastructure, platform and software-as-a-service.
- Methods:
  - Scalability and performance of datacenters: definitions, fundamental laws, queuing network theory basics.
  - Reliability and availability of datacenters: definitions, fundamental laws, reliability block diagrams.

Contents

CHAPTER 1

Introduction

1.1 Computing infrastructures

Definition (*Computing infrastructure*)**.** Computing infrastructure refers to the technological framework comprising hardware and software components designed to facilitate computation for other systems and services.

Data centers encompass servers tailored for diverse functions:

- *Processing Servers.*
- *Storage Servers.*
- *Communication Servers.*

1.1.1 Virtual machines

Virtual machines offer a comprehensive stack comprising an operating system, libraries, and applications. Applications rely on a guest operating system.

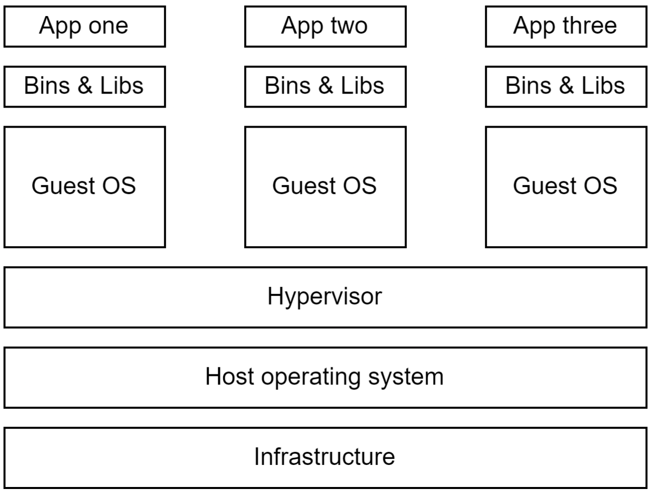


Figure 1.1: Virtual machine's structure

In this configuration, each operating system perceives the hardware as exclusively dedicated to itself. Virtualization results in significant power efficiency gains by consolidating the power consumption of individual machines, typically saving around 60%. Virtualization enables hot swaps, delivering two key advantages:

1. *Maintainability.*
2. *Availability*: overloaded machines can be supplemented by others.

1.1.2 Containers

Containers encapsulate applications along with their dependencies into a uniform unit for streamlined software development and deployment.

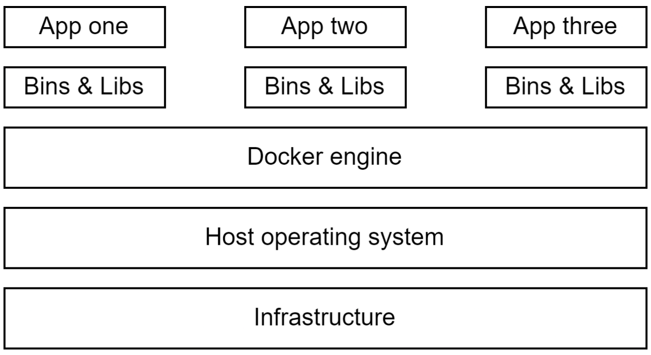


Figure 1.2: Container's structure

Containers are employed to execute specific services and offer a lighter alternative to virtual machines.

1.1.3 Summary

Data centers offer several advantages, including reduced IT costs, enhanced performance, automatic software updates, seemingly limitless storage capacity, improved data reliability, universal document accessibility, and freedom from device constraints. However, it also presents challenges such as the need for a stable internet connection, poor compatibility with slow connections, limited hardware capabilities, privacy and security concerns, increased power consumption, and delays in decision-making.

1.2 Edge computing systems

Edge computing is a distributed computing model in which data processing occurs as close as possible to where the data is generated, improving response times and saving on bandwidth. Processing data near the location where it is generated brings significant advantages in terms of processing latency, reduced data traffic, and increased resilience in case of data connection interruptions. Edge computing systems can be categorized as follows:

- *Cloud*: providing virtualized computing, storage, and network resources with highly elastic capacity.

- *Edge servers*: utilizing on-premises hardware resources for more computationally intensive data processing.
- *IoT and AI-enabled edge sensors*: enabling data acquisition and partial processing at the edge of the network.

Edge computing offers several advantages, including high computational capacity, distributed computing capabilities, enhanced privacy and security, and reduced latency in decision-making. However, it comes with drawbacks such as the requirement for a power connection and dependence on a connection with the Cloud.

1.2.1 Embedded PC

An embedded system refers to a computer system that comprises a computer processor, computer memory, and input/output peripheral devices, all serving a specific function within a larger mechanical or electronic system. Advantages of this approach include its pervasiveness in computing, high-performance units, availability of development boards, ease of programming similar to personal computers, and the support of a large community. On the other hand, it has disadvantages such as relatively high power consumption and the necessity for some hardware design work to be done.

1.2.2 Internet of things

The internet of things (IoT) encompasses devices equipped with sensors, processing capabilities, software, and other technologies. These devices are designed to connect and exchange data with other devices and systems over the internet or other communication networks. Advantages of IoT devices include their high pervasiveness, wireless connectivity, battery-powered operation, low costs, and their ability to sense and actuate. However, these devices also come with several disadvantages, such as their low computing ability, constraints on energy usage, limitations in memory (RAM/FLASH), and difficulties in programming them.

CHAPTER 2

Hardware infrastructures

2.1 Introduction

Over the past few decades, there has been a significant shift in computing and storage, transitioning from PC-like clients to smaller, often mobile devices, coupled with expansive internet services. Concurrently, traditional enterprises are increasingly embracing cloud computing.

This shift offers several user experience improvements, including ease of management and ubiquitous access.

For vendors, Software-as-a-Service (SaaS) facilitates faster application development, making changes and improvements easier. Moreover, software fixes and enhancements are streamlined within data centers, rather than needing updates across millions of clients with diverse hardware and software configurations. Hardware deployment is simplified to a few well-tested configurations.

Server-side computing enables the swift introduction of new hardware devices, such as hardware accelerators or platforms, and supports many application services running at a low cost per user. Certain workloads demand substantial computing capability, making data centers a more natural fit compared to client-side computing.

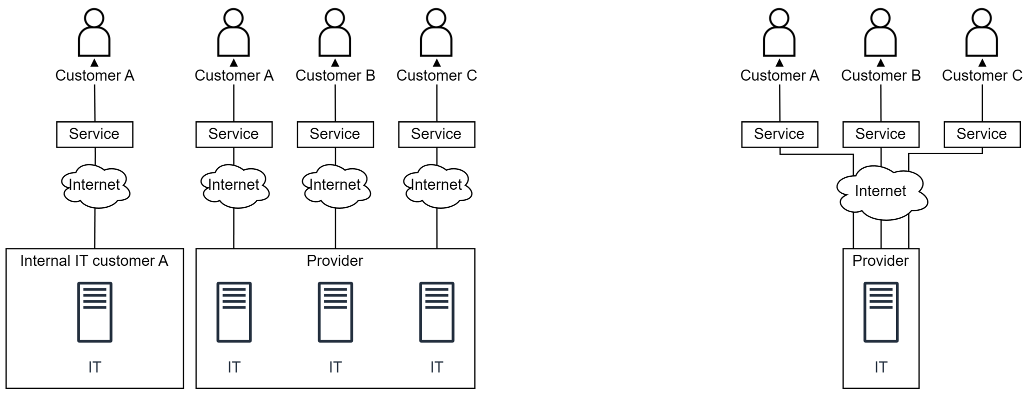
2.1.1 Warehouse-scale computers

The rise of server-side computing and the widespread adoption of internet services have given rise to a new class of computing systems known as warehouse-scale computers (WSCs). In warehouse-scale computing, the program:

- Operates as an internet service.
- Can comprise tens or more individual programs.
- These programs interact to deliver complex end-user services like email, search, maps, or machine learning.

Data centers are facilities where numerous servers and communication units are housed together due to their shared environmental needs, physical security requirements, and for the sake of streamlined maintenance. Traditional data centers typically accommodate a considerable number of relatively small- or medium-sized applications. Each application operates on a dedicated hardware infrastructure, isolated and safe guarded against other systems within the same facility. These applications typically do not communicate with one another. Moreover, these data centers host hardware and software for multiple organizational units or even different companies.

In contrast, warehouse-scale computers are owned by a single organization, employ a relatively uniform hardware and system software platform, and share a unified systems' management layer.



(a) Data center (b) Data warehouse

Figure 2.1: Structures of data centers and data warehouses

Warehouse-scale computers operate a reduced quantity of highly expansive applications, often internet services. Their shared resource management infrastructure affords considerable deployment flexibility. Designers are driven by the imperatives of homogeneity, single-organization control, and cost efficiency, prompting them to adopt innovative approaches in crafting WSCs.

Originally conceived for online data-intensive web workloads, warehouse-scale computers have expanded their capabilities to drive public cloud computing systems, such as those operated by Amazon, Google, and Microsoft. These public clouds do accommodate numerous small applications, resembling a traditional data center setup. However, all these applications leverage virtual machines or containers and access shared, large-scale services for functionalities like block or database storage and load balancing, aligning seamlessly with the WSC model.

The software operating on these systems is designed to run on clusters comprising hundreds to thousands of individual servers, far surpassing the scale of a single machine or a single rack. The machine itself constitutes this extensive cluster or aggregation of servers, necessitating its consideration as a single computing unit.

2.1.2 Geographical distribution of data centers

Frequently, multiple data centers serve as replicas of the same service, aiming to reduce user latency and enhance serving throughput. Requests are typically processed entirely within one data center.

Definition (*Geographic area*)**.** Geographic areas partition the world into sectors, each defined by geopolitical boundaries.

Within each geographic area, there are at least two computing regions. Customers perceive regions as a more detailed breakdown of the infrastructure. Notably, multiple data centers within the same region are not externally visible. The perimeter of each computing region is defined by latency (with a round trip latency of two milliseconds), which is too far for synchronous replication but sufficient for disaster recovery.

Definition (*Availability zone*)**.** Availability zones represent more granular locations within a single computing region.

They enable customers to operate mission-critical applications with high availability and fault tolerance to datacenter failures by providing fault-isolated locations with redundant power, cooling, and networking. Application-level synchronous replication among availability zones is implemented, with a minimum of three zones being adequate for ensuring quorum.

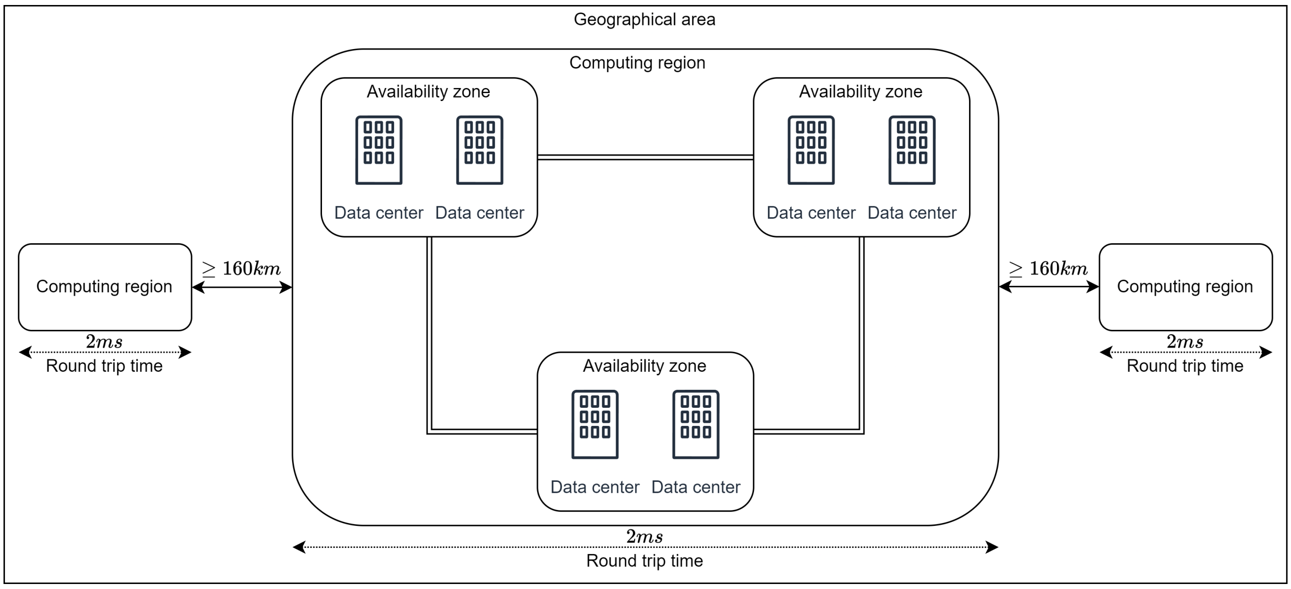


Figure 2.2: Geographical area structure

2.2 Warehouse-scale computers

Services offered by warehouse-scale computers need to ensure high availability, usually targeting a minimum uptime of 99.99%, equating to one hour of downtime per year. Maintaining flawless operation is challenging when managing a vast array of hardware and system software components. Therefore, WSC workloads must be crafted to gracefully handle numerous component faults, minimizing or eliminating any adverse effects on service performance and availability.

2.2.1 Architecture

While the hardware implementation of WSCs may vary considerably, the architectural organization of these systems remains relatively consistent.

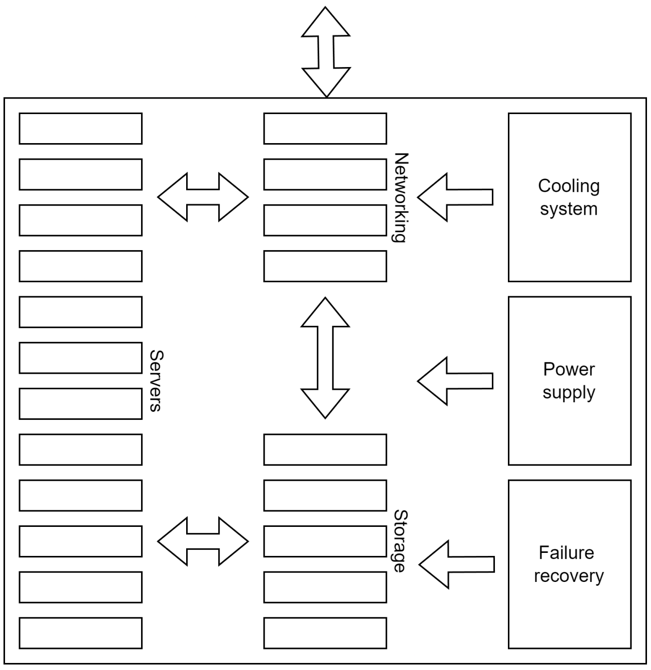


Figure 2.3: Architecture of warehouse-scale computers

Servers Servers resemble standard PCs but are designed with form factors that enable them to fit into racks, such as rack (one or more), blade enclosure format, or tower configurations. They can vary in terms of the number and type of CPUs, available RAM, locally attached disks (HDD, SSD, or none), as well as the inclusion of other specialized devices like GPUs, DSPs, and coprocessors.

Storage Disks and Flash SSDs serve as the fundamental components of contemporary WSC storage systems. These devices are integrated into the data-center network and overseen by advanced distributed systems. Examples of storage configurations include Direct Attached Storage (DAS), Network Attached Storage (NAS), Storage Area Networks (SAN), and RAID controllers.

Networking Communication equipment facilitates network interconnections among devices within the system. These include hubs, routers, DNS or DHCP servers, load balancers, switches, firewalls, and various other types of devices.

2.3 Servers

Servers housed within individual shelves form the foundational components of warehouse-scale computers. These servers are interconnected through hierarchical networks and are sustained by a shared power and cooling infrastructure.

They resemble standard PCs but are designed with form factors that enable them to fit into shelves, such as rack (one or more), blade enclosure format, or tower configurations. Servers are typically constructed in a tray or blade enclosure format, housing the motherboard, chipset, and additional plug-in components.

Motherboard The motherboard offers sockets and plug-in slots for installing CPUs, memory modules (DIMMs), local storage (such as Flash SSDs or HDDs), and network interface cards (NICs) to meet various resource requirements.

The chipset and additional components of these systems include:

- *Number and type of CPUs*: these systems support a varying number of CPU sockets, typically ranging from 1 to 8, and can accommodate processors such as Intel Xeon Family, AMD EPYC, etc.
- *Available RAM*: they offer a range of DIMM slots, typically from 2 to 192, for memory modules.
- *Locally attached disks*: these systems come with between 1 and 24 drive bays for local storage. They support both HDD and SSD options, with specific configurations detailed in lectures. Users can choose between SAS for higher performance (albeit at a higher cost) or SATA for entry-level servers.
- *Other special purpose devices*: these systems can integrate specialized components like GPUs or TPUs, with support for various models including NVIDIA Pascal, Volta, A100, etc.
- *Form factor*: they come in different sizes, ranging from 1U to 10U, and can also be configured as tower systems.

2.3.1 Racks

Racks serve as specialized shelves designed to house and interconnect all IT equipment. They are utilized for storing rack servers and are measured in rack units, denoted as U, with one rack unit (1U) equivalent to 44.45 mm (1.75 inches). One of the advantages of using racks is that they allow designers to stack additional electronic devices alongside the servers. IT equipment must adhere to specific sizes to fit into the rack shelves.

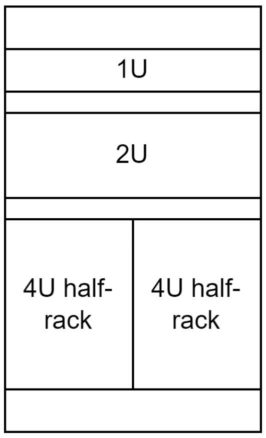


Figure 2.4: Rack elements dimensions

The rack serves as the shelf that securely holds together tens of servers. It manages the shared power infrastructure, including power delivery, battery backup, and power conversion. The width and depth of racks vary across WSCs, with some adhering to classic 19-inch width and 48-inch depth, while others may be wider or shallower. It is often convenient to connect network cables at the top of the rack, leading to the adoption of rack-level switches appropriately called Top of Rack (TOR) switches.

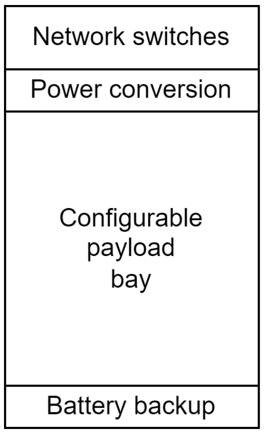


Figure 2.5: Rack structure

Advantages of using racks include the ease of failure containment, as identifying and replacing malfunctioning servers is a straightforward process. Additionally, racks offer simplified cable management, making it efficient to organize cables. Moreover, they provide cost-effectiveness by offering computing power and efficiency at relatively lower costs.

However, there are also drawbacks to consider. The high overall component density within racks leads to increased power usage, requiring additional cooling systems. Consequently, this results in higher power consumption. Furthermore, maintaining multiple devices within racks becomes considerably challenging as the number of racks increases, making maintenance a complex task.

2.3.2 Towers

A tower server closely resembles a traditional tower PC in appearance and functionality. Its advantages include scalability and ease of upgrade, allowing for customization and upgrades as needed. Tower servers are also cost-effective, often being the cheapest option among server types. Additionally, due to their low overall component density, they cool down easily.

However, tower servers have their drawbacks. They consume a significant amount of physical space and can be difficult to manage. Furthermore, while they provide a basic level of performance suitable for small businesses with a limited number of clients, they may not meet the performance needs of larger enterprises. Additionally, complicated cable management is a challenge, as devices are not easily routed together within the tower server setup.

2.3.3 Blade servers

Blade servers represent the latest and most advanced type of servers available in the market today. They can be described as hybrid rack servers, with servers housed inside blade enclosures forming a blade system. The primary advantage of blade servers lies in their compact size, making them ideal for conserving space in data centers.

Advantages of blade servers include their small size and form-factor, requiring minimal physical space. They also simplify cabling tasks compared to tower and rack servers, as the cabling involved is significantly reduced. Additionally, blade servers offer centralized management, allowing all blades to be connected through a single interface, which facilitates easier maintenance and monitoring. Furthermore, blade servers support load balancing, failover, and scalability through a uniform system with shared components, enabling simple addition or removal of servers.

However, blade servers do have drawbacks. Their initial configuration or setup can be expensive and may require significant effort, particularly in complex environments. Blade servers often entail vendor lock-in, as they typically require the use of manufacturer-specific blades and enclosures, limiting flexibility and potentially increasing long-term costs. Moreover, due to their high component density and powerful nature, blade servers require special accommodations to prevent overheating, necessitating well-managed heating, ventilation, and air conditioning systems in data centers hosting blade servers.

2.3.4 Building structure

The IT equipment is housed in corridors and arranged within racks. It's important to note that server racks are never positioned back-to-back. The corridors where servers are situated are divided into cold aisles, providing access to the front panels of the equipment, and warm aisles, where the back connections are located. Cold air flows from the front (cool aisle), cooling down the equipment, and exits the room through the back (warm aisle).

2.4 Hardware accelerators

The rate of complexity in technology doubles approximately every 3.5 months, contrasting with Moore's Law, which historically predicted a doubling of computing capacity every 18-24 months. However, in the present era, Moore's Law has slowed down, with computing capacity doubling every 4 years or longer.

The emergence and widespread adoption of deep learning models have ushered in a new era where specialized hardware plays a crucial role in powering a wide range of machine learning solutions. Since 2013, the computational requirements for AI training have been doubling every 3.5 months, significantly outpacing the traditional Moore's Law projection of 18-24 months. To meet the escalating computational demands of deep learning tasks, WSCs are deploying specialized accelerator hardware such as GPUs, TPUs, and FPGAs. These accelerators are optimized to handle the intensive processing required for training and inference tasks in deep learning applications.

2.4.1 Graphical processing unit

GPUs have revolutionized data processing by enabling data-parallel computations, where the same program can be executed simultaneously on numerous data elements in parallel. This parallelization technique is particularly effective for scientific codes, which are often structured around matrix operations.

Harnessing the power of GPUs typically involves using high-level languages like CUDA and OpenCL. Compared to traditional CPU-based processing, GPUs can deliver remarkable speed boosts, sometimes performing computations up to 1000 times faster.

2.4.2 Neural networks

Neural networks are a computational model inspired by the human brain, specifically the perceptron. They comprise interconnected nodes, or neurons, organized in layers to process and analyze data. Neural networks are utilized to learn data representation, enabling them to learn features and function as classifiers or regressors.

Neural networks have a rich history dating back to the 1940s, with notable developments occurring in the 1980s. In recent years, there has been a resurgence of interest in neural networks due to factors such as increased data availability and computational power, with some regarding them as among the top breakthroughs in 2013.

In natural neurons, information is transmitted through chemical mechanisms. Dendrites gather charges from synapses, which can be either inhibitory or excitatory. Once a threshold is reached, the accumulated charge is released, causing the neuron to fire. Like its biological counterpart, an artificial neuron receives input signals, which are weighted and summed. This sum undergoes an activation function, determining the output signal of the neuron:

$$h_j(x|w,b) = h_j\left(\sum_{i=1}^{I} w_j x_i - b\right) = h_j\left(\sum_{i=0}^{I} w_i x_i\right) = h_j\left(w^T x\right)$$

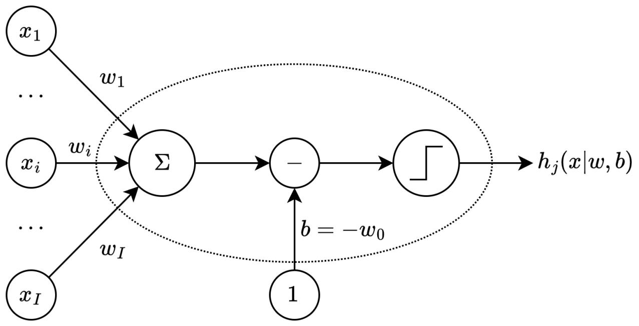


Figure 2.6: Artificial neuron

Neural networks are structured into at least three layers:

- *Input layer*: this is where data is initially introduced into the network.
- *Hidden layers*: intermediate layers that process and transform the input data.
- *Output layers*: the final layer that provides the network's ultimate results or predictions.

These layers are interconnected through weighted connections. Activation functions, which must be differentiable, determine the output of each neuron. Importantly, the output of a neuron is influenced solely by the inputs from the preceding layer.

Neural networks are inherently non-linear models. Their behavior is characterized by various factors, including the number of neurons, the choice of activation functions, and the specific values of the weights assigned to connections within the network. These factors collectively determine the network's ability to learn and make accurate predictions from the input data.

Learning in neural networks involves several key processes:

- *Activation functions*: neurons within the network make decisions based on input data through activation functions.
- *Weights*: connections between neurons are adjusted during training, either strengthened or weakened, to optimize performance.

Neural networks learn from historical data and examples provided during training. This process often involves backpropagation, which utilizes techniques like gradient descent and the chain rule to calculate errors and adjust the model accordingly. Errors are evaluated and used to refine the network's parameters, improving its ability to make accurate predictions or classifications.

Neural networks come in various types, each tailored for specific tasks:

- *Feed forward neural network*: standard neural network where information flows in one direction, from input to output.
- *Convolutional neural networks* (CNNs): designed for processing grid-like data such as images, with specialized layers for feature extraction and pattern recognition.
- *Recurrent neural networks* (RNNs): suitable for sequential data where past information influences the present, often used in natural language processing and time series analysis.

These types of neural networks find applications across diverse domains:

- *Image recognition*: utilized in technologies like FaceNet and YOLO for tasks such as facial recognition, object detection, and instance segmentation.
- *Natural language processing*: leveraged by models like BERT and GPT for applications like chatbots, sentiment analysis, and speech-to-text translation.

The potential for innovation and future development in neural networks is extensive. With the rapid growth of neural network applications, they are continuously expanding into new and diverse areas such as social media, aerospace, e-commerce, finances, and beyond. Additionally, advancements in generative AI are driving innovation in fields like art, music, and content generation, allowing for the creation of novel content through sophisticated generative models.

Neural networks and GPUs GPUs are commonly employed for training neural networks. However, the performance of such a synchronous system is constrained by the slowest learner and the slowest messages transmitted through the network. As the communication phase is a critical component, a high-performance network is essential for expediting parameter reconciliation across learners.

In configurations involving GPUs, a CPU host is typically connected to a PCIe-attached accelerator tray housing multiple GPUs. Within this tray, GPUs are interconnected using high-bandwidth interfaces such as NVlink, facilitating efficient communication and data exchange between the GPUs.

In the A100 GPU, each NVLink lane supports a data rate of $50 \times 4\, Gb/s$ in each direction. The total number of NVLink lanes increases from six lanes in the V100 GPU to 12 lanes in the A100 GPU, resulting in a total bandwidth of $600\, GB/s$. With the H100 GPU, each GPU can have up to 18 lanes, leading to a total bandwidth of $900\, GB/s$.

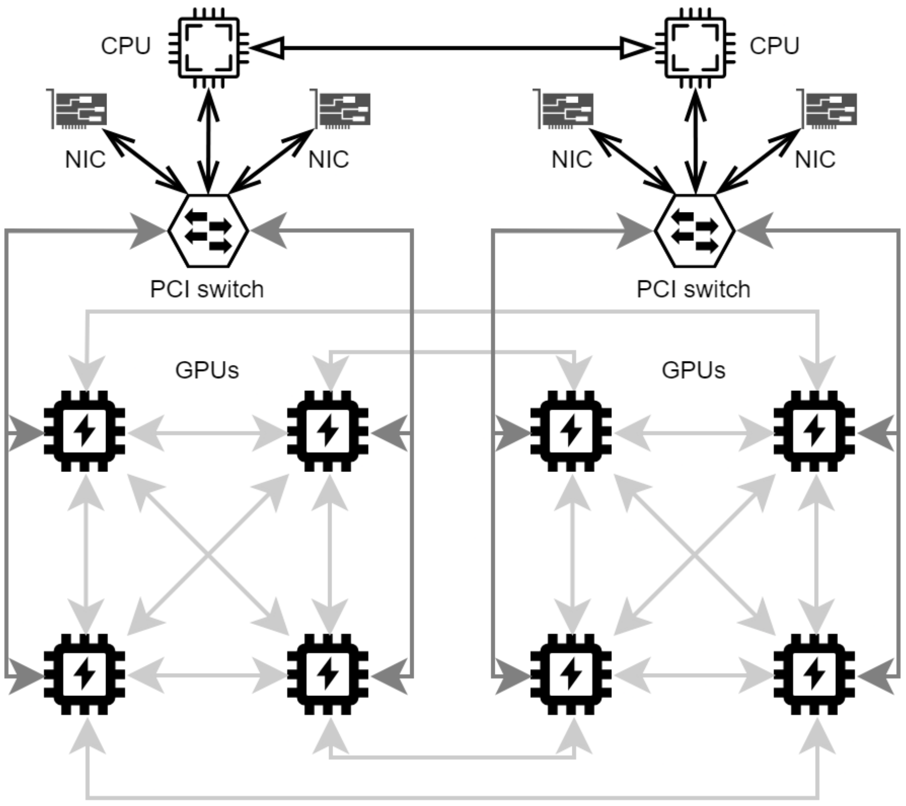


Figure 2.7: Neural network training

2.4.3 Tensor processing unit

Although GPUs are well-suited to machine learning (ML), they are still considered relatively general-purpose devices. However, in recent years, designers have increasingly specialized them into ML-specific hardware. These custom-built integrated circuits are developed specifically for machine learning tasks and are tailored to frameworks like TensorFlow.

These ML-specific hardware units have been powering Google data centers since 2015, alongside CPUs and GPUs. In TensorFlow, the basic unit of operation is a Tensor, which is an n-dimensional matrix.

Tensor Processing Units (TPUs) are extensively used for both training and inference tasks. The main versions of Tensor Processing Units are:

- *TPUv1*: inference-focused accelerator connected to the host CPU through PCIe links.

- *TPUv2*: supports both training and inference operations, providing enhanced flexibility and performance. In TPUv2, each Tensor Processing Unit (TPU) core consists of an array for matrix computations unit (MXU) and a connection to high bandwidth memory (HBM), which is used to store parameters and intermediate values during computation. Specifically, TPUv2 is equipped with 8 GiB of HBM for each TPU core, ensuring ample storage capacity. Additionally, there is one MXU allocated for each TPU core, facilitating efficient matrix computations. The hardware layout of TPUv2 comprises 4 chips, with each chip housing 2 cores. This configuration optimizes performance and scalability for machine learning tasks. Within a rack, numerous TPUv2 accelerator boards are interconnected via a custom high-bandwidth network, facilitating a collective ML compute power of 11.5 petaflops. This network's high bandwidth capabilities ensure swift parameter reconciliation with precisely controlled tail latencies. A TPU Pod, consisting of 64 units, can accommodate up to 512 total TPU cores and 4 TB of total memory. This configuration maximizes both processing power and memory capacity for demanding machine learning tasks.

- *TPUv3*: Google's initial venture into liquid-cooled accelerators within their data centers. With a performance boost of 2.5 times compared to its predecessor, TPUv2, this supercomputing-class computational power enables various new ML capabilities such as AutoML and rapid neural architecture search. The TPUv3 Pod offers an impressive maximum configuration, boasting 256 devices, totaling 2048 TPU v3 cores. This setup delivers an astounding 100 petaflops of computing power and accommodates 32 TB of TPU memory, ensuring enhanced performance and scalability for demanding machine learning tasks.

- *TPUv4*: announced in June 2021, TPUv4 marks the latest advancement in Google's Tensor Processing Unit lineup. Initially deployed to bolster Google's in-house services, TPUv4 is not yet available as a cloud service. A single TPUv4 pod comprises 4096 devices, showcasing a remarkable performance increase of approximately 2.7 times compared to its predecessor, TPUv3. With computing capabilities equivalent to around 10 million laptops, TPUv4 sets a new standard for high-performance machine learning hardware.

- *TPUv5*: it has two versions (v5e and v5p). The latter one was announced in December 2023 and has been available since early this year. Compared to TPUv4, v5p can train large language models like GPT3-175B 2.8 times faster. On the other hand, although v5e is slower, it offers more relative performance per dollar compared to v5p.

On the other hand, AWS offers Trainium2, specifically optimized for Large Language Models (LLMs). Additionally, they provide Graviton4, based on ARM architecture, offering 30% greater energy efficiency for general AI tasks. For AI inference, AWS offers Inferentia2, which boasts 2.3 times higher throughput and up to 70% lower cost per inference compared to comparable EC2 instances.

2.4.4 Field programmable gate array

FPGAs (Field-Programmable Gate Arrays) are arrays of logic gates that users can program or configure in the field, without relying on the original designers. These devices consist of carefully designed and interconnected digital sub-circuits that efficiently implement common functions, providing extremely high levels of flexibility. The individual digital sub-circuits within FPGAs are known as configurable logic blocks (CLBs).

VHDL and Verilog are hardware description languages used to describe hardware. They enable users to create textual representations of hardware components and their interconnections. HDL code resembles a schematic, using text to define components and establish connections between them.

	Advantages	**Disadvantages**
CPU	Easily programmable and compatible Rapid design space Efficiency	Simple models Small training sets
GPU	Parallel execution	Limited flexibility
TPU	Fast for ML	Limited flexibility
FPGA	High performance Low cost Low power consumption	Limited flexibility High-level synthesis

2.5 Storage solutions

During the 80s and 90s, data was mainly produced by humans. However, in contemporary times, machines generate data at an unprecedented pace.

Various forms of media such as images, videos, audios, and social media platforms have emerged as significant sources of big data. Moreover, the widespread deployment of sensors, surveillance cameras, digital medical imaging devices, and other technologies has further accelerated the accumulation of data. This data deluge is further augmented by the integration of Industry 4.0 technologies and artificial intelligence, ushering in a new era of data-centricity and innovation.

The trend favors a centralized storage approach, which offers several advantages: By limiting redundant data, it streamlines storage efficiency. Automation of replication and backup processes ensures data reliability and security. This centralized model ultimately leads to reduced management costs.

The current trend leans towards favoring a centralized storage strategy. This approach helps in minimizing redundant data, automating replication and backup processes, and ultimately reducing management costs.

HDDs have long dominated the storage technology landscape, characterized by magnetic disks with mechanical interactions. However, recent technological advancements have introduced SSDs, which differ significantly. SSDs have no mechanical or moving parts and are constructed using transistors, specifically NAND flash-based devices. Furthermore, NVMe (Non-Volatile Memory Express) has emerged as the latest industry-standard for running PCIe SSDs. Despite these innovations, tapes persist as a reliable storage solution unlikely to fade away.

Certain large storage servers employ SSDs as caches for multiple HDDs. Similarly, some latest-generation main boards integrate a small SSD with a larger HDD to enhance disk speed. Additionally, certain HDD manufacturers produce Solid State Hybrid Disks (SSHDs) that combine a small SSD with a large HDD within a single unit.

2.5.1 Disk drives

In the view of an operating system, disks are perceived as a compilation of data blocks capable of independent reading or writing. To facilitate their organization and management, each block is assigned a unique numerical address known as the Logical Block Address (LBA). Usually, the operating system groups these blocks into clusters, which serve as the smallest unit that the OS can read from or write to on a disk. Cluster sizes typically vary from one disk sector (512 bytes) to 128 sectors (64 kilobytes).

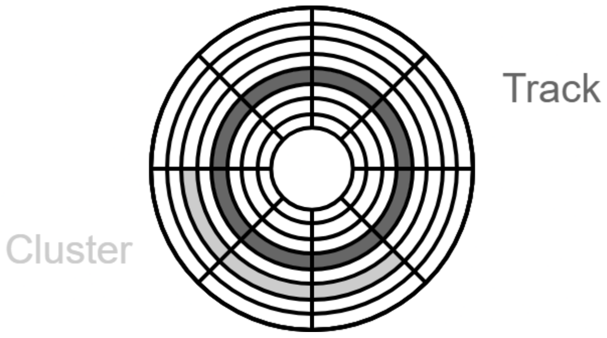


Figure 2.8: Hard disk drive structure

Clusters encompass two crucial components:

1. *File data*: this refers to the actual content stored within files.
2. *Metadata*: this includes essential information necessary for supporting the file system, which consists of:
   - File names.
   - Directory structures and symbolic links.
   - File size and file type.
   - Creation, modification, and last access dates.
   - Security information such as owners, access lists, and encryption details.
   - Links directing to the Logical Block Address (LBA) where the file content is located on the disk.

Hence, the disk can harbor various types of clusters:

- *Fixed-position metadata*: reserved to bootstrap the entire file system.
- *Variable-position metadata*: used to store the folder structure.
- *File data*: housing the actual content of files.
- *Unused space*: available to accommodate new files and folders.

Reading To read a file, the process involves:

1. Accessing the metadata to locate its blocks.
2. Accessing the blocks to read its content.

Writing The process of writing a file involves the following steps:

1. Accessing the metadata to locate free space.
2. Writing the data into the assigned blocks.

Since the file system operates on clusters, the actual space occupied by a file on a disk is always a multiple of the cluster size, denoted by c. This is given by the formula:

$$a = \left\lfloor \frac{s}{c} \right\rfloor \cdot c$$

Here, a is the actual size on disk, s is the file size, and c is the cluster size. The wasted disk space, denoted by w, due to organizing the file into clusters can be computed as:

$$w = a - s$$

This unused space is referred to as the internal fragmentation of files.

Example:
Let's consider a hard disk with a cluster size of 8 bytes and a file with a size of 27 bytes. The actual size on the disk would be:

$$a = \left\lfloor \frac{s}{c} \right\rfloor \cdot c = \left\lfloor \frac{27}{8} \right\rfloor \cdot 8 = 32\, byte$$

And the wasted disk space would be:

$$w = a - s = 32 - 27 = 5\, byte$$

Deleting Deleting a file involves updating the metadata to indicate that the blocks previously allocated to the file are now available for use by the operating system. However, the deletion process does not physically remove the data from the disk. Instead, the data remains intact until new data is written to the same clusters. In other words, when new files are written to the same clusters, the old data is overwritten by the new data.

External fragmentation As the lifespan of the disk progresses, there may not be sufficient contiguous space available to store a file. In such instances, the file is divided into smaller chunks and distributed across the free clusters scattered throughout the disk. This process of splitting a file into non-contiguous clusters is known as external fragmentation. It's important to note that external fragmentation can significantly degrade the performance of a hard disk drive (HDD).

2.5.2 Hard disk drives

A hard disk drive (HDD) utilizes rotating disks (platters) coated with magnetic material for data storage. Information can be accessed in a random-access manner, allowing for the storage or retrieval of individual data blocks in any order, rather than sequentially. The HDD comprises one or more rigid ("hard") rotating disks (platters) with magnetic heads positioned on a moving actuator arm, facilitating the reading and writing of data onto the surfaces.

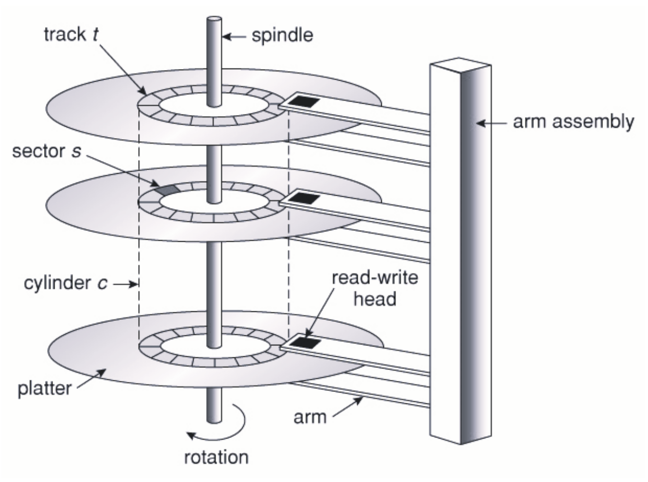


Figure 2.9: Hard disk drive structure

Externally, hard drives present a multitude of sectors (blocks), typically comprising 512 or 4096 bytes each. Each sector write is indivisible and comprises a header along with an error correction code. However, the writing of multiple sectors can be susceptible to interruption, resulting in what is known as a torn write scenario, where only a portion of a multi-sector update is successfully written to the disk.

Regarding drive geometry, sectors are organized into tracks, with a cylinder representing a specific track across multiple platters. These tracks are arranged in concentric circles on the platters, and a disk may consist of multiple double-sided platters. The drive motor maintains a constant rate of rotation for the platters, typically measured in revolutions per minute (RPM).

Present-day technology specifications include:

- *Diameter*: approximately 9 cm (3.5 inches), accounting for two surfaces.
- *Rotation speed*: ranges from 7200 to 15000 revolutions per minute (RPM).
- *Track density*: reaching up to 16,000 Tracks Per Inch (TPI).
- *Heads*: can be parked either close to the center or towards the outer diameter, particularly in mobile drives.
- *Disk buffer cache*: embedded memory within a hard disk drive, serving as a buffer between the disk and the computer.

Cache Numerous disks integrate caches, often referred to as track buffers, which typically consist of a small amount of RAM ranging from 16 to 64 MB. These caches serve several purposes:

1. *Read caching*: This minimizes read delays caused by seeking and rotation.
2. *Write caching*: There are two primary types of write caching:
    - Write back cache: In this mode, the drive acknowledges writes as complete once they have been cached. However, this feature can be risky as it may lead to an inconsistent state if power is lost before the write back event.

- Write through cache: Here, the drive acknowledges writes as complete only after they have been written to the disk.

Today, some disks incorporate flash memory for persistent caching, resulting in hybrid drives.

Standard disk interfaces The standard disk interfaces are:

- *ST-506*: an ancient standard characterized by commands (read/write) and addresses in cylinder/head/sector format stored in device registers.
- *ATA*: advanced Technology Attachment, evolved from ST-506, and eventually standardized as IDE (Integrated Drive Electronics).
- *IDE*: integrated Drive Electronics, also known as Parallel ATA, used primarily in older systems.
- *SATA*: serial ATA, the current standard for connecting storage devices, featuring recent versions that support Logical Block Addresses (LBA).
- *SCSI*: a packet-based interface, akin to TCP/IP, with devices translating Logical Block Addresses (LBA) to internal formats such as cylinder/head/sector. It is transport-independent and utilized in various devices including USB drives, CD/DVD/Blu-ray drives, and Firewire connections.
- *iSCSI*: a variant of SCSI over TCP/IP and Ethernet, enabling storage over network connections.

Transfer time With hard disk drives we can have four types of delay:

- *Rotational delay*: this delay represents the time it takes for the desired sector to rotate under the read head. It's influenced by the disk's rotational speed, measured in revolutions per minute (RPM). The full rotation delay is calculated as:

  $$R = \frac{1}{DiskRPM}$$

  While the average rotation time is typically half of the full rotation delay, given by:

  $$T_{rotation} = \frac{60 \cdot R}{2}$$

  in seconds.
- *Seek delay*: this delay refers to the time it takes for the read head to move to a different track on the disk. The seek time includes components such as acceleration, coasting, deceleration, and settling. Modeling seek time with a linear dependency on distance, we find that the average seek time $t_{seek_{AVG}}$ is one-third of the maximum seek time $t_{seek_{MAX}}$.
- *Transfer time*: this delay is the time required to read or write bytes of data from or to the disk. It encompasses the final phase of the input/output (I/O) process, considering the data being read from or written to the disk surface. This includes the time for the head to pass over the sectors and the I/O transfer process, taking into account factors such as rotation speed and storage density.

- *Controller overhead*: this delay encompasses the additional time incurred for managing the requests sent to the disk controller. It involves tasks such as buffer management for data transfer and the time taken to send interrupts.

The service time, also known as I/O time, is determined by summing up various elements:

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} + T_{overhead}$$

When factoring in the queue time, which represents the wait time for the resource, we can calculate the response time $\tilde{R}$ as follows:

$$\tilde{R} = T_{queue} + T_{I/O}$$

Here, T_{queue} is influenced by factors such as queue length, resource utilization, the mean and variance of disk service time distribution, and the distribution of request arrivals.

Example:
Let's consider a hard disk with the following specifications: a read/write sector size of 512 bytes (0.5 KB), a data transfer rate of 50 MB/s, a rotation speed of 10000 RPM, a mean seek time of 6 ms, and an overhead controller time of 0.2 ms. The service time is calculated as:

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer} + T_{overhead} = 6 + T_{rotation} + T_{transfer} + 0.2$$

Given:

$$T_{rotation} = 60 \cdot \frac{1000}{2 \cdot DiskRPM} = 60 \cdot \frac{1000}{2 \cdot 10000} = 3.0 \text{ ms}$$

$$T_{transfer} = 0.5 \cdot \frac{1000}{50 \cdot 1024} = 0.01 \text{ ms}$$

Hence:

$$T_{I/O} = 6 + T_{rotation} + T_{transfer} + 0.2 = 6 + 3 + 0.01 + 0.2 = 9.21 \text{ ms}$$

The previous analysis of service times reflects a highly pessimistic scenario, assuming the worst-case conditions where disk sectors are extensively fragmented. This scenario occurs when files are tiny, typically occupying just one block, or when the disk suffers from significant external fragmentation. Consequently, each access to a sector necessitates both rotational latency and seek time.

However, in many practical situations, these extreme conditions are not encountered. Files tend to be larger than a single block and are stored contiguously, mitigating the need for frequent seeks and rotational latency.

We can assess the data locality of a disk by quantifying the percentage of blocks that can be accessed without requiring seek or rotational latency:

$$T_{I/0} = (1 - DL)(T_{seek} + T_{rotation}) + T_{transfer} + T_{overhead}$$

Example:
Let's revisit the previous scenario but now with a data locality of 75%. In this case, the time is computed as:

$$T_{I/0} = (1 - DL)(T_{seek} + T_{rotation}) + T_{transfer} + T_{overhead} = (0.25)(6 + 3) + 0.01 + 0.2 = 2.46 \text{ ms}$$

Here, DL represents the data locality factor. Substituting the given values, we find the resulting time to be 2.46 ms.

Disk scheduling Caching is instrumental in enhancing disk performance, although it cannot fully compensate for inadequate random access times. The core concept lies in reordering a queue of disk requests to optimize performance. Estimating the request length becomes viable by considering the data's position on the disk. Various scheduling algorithms facilitate this optimization process:

- *First come, first served* (FCFS): this is the most basic scheduler, serving requests in the order they arrive. However, it often results in a significant amount of time being spent seeking.
- *Shortest seek time first* (SSTF): this scheduler minimizes seek time by always selecting the block with the shortest seek time. SSTF is optimal and relatively easy to implement, but it may suffer from starvation.
- *SCAN*, also known as the elevator algorithm: in SCAN, the head of the disk sweeps across the disk, servicing requests in a linear order. This approach offers reasonable performance and avoids starvation, but average access times are higher for requests at the extremes of the disk.
- *C-SCAN*: Similar to SCAN, but it only services requests in one direction, providing fairer treatment to requests. However, it typically exhibits worse performance compared to SCAN.
- *C-LOOK*: this is akin to C-SCAN, but with a twist: the head of the disk only moves as far as the last request in the queue, effectively reducing the range of movement.

These scheduling algorithms can be implemented in various ways:

- *Operating system scheduling*: requests are reordered based on Logical Block Address (LBA). However, the OS lacks the ability to account for rotation delay.
- *On-disk scheduling*: the disk possesses precise knowledge of the head and platter positions, allowing for the implementation of more advanced schedulers. However, this approach requires specialized hardware and drivers.
- *Disk command queue*: found in all modern disks, this queue stores pending read/write requests, known as Native Command Queuing (NCQ). The disk may reorder items in the queue to enhance performance.

Joint operating system and on-disk scheduling can introduce challenges and potential issues.

2.5.3 Solid state drive

Solid-state storage devices, in contrast to traditional hard disk drives (HDDs), do not contain any mechanical or moving parts. Instead, they are constructed using transistors, similar to memory and processors. Unlike typical RAM, they can retain information even in the event of power loss. These devices include a controller and one or more solid-state memory components. They are designed to utilize traditional HDD interfaces and form factors, although this may be less true as technology evolves. Furthermore, they offer higher performance compared to HDDs.

An SSD's components are arranged in a grid structure. Each cell within this grid can store varying amounts of data, ranging from one bit (single-level cell) to multiple bits (multi-level cell, triple-level cell, etc.).

Internal organization NAND flash memory is structured into pages and blocks. Each page contains several logical block addresses (LBAs), while a block generally comprises multiple pages, collectively holding approximately $128 - 256\ KB$ of data:

- *Blocks* (or erase blocks): these are the smallest units that can be erased, typically consisting of multiple pages. They can be cleaned using the ERASE command.
- *Pages*: these are the smallest units that can be read/written. They are subunits of an erase block and contain a specific number of bytes that can be read/written in a single operation through the READ or PROGRAM commands. Pages can exist in three states:
  - *Empty* (or ERASED): these pages do not contain any data.
  - *Dirty* (or INVALID): these pages contain data, but this data is either no longer in use or has never been used.
  - *In use* (or VALID): these pages contain data that can be read.

Pages that are empty are the only ones that can be written to. Erasing is limited to pages that are dirty, and this must occur at the block level, where all pages within the block must be dirty or empty. Reading is only meaningful for pages in the "in use" or "valid" state. If there are no empty pages available, a dirty page must be erased. If there are no blocks containing only dirty or empty pages available, special procedures should be followed to gather empty pages across the disk. Resetting the original voltage to neutral is necessary before applying a new voltage to erase the value in flash memory.

It's worth noting that while writing and reading a single page of data from an SSD is possible, an entire block must be deleted to release it. This discrepancy is one of the underlying causes of the write amplification issue. Write amplification occurs when the actual volume of data physically written to the storage media is a multiple of the intended logical amount.

Example:
Let's examine an SSD with these specifications: page size of $4\ KB$, block size of 5 pages, drive size of 1 block, read speed of $2\ KB/s$, and write speed of $1\ KB$/s.

First, let's write a $4 : KB$ text file to the brand-new SSD. The overall writing time will be 4 seconds.

Next, let's write a $8 : KB$ picture file to the almost brand-new SSD. The overall writing time will be 8 seconds.

Now, suppose the text file in the first page is no longer needed.

Finally, let's write a $12 : KB$ picture to the SSD. It will take 24 seconds. In this case, we need to perform the following steps:

- Read the block into the cache.
- Delete the page from the cache.
- Write the new picture into the cache.
- ERASE the old block on the SSD.

> - Write the cache to the SSD.
>
> The operating system only thought it was writing $12 : KB$ of data when, in fact, the SSD had to read $8 : KB$ ($2 : KB/s$) and then write $20 : KB$ ($1 : KB/s$), the entire block. The writing should have taken 12 seconds but actually took $4 + 20 = 24$ seconds, resulting in a write speed of $0.5 : KB/s$ instead of $1 : KB/s$.

As time goes on, write amplification significantly diminishes the performance of an SSD. This degradation occurs due to the wear-out of flash cells, resulting from the breakdown of the oxide layer within the floating-gate transistors of NAND flash memory. During the erasing process, the flash cell is subjected to a relatively high charge of electrical energy.

Each time a block is erased, this high electrical charge progressively degrades the silicon material. After numerous write-erase cycles, the electrical characteristics of the flash cell start to deteriorate, leading to unreliability in operation.

Flash transition layer Establishing a direct mapping between logical and physical pages is not practical. To address this, an SSD component called the FTL (Flash Translation Layer) is utilized to emulate the functionality of a traditional HDD. The FTL manages data allocation and performs address translation efficiently to mitigate the effects of Write Amplification. One method used by the FTL to reduce Write Amplification is through a technique called Log-Structured FTL. This involves programming pages within an erased block sequentially, from low to high pages.

Garbage collection Additionally, the FTL employs garbage collection to recycle pages containing old data (marked as Dirty/Invalid) and wear leveling to evenly distribute write operations across the flash blocks. This ensures that all blocks within the device wear out at approximately the same rate. Garbage collection incurs significant costs due to the necessity of reading and rewriting live data. Ideally, garbage collection should reclaim blocks composed entirely of dead pages. The expense of garbage collection is directly related to the volume of data blocks requiring migration. To address this challenge, several solutions can be implemented:

- Increase device capacity by over provisioning with extra flash storage.
- Postpone cleaning operations to less critical periods.
- Execute garbage collection tasks in the background during periods of lower disk activity.

During background garbage collection, SSDs operate under the assumption that they can identify which pages are invalid. However, a prevalent issue arises as most file systems do not permanently delete data. To address this challenge, a new SATA command called TRIM has been introduced. With TRIM, the operating system informs the SSD about specific logical block addresses (LBAs) that are invalid and can be subjected to garbage collection. Operating system support for TRIM is available in various platforms such as Windows 7, OSX Snow Leopard, Linux 2.6.33, and Android 4.3.

Mapping table The size of the page-level mapping table can be excessively large, particularly in scenarios like a $1\,TB$ SSD where each $4\,KB$ page requires a 4-byte entry, necessitating $1\,GB$ of DRAM for mapping. To mitigate the costs associated with mapping, several approaches have been developed:

- *Block-based mapping*: employing a coarser grain approach to mapping. This approach encounters a write issue: the FTL needs to read a significant amount of live data from the old block and transfer it to a new one.
- *Hybrid mapping*: utilizing multiple tables for mapping. The Flash Translation Layer (FTL) manages two distinct tables:
  - *Log blocks*: for page-level mapping.
  - *Data blocks*: for block-level mapping.

  When searching for a specific logical block, the FTL consults both the page mapping table and the block mapping table sequentially.
- *Page mapping combined with caching*: implementing strategies to exploit data locality and optimize mapping efficiency. The fundamental concept is to cache the active portion of the page-mapped FTL. When a workload primarily accesses a limited set of pages, the translations for those pages are retained in the FTL memory. This approach offers high performance without incurring high memory costs, provided that the cache can accommodate the required working set. However, there is a potential overhead associated with cache misses.

Wear leveling The Flash memory's Erase/Write (EW) cycle is limited, leading to skewness in the cycles that can shorten the SSD's lifespan. It's essential for all blocks to wear out at a similar pace to maintain longevity. While the Log-Structured approach and garbage collection aid in spreading writes, blocks may still contain cold data. To address this, the FTL periodically reads all live data from these blocks and rewrites it elsewhere. However, wear leveling can increase the SSD's write amplification and reduce performance. A simple policy involves assigning each Flash Block an EW cycle counter, ensuring that the difference between the maximum and minimum EW cycles remains below a certain threshold, denoted as e.

Summary Flash-based SSDs are increasingly common in laptops, desktops, and datacenter servers, despite their higher cost compared to conventional HDDs. Flash memory has limitations, including a finite number of write cycles, shorter lifespan, and the need for error correcting codes and over-provisioning (extra capacity). SSDs also exhibit different read/write speeds compared to HDDs. Unlike HDDs, SSDs are not affected by data locality and do not require defragmentation.

The Flash Translation Layer (FTL) is a crucial component of SSDs, responsible for tasks such as data allocation, address translation, garbage collection, and wear leveling. However, the controller often becomes the bottleneck for transfer rates in SSDs.

2.5.4 Comparison

To assess these two memory types, we can utilize two metrics:

- *Unrecoverable Bit Error Ratio* (UBER): this metric indicates the rate of data errors, expressed as the number of data errors per bits read.
- *Endurance rating*: measured in Terabytes Written (TBW), this indicates the total data volume that can be written into an SSD before it's likely to fail. It represents the number of terabytes that can be written while still meeting the specified requirements.

2.6 Storage systems

The storage systems can be classified as:

- *Direct Attached Storage* (DAS): these storage systems are directly linked to a server or workstation, appearing as disks/volumes within the client operating system.
- *Network Attached Storage* (NAS): NAS entails a computer connected to a network, offering file-based data storage services (e.g., FTP, network file system, and SAMBA) to other devices on the network. It is recognized as a File Server by the client operating system.
- *Storage Area Networks* (SAN): SAN comprises remote storage units connected to servers through specific networking technologies (e.g., fiber channel). They are perceived as disks/volumes by the client operating system.

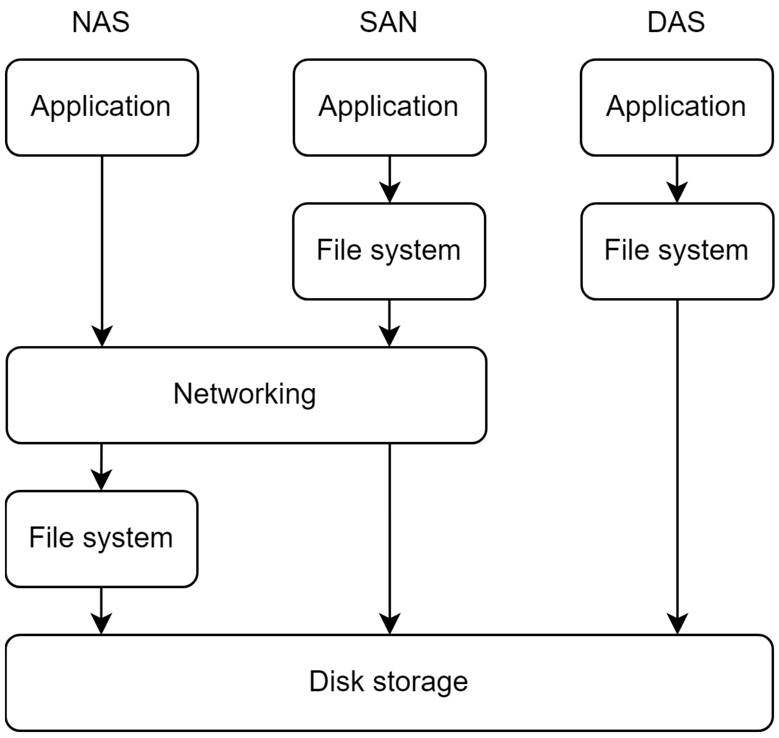


Figure 2.10: Storage system classification

2.6.1 Direct Attached Storage

Direct Attached Storage (DAS) refers to a storage system directly connected to a server or workstation. It serves to distinguish non-networked storage from Storage Area Networks (SAN) and Network Attached Storage (NAS). Key characteristics of DAS include limited scalability and complex manageability. Accessing files from other machines typically requires utilizing the file sharing protocol of the operating system.

DAS can be internal or external; it does not exclusively entail internal drives. Any external disks connected via a point-to-point protocol to a PC can be classified as DAS.

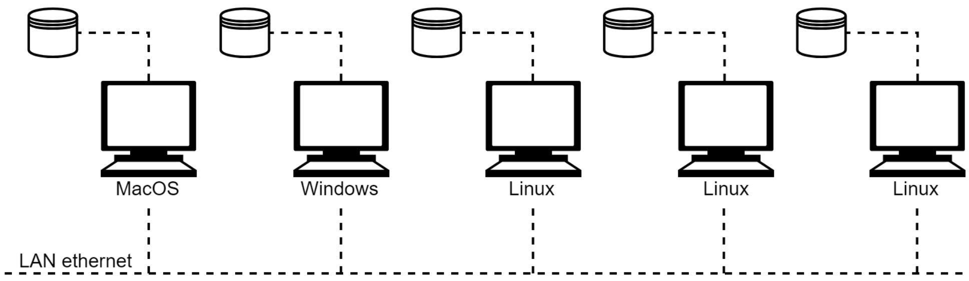


Figure 2.11: Direct Attached Storage architecture

2.6.2 Network Attached Storage

A Network Attached Storage (NAS) unit is a computer connected to a network, offering file-based data storage services exclusively to other devices within the network. NAS systems typically comprise one or more hard disks, often configured into logical redundant storage containers or RAID setups. They provide file-access services to hosts connected via TCP/IP networks through protocols like Networked File Systems or SAMBA. Each NAS element is assigned its own IP address, facilitating individual identification and management.

NAS systems exhibit good scalability, allowing for the addition of devices within each NAS element or the expansion of the number of NAS elements themselves.

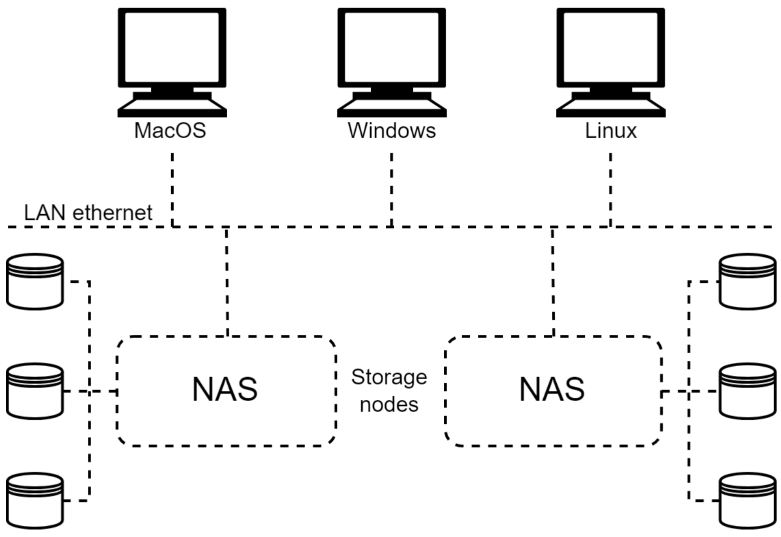


Figure 2.12: Network Attached Storage architecture

NAS and DAS The primary distinctions between Network Attached Storage (NAS) and Direct Attached Storage (DAS) are as follows:

- DAS serves as a mere extension of an existing server and may not necessarily be networked.
- NAS is purposefully designed as a convenient and self-contained solution for sharing files across a network.
- The performance of NAS primarily relies on the speed and congestion levels of the network.

2.6.3 Storage Area Network

Storage Area Networks (SANs) are remote storage units connected to PCs/servers through specific networking technologies. SANs feature a dedicated network solely devoted to accessing storage devices, typically comprising two distinct networks: one for TCP/IP communication and another dedicated network, like fiber channel. They offer high scalability by simply increasing the number of storage devices connected to the SAN network.

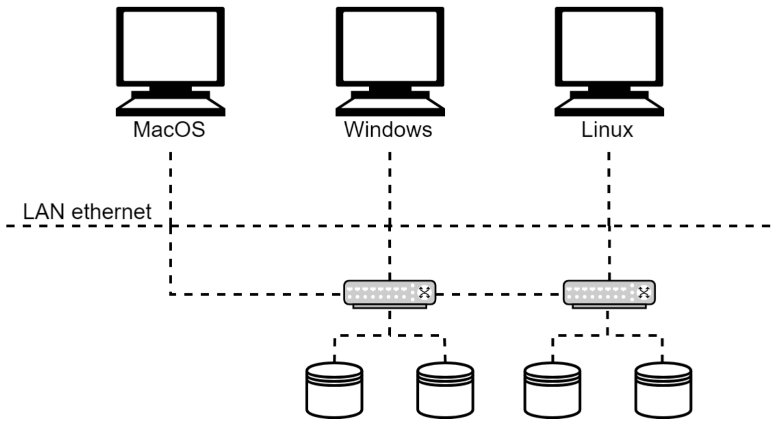


Figure 2.13: Storage Area Network architecture

NAS and SAN AS provides both storage and a file system, contrasting with SAN, which solely offers block-based storage, leaving file system concerns on the client side. A way to loosely differentiate NAS from SAN is:

- NAS presents itself to the client OS as a file server, allowing the client to map network drives to shares on that server.
- A disk accessible through SAN appears to the client OS as a disk, visible in disk and volume management utilities alongside the client's local disks, and available for formatting with a file system.

Traditionally NAS is utilized for low-volume access to a large amount of storage by many users, while SAN serves as the solution for petabytes (10^{12}) of storage and simultaneous access to files, such as streaming audio/video.

2.6.4 Summary

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

2.7 Buildings structure

Usually only half of the available building is used for the actual servers, while the other parts are used for cooling and energy management.

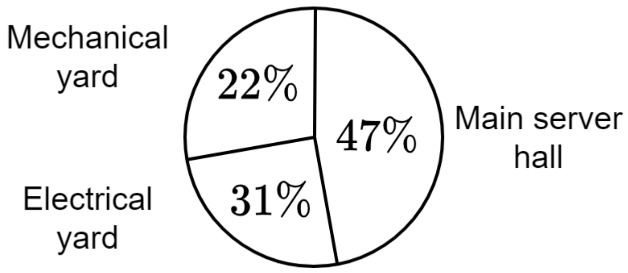


Figure 2.14: Usual building usage

In addition to servers, the WSC comprises critical components associated with power delivery, cooling, and building infrastructure, all of which warrant consideration.

2.7.1 Backup power generation

To safeguard against power failure, battery systems and diesel generators are employed to back up the external power supply. A typical Uninterruptible Power Supply (UPS) integrates three functions within a single system:

- It incorporates some form of energy storage (electrical, chemical, or mechanical) to bridge the time gap between utility failure and the availability of generator power.
- It includes a transfer switch that selects the active power input, whether it be utility power or generator power.
- It conditions the incoming power feed by eliminating voltage spikes or sags, as well as harmonic distortions in the AC feed.

2.7.2 Cooling systems

IT equipment generates substantial heat, necessitating an expensive cooling system within the data center. This system typically includes coolers, heat exchangers, and cold water tanks.

Open loop The most basic cooling setup is fresh air cooling, also known as air economization, which essentially involves opening windows. This operates as a single open-loop system. Free cooling, or open-loop cooling, utilizes cold outside air to assist in either producing chilled water or directly cooling servers. While it's not entirely free in terms of cost, it incurs very low energy expenses compared to traditional chillers.

Closed loop Closed-loop systems come in various configurations, with the air circuit on the data center floor being the most common. The objective is to extract and expel heat from the servers, directing it to a heat exchanger. Cold air is directed towards the servers, where it absorbs heat and then proceeds to the heat exchanger to be cooled again for the subsequent cycle through the servers.

Two-loop system The airflow path from the underfloor plenum, through the racks, and back to the CRAC (Computer Room Air Conditioning, a term dating back to the 1960s) delineates the primary air circuit, also known as the first loop. The second loop, comprising the liquid supply within the CRAC units, directs from the CRAC to external heat exchangers, often positioned on the building roof. These heat exchangers then release the heat into the environment.

Three-loop system A three-loop system for cooling typically involves three distinct circuits to manage the thermal environment within a data center or similar facility. Here's a brief description of each loop:

- Primary air circuit: this loop involves airflow from the underfloor plenum, through the equipment racks, and back to the CRAC (Computer Room Air Conditioning) units. It's responsible for regulating the temperature of the air within the data center space.
- Secondary liquid cooling loop: the second loop consists of a liquid coolant circulating within the CRAC units. This coolant absorbs heat from the air as it passes through the CRAC, helping to cool down the equipment and maintain optimal operating temperatures.
- Tertiary heat rejection loop: in this loop, the heated liquid coolant from the CRAC units is transported to external heat exchangers, typically located on the building's roof. These heat exchangers release the absorbed heat into the environment, completing the cooling cycle and ensuring efficient heat dissipation from the data center.

Chillers and cooling towers A water-cooled chiller operates similarly to a water-cooled air conditioner. Cooling towers are utilized to cool a water stream by evaporating a portion of it into the atmosphere. However, their effectiveness diminishes in very cold climates as they require supplementary mechanisms to prevent ice formation.

Summary Each cooling topology involves trade-offs concerning complexity, efficiency, and cost:

- Fresh air cooling exhibits high efficiency but is not universally applicable, necessitates the filtration of airborne particles, and may introduce intricate control challenges.
- Two-loop systems are straightforward to set up, relatively cost-effective to build, and provide isolation from external contaminants. However, they generally exhibit lower operational efficiency.
- Three-loop systems are the most costly to establish and involve moderately complex controls. However, they offer protection against contaminants and boast good efficiency.

Liquid cooling An in-rack cooler integrates an air-to-water heat exchanger at the rear of a rack, allowing hot air emitted by the servers to pass over coils cooled by water. This setup significantly shortens the distance between server exhaust and the input of the CRAC (Computer Room Air Conditioning) unit.

In-row cooling functions similarly to in-rack cooling, but instead of having the cooling coils within the rack, they are positioned adjacent to the rack.

Direct cooling of server components is achievable through cold plates, which are essentially local liquid-cooled heat sinks. However, it's impractical to cool all compute components using cold plates. Typically, components with the highest power dissipation are targeted for liquid cooling, while other components rely on air cooling. The liquid circulating through the heat sinks carries the heat to a liquid-to-air or liquid-to-liquid heat exchanger, which can be situated near the tray or rack, or be integrated into the data center infrastructure, such as a cooling tower.

Container-based datacenters Container-based data centers represent a further advancement from in-row cooling by housing server racks within a container, typically ranging from six to twelve meters in length. These containers integrate heat exchange and power distribution systems directly into the container structure.

2.7.3 Datacenter power consumption

Data center power consumption has become a critical issue due to its potential to reach several megawatts (MWs). Cooling requirements typically consume around half of the energy needed by the IT equipment, including servers, network infrastructure, and disks. Moreover, the energy transformation processes within data centers often result in significant energy wastage.

In terms of global impact, data centers account for approximately 3% of the world's electricity supply, exceeding the annual electricity consumption of countries like the UK, which stands at 300 terawatt-hours (TWh). This substantial power consumption contributes to about 2% of total greenhouse gas emissions, comparable to the emissions produced by worldwide air traffic before the pandemic.

The carbon dioxide (CO2) emissions from data centers are substantial, equivalent to the emissions of entire countries such as the Netherlands or Argentina. This highlights the urgent need for energy-efficient solutions and sustainable practices within the data center industry to mitigate its environmental impact.

The power consumption of data centers poses a significant challenge, as it can scale up to multiple megawatts (MWs). Typically, cooling operations account for approximately half of the total energy consumption, encompassing the energy demand of IT equipment such as servers, network infrastructure, and disks. Additionally, the energy transformation processes within data centers result in substantial wastage, further contributing to the overall energy consumption.

Power usage effectiveness Power Usage Effectiveness (PUE) represents the ratio of the total energy consumed by a data center facility to the energy supplied to the computing equipment, expressed as:

$$\text{PUE} = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}}$$

Here, total facility power encompasses the energy consumption of IT systems, including servers, network devices, and storage units, along with other equipment such as cooling systems, Uninterruptible Power Supplies (UPS), switch gear, generators, lighting, and fans. Data Center Infrastructure Efficiency (DCiE) is the reciprocal of Power Usage Effectiveness (PUE).

2.7.4 Taxonomy

Data center availability is categorized into four distinct tier levels, each with its own set of requirements.

Tier level	Requirements
1	No redundancy 99.671% uptime per year Maximum of 28.8 hours of downtime per year
2	Some cooling and power redundancies 99.741% uptime per year No more than 22 hours of downtime per year
3	$N + 1$ fault tolerance 99.982% uptime Less than 1.6 hours of downtime per year
4	$2N$ or $2N + 1$ fault tolerance No single points of failure 99.995% uptime per year Less than 26.3 minutes of downtime per year

2.8 Networking

Here are various network architectures:

1. *Monolithic app*: this setup requires minimal network resources and relies on proprietary protocols.

2. *Client-server*: high network demands within the enterprise, applications confined within the enterprise, and a combination of TCP/IP and proprietary protocols.

3. *Web applications*: utilizes ubiquitous TCP/IP, accessible from anywhere, and servers are segmented into multiple units.

4. *Microservices*: infrastructure shifts to cloud providers, servers segmented into microservices, resulting in increased server-to-server traffic.

A data center comprises all physical infrastructure necessary to support cloud computing services. This infrastructure is typically co-located within a room, building, or set of adjacent buildings. Applications within the data center may include: cloud computing, cloud storage, and web services. These applications facilitate the consolidation of computation and network resources, often in very large data centers ranging from 1,000 to 1,000,000 servers.

Demand increasing As server performance improves over time, there is a natural increase in demand for inter-server bandwidth. By doubling the number of compute or storage elements, we can easily double the aggregate compute capacity or storage.

However, networking poses a challenge for straightforward horizontal scaling. While doubling leaf bandwidth is straightforward—by doubling the number of servers, we also double the network ports and thus the bandwidth—dealing with bisection bandwidth becomes more complex when assuming that every server needs to communicate with every other server.

Definition (*Bisection bandwidth*). Bisection bandwidth refers to the bandwidth across the narrowest line that evenly divides the cluster into two parts.

This metric serves as a characterization of network capacity, as it represents the capacity for randomly communicating processors to transmit data across the central portion of the network. When assuming that every server must communicate with every other server, it becomes necessary to double not only leaf bandwidth but also bisection bandwidth.

Datacenter networking Data center networking can be categorized into three primary types:

- *Switch-centric architectures*: utilize switches for packet forwarding tasks.
- *Server-centric architectures*: employ servers equipped with multiple Network Interface Cards (NICs) to serve as switches alongside their computational functions.
- *Hybrid architectures*: blend both switches and servers to handle packet forwarding duties.

2.8.1 Switch-centric architecture

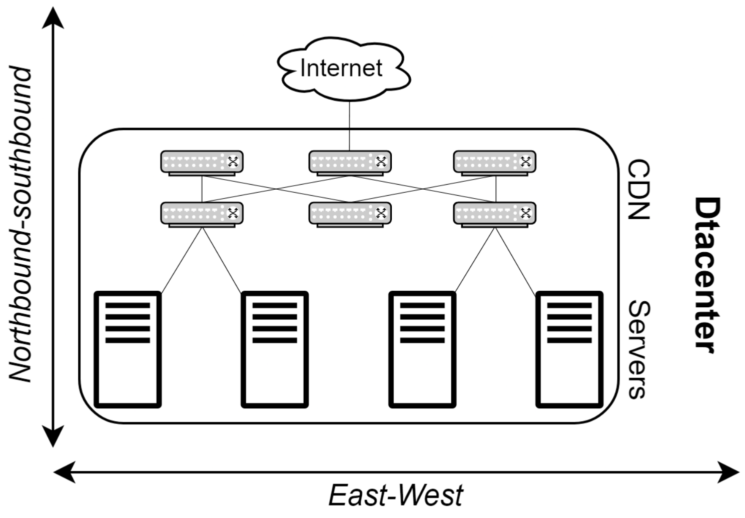


Figure 2.15: Switch-centric architecture

The traffic is divided into two main flows:

- Northbound-southbound: direct connection between internet and servers.
- East-West: inter-server communication within a data center, often referred to as East-West traffic, involves several key aspects:
  - Storage replication, typically characterized by numerous data flows with relatively few instances of data.
  - In distributed file systems like Hadoop, maintaining at least three copies of the same data, commonly with two copies residing within the same rack and one in a different rack.
  - Virtual Machine (VM) migration, a process integral to tasks such as Network Function Virtualization (NFV), where data undergoes processing across a series of VMs, such as firewalls, web servers, parental control systems, and accounting servers.

It's notable that East-West traffic typically surpasses North-South traffic in terms of volume and significance within the data center environment. This type of traffic can be: unicast, multicast, and incast.

2.8.2 Classical 3-tier architecture

The classical 3-tier switch-centric architecture is structured into three distinct layers, as illustrated in the diagram below:

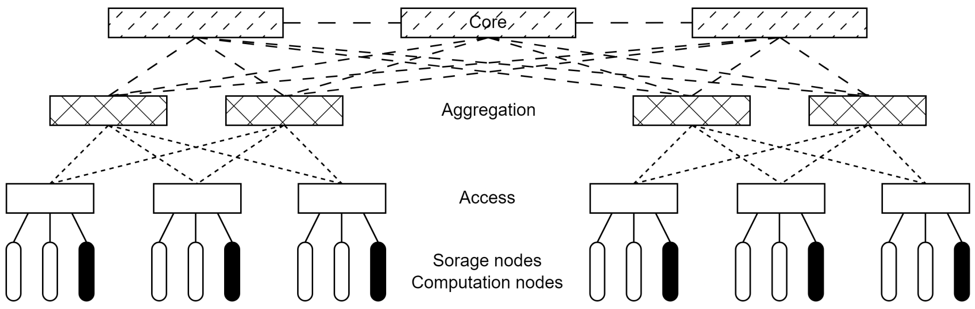


Figure 2.16: Classical 3-tier architecture

This configuration represents a straightforward Data Center Network (DCN) topology, where servers interface with the DCN via access switches. Each access-level switch links to a minimum of two aggregation-level switches, which, in turn, connect to core-level switches (gateways).

Server structure Servers can be categorized into two main architectures:

- *Top-of-Rack* (ToR): in a rack, all servers are linked to a Top-of-Rack (ToR) access switch, with both the servers and the ToR switch situated within the same rack. Aggregation switches are housed either in dedicated racks or in shared racks alongside other ToR switches and servers. This setup restricts the number of cables, resulting in simpler cabling. Moreover, the number of ports per switch is constrained, leading to lower costs. However, this configuration suffers from limited scalability and heightened complexity in switch management due to the proliferation of switches.

- *End-of-Row* (EoR): aggregation switches are strategically placed at the end of each corridor, serving as the termination point for a row of racks. Servers within racks establish direct connections to the aggregation switch situated in another rack. Due to this arrangement, aggregation switches require a greater number of ports, resulting in more intricate cabling and necessitating longer cables, which in turn incur higher costs. Typically, a patch panel is employed to facilitate connections between servers and the aggregation switch. Despite the more complex cabling, this setup offers simpler switch management owing to the reduced number of switches involved.

Bandwidth increasing Boosting bandwidth can be achieved through various methods. Firstly, augmenting the number of switches at both the core and aggregation layers facilitates enhanced bandwidth. Additionally, employing routing protocols like Equal Cost Multiple Path (ECMP) enables equitable distribution of traffic across various routes, further amplifying bandwidth capabilities. While this solution is straightforward, it can prove to be exceedingly costly in expansive data centers for several reasons:

- The upper layers necessitate swifter network equipment, which tends to be more expensive.
- Each layer is serviced by switches of varying types, compounding acquisition costs and introducing complexities in management, spare-part stocking, and energy consumption.

Cumulatively, these factors contribute to significantly elevated expenses, both in terms of upfront investments and ongoing operational costs.

2.8.3 Leaf-spine architecture

The Leaf-spine switch-centric architecture is structured into two distinct layers:

- Leaf: ToR switch
- Spine: dedicated switches (aggregation switches)

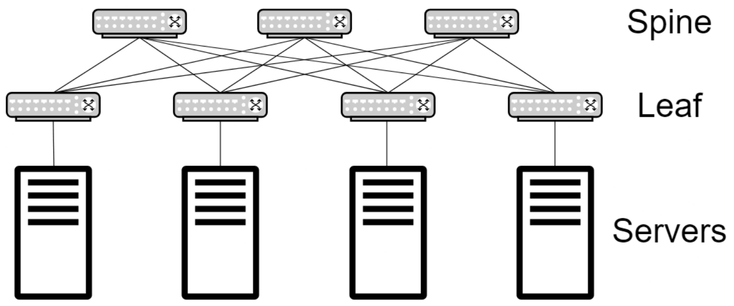


Figure 2.17: Leaf-spine architecture

Spine-leaf topologies draw inspiration from the realm of telephony, introducing a non-folded Clos structure. This architecture features fully interconnected stages, ensuring that each matrix within one stage is linked, at least once, to every matrix in the subsequent stage, and vice versa.

Clos networks Let's denote k as the number of middle stage switches and n as the number of inputs and outputs of switches in the side stages. If $k \geq n$, there always exists a method to reorganize communications, allowing for the creation of a path between any pair of inactive input/output channels. Moreover, if $k \geq 2n - 1$, a path between any pair of idle input/output channels is guaranteed to be available. It's important to note that t serves as a flexible design parameter. Consequently, the total number of input/output channels,$N = t \cdot n$, can be adjusted freely by increasing the size of middle-stage switches. However, it's crucial to remember that a Data Center Network (DCN) operates as a packet-switched network.

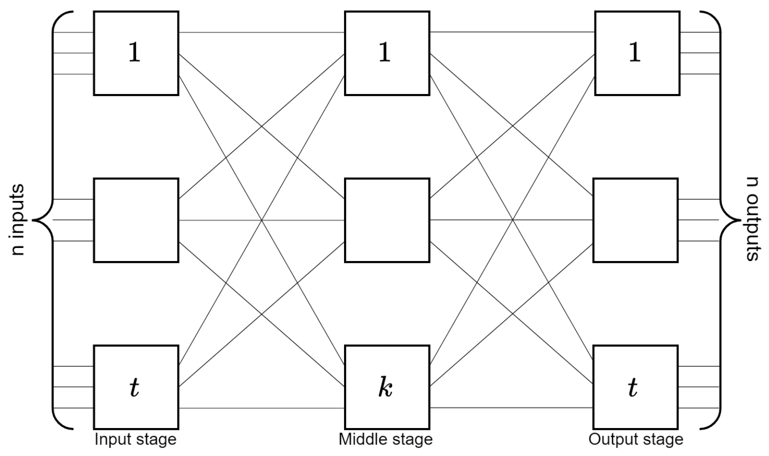


Figure 2.18: Clos network

In the Clos topology, where $n = m = k$, each switching module functions uni-directionally with k input and k output ports per module. The central stage comprises k matrices, while the side stages consist of t matrices. Path traversal encompasses three modules.

In the Leaf and Spine topology, every switching module operates bidirectionally:

- Leaf Configuration: comprises t switching modules, each featuring $2k$ bidirectional ports per module.
- Spine Configuration: consists of k switching modules, with each module having t bidirectional ports.

In both configurations, paths traverse either one or three modules.

Advantages The Clos design in Data Center Networks (DCNs) offers several advantages:

- Utilization of homogeneous equipment throughout the network, simplifying management and maintenance.
- Routing serves as the Fundamental Interconnect Model, eliminating the need for Learning and Forwarding or Spanning Tree Protocol (STP).
- Implementation of Equal Cost Multipath (ECMP) strategy with routing protocols like IS-IS, SPB, or TRILL, enhancing network efficiency and resilience.
- Consistent number of hops for any pair of nodes, ensuring predictable and reliable communication.
- Small blast radius, minimizing the impact of network issues and failures.

Upscaling We begin with a two-tier network structure and introduce an additional row of switches. Alternatively, we can consider transforming each spine-leaf group into a pod and incorporate a super spine tier. This architecture represents a highly scalable and cost-effective Data Center Network (DCN) design, focusing on optimizing bisection bandwidth. It can be constructed using standard Gigabit Ethernet switches with identical port counts. This approach is adopted by major tech giants like Microsoft and Amazon.

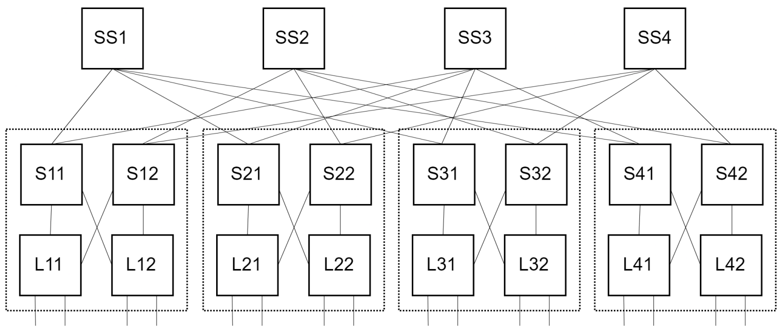


Figure 2.19: Pod-based three-tier Clos

A Point Of Delivery (POD) refers to a module or cluster comprising network, compute, storage, and application elements collaboratively functioning to provide a network service. The POD represents a standardized and replicable pattern, with its constituent components enhancing the modularity, scalability, and manageability of data infrastructure.

Generalization of the architecture A leaf with $2k^2$ bidirectional ports is configured with k ports connecting to servers and k ports linking to the data center network. It's important to note that this setup cannot directly interconnect $2k^2$ servers as it would lead to network blocking. This configuration is constructed using $2k$ switches with $2k$ ports each.

The Point Of Delivery (PoD) is replicated P times, each consisting of k servers and $2P$ switches with $2k$ ports, and k switches with P ports. In a Fat-tree design, choosing $P = 2k$, we have $2k^3$ servers, $5k^2$ switches with $2k$ ports each, and $2k$ PoDs at the edge layer, each containing $2k^2$ servers.

At the edge layer, each edge switch is directly connected to k servers in a pod and k aggregation switches. A fat-tree network with $2k$-port commodity switches can accommodate a total of $2k^3$ servers. Additionally, there are $2k^2$ core switches with $2k$-port each, with each one connected to $2k$ pods. Each aggregation switch is connected to k core switches, highlighting partial connectivity at the switch level.

VL2 network The VL2 Network is an economical hierarchical fat-tree-based Data Center Network (DCN) architecture designed to provide high bisection bandwidth. It employs three types of switches: intermediate, aggregation, and top-of-rack (ToR) switches.

The network comprises $\frac{D_A}{2}$ intermediate switches, D_I aggregation switches, and $\frac{1}{2}D_AD_I$ ToR switches. The total number of servers in a VL2 network is calculated as $20\frac{D_AD_I}{4}$. A key feature of the VL2 Network is its use of a load-balancing technique called valiant load balancing (VLB).

2.8.4 Switches

Commodity switches are known for their affordability and simplicity, as well as their rapid evolution. While they offer intermittent capacity, they are suitable for scenarios where only a single operator is involved. On the other hand, WAN switches are more complex and expensive, with slower evolution. They provide the highest level of availability and allow intermittent capacity, with numerous protocols available to support interoperability among multivendor WANs.

2.8.5 Server-centric and hybrid architectures

A server-centric architecture, proposed for constructing containerized data centers, aims to lower implementation and maintenance expenses by solely utilizing servers to establish the DCN. It employs a 3D-Torus topology to directly interconnect the servers, leveraging network locality to enhance communication efficiency. However, drawbacks include the necessity for servers with multiple Network Interface Cards (NICs) to assemble a 3D Torus network, as well as the presence of lengthy paths and heightened routing complexity.

CHAPTER 3

Dependability

3.1 Introduction

Definition (*Dependability*)**.** Dependability is a measure of how much we trust a system.

Dependability can also be construed as the system's capacity to execute its intended functions while embodying:

- *Reliability*: the continuity of accurate service.
- *Availability*: the readiness for accurate service.
- *Maintainability*: the ease of maintenance.
- *Safety*: the absence of catastrophic consequences.
- *Security*: the preservation of data confidentiality and integrity.

Significant efforts are often directed towards functional verification, ensuring that the implementation aligns with specifications, fulfills requirements, meets constraints, and optimizes selected parameters. However, even when all these aspects are addressed, system failures may still occur. Such failures are typically attributable to some form of breakdown.

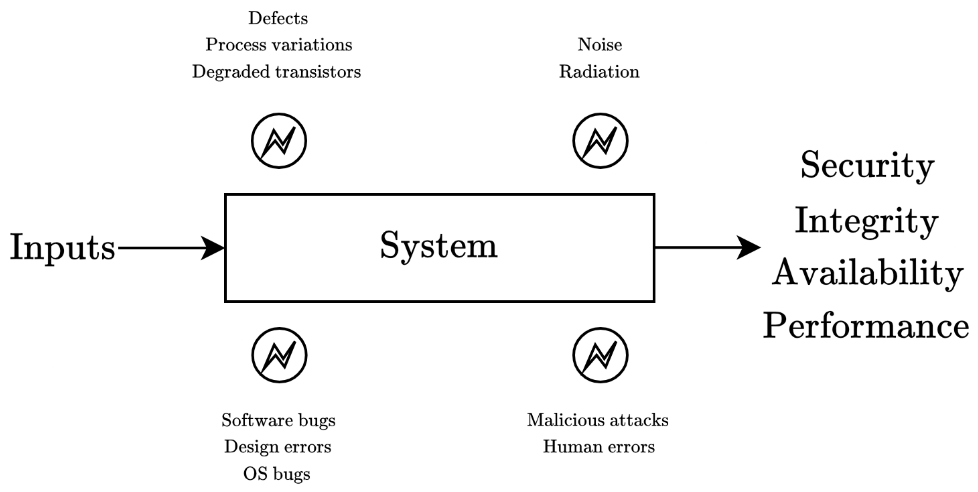


Figure 3.1: Dependability

A single system failure has the potential to impact a significant number of individuals. The costs associated with a failure can be substantial, particularly if it results in economic losses or physical damage. Systems lacking in dependability are less likely to be utilized or embraced. Moreover, undependable systems can lead to information loss, incurring significant recovery costs.

To address these issues, specific standards have been established for various systems:

- ISO 26262 for automotive.
- CENELEC 50128 (software) and 50129 (hardware) for railways.
- RTCA DO-178C (software) and DO-254 (hardware) for airborne systems.
- ESA ECSS-E-ST-40C (software) and ECSS-Q-ST-60-02C (hardware) for space systems.

3.2 Dependability principles

Dependability is a critical consideration both during the design phase and during runtime operations. During the design phase, it is essential to:

- Analyze the system under development.
- Evaluate and measure its dependability properties.
- Make necessary modifications to the design as needed.

During runtime, the focus shifts to:

- Detecting any malfunctions or failures that occur.
- Investigating and understanding the root causes of these issues.
- Taking appropriate reactive measures to address and mitigate the impact of the malfunctions.

Failures are commonplace in both the development and operational stages: while those in development should be averted, operational failures, being unavoidable due to the nature of system components, must be managed effectively. Design processes should factor in these potential failures to ensure that control and safety measures remain intact even when failures arise. Moreover, the effects of these failures should be predictable and deterministic rather than catastrophic.

Once upon a time, dependability was primarily a concern in safety-critical and mission-critical application domains like space exploration, nuclear facilities, and avionics. This was largely due to the significant cost associated with ensuring dependability, which was deemed acceptable only when absolutely necessary. In non-critical systems, operational failures can lead to economic losses and damage to reputation, as seen in consumer products. However, in mission-critical systems such as satellites, automatic weather stations, surveillance drones, and unmanned vehicles, operational failures can result in serious or irreversible consequences for the mission at hand. Safety-critical systems, on the other hand, pose a direct threat to human life if they fail during operation. Examples include aircraft control systems, medical instrumentation, railway signaling, and nuclear reactor control systems.

3.3 Datacenters dependability

In the context of data centers, downtime is a major concern. Aberdeen Research provides statistics on downtimes and incidents:

- Average performing facilities experience downtime of 60 minutes with 2.3 incidents per year.
- Best-in-class organizations have downtime as low as 6 minutes with only 0.3 incidents per year.

Within a data center, the components requiring dependability include:

- Computing systems, sensors, and actuators (referred to as nodes).
- Network infrastructure facilitating communication.
- Cloud services encompassing data storage and manipulation capabilities.

The smooth operation of all these elements is essential for the system's overall functionality.

3.3.1 Dependability requirements

To ensure dependability, a resilient computing system must adhere to the failure avoidance paradigm, which includes:

- Employing a conservative design approach.
- Validating the design thoroughly.
- Conducting detailed testing of both hardware and software components.
- Implementing an infant mortality screen to identify and address early failures.
- Focusing on error avoidance strategies.
- Incorporating mechanisms for error detection and masking during system operation.
- Utilizing on-line monitoring tools.
- Implementing diagnostics for identifying issues.
- Enabling self-recovery and self-repair capabilities.

In practice, dependable systems can be obtained through:

- Robust design, which focuses on error-free processes and design practices.
- Robust operation, achieved through fault-tolerant measures such as monitoring, detection, and mitigation.

Safety-critical systems Safety-critical systems encompass all components collaborating to fulfill the safety-critical mission. These components may encompass input sensors, digital data devices, hardware, peripherals, drivers, actuators, controlling software, and other interfaces. Their development necessitates thorough analysis, comprehensive design, and rigorous testing.

3.3.2 Dependability in practice

Dependability can be enhanced at various levels:

- At the technological level: by incorporating reliable and robust components during design and manufacturing.
- At the architectural level: by integrating standard components with solutions designed to handle potential failures effectively.
- At the application level: by developing algorithms or operating systems that can mask and recover from failures.

However, it's important to note that regardless of the level, enhancing dependability typically entails increased costs and a potential reduction in performance.

The primary challenges of dependability include:

- Designing robust systems using unreliable and cost-effective Commercial Off-The-Shelf (COTS) components and integrating them seamlessly.
- Addressing new challenges arising from technological advancements, such as process variations, stressed working conditions, and the emergence of failure mechanisms previously unnoticed at the system-level due to smaller geometries.
- Striking the optimal balance between dependability and costs, which is contingent upon factors like the application field, working scenario, employed technologies, algorithms, and applications.

In the application scenario achieving 100% dependability may entail significant costs and overheads but is deemed justified.

Dependability becomes a trade-off with performance and power consumption, necessitating careful consideration of trade-offs.

3.4 Reliability and availability

Dependability can be subdivided into the following components:

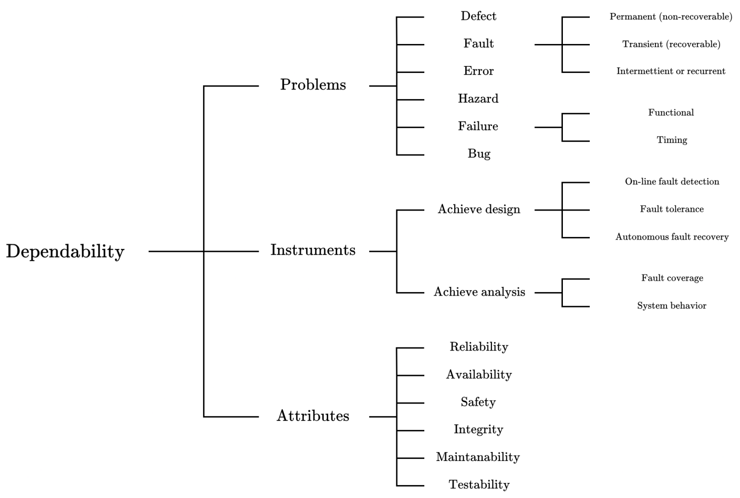


Figure 3.2: Dependability components

3.4.1 Reliability

Definition (*Reliability*)**.** Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time.

We denote $R(t)$ as the probability of the system operating correctly in a designated environment until time t, expressed as:

$$R(t) = \mathrm{P}(\text{not failed during } [0,t])$$

assuming it was operational at time $t = 0$. If a system is required to function for ten-hour intervals, then the reliability target is set for ten hours. $R(t)$ is a monotonically decreasing function ranging from 1 to 0 over 0 to infinity:

$$\lim_{x \to +\infty} R(t) = 0$$

This metric is often applied to systems where even brief periods of malfunction are intolerable, such as those with stringent performance, timing, or safety requirements, or those that are challenging or impossible to repair.

Unreliability The unreliability is the complementary measure to reliability and is calculated as:

$$1 - R(t)$$

3.4.2 Availability

Definition (*Availability*)**.** Availability is the e degree to which a system or component is operational and accessible when required for use.

It is defined as:

$$\text{Availability} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

We denote as $A(t)$ the probability that the system will be operational at time t:

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

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

Unavailability The unavailability is the complementary measure to availability and is calculated as:

$$1 - A(t)$$

Definition (*Repairable system*)**.** A system is sad to be repairable if $A(t) \geq R(t)$.

Definition (*Unrepairable system*)**.** A system is sad to be unrepairable if $A(t) = R(t)$.

3.4.3 Other indices

Definition (*Mean time to failure*)**.** The mean time to failure represents the average duration before any failure occurs.

Definition (*Mean time between failures*)**.** The mean time between failures denotes the average duration between two consecutive failures.

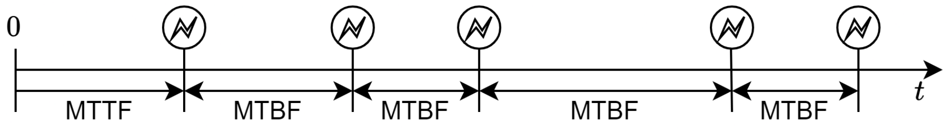


Figure 3.3: Example of MTTF and MTBF

The MTTF can alternatively be calculated as the integral of reliability:

$$\text{MTTF} = \int_0^\infty R(t)dt$$

Additionally, the MTBF can be defined as:

$$\text{MTBF} = \frac{\text{total operating time}}{\text{number of failures}}$$

Definition (*Failures in time*)**.** Failures in time refer to the reciprocal of the mean time between failures.

The failures in time, denoted as λ, are computed as:

$$\lambda = \frac{\text{number of failures}}{\text{total operating time}} = \frac{1}{\text{MTBF}}$$

Typically, it is computed based on a scale of one billion hours.

Definition (*Infant mortality*)**.** Infant mortality measures failures occurring in new systems, often observed during testing phases rather than during production.

Definition (*Random failures*)**.** Random failures occur sporadically throughout the lifespan of a system.

Definition (*Wear out*)**.** Wear out refers to the failure of components at the end of their operational life, potentially leading to system failure.

Preemptive maintenance can mitigate this type of failure.

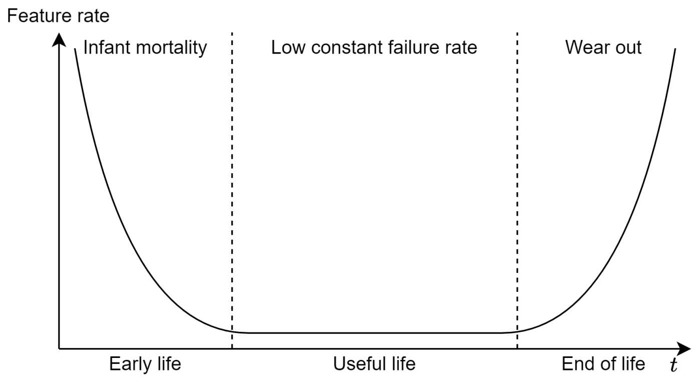


Figure 3.4: Product reliability curve

3.4.4 Defect identification

To identify defective products and determine the Mean Time To Failure (MTTF), a burn-in test can be conducted. During this test, the system is subjected to elevated levels of temperature, voltage, current, and humidity to accelerate wear and tear.

In scenarios where systems exhibit low reliability but must remain operational, quick repairs of system failures that do not compromise data integrity may help mitigate the impact of low reliability. For example, in a database management system, low reliability might not present significant issues.

On the other hand, ensuring high reliability in systems can pose greater challenges.

Utilizing information from the reliability function $R(t)$ allows for the computation of a complex system's reliability over time, representing its expected lifetime. This process involves calculating the MTTF and determining the overall reliability by aggregating the reliability of individual components.

Definition (*Fault*)**.** A fault refers to a defect within the system.

Definition (*Error*)**.** An error represents a deviation from the intended operation of the system or subsystem.

Definition (*Failure*)**.** A failure occurs when the system is unable to perform its designated function.

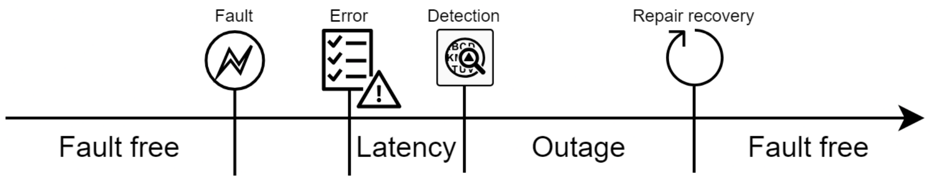


Figure 3.5: Reliability measures

3.4.5 Reliability block diagrams

Reliability block diagrams (RBDs) provide an inductive model where a system is partitioned into blocks representing distinct elements such as components or subsystems. Each block in the RBD has its own reliability, which can be previously calculated or modeled. These blocks are then combined to represent all possible success paths within the system.

Assuming that failures occur according to a Poisson model, the time between two successive failures can be modeled as follows:

- Probability density function:

$$f(t, \lambda) = \lambda e^{-\lambda t} \qquad t \geq 0, \lambda > 0$$

- Cumulative density function:

$$\text{P}(T \leq t) = \int_0^t f(s, \lambda) ds = 1 - e^{-\lambda t}$$

- Expected value:

$$\mathbb{E}\left[T\right] = \frac{1}{\lambda}$$

- Variance:

$$\sigma^2(T) = \frac{1}{\lambda^2}$$

As a result, reliability can be defined as:

$$R(t) = \mathrm{P}(T \geq t) = e^{-\lambda t}$$

Here, $\lambda(t)$ represents the failure rate. RBDs provide an approach to compute the reliability of a system based on the reliability of its components.

Reliability in series In the case of components arranged in series, all components must be operational for the system to function properly:

$$R_S(t) = R_{C1}(t) \cdot R_{C2}(t)$$

For systems composed of components with reliability following an exponential distribution (a common case), the Mean Time To Failure (MTTF) of the system S can be expressed as:

$$\mathrm{MTTF}_S = \frac{1}{\lambda_S} = \frac{1}{\sum_{i=1}^{n} \frac{1}{\mathrm{MTTF}_i}}$$

ì A special case occurs when all components are identical:

$$\mathrm{MTTF}_S = \frac{\mathrm{MTTF}_1}{n}$$

Availability in series The availability A_S is calculated as:

$$A_S = \prod_{i=1}^{n} \frac{\mathrm{MTTF}_i}{\mathrm{MTTF}_i + \mathrm{MTTR}_i}$$

When all components are identical:

$$A = \left(\frac{\mathrm{MTTF}_1}{\mathrm{MTTF}_1 + \mathrm{MTTR}_1} \right)^n$$

Reliability in parallel In the case of components arranged in parallel, if at least one component is operational, the system functions properly:

$$R_S(t) = R_{C1}(t) + R_{C2}(t) - R_{C1}(t) \cdot R_{C2}(t)$$

For a system P composed of n components:

$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t))$$

Availability in parallel The availability A_p is given by:

$$A_p(t) 1 - \prod_{i=1}^{n} (1 - A_i(t))$$

Redundancy A system may consist of two parallel replicas:

- The primary replica operates continuously.
- The redundant replica, typically disabled, is activated when the primary replica fails.

For operational redundancy, the following are necessary a mechanism to ascertain whether the primary replica is functioning correctly (online self-check), and a dynamic switching mechanism to deactivate the primary replica and activate the redundant one. In the standby model, we calculate the system reliability using various approaches:

- When failure rates are equal and switching is perfect:
$$R_S = e^{-\lambda t}(1 + \lambda t)$$
- In cases of unequal failure rates with perfect switching:
$$R_S = e^{-\lambda t} + \lambda_1 \frac{e^{-\lambda_1 t} - e^{-\lambda_2 t}}{\lambda_2 - \lambda_1}$$
- When failure rates are equal, but switching is imperfect:
$$R_S = e^{-\lambda t}(1 + R_V \lambda t)$$
- For unequal failure rates with imperfect switching:
$$R_S = e^{-\lambda t} + R_V \lambda_1 \frac{e^{-\lambda_1 t} - e^{-\lambda_2 t}}{\lambda_2 - \lambda_1}$$

In these equations, R_S represents the system reliability, λ stands for the failure rate, t denotes the opening time, and R_V indicates the switching reliability.

In a broader context, a system with one primary replica and n redundant replicas (identical and perfectly switchable) can be represented by the reliability function:

$$R(t) = e^{-\lambda t} \sum_{i=0}^{n-1} \frac{(\lambda t)^i}{i!}$$

For a system comprising n identical replicas where at least r replicas must function correctly for the entire system to operate correctly. In this case we the reliability is computed as:

$$R_S(t) = R_V \sum_{i=r}^{n} R_C^i (1 - R_C)^{n-i} \frac{n!}{i!(n-i)!}$$

Here, R_S represents the system reliability, R_C denotes the component reliability, R_V signifies the voter reliability, n stands for the number of components, and r indicates the minimum number of components required to remain operational.

Triple modular redundancy Considering triple modular redundancy where 2 out of 3 components function properly, and the voter works properly:

$$\text{MTTF}_{TMR} = \frac{5}{6}\text{MTTR}_{simplex}$$

Thus, MTTF_{TMR} is shorter than $MTTR_{simplex}$, allowing tolerance for transient and permanent faults, leading to higher reliability for shorter missions.

Redundancy proves beneficial when $R_{TMR}(t) > R_C(t)$ for mission times shorter than 70% of MTTF_C. Hence, redundancy is advantageous for specific mission durations.

CHAPTER 4

Redundant Arrays of Independent Disks

4.1 Introduction

Redundant Arrays of Independent Disks (RAID) is a storage technology introduced in the 1980s by Patterson. Its purpose is to enhance the performance, capacity, and reliability of storage systems. RAID achieves this by combining multiple independent disks into a single logical unit with improved performance.

This approach stands in contrast to Just a Bunch of Disks (JBOD), where each disk functions as a separate entity with its own mount point. Instead, RAID stripes data across several disks, allowing for parallel access. This striping enables high data transfer rates for large data accesses and high I/O rates for small, frequent data accesses. Additionally, it facilitates load balancing across the disks.

RAID employs two primary techniques: data striping to boost performance and redundancy to enhance reliability. These orthogonal methods work together to provide improved storage capabilities for various computing needs.

The distribution of data in RAID systems is facilitated through I/O virtualization. This means that data is distributed transparently across the disks, requiring no action from the users.

4.1.1 Data striping

Data striping in RAID involves writing data sequentially, such as vectors, files, or tables, onto multiple disks in units called stripes. These stripes can be defined by various units, such as bits, bytes, or blocks, and are written according to a cyclic algorithm, typically a round-robin approach.

The stripe unit refers to the size of the data unit that is written on a single disk, while the stripe width indicates the number of disks considered by the striping algorithm.

There are several benefits to data striping:

- Multiple independent I/O requests can be executed in parallel by several disks, reducing the queue length and overall response time of the disks.
- Single multiple-block I/O requests can be executed by multiple disks in parallel, increasing the transfer rate of individual requests.

Indeed, as the number of physical disks in a RAID array increases, so do the potential size and performance gains. However, this also comes with a corresponding increase in the probability of failure of any one disk within the array. This heightened risk of disk failure underscores the importance of introducing redundancy into the system.

Redundancy in RAID serves as a safeguard against data loss in the event of disk failure. By duplicating or distributing data across multiple disks in a redundant manner, the system can withstand the failure of one or more disks without losing critical data. Redundancy mechanisms such as mirroring (RAID 1), parity (RAID 5), or both (RAID 6) provide fault tolerance and ensure data integrity even in the face of hardware failures.

Performance In redundancy-based RAID configurations, error correcting codes are computed and stored on disks separate from those holding the primary data. These error correcting codes provide the ability to tolerate data loss resulting from disk failures. However, the implementation of redundancy introduces overhead in terms of performance, particularly during write operations.

During write operations, updates must also be made to the redundant information, which results in a performance penalty compared to traditional writes. This is because the additional computation and storage required for redundancy checking and updating the error correcting codes increase the time required to complete write operations.

RAID Indeed, hard drives are invaluable storage devices, offering relatively fast and persistent storage. However, they have their shortcomings, primarily in terms of coping with disk failure and limited capacity.

To address these limitations, RAID (Redundant Array of Independent Disks) technology comes into play. RAID leverages multiple disks to create the illusion of a single, larger, faster, and more reliable disk. Externally, RAID operates seamlessly, appearing as a single disk to users. This transparency means that data blocks are read and written just as they would be on a single disk, without the need for software to explicitly manage multiple disks or perform error checking and recovery tasks.

Internally, however, RAID is a sophisticated computer system. Disks are managed by dedicated CPUs and specialized software, with support from RAM and non-volatile memory. RAID offers a wide array of configuration options, known as RAID levels, each tailored to different needs and priorities, such as performance, redundancy, or a balance of both.

RAID offers various levels, each with its own method of data organization and redundancy. Here's an overview of the commonly known RAID levels:

1. RAID 0: striping only. Data is striped across multiple disks without redundancy. This offers improved performance by spreading data across disks, but there's no fault tolerance, meaning if one disk fails, all data is lost.

2. RAID 1: mirroring only. Data is mirrored across pairs of disks. This provides redundancy, as data is duplicated on each disk. If one disk fails, data remains intact on the mirrored disk.

3. RAID 0+1 (also known as RAID 01): a nested RAID level combining RAID 0 (striping) and RAID 1 (mirroring). Data is striped across mirrored sets of disks. This offers both performance benefits and redundancy.

4. RAID 1+0 (also known as RAID 10): another nested RAID level combining RAID 1 (mirroring) and RAID 0 (striping). Data is mirrored first and then striped across the mirrored sets. This also provides both performance and redundancy.
5. RAID 2: bit interleaving, which is not commonly used in practice.
6. RAID 3: byte interleaving with dedicated parity disk. Data is striped across disks, and parity information is stored on a dedicated disk for fault tolerance.
7. RAID 4: block interleaving with dedicated parity disk. Similar to RAID 3, but operates at the block level rather than the byte level.
8. RAID 5: block interleaving with distributed parity blocks. Data and parity information are striped across multiple disks, with parity blocks distributed among the disks. This provides fault tolerance with improved performance compared to RAID 3 and 4.
9. RAID 6: similar to RAID 5 but with greater redundancy. It can tolerate the failure of up to two disks simultaneously, as it utilizes double parity to protect against data loss.

4.1.2 Performance

Sequential transfer rate We typically examine sequential and random workloads. Assuming disks in the array have a sequential transfer rate S:

$$S = \frac{\text{transfer size}}{\text{time to access}}$$

Example:
Consider a 10 megabyte transfer and a hard disk with the following characteristics:

- Average seek time of seven milliseconds.
- Average rotational delay of three milliseconds.
- Transfer rate of 50 megabytes per second.

The transfer rate in this case will be:

$$S = \frac{10\,MB}{7\text{ ms} + 3\text{ ms} + \frac{10\,MB}{50\,MB/s}} = 47.62\,MB/s$$

Random transfer rate Again, we focus on sequential and random workloads. Assuming disks in the array have a random transfer rate R:

$$R = \frac{\text{transfer size}}{\text{time to access}}$$

Example:
Consider a 10 kilobyte transfer and a hard disk with the following characteristics:

- Average seek time of seven milliseconds.

- Average rotational delay of three milliseconds.
- Transfer rate of 50 megabytes per second.

The transfer rate in this case will be:

$$S = \frac{10\,KB}{7 \text{ ms} + 3 \text{ ms} + \frac{10\,MB}{50\,MB/s}} = 0.98\,MB/s$$

4.1.3 Consistent update

Mirrored writes should be atomic, meaning that all copies are written, or none are written. However, this is difficult to guarantee. Many RAID controllers include a write-ahead log, which is a battery-backed, non-volatile storage of pending writes. A recovery procedure ensures that the out-of-sync mirrored copies are recovered.

4.1.4 Main levels comparison

		RAID 0	**RAID 1**	**RAID 4**	**RAID 5**
	Capacity	N	$\frac{N}{2}$	$N-1$	$N-1$
	Reliability	0	1 or $\frac{N}{2}$	1	1
Throughput	**Sequential read**	$N \cdot S$	$\frac{N}{2} \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
	Sequential write	$N \cdot S$	$\frac{N}{2} \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
	Random read	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
	Random write	$N \cdot R$	$\frac{N}{2} \cdot R$	$\frac{R}{2}$	$\frac{N}{4} \cdot R$
Latency	**Read**	D	D	D	D
	Write	D	D	$2D$	$2D$

Here, N refers to the number of drives, S represents sequential access speed, R denotes random access speed, and D signifies the latency required to access a single disk.

4.2 RAID zero

Data is inscribed onto a solitary logical disk and partitioned into multiple blocks dispersed across disks based on a striping algorithm. This method is preferred when prioritizing performance and capacity over reliability, necessitating a minimum of two drives. It offers cost-effectiveness since redundancy isn't utilized, meaning error-correcting codes aren't calculated or stored. It excels in write performance due to the absence of redundant data updates and parallelization. However, a single disk failure can lead to irreversible data loss.

4.2.1 Striping

The core concept behind striping is to depict an array of disks as a unified large disk, thus maximizing parallelism by distributing data across all N disks.

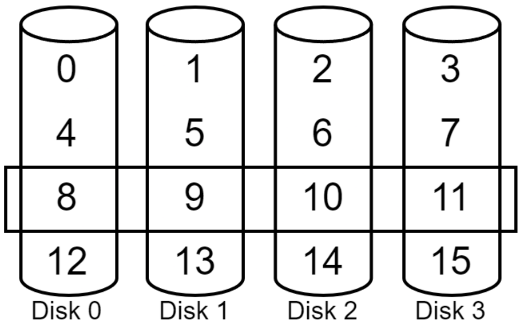


Figure 4.1: Data striping

Sequential accesses are evenly distributed across all drives, while random accesses occur naturally across all drives as well. The access to a specific data blocks can be done by computing the number of disk, and the offset:

$$\text{Disk} = \text{logical block number}\%\text{number of disks}$$

$$\text{Disk} = \frac{\text{logical block number}}{\text{number of disks}}$$

Chunk sizing Chunk size influences array performance:

- Smaller chunks lead to increased parallelism.
- Larger chunks result in reduced seek times.

Typical arrays utilize 64 kilobytes chunks.

4.2.2 Summary

The following is a summary of the features of RAID 0:

- The system has a capacity of N, meaning that it can hold all of the data on all drives.
- The system has a reliability of 0, meaning that if any drive fails, data will be permanently lost. The Mean Time to Data Loss is equal to the Mean Time to Failure.
- The system supports sequential read and write operations of $N \times S$, which means that data can be read and written in parallel across all drives.
- The system also supports random read and write operations of $N \times R$, which means that data can be read and written in a random pattern across all drives.

4.3 RAID one

RAID one is a type of data storage configuration where data is duplicated (mirrored) to a second disk. This provides high reliability, as when a disk fails, the second copy can be used. Reads of data can be retrieved from the disk with shorter queueing, seek, and latency delays. However, writes are slower than standard disks due to the duplication, and the cost of using 50% of the capacity is higher.

In principle, a RAID 1 can mirror the content over more than one disk. This provides resiliency to errors even if more than one disk breaks. Additionally, with a voting mechanism, it allows for the identification of errors not reported by the disk controller. However, in practice, this is never used because the overhead and costs are too high.

4.3.1 Mirroring

The key idea for mirroring is to make two copies of all data. This provides data redundancy, which means that if one copy of the data is lost or corrupted, the other copy can be used to recover the data. This provides a level of fault tolerance and data protection that is not present in RAID 0, which offers high performance but no error recovery. By making two copies of all data, mirroring provides both high performance and data protection.

If multiple disks are available, they can be coupled in an even number to achieve a total capacity that is halved. Each disk can have a mirror, and RAID levels can be combined to achieve the desired performance and redundancy. The two possible combinations are RAID 01 and RAID 10.

4.3.2 Multiple levels RAID

RAID levels can be combined by considering the number of groups of disks and applying RAID x to each group of n disks. Then, RAID y can be applied to the m groups of n disks, resulting in a total of $n \times m$ disks.

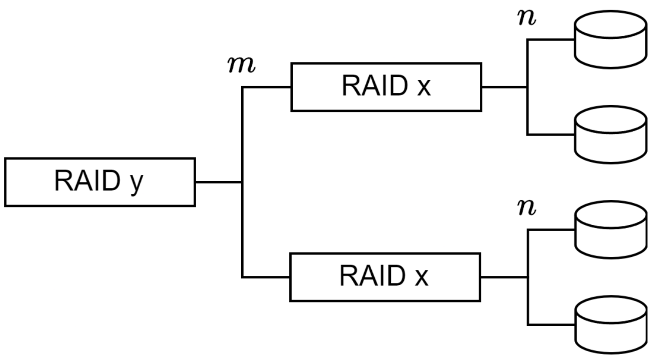


Figure 4.2: RAID levels

RAID 01 RAID level 0 + 1 is a type of RAID (Redundant Array of Independent Disks) configuration where data is first striped across multiple drives using RAID 0, and then mirrored using RAID 1. This provides both high performance and data redundancy. In the event of a failure of one of the drives in the RAID 0 stripe, the data is still available on the other drives, and the RAID 1 mirror provides an additional copy of the data for protection. The minimum number of drives required for this configuration is four.

RAID 10 RAID 10 is a type of RAID (Redundant Array of Independent Disks) level that combines the mirroring of RAID 1 with the striping of RAID 0. In RAID 10, data is first mirrored across two or more drives (RAID 1), and then the mirrored data is striped across the remaining drives (RAID 0). This provides both data redundancy and improved performance for read and write operations. RAID 10 is commonly used in databases with very high workloads, particularly those that require fast writes. A minimum of four drives are required to implement RAID 10.

4.3.3 Comparison

RAID 0, 1, 0+1, and 1+0 organizations differ in the order of block allocation. Both RAID 10 and RAID 01 have the same performance. The storage capacity of RAID 10 and RAID 01 is the same. The main difference between these RAID organizations is their fault tolerance levels:

- RAID 0+1 has less fault tolerance on most RAID controller implementations.
- RAID 1+0 has a larger fault tolerance level.

4.3.4 Summary

The following is a summary of the features of RAID 1:

- The system has a capacity of $\frac{N}{2}$, meaning it can store half of the total data.
- The system has a reliability of 1, meaning that if one drive fails, the system can still function without data loss. However, if multiple drives fail, the system may not be able to recover all of the data. In the best-case scenario, $\frac{N}{2}$ drives can fail without data loss.
- The system supports sequential read and write operations of $\frac{N}{2} \times S$, which means that it can read and write half of the data in the system at a time. This results in a throughput of $\frac{N}{2} \times S$.
- The system also supports random read and write operations of $\frac{N}{2} \times R$, which means that it can read and write half of the read blocks in the system at a time. However, since only half of the read blocks are actually being read, this results in half of the read throughput.

In the best scenario, it is possible to have $\frac{N}{2} \times R$ random writes, where N is the number of disks and R is the number of random writes per disk. This is because we have two copies of all data, so the throughput is halved. On the other hand, reads can parallelize across all disks, so it is possible to have $N \times R$ random reads in the best scenario.

4.4 RAID four

In RAID four, the N-th disk only stores parity information for the other $N - 1$ disks. The parity bit is computed using the XOR operation.

Writes There are two methods for updating parity when blocks are written:

- Additive parity involves reading other blocks and then updating the parity block. This method calculates the new parity value by XORing the old parity value with the new parity value.
- Subtractive parity involves updating the disk and then computing the new parity value. This method calculates the new parity value by XORing the old parity value with the new parity value.

Reads In RAID 4, reads are not a problem because the data is distributed evenly across all non-parity blocks in the stripe. This ensures that read operations can be performed efficiently without any significant performance reduction due to the parity disk.

RAID 4 has the same read and write performance, with parallelization across all non-parity blocks in the stripe. This means that all writes on the same stripe update the parity drive once.

Random writes In RAID 4, random writes can cause a bottleneck due to the need to update the parity drive, which can cause serialization. The process of writing to a RAID 4 array involves reading the target block and the parity block, calculating the new parity block using subtraction, and then writing the target block and the new parity block. This process can cause performance issues, especially if the parity drive is slow or has limited bandwidth.

4.4.1 Summary

The following is a summary of the features of RAID 1:

- The system has a capacity of $N - 1$, meaning that the space on the parity drive is lost.
- The system has a reliability of 1, meaning that one drive can fail. This can result in massive performance degradation during a partial outage.
- The system supports sequential read and write operations of $(N - 1) \times S$, which means that it can perform parallelization across all non-parity blocks in the stripe.
- The system supports parallelization of read operations over all drives except the parity drive, with a maximum of $(N - 1) \times R$ simultaneous reads.
- The system supports random write operations of $\frac{R}{2}$, which means that writes serialize due to the parity drive. Each write requires one read and one write of the parity drive.

4.5 RAID five

In RAID 5, the parity blocks are distributed evenly across all N disks. Unlike RAID 4, writes are spread evenly across all drives. Random writes in RAID 5 involve the following steps:

1. Read the target block and the parity block.
2. Calculate the new parity block by subtracting the old parity block from the target block.
3. Write the target block and the new parity block.

This results in a total of 4 operations (2 reads and 2 writes) being distributed evenly across all drives.

4.5.1 Summary

The following is a summary of the features of RAID 1:

- The system has a capacity of $N - 1$, meaning that the space on the parity drive is lost.

- The system has a reliability of 1, meaning that one drive can fail. This can result in massive performance degradation during a partial outage.
- The system supports sequential read and write operations of $(N-1) \times S$, which means that it can perform parallelization across all non-parity blocks in the stripe.
- The system supports parallelization of read operations over all drives, with a maximum of $N \times R$ simultaneous reads.
- The system supports random write operations of $\frac{N}{4}R$, which means that writes parallelize over all drives. Each write requires two reads and two writes.

4.6 RAID six

RAID 6 is a type of RAID level that provides more fault tolerance than RAID 5. It allows for two concurrent failures to be tolerated, which means that if two of the disks in the array fail, the data can still be recovered from the remaining disks. RAID 6 uses Solomon-Reeds codes with two redundancy schemes, which means that there are two different ways to encode the data on the disks. The (P+Q) distribution is distributed and independent, which means that the parity blocks are spread across all the disks in the array. This provides a higher level of fault tolerance than RAID 5. To implement RAID 6, a minimum of four data disks and two parity disks are required. Each write operation requires six disk accesses, as the data must be written to both the P and Q parity blocks, which can result in slow writes.

4.7 Comparison

RAID	Capacity	Reliability	Read write performance	Rebuild performance	Applications
0	100%	N/A	Very good	Good	Non critical data
1	50%	Excellent	Very good, good	Good	Critical information
5	$\frac{(N-1)}{N}$	Good	Good, fair	Poor	Database
6	$\frac{(N-2)}{N}$	Excellent	Very good, poor	Poor	Critical information
01	50%	Excellent	Very good, good	Good	Critical information with performance

Based on the context information provided, the best performance and most capacity can be achieved with RAID 0. This is because RAID 0 allows for the striping of data across multiple drives, which can significantly increase read and write speeds. However, it is important to note that RAID 0 does not provide any data redundancy or fault tolerance, so it is not recommended for use in critical data storage scenarios.

The greatest error recovery can be achieved with RAID 1 (1+0 or 0+1). This is because RAID 1 provides data redundancy by mirroring data across multiple drives. If one drive fails, the data can still be accessed from the other drives. However, it is important to note that RAID 1 can also result in a decrease in performance compared to RAID 0.

The balance between space, performance, and recoverability can be achieved with RAID 5. This is because RAID 5 provides a balance between the three by allowing for data striping across multiple drives while also providing some level of data redundancy. However, it is important to note that RAID 5 can also result in a decrease in performance compared to RAID 0.

4.7.1 Hot spares

Many RAID systems include a hot spare, which is an idle, unused disk installed in the system. In the event of a drive failure, the array is immediately rebuilt using the hot spare, allowing for seamless data recovery and minimizing downtime.

4.7.2 Implementation

RAID is a data storage technology that allows multiple hard drives or solid-state drives to be combined into a single logical unit that can be accessed as if it were a single drive. RAID can be implemented in either hardware or software.

Hardware RAID is implemented using specialized hardware controllers that are built into the motherboard or a separate expansion card. Hardware RAID is generally faster and more reliable than software RAID, as it offloads the processing of managing the RAID array to dedicated hardware. However, migrating a hardware RAID array to a different hardware controller can be challenging and may not always work successfully.

On the other hand, software RAID is implemented using software drivers that run on the computer's central processing unit (CPU). Software RAID is simpler to migrate and cheaper than hardware RAID, but it has worse performance and weaker reliability due to the consistent update problem. This problem occurs because software RAID relies on the CPU to manage the RAID array, which can cause performance issues if the CPU is not powerful enough or if there are other processes running on the computer that require CPU resources.

CHAPTER 5

Performance

5.1 Introduction

Definition (*Computer performance*)**.** Computer performance refers to the overall effectiveness of a computer system, taking into account factors such as throughput, individual response time, and availability.

It can be measured by assessing the amount of useful work that a computer system or network can accomplish in a given amount of time and resources.

Commonly, systems are predominantly validated against functional requirements rather than quality ones. Ensuring quality needs different skills, which can be hard to find. There's a trend to hurry products to market quickly, making them seem more appealing. Early on, there's often not much info about quality, but understanding it is crucial for cost and performance. This matters not just when designing and sizing the system but also as it evolves over time.

5.1.1 System quality evaluation

Intuition and trends evaluation Intuition and trend extrapolation involve making quick and flexible assessments based on gut feelings and projected patterns. However, individuals with sufficient expertise in these areas are rare, and the accuracy of their predictions may sometimes be questionable.

Experimental evaluation Assessing alternatives through experimentation can be highly beneficial, and at times, it's the preferred method. However, it can also be costly, sometimes excessively so. Another downside is that experiments may only provide detailed insights into system behavior under specific conditions, lacking broader generalizations. On the upside, they tend to offer excellent accuracy, albeit at the expense of being labor-intensive and less adaptable.

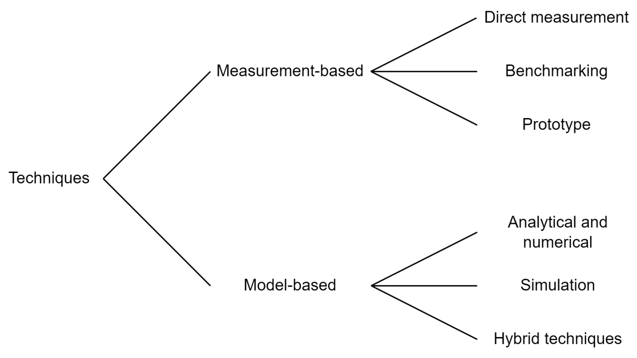


Figure 5.1: Quality evaluation techniques

5.1.2 Model-based approach

Given the complexity of systems, abstraction is necessary, achieved through the use of models. Frequently, models become the primary tool for handling these complexities, especially during phases like design. They guide design decisions by extracting essential aspects of system behavior from the intricate details.

Definition (*Model*)**.** A model serves as a simplified representation of a system, capturing its essential features while being less complex than the actual system.

It enables making predictions and evaluations about the system's behavior.

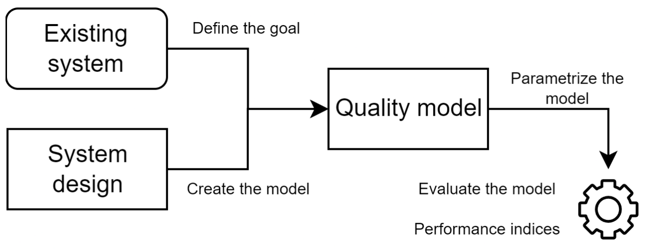


Figure 5.2: Model definition

The main model-based approaches include:

- *Analytical and numerical techniques*: these methods rely on mathematical techniques, often utilizing results from probability theory and stochastic processes. While they are highly efficient and precise, they are only available for very specific cases due to their limitations.
- *Simulation techniques*: these methods involve replicating the behavior of the model. They are versatile but may lack accuracy, particularly in scenarios involving rare events. Additionally, achieving high accuracy can result in lengthy solution times.
- *Hybrid techniques*: these approaches combine analytical or numerical methods with simulation, leveraging the strengths of both to address various aspects of the system.

5.2 Queueing networks

Queueing theory is the study of what occurs when there are numerous tasks, limited resources, resulting in lengthy queues and delays. Queueing network modeling is a specific method for computer system modeling. It represents the computer system as a network comprising queues.

Definition (*Queueing networks*)**.** A network of queues consists of service centers, representing system resources, and customers, representing users or transactions.

Queueing theory comes into play whenever queues emerge. In computer systems, queues are ubiquitous:

- The CPU utilizes a time-sharing scheduler.
- Disk operations involve a queue of requests awaiting block reads or writes.
- Routers manage queues of packets awaiting routing.
- Databases feature lock queues, where transactions await record locks.

Queueing theory aids in performance prediction, particularly for capacity planning, and is rooted in stochastic modeling and analysis. The success of queueing networks lies in their ability to abstract low-level system details, focusing instead on high-level performance characteristics.

Single queue In a single queue scenario, customers from a certain population arrive at a service facility. This facility, equipped with one or more servers, provides the required service to customers. If a customer cannot access a server immediately, they join a queue or buffer until a server becomes available. Upon completion of service, the customer departs, and the server selects the next customer from the buffer based on the service discipline or queuing policy.

5.2.1 Queueing model

Queuing models encompass various aspects, including:

- *Arrival of customers*: customer arrivals represent jobs entering the system, detailing their frequency, speed, and types. The average arrival rate, denoted as λ, is of interest.

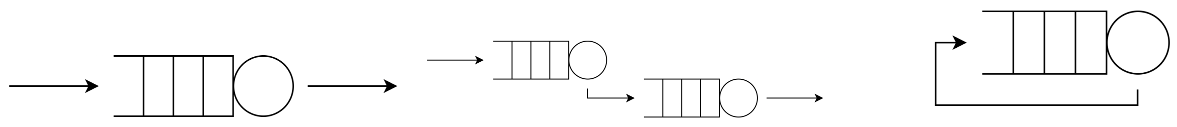

(a) Data from external source (b) Data from another queue (c) Data from same queue

Figure 5.3: Types of data arrival

- *Service*: the service aspect represents the duration a job spends being served, indicating the time a server dedicates to satisfying a customer. Similar to inter-arrival time, the key characteristics of service time include its average duration and distribution function. If the average duration of a service interaction between a server and a customer is $\frac{1}{\mu}$, then μ represents the maximum service rate. Possible situations include:

  - *Single server*: the service facility can handle only one customer at a time. Waiting customers remain in the buffer until selected for service, with the selection process depending on the service discipline.
  - *Infinite server*: there are always enough servers available for every arriving customer, eliminating queues and buffers.
  - *Multiple server facilities*: these facilities have a fixed number of servers, denoted as c, each capable of serving a customer simultaneously. If the number of customers in the facility is less than or equal to c, there is no queueing, and each customer has direct access to a server. However, if there are more than c customers, the additional customers must wait in the buffer.

- *Queue*: if the number of jobs exceeds the system's parallel processing capacity, they queue in a buffer. Customers unable to receive immediate service wait in the buffer until a server becomes available. When the buffer reaches its finite capacity, two options arise:

  - The facility being full is communicated to the arrival process, and arrivals are suspended until spare capacity is available, or a customer leaves.
  - Arrivals continue, but incoming customers are turned away until spare capacity is available again.

  If the buffer's capacity is so large that it never affects customer behavior, it's considered infinite. When a job in service leaves the system, a job from the queue can enter the now vacant service center. The service discipline or queuing policy dictates which job in the queue starts its service. In cases where multiple customers are waiting, a rule determines which waiting customer gains access to a server next. Common service disciplines include FCFS (First-Come-First-Serve or FIFO), LCFS (Last-Come-First-Serve or LIFO), RSS (Random-Selection-for-Service), and PRI (Priority), where different priorities are assigned to elements of a population.

- *Population*: The characteristic of interest regarding the population is typically its size. If the population size is fixed at a value N, no more than N customers will ever require service simultaneously. When the population is finite, the arrival rate of customers is influenced by the number already in the service facility. If the population size is so large that it has no noticeable impact on the arrival process, we assume it to be infinite. Ideally, members of the population are indistinguishable from one another. However, if they are not, we categorize the population into classes, where members exhibit similar behavior. Different classes may differ in characteristics such as arrival rate or service demand. Identifying these classes is part of workload characterization.

- *Routing*: when a job completes service at a station and has multiple potential routes, a suitable selection policy must be established. This policy, dictating how the next destination is chosen, is termed routing. Routing specifications are necessary at points where jobs leaving a station can have more than one destination. The main routing algorithms we will consider include:

  - *Probabilistic*: each path is assigned a probability of being chosen by the departing job.

- *Round robin*: the destination selected by the job rotates among all possible exits.
- *Join the shortest queue*: jobs can assess the queue length of potential destinations and opt for the one with the fewest waiting jobs to be served.

For many systems, we can conceptualize the system as a collection of resources and devices, with customers or jobs circulating among them. Each resource in the system can be associated with a service center, and customers are routed among these service centers. After receiving service at one service center, a customer may move on to other service centers based on a predefined pattern of behavior that corresponds to the customer's requirements.

5.2.2 Classification

A queueing network can be depicted as a graph, where nodes represent the service centers k and arcs signify potential transitions of users from one service center to another. Together, nodes and arcs define the network's topology.

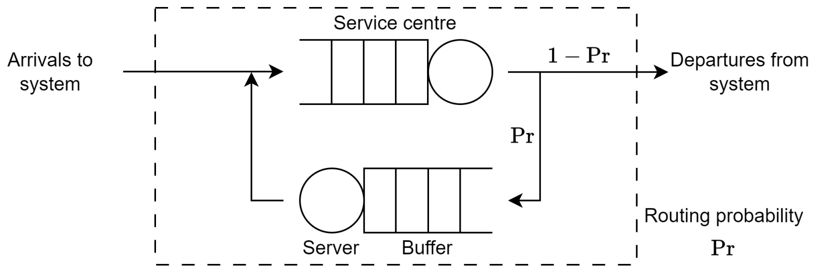


Figure 5.4: Queueing network representation

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 remains within the system. In closed models, we have a parameter N that accounts for the fixed population of jobs continuously circulating within the system.
- *Mixed*: there are classes of customers within the system exhibiting open and closed patterns of behavior, respectively.

Graphical notation Graphical notation is not standardized, but typically corresponds to a graph where edges represent the flow of customers in the network.

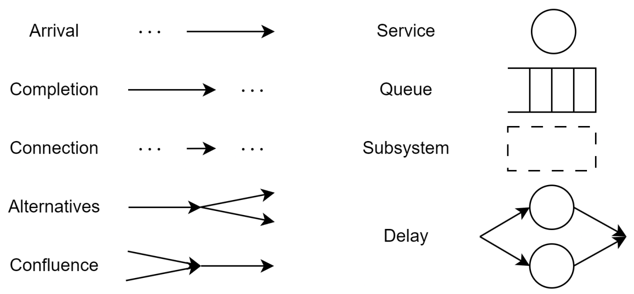


Figure 5.5: Graphical notation

5.3 Operational law

Operational laws serve as straightforward equations that offer an abstract representation or model of the average behavior of various systems. These laws are highly general and impose minimal assumptions about the behavior of the random variables characterizing the system. One notable advantage of these laws lies in their simplicity; they can be swiftly and easily applied. They are based on observable variables, which are values that we could derive from observing a system over a finite period. The underlying assumptions include:

- The system receives requests from its environment.
- Each request generates a job or customer within the system.
- Upon processing the job, the system responds to the environment by completing the corresponding request.

Measurable variables When observing 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, where $B \leq T$.
- N: the average number of jobs in the system.

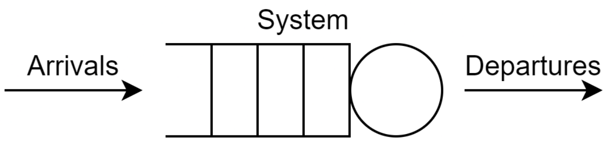


Figure 5.6: Queueing network

From these observed values, we can derive the following four important quantities:

- $\lambda = \frac{A}{T}$: the arrival rate.
- $X = \frac{C}{T}$: the throughput or completion rate.
- $U = \frac{B}{T}$: the utilization.
- $S = \frac{B}{C}$: the mean service time per completed job.

We will assume that the system is job flow balanced, meaning that the number of arrivals is equal to the number of completions during an observation period, i.e., $A = C$. This assumption is testable because an analyst can verify whether it holds true. It can be strictly satisfied by selecting a suitable measurement interval. Note that if the system is job flow balanced, the arrival rate will be the same as the completion rate, that is:

$$\lambda = X$$

Operational law A system can be viewed as comprising several devices or resources, each of which can be treated as a system on its own within the framework of operational laws. Each of these devices or resources can be considered a system in its own right within the framework of operational laws. An external request triggers a job within the system, which may then circulate among the resources until all necessary processing is completed. As it reaches each resource, it is treated as a request, generating a job internal to that resource. A system can thus be seen as comprising multiple devices or resources, each of which can be analyzed individually as a system under operational laws. We define the following variables:

- T: the length of time we observe the system.
- A_k: the number of request arrivals we observe for resource k.
- C_k: the number of request completions we observe at resource k.
- B_k: the total amount of time during which resource k is busy ($B_k \leq T$).
- N_k: the average number of jobs in resource k (either in queue or being served).

From these observed values, we can derive the following four important quantities for resource k:

- $\lambda_k = \frac{A_k}{T}$: the arrival rate.
- $X_k = \frac{C_k}{T}$: the throughput or completion rate.
- $U_k = \frac{B_k}{T}$: the utilization.
- $S_k = \frac{B_k}{C_k}$: the mean service time per completed job.

5.3.1 Utilization law

Using the previously determined observed values, we find:

$$X_k S_k = \frac{C_k}{T} \cdot \frac{B_k}{C_k} = \frac{B_k}{T} = U_k$$

From this, we derive the utilization law as:

$$U_k = X_k S_k$$

5.3.2 Little's law

Little's Law states:

$$N = XR$$

Here, N is the average number of requests in the system, X is the system throughput, measured in requests per second, and R is the average system residence time per request. Little's Law can be applied to the entire system as well as to some subsystems. If the system throughput is X requests per second, 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.

Little's law derivation Let's denote W as the accumulated time in the system (in jobs-seconds). Then, we can express:

- N, the average number of requests in the system, as $N = \frac{W}{T}$.
- R, the average system residence time per request, as $R = \frac{W}{C}$.

Thus, we can write:

$$N = \frac{W}{T} = \frac{C}{T} \cdot \frac{W}{C} = XR$$

Thus, $N = XR$.

Little's law level 1 Little's Law can be applied to the single server disk (without the queue). In this case:

- $N(1)$ represents the percentage of time in which the disk server is busy, corresponding to U_{disk}.
- $R(1)$ represents the average service time of requests.
- $X(1)$ represents the rate of serving requests.

Little's Law level 2 Now, let's include the queue. In this context:

- $N(2)$ represents the number of users in the service center (waiting and in service).
- $R(2)$ is the time spent in the service center (waiting time and service time).
- $X(2)$ is the throughput of the disk server and corresponds to $X(1)$.

Little's Law level 3 Let's focus on the central subsystem. In this scenario:

- $N(3)$ represents the total number of users in the subsystem (e.g., requests for web pages per second).
- $R(3)$ represents the average time spent in the subsystem by each request.
- $X(3)$ is the subsystem throughput (e.g., number of web pages served per second).

Little's Law level 4 Now, let's consider the complete system. Here:

- $N(4)$ is the total number of users in the system, which is fixed since we have a closed system.
- $R(4)$ is the total amount of time spent in service, waiting, and at the client-side terminals (think time, e.g., the time a user spends reading a web page and processing a request).
- $X(4)$ is the rate at which requests reach the system from the client terminals, corresponding to $X(3)$.

5.3.3 Interactive response time law

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 for some other reason. The think time represents the time between processing being completed and the job becoming available as a request again.

The interactive response time law states:

$$R = \frac{N}{X} - Z$$

The response time in an interactive system is the residence time minus the think time. Note that if the think time is zero ($Z = 0$), then $R = \frac{N}{X}$, and the interactive response time law simplifies to Little's Law.

5.3.4 Forced Flow Law

During an observation interval, we can tally not only completions external to the system but also the number of completions at each resource within the system.

Let C_k be the number of completions at resource k. We define the visit count, V_k of the k-th resource to be the ratio of the number of completions at that resource to the number of system completions:

$$V_k = \frac{C_k}{C}$$

It's important to note that:

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

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

$$X_k = V_k X$$

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. By rewriting $C_k = V_k C$ and applying $\frac{C_k}{T} = V_k \frac{C}{T}$, we can derive the forced flow law:

$$X_k = V_k X$$

5.3.5 Utilization Law with visits

If we are aware of the processing required by each job at a resource, we can calculate the resource's utilization. Let's assume that each time a job visits the k-th resource, it requires a certain amount of processing, denoted by S_k. It's important to note that the service time S_k

is not necessarily the same as the response time of the job at that resource. Generally, a job might have to wait for some time before processing begins.

The total amount of service that a job in the system generates at the k-th resource is termed the service demand, denoted as D_k:

$$D_k = S_k V_k$$

The utilization of a resource, represented by U_k, indicates the percentage of time the k-th resource is in use processing a job. As a result, the utilization law can be expressed as:

$$U_k = X_k S_k = (X V_k) S_k = D_k X$$

The utilization of a resource is equal to the product oft the throughput of that resource and the average service time at that resource, and the system-level throughput and the average service demand at that resource:

$$U_k = D_k X$$

The average service time S_k accounts for the average time a job spends at station k when it is served. The average service demand D_k accounts for the average time a job spends at station k during its stay in the system. Depending on how jobs move in the system, the demand can be less than, greater than, or equal to the average service time of station k.

5.3.6 Response and residence times

When considering nodes characterized by visits different from one, we can define two permanence times:

- *Response time* $\tilde{R}_k$: this accounts for the average time spent in station k when the job enters the corresponding node. It represents the time for the single interaction, such as a disk request.
- *Residence time* R_k: this accounts for the average time spent by a job at station k during its stay in the system. It can be greater or smaller than the response time depending on the number of visits.

It's important to note the relation between residence time and response time is similar to the one between demand and service time:

$$\begin{cases} D_k = V_k S_k \\ R_k = V_k \tilde{R}_k \end{cases}$$

Also, note that for single queue open systems or tandem models, $V_k = 1$. This implies that the average service time and service demand are equal, and response time and residence time are identical.

5.3.7 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.

Applying Little's Law to the k-th resource, we find that:

$$N_k = X_k \tilde{R}_k$$

Here, N_k is the mean number of jobs at the resource, and $\tilde{R}_k$ is the average time spent at the resource for a single interaction (e.g., a disk request).

From the forced flow law, we know that $X_k = XV_k$. Thus, we can deduce that:

$$\frac{N_k}{X} = V_k\tilde{R}_k = R_k$$

The total number of jobs in the system is clearly the sum of the number of jobs at each resource, i.e. $N = N_1 + \cdots + N_M$ if there are M resources.

From Little's Law $R = \frac{N}{X}$, and so, $R_k = \frac{N_k}{X} = V_k\tilde{R}_k$: Thus, the general response time law is:

$$R = \sum_k V_k\tilde{R}_k = \sum_k R_k$$

The average response time of a job in the system is the sum of the product of the average time for the individual access at each resource and the number of visits it makes to that resource. The average response time of a job in the system is the sum of the resources' residence time.

5.3.8 Summary

Operational laws offer simple equations that serve as an abstract representation or model of the average behavior of nearly any system. These laws are highly general and make minimal assumptions about the behavior of the random variables characterizing the system. Moreover, their simplicity allows for quick and easy application, making them valuable tools in system analysis and modeling.

5.4 Performance bounds

Performance bounds offer valuable insights into the fundamental factors influencing the performance of a computer system. They can be computed swiftly and effortlessly, making them an ideal initial modeling technique. Moreover, performance bounds allow for the simultaneous treatment of multiple alternatives, providing a comprehensive understanding of system performance.

5.4.1 Bounding analysis

Bounding analysis focuses on single-class systems and aims to establish asymptotic bounds, both upper and lower, on performance indices such as throughput (X) and response time (R). These bounds are viewed as functions of variables like the number of users or arrival rate. By employing bounding analysis, critical influences of system bottlenecks can be highlighted and quantified, offering valuable insights into system performance.

Bottleneck Bottleneck refers to the resource within a system that experiences the highest service demand, identified as $\max_k D_k$ where D_k represents the service demand at each resource. This resource, also known as the bottleneck device, plays a crucial role in limiting the overall performance of the system. Typically, the bottleneck resource exhibits the highest utilization within the system, making it a key determinant of system performance.

Bounding analysis evaluation Bounding analysis offers several advantages. It effectively highlights and quantifies the critical influence of the system bottleneck, shedding light on the factors limiting system performance. Moreover, bounding analysis techniques can be computed swiftly, even by hand, making them accessible for quick assessments. This approach proves especially useful in system sizing activities. It enables the evaluation of numerous candidate configurations, with a focus on the dominant resource, such as the CPU. By treating multiple configurations as a single alternative, bounding analysis streamlines decision-making based on preliminary estimates.

In terms of parameters and performance quantities, bounding analysis considers parameters such as the number of service centers (K), the sum of service demands at the centers (D), the largest service demand at any single center ($D_{\max}$), and the average think time for interactive systems (Z). Performance quantities of interest include the system throughput (X) and the system response time (R).

5.4.2 Asymptotic bounds

Asymptotic bounds are established by examining the extreme conditions of light and heavy loads, yielding both optimistic and pessimistic scenarios. These bounds provide insight into system performance under different operational conditions:

- *Optimistic bounds*: represent the upper limit for system throughput (X) and the lower limit for system response time (R).
- *Pessimistic bounds*: represent the lower limit for system throughput (X) and the upper limit for system response time (R).

These bounds are determined under two extreme conditions: light load and heavy load. The analysis assumes that the service demand of a customer at a center remains consistent, irrespective of the number of other customers present in the system or their locations within the service centers.

Open models asymptotic bounds In open models, where less information is available compared to closed models, asymptotic bounds provide insights into system behavior under varying conditions:

- *X bound*: represents the maximum arrival rate that the system can effectively process. If the arrival rate λ exceeds this bound, the system becomes saturated, leading to indefinite wait times for new jobs. The X bound λ_{sat} is calculated as the reciprocal of the maximum service demand $D_{\max}$.
- *R bound*: Refer to the largest and smallest possible response times experienced at a given arrival rate λ. These bounds are explored only when the arrival rate is less than the saturation arrival rate λ_{sat}, as the system becomes unstable otherwise. Two extreme situations are considered:

  1. When no customers interfere with each other, resulting in response time R equal to the sum of all service demands ($D = \sum_k D_k$).
  2. When there is no pessimistic bound on response times due to batch arrivals. As the batch size n increases, more customers wait increasingly longer times, leading to no pessimistic bound on response times regardless of how small the arrival rate λ might be.

These bounds help evaluate system performance and stability under varying load conditions, providing crucial insights for system design and optimization.

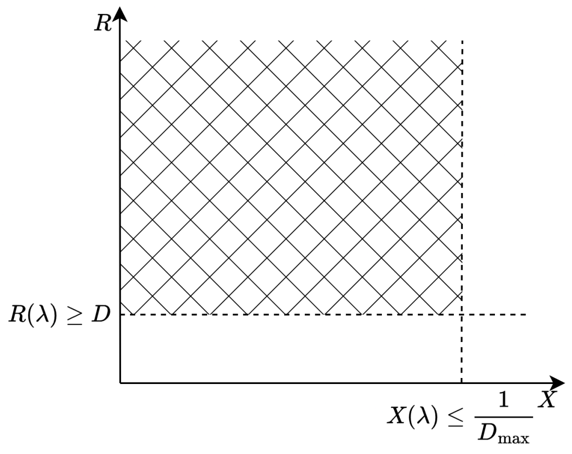


Figure 5.7: Open loop asymptotic evaluation

Closed models asymptotic bounds In closed models, we establish asymptotic bounds under both light and heavy load conditions to gain insights into system performance. Initially, we focus on determining bounds for system throughput X, which are then translated into bounds for system response time R using Little's Law. We have these following cases:

- *Light load (lower bounds)*: With one customer in the system, the system throughput X can be expressed as:

  $$X = \frac{1}{D+Z}$$

  Here, D represents the sum of service demands and Z is the think time.

  As additional customers are added, the smallest X is obtained when new jobs queue behind existing ones. This scenario results in:

  $$X = \frac{N}{ND+Z}$$

  Here, N is the total number of customers in the system. The limit of this expression approaches $X = \frac{1}{D}$ as N increases.

  In closed models, the peak system response time arises when every job, at each station, encounters all the other $N-1$ customers ahead of it in the queue.

- Light load (upper bounds): the largest X is achieved when jobs consistently find the queue empty, and service begins immediately. In this case:

  $$X = \frac{N}{D+Z}$$

  The minimum response time is achieved when a job consistently encounters an empty queue and begins service immediately.

- Heavy load (upper bound): the system's utilization at each resource U_k is constrained to be less than or equal to one. Since the bottleneck resource is the first to saturate, $X(N)$ cannot exceed $\frac{1}{D_{\max}}$. Thus, the bounds for $X(N)$ are:

$$\frac{N}{ND + Z} \leq X(N) \leq \min\left(\frac{1}{D_{\max}, \frac{N}{D+Z}}\right)$$

The parameter N^* indicates the population size at which either the light or the heavy load optimistic bound is applied, determined by

$$N^* = \frac{D + Z}{D_{\max}}$$

Considering $X(N) = \frac{N}{(R(N)+Z)}$, the bounds for $R(N)$ are:

$$\frac{N}{ND + Z} \leq \frac{N}{R(N) + Z} \leq \min\left(\frac{1}{D_{\max}, \frac{N}{D+Z}}\right)$$

Consequently, the bounds for $R(N)$ are established as:

$$\max\left(D_{\max} - Z, D\right) \leq R(N) \leq ND$$

5.4.3 What-if analysis

What-if analysis is a decision-making technique used to explore the potential outcomes of various scenarios by changing one or more input variables in a model or system. It allows users to assess the impact of different decisions or changes by simulating how they would affect the results. By adjusting parameters and observing the resulting changes, decision-makers can gain insights into potential risks, opportunities, and optimal strategies.

CHAPTER 6

Software infrastructures

6.1 Introduction

Cloud Computing represents a broad, scalable network of computing, storage, and networking resources readily available to the public. Users can access these resources via web service calls over the Internet, enabling both short-term and long-term usage based on a pay-per-use model.

6.1.1 Virtualization

Virtualization entails dividing and sharing hardware resources such as CPUs and RAM among multiple Virtual Machines (VMs). A Virtual Machine Monitor (VMM) manages the allocation of these physical resources to the VMs, ensuring performance isolation and security.

Without virtualization, software is tightly coupled with specific hardware, complicating the process of moving or modifying applications. In such environments, failures or crashes are confined to individual servers, operating systems (OS), and applications, leading to low CPU utilization. Virtualization decouples software from hardware, providing greater flexibility through pre-configured VMs. OS and applications can be managed as unified entities, simplifying deployment and management. Virtualization has significantly influenced the evolution of IT systems, particularly through server consolidation and the facilitation of Cloud Computing.

Server consolidation Server consolidation involves transitioning from physical to VMs. This approach enables the seamless movement of VMs without disrupting the applications running within them. Workloads can be automatically balanced according to predefined limits and guarantees, optimizing resource utilization and safeguarding servers and applications from component and system failures. The benefits of server consolidation include:

- Running multiple OS instances on the same hardware.
- Reducing hardware requirements, leading to cost savings in acquisition and management, and promoting environmentally friendly practices (green IT).
- Allowing the continued use of legacy software.
- Ensuring application independence from underlying hardware.

6.2 Cloud Computing

Cloud Computing is a model designed to enable convenient, on-demand network access to a shared pool of configurable computing resources. These resources encompass networks, servers, storage, applications, and services, which can be rapidly provisioned and released with minimal management effort or service provider interaction.

6.2.1 Cloud computing services

Cloud Computing offers a diverse array of services:

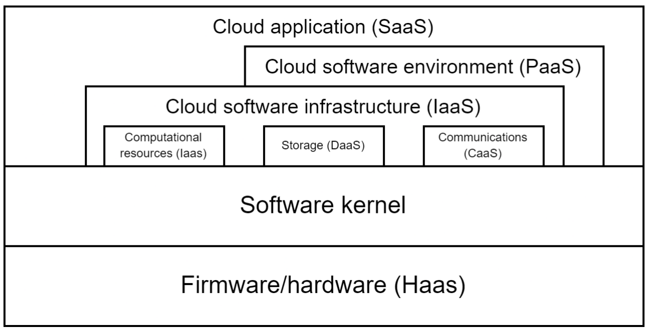

Cloud application (SaaS)
Cloud software environment (PaaS)
Cloud software infrastructure (IaaS)
Computational resources (Iaas)
Storage (DaaS)
Communications (CaaS)
Software kernel
Firmware/hardware (Haas)

Figure 6.1: Cloud Computing services

The primary service layers include:

- *Cloud application layer*: this layer is primarily composed of Software-as-a-Service (SaaS). Users access services through web portals, often paying for usage. Cloud applications are either developed within cloud software environments or use cloud infrastructure components. Examples include Gmail and Google Docs.

- *Cloud software environment layer*: this layer is mainly represented by Platform-as-a-Service (PaaS). It serves application developers by offering a programming environment with a well-defined API. This environment facilitates application-platform interaction, accelerates deployment, and supports scalability. Examples in Deep Learning include Microsoft Azure Machine Learning and Google TensorFlow.

- *Cloud software infrastructure layer*: this layer includes three main types of services:

  – *Infrastructure-as-a-Service* (IaaS): offers computational resources, including VMs and dedicated hardware. Benefits include flexibility (dynamic allocation and scaling of resources) and super-user access (fine-grained settings and customization). Challenges include performance interference (due to shared physical resources) and difficulty in guaranteeing consistent performance levels. Examples include Amazon Web Services, and Microsoft Azure.

  – *Data-as-a-Service* (DaaS): provides storage capabilities, enabling users to store data on remote disks accessible from anywhere. Key requirements include high dependability, replication for reliability, and data consistency. Examples include Dropbox, iCloud, and Google Drive.

- *Communication-as-a-Service* (CaaS): manages communications, ensuring Quality of Service (QoS) in cloud environments. Key aspects include service-oriented communication capabilities, network security, and dynamic provisioning. Examples include VoIP and video conferencing services.

6.2.2 Clouds taxonomy

Public clouds Public clouds offer extensive infrastructure available for rental, emphasizing customer self-service. Service Level Agreements (SLAs) are prominently advertised, with resources accessed remotely via the Internet. Accountability is managed through e-commerce mechanisms, including web-based transactions and customer service provisions.

Private clouds Private clouds involve internally managed data centers where an organization sets up a virtualization environment on its own servers. Benefits include total control over the infrastructure and leveraging virtualization advantages. Limitations include capital investment and less flexibility compared to public clouds. Private clouds are suitable for organizations with significant IT investments prioritizing control and security.

Community clouds Community clouds are managed by multiple federated organizations, combining resources for shared use. They resemble private clouds but require a more complex accounting system. Community clouds can be hosted locally or externally, typically using the infrastructure of participating organizations or a specific hosting entity.

Hybrid clouds Hybrid clouds combine elements of public, private, and community clouds. They are used by companies with private cloud infrastructure that may need additional resources during demand spikes. Common interfaces are crucial for simplifying deployment across hybrid environments, ensuring consistency in managing VMs, addresses, and storage. The Amazon EC2 model is a leading example of such cloud infrastructure.

6.2.3 Edge Computing

While Cloud Computing has been the primary solution for large-scale data storage and processing, the rise of intelligent mobile devices and IoT technologies demands real-time responses, context-awareness, and mobility. Due to WAN-induced delays and the need for location-agnostic resource provisioning, Edge Computing has emerged. This approach brings computing and storage closer to data sources, enabling faster response times and improved support for these requirements.

6.3 Virtualization

Definition (*Machine*)**.** A machine serves as an execution environment capable of running programs.

Computer architecture defines sets of instructions organized into various levels that programs can utilize. The OS plays a crucial role by creating new instructions and facilitating program access to devices and hardware resources.

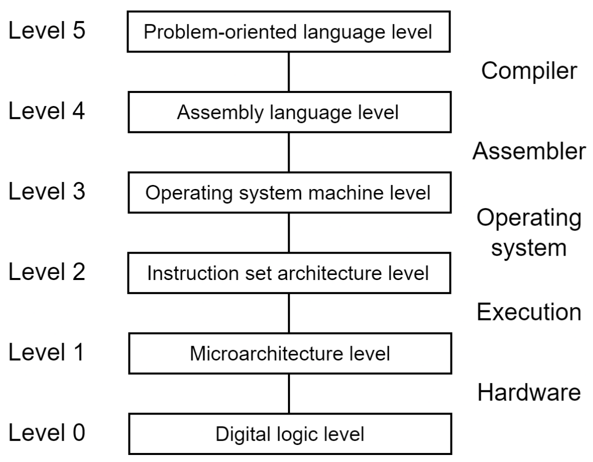


Figure 6.2: Machine's architectures levels

The main levels of virtualization include:

- *Instruction Set Architecture* (ISA): this is the second level, marking the boundary between hardware and software. ISA can be subdivided into:
  - *User*: consists of instructions visible to the application program, used for basic arithmetic operations and interfacing directly with the hardware.
  - *System*: comprises instructions visible to the OS, which manages hardware resources. This ISA abstracts CPU complexity, defining how applications access memory and facilitating hardware communication.
- *Application Binary Interface* (ABI): this is the third level, serving as the boundary between OS and software. The ABI includes aspects of the ISA visible to an application program. System calls are performed indirectly through the OS using shared hardware resources. Each machine level executes only its designated instructions, ensuring proper interaction and execution.

6.3.1 Virtual Machines

A VM is a logical abstraction providing a virtualized execution environment:

- It offers software behavior identical to that of a physical machine.
- It consists of both physical hardware and virtualizing software.
- It may present different resources than a physical machine.
- It may have varying performance levels compared to physical counterparts.

The tasks of a VM include mapping virtual resources or states to corresponding physical ones and using physical machine instructions or calls to execute virtual instructions.

There are two primary types of VMs:

1. *System VMs*: these emulate an entire physical computer system, including its hardware resources. The virtualizing software is positioned at the ISA interface. System VMs offer a complete system environment capable of supporting an OS and multiple user processes, granting the OS access to underlying hardware resources. The virtualization software, commonly called VMM, operates directly on the hardware or on top of another OS.

2. *Process VMs*: these provide a virtualized environment for individual processes. The virtualizing software is positioned at the ABI interface and is referred to as Runtime Software. Runtime software supporting process VMs typically spans three levels of the architecture.

Definition (*Host*)**.** The host is the underlying platform that supports the virtualized environment.

Definition (*Guest*)**.** The guest is the software that operates within the VM environment.

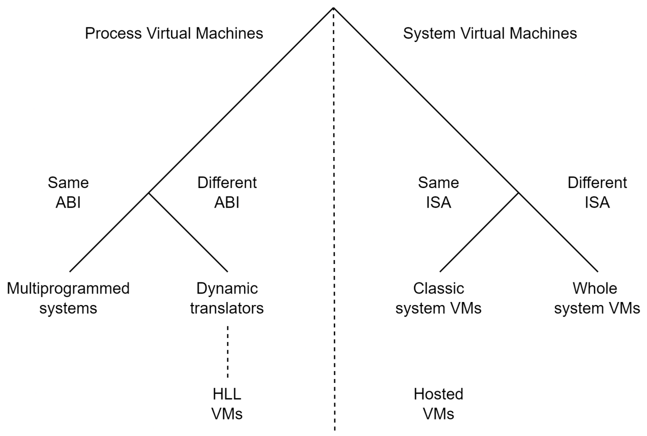


Figure 6.3: Virtual Machines taxonomy

System VMs System VMs can use the same or different ISA. Based on this, we have:

- *Classic systems* (same ISA): the VMM resides directly on the hardware, managing interactions between the guest OS and hardware resources. This configuration efficiently supports the execution of different OS on the same hardware platform.
- *Whole system* (different ISA): the VMM resides on top of an existing OS, with all software components virtualized. Due to the different ISAs, both application and OS code require emulation via binary translation.

Process VMs Process VMs can use the same or different ABI. Based on this, we have:

- *Multi-programmed systems* (same ABI): VMs are formed by the OS call interface and the user ISA. This approach supports multiple users by giving each process the illusion of having exclusive access to a complete machine, with the OS managing resource distribution.
- *High-level language system* (different ABI): provide isolated execution environments for applications or multiple instances. They translate application bytecode into OS-specific executable code while minimizing dependencies on hardware and OS. Applications run normally but with characteristics like isolation and multi-environment support, without adhering to traditional installation processes.

Emulation Emulation involves developing software technologies that enable an application or OS to operate in an environment different from its original platform. An emulator reads all bytes in the system memory it seeks to replicate. A common emulation method is interpretation, where an interpreter program sequentially fetches, decodes, and emulates the execution of individual source instructions. This method, though often slow, allows for the replication of different platform environments.

6.3.2 Implementation levels

In implementing virtualization within a typical layered system architecture, additional layers are introduced between existing layers. Depending on their placement, various types of virtualization can be achieved:

- *Hardware-level virtualization*: the virtualization layer is situated between the hardware and the OS. This configuration alters the interface seen by the OS and applications, which might differ from the physical hardware interface.
- *Application-level virtualization*: this layer is placed between the OS and specific applications. It ensures that applications receive a consistent interface, allowing them to run within their own environment independently of the OS.
- *System-level virtualization*: the virtualization layer provides the interface of a physical machine to a secondary OS and the applications running within it. This layer is positioned between the system's primary OS and other OS instances, enabling multiple OS to run on a single hardware infrastructure.

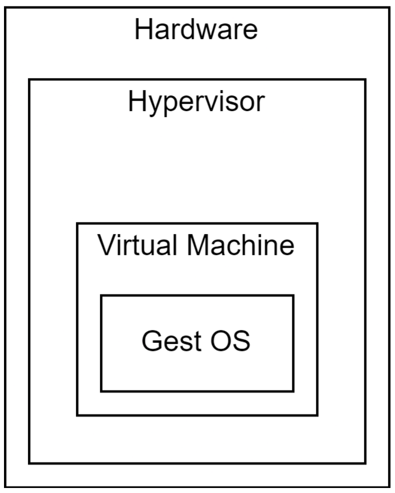
6.3.3 Virtual Machines properties

VMs possess several key properties:

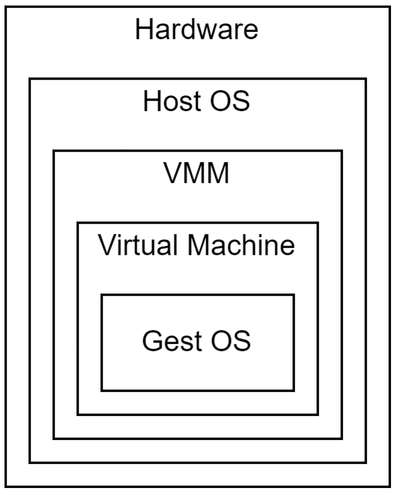
- *Partitioning*: multiple OS can run on a single physical machine with partitioned resources.
- *Isolation*: ensuring that each VM operates independently and securely.
- *Advanced resource control*: efficient and flexible management of resources.
- *Encapsulation*: the state of a VM can be saved in a file, simplifying replication and migration processes.
- *Hardware independence*: VMs can run on various hardware platforms without modification.

6.3.4 Virtual Machines Managers

A Virtual Machine Manager (VMM), also known as a Hypervisor, is responsible for mediating access to hardware resources on the physical host system. It intercepts and handles any privileged or protected instructions issued by the VMs. Typically, a VMM runs VMs with OS, libraries, and utilities compiled for the same type of processor and instruction set as the physical machine.



(a) Hypervisor type one

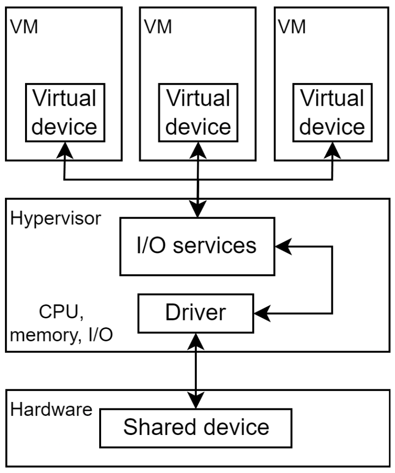


(b) Hypervisor type two

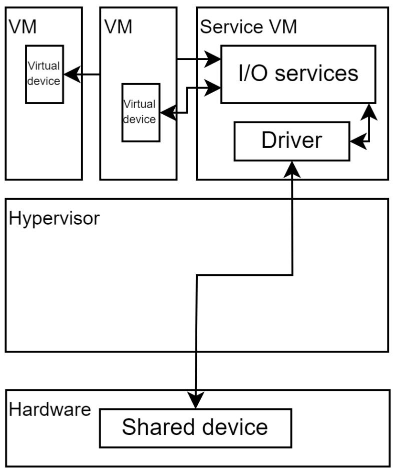
Figure 6.4: Hypervisor types

Hypervisors are classified into two types:

- *Type one* (bare metal): directly controls the hardware without requiring an underlying OS. The architecture can be:
  - *Monolithic*: device drivers run within the Hypervisor itself, offering better performance and isolation but limited to hardware supported by the Hypervisor.
  - *Microkernel*: device drivers run within a service VM, resulting in a smaller Hypervisor. This approach leverages the driver ecosystem of an existing OS and allows the use of third-party drivers.



(a) Monolithic



(b) Microkernel

Figure 6.5: Type one Hypervisor architectures

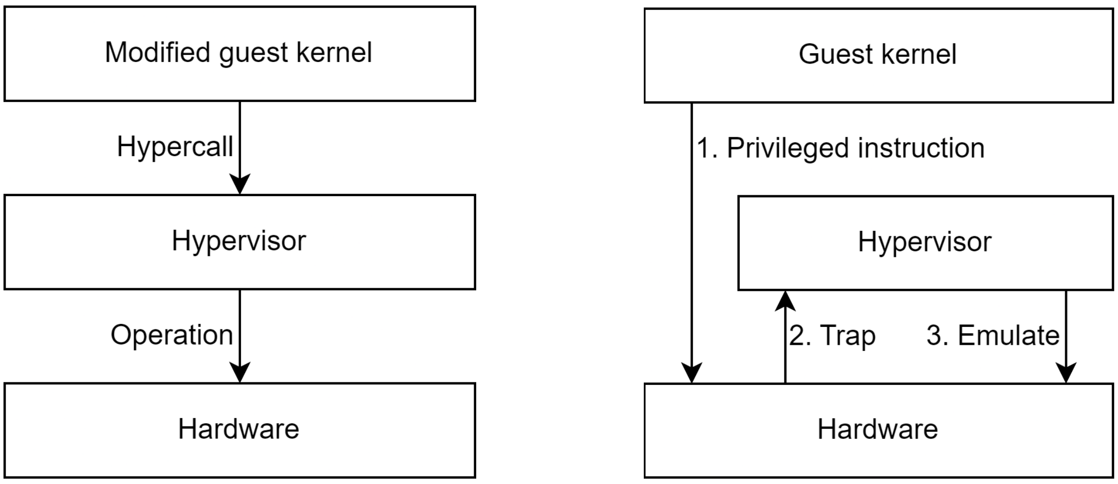
- *Type two* (VMM): requires a host OS for CPU scheduling and memory management. The host OS runs the VM, and the guest OS runs within the VM. This type is easily configurable but needs caution to prevent conflicts between the host OS and guest OS.

6.3.5 System level virtualization techniques

There are various methods to implement system level virtualization:

- *Paravirtualization*: involves collaboration between the guest OS and the VM. The VMM provides a virtual interface that resembles the underlying hardware, aiming to reduce resource-intensive tasks in the virtualized environment. This approach simplifies VMM implementation and enhances performance but requires modifications to the guest OS and is incompatible with traditional OSs.

- *Full virtualization*: provides a complete simulation of the underlying hardware. The Hypervisor intercepts and manages protected instructions, allowing unmodified OSs to run. However, this approach relies on Hypervisor mediation for communication between guests and hosts and requires specific hardware support.



(a) Paravirtualization (b) Full virtualization

Figure 6.6: System level virtualization techniques

6.4 Containers

Containers are lightweight virtualization solutions, particularly significant in DevOps contexts. They encapsulate pre-configured packages containing all necessary components for code execution. Their primary advantage lies in predictable, repeatable, and immutable behavior. When duplicating a master container onto another server, its execution remains consistent and error-free across environments.

Contrasting containers with VMs, the former offer virtualization at the OS level, sharing the host system kernel with other containers. In contrast, VMs provide hardware virtualization, with applications dependent on guest OSs. Containers possess several key characteristics:

- *Flexibility*: they can containerize even complex applications.

- *Lightweight*: containers leverage and share the host kernel, minimizing resource usage.

- *Interchangeable*: updates can be seamlessly distributed without disruption.

- *Portable*: they can be created locally, deployed in the cloud, and run anywhere.

- *Scalable*: containers enable automatic replication and distribution of replicas.
- *Stackable*: they can be vertically stacked and deployed on the fly.

Containers streamline application deployment, enhance scalability, and promote modular application development, where modules remain independent and uncoupled.

6.4.1 Docker

Docker simplifies software deployment by utilizing containers. It is an open-source platform that facilitates the building, shipping, and running of applications across diverse environments. Docker operates based on DockerFiles, which are text files containing commands to assemble application images via the command line. When executed with the Docker build command, they create immutable Docker images—snapshots of the application. Containers created with Docker can be run on various platforms.

6.4.2 Kubernetes

Kubernetes, an open-source project from Google, is well-suited for managing medium to large clusters and complex applications. It offers a comprehensive and customizable solution for efficiently coordinating large-scale node clusters in production. Kubernetes enables running containers across diverse machines within a cluster. It allows for scaling the performance of applications by adjusting the number of containers dynamically. In the event of machine failure, Kubernetes can automatically initiate new containers on alternative machines, ensuring continuous operation.