Computing Infrastructures

Christian Rossi

Academic Year 2023-2024

Abstract

This course covers several key topics related to datacenters. The hardware infrastructure section includes discussions on the basic components and rack structure, cooling mechanisms, Hard Disk Drives and Solid State Disks, RAID architectures, and hardware accelerators. The software infrastructure segment explores virtualization concepts and technologies, hypervisors, and containers, as well as the computing architectures of cloud, edge, and fog computing, and the different models of service delivery such as infrastructure-as-a-service, platform-as-a-service, and software-as-a-service. Additionally, the course delves into methods for assessing scalability and performance in datacenters, including definitions, fundamental laws, and basics of queuing network theory, as well as reliability and availability, covering definitions, fundamental laws, and 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 that facilitate computation for other systems and services.

Data centers house servers designed for various specialized functions: processing, storage, and communication.

Data centers provide numerous benefits, including lower IT costs, enhanced performance, automatic software updates, virtually unlimited storage capacity, improved data reliability, universal document accessibility, and freedom from device constraints.

However, they also pose challenges such as the necessity 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 where data processing occurs as close as possible to the data generation source, improving response times and saving on bandwidth. Processing data near its point of origin offers significant advantages, including reduced processing latency, decreased data traffic, and enhanced resilience during data connection interruptions.

Edge computing systems can be categorized into the following types:

- *Cloud*: provides virtualized computing, storage, and network resources with highly elastic capacity.
- *Edge servers*: utilizes on-premises hardware resources for more computationally intensive data processing.
- *IoT and AI-Enabled edge sensors*: facilitates data acquisition and partial processing at the network's edge.

Edge computing offers several benefits, including high computational capacity, distributed computing capabilities, enhanced privacy and security, and reduced latency in decision-making.

However, it also has drawbacks, such as the need for a power connection and reliance on cloud connectivity.

1.2.1 Embedded PCs

An embedded system refers to a computer system comprising a processor, memory, and input and output peripheral devices, all serving a specific function within a larger mechanical or electronic system. Advantages of embedded PCs include their ubiquity in computing, high performance, availability of development boards, ease of programming similar to personal computers, and support from a large community. However, they also have disadvantages, such as relatively high power consumption and the necessity for some hardware design work.

1.2.2 Internet of Things

The Internet of Things (IoT) includes devices equipped with sensors, processing capabilities, software, and other technologies designed to connect and exchange data with other devices and systems over the internet or other communication networks. Advantages of IoT devices include their widespread presence, wireless connectivity, battery-powered operation, low costs, and ability to sense and actuate. However, these devices also have several disadvantages, such as limited computing power, energy constraints, limited memory, and programming challenges.

CHAPTER 2

Hardware infrastructures

2.1 Introduction

Over the past few decades, there has been a significant shift in computing and storage from PC-like clients to smaller, often mobile devices coupled with expansive internet services. Simultaneously, traditional enterprises are increasingly adopting cloud computing. This shift offers numerous benefits, including ease of management and ubiquitous access for users.

For vendors, Software-as-a-Service (SaaS) accelerates application development and facilitates easier implementation of changes and improvements. Software fixes and enhancements are streamlined within data centers, eliminating the need for updates across millions of clients with diverse hardware and software configurations. Hardware deployment is simplified to a few well-tested configurations.

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

2.1.1 Geographical distribution of data centers

Frequently, multiple data centers serve as replicas of the same service 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.

Availability zones enable customers to operate mission-critical applications with high availability and fault tolerance to data center 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.

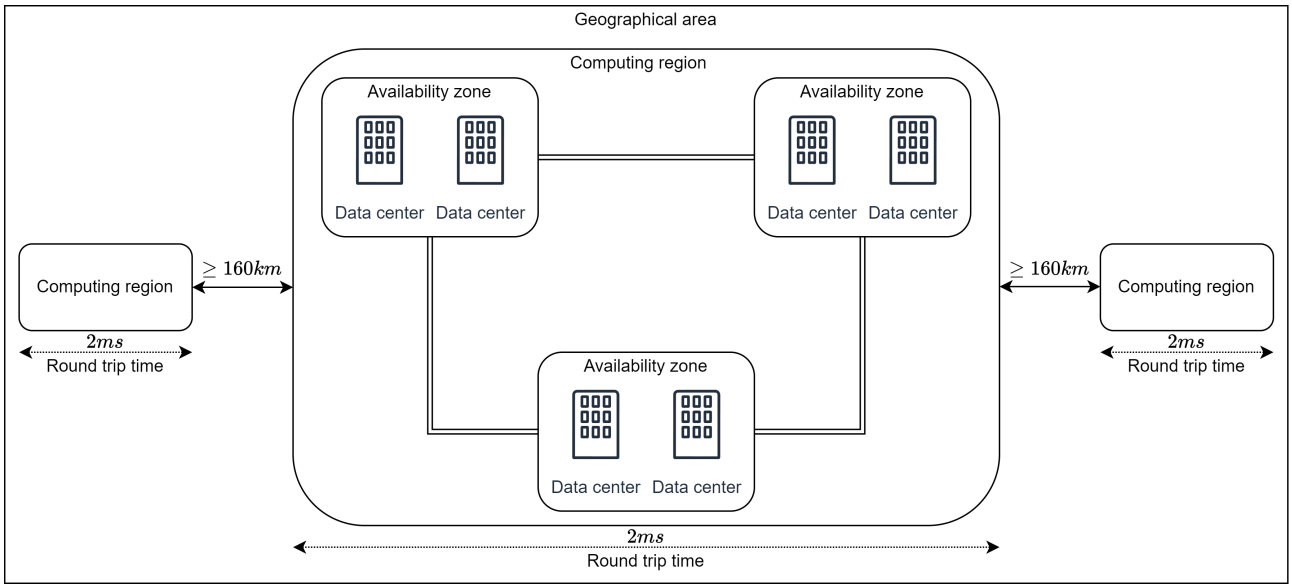


Figure 2.1: Geographical area structure

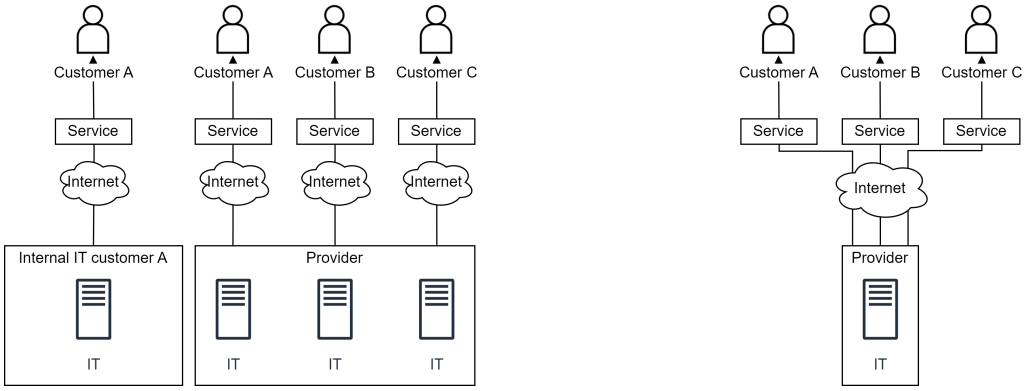
2.2 Warehouse-Scale Computers

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 WSC, 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 (ML).

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

In contrast, WSCs 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 2.2: Structures of data centers and data warehouses

WSCs 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, WSCs have expanded their capabilities to drive public cloud computing systems, such as those operated by Amazon, Google, and Microsoft. These public clouds 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.

Services offered by WSCs 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.

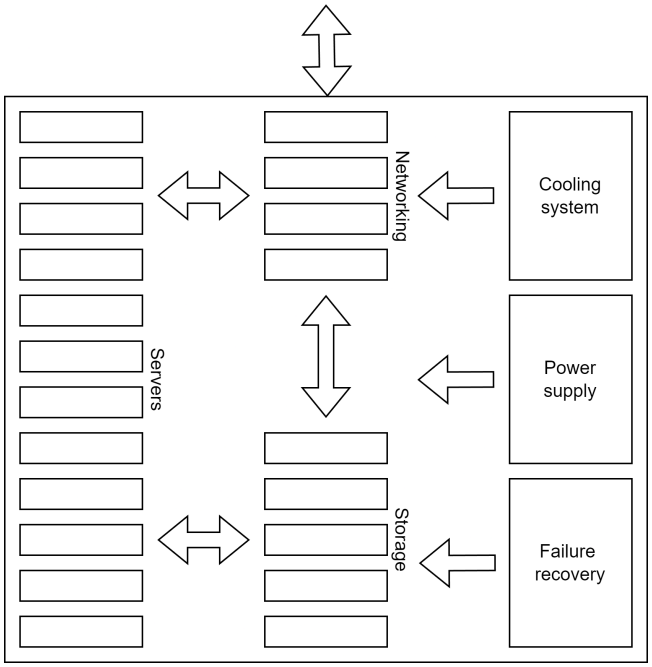


Figure 2.3: Architecture of WSC

Servers Servers resemble standard PCs but are designed with form factors that enable them to fit into racks, such as racks, 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 and coprocessors.

Storage Disks and Flash SSDs serve as the fundamental components of contemporary WSC storage systems. T These devices are integrated into the data-center network and overseen by advanced distributed systems.

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 WSC. 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 racks, blade enclosure formats, 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:

- *Central Processing Units* (CPU): varying number of CPU sockets, typically ranging from 1 to 8, and can accommodate multiple processors.

- *Random Access Memory* (RAM): from 2 to 192 memory modules.
- *Disks*: between 1 and 24 drive bays for local storage, supporting both HDD and SSD options.
- *Special purpose devices*: specialized components like GPUs or TPU.

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, with one rack unit (1U) equivalent to 44.45 mm (1.75 inches).

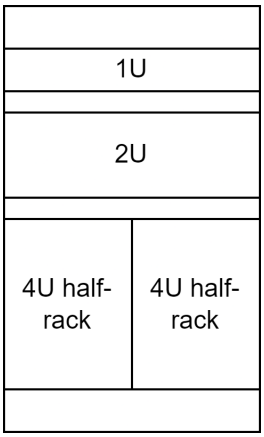


Figure 2.4: Rack elements dimensions

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. It is often convenient to connect network cables at the top of the rack.

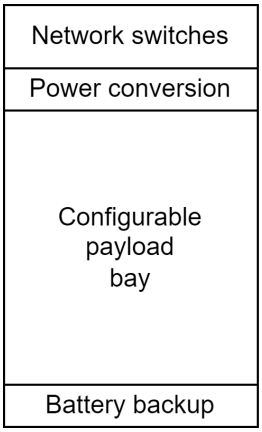


Figure 2.5: Rack structure

Advantages of using racks include the ease of failure containment, as identifying and replacing malfunctioning servers is straightforward. 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.

The high overall component density within racks leads to increased power usage, requiring additional cooling systems. 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 enhancements 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 consume a significant amount of physical space and can be difficult to manage. 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

lade 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.

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 Rack positioning

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.

2.4 Hardware accelerators

The rise and extensive adoption of Deep Learning models have ushered in a new era where specialized hardware is essential for powering various machine learning (ML) solutions. Since 2013,

the computational demands for AI training have doubled approximately every four months, significantly outpacing the traditional Moore's Law projection of two years. To meet the increasing computational requirements of Deep Learning tasks, WSC are now deploying specialized accelerator hardware optimized for the intensive processing needed for training and inference tasks in Deep Learning applications.

2.4.1 Graphical Processing Unit

Graphical Processing Unit (GPU) have transformed data processing by enabling data-parallel computations, where the same program can be executed simultaneously on multiple data elements. This parallelization is particularly effective for scientific codes, which often revolve around matrix operations. Using high-level languages, GPUs offer significant speed improvements over traditional CPU-based processing.

2.4.2 Neural Networks

Neural Networks (NN) , inspired by the human brain, consist of interconnected nodes or neurons arranged in layers to process and analyze data. They learn data representation, enabling them to function as classifiers or regressors. The resurgence of interest in NN is attributed to increased data availability and computational power.

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. Similarly, an artificial neuron receives input signals, which are weighted and summed. This sum undergoes an activation function, determining the neuron's output:

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

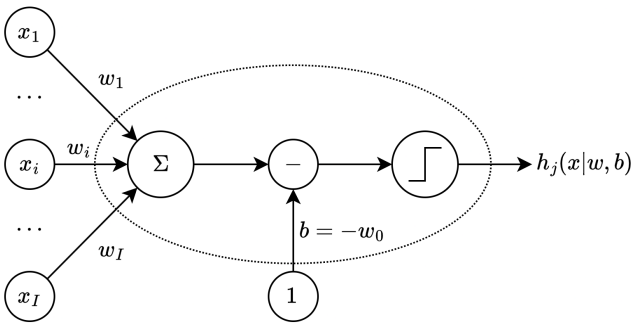


Figure 2.6: Artificial neuron

NN are structured into three main layers:

- The input layer, where data is introduced.
- The hidden layers, which process and transform the input data.
- The output layer, which provides the final results or predictions.

These layers are connected through weighted connections and activation functions, which determine the output of each neuron. Importantly, the output is influenced solely by the preceding layer's inputs.

NN are inherently non-linear models. Their behavior is shaped by various factors, including the number of neurons, the choice of activation functions, and the specific values of the weights. These factors collectively determine the network's ability to learn and make accurate predictions.

NN learn from historical data during training, often involving back propagation techniques like gradient descent to calculate errors and adjust the model. Errors are evaluated and used to refine the network's parameters, improving its prediction or classification accuracy. Various types of NN are tailored for specific tasks, including:

- *Feed Forward NN* (FFNN): where information flows in one direction from input to output.
- *Convolutional NN* (CNN): designed for processing grid-like data, such as images, with specialized layers for feature extraction.
- *Recurrent NN* (RNN): suitable for sequential data, where past information influences the present, often used in natural language processing and time series analysis.

These networks are applied in diverse domains such as image recognition (e.g., facial recognition, object detection) and natural language processing (e.g., chat bots, sentiment analysis).

GPUs are commonly used for training NN. However, the performance of such systems is constrained by the slowest learner and the slowest messages transmitted through the network, making a high-performance network essential. Typically, a CPU host is connected to a PCIe-attached accelerator tray housing multiple GPUs, which are interconnected using high-bandwidth interfaces like NVlink.

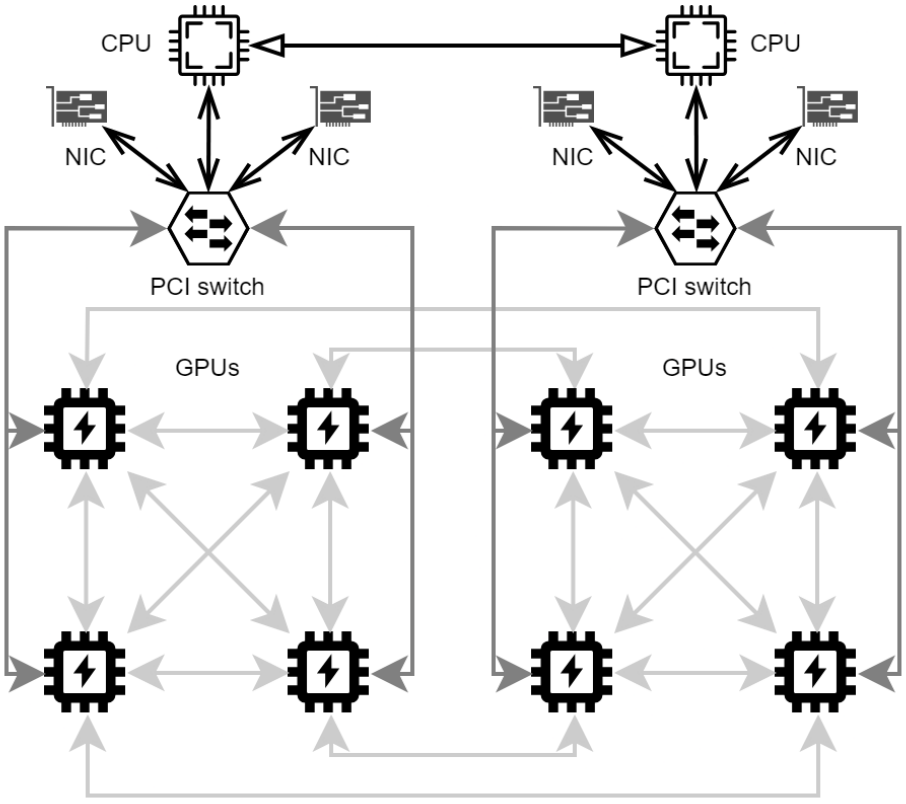


Figure 2.7: NN training

2.4.3 Tensor Processing Unit

Although GPUs are well-suited for ML, they are still considered relatively general-purpose devices. Recently, designers have specialized them into ML-specific hardware units, such as Tensor Processing Units (TPU), which have been powering Google data centers since 2015 alongside CPUs and GPUs. In TensorFlow, the basic unit of operation is a Tensor, an n-dimensional matrix.

TPUs are extensively used for both training and inference tasks, with the main versions being:

- *TPUv1*: inference-focused accelerator connected to the host CPU via PCIe links.
- *TPUv2*: supports both training and inference, providing enhanced flexibility and performance. Each TPU core consists of an array for matrix computations (MXU) and a connection to high-bandwidth memory (HBM).
- *TPUv3*: offers a 2.5 times performance boost compared to TPUv2.
- *TPUv4*: delivers a performance increase of approximately 2.7 times compared to TPUv3.
- *TPUv5*: an train large language models 2.8 times faster than TPUv4.

2.4.4 Field Programmable Gate Array

Field Programmable Gate Arrays (FPGA) 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, offering high levels of flexibility. The individual sub-circuits within FPGAs are known as Configurable Logic Blocks (CLBs). Hardware description languages like VHDL and Verilog describe hardware, enabling users to create textual representations of hardware components and their interconnections, similar to a schematic.

2.4.5 Summary

	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 primarily generated by humans. Today, machines produce data at an unprecedented pace.

Various forms of media, such as images, videos, audio files, and social media platforms, have emerged as significant sources of big data. The integration of Industry 4.0 technologies and Artificial Intelligence has further increased data production, ushering in a new era of data-centricity and innovation.

The trend now favors a centralized storage approach, which offers several advantages: reducing redundant data, streamlining storage efficiency, and automating replication and backup processes to ensure data reliability and security. This centralized model ultimately leads to reduced management costs.

Historically, HDDs have dominated the storage technology landscape, characterized by magnetic disks with mechanical interactions. However, recent advancements have introduced SSDs, which have no mechanical parts and are constructed using transistors, specifically NAND flash-based devices. Additionally, NVMe (Non-Volatile Memory Express) has emerged as the latest industry standard. Despite these innovations, tapes persist as a reliable storage solution.

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

From the perspective of an OS, disks are viewed as collections of data blocks that can be read or written independently. To facilitate their organization and management, each block is assigned a unique numerical address known as the Logical Block Address (LBA). The OS typically groups these blocks into clusters, which are the smallest units the OS can read from or write to on a disk. Cluster sizes typically range from one disk sector (512 bytes) to 128 sectors (64 kilobytes).

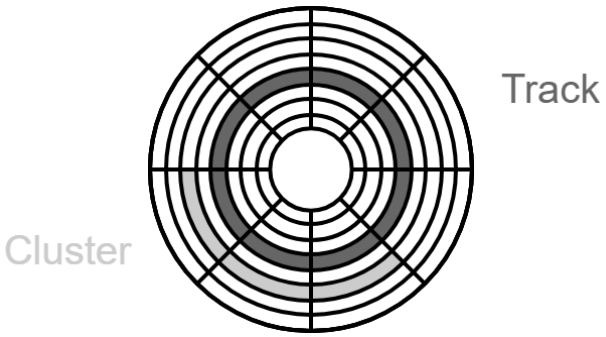


Figure 2.8: HDD structure

Clusters encompass two crucial components:

1. *File data*: The actual content stored within files.
2. *Metadata*: Essential information supporting the file system, including:
    - File names.
    - Directory structures and symbolic links.
    - File size and type.
    - Creation, modification, and last access dates.
    - Security information such as owners, access lists, and encryption details.
    - Links to the 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.

To read a file, the process involves:

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

Writing a file involves:

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:

$$a = \left\lceil \frac{s}{c} \right\rceil \cdot c$$

Here, a is the actual size on disk, s is the file size, and c is the cluster size. The wasted disk space, 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 eight bytes and a file with a size of twenty-seven bytes. The actual size on the disk would be:

$$a = \left\lceil \frac{s}{c} \right\rceil \cdot c = \left\lceil \frac{27}{8} \right\rceil \cdot 8 = 32\,\text{B}$$

And the wasted disk space would be:

$$w = a - s = 32 - 27 = 5\,\text{B}$$

Deleting a file involves updating the metadata to indicate that the blocks previously allocated to the file are now available for use by the OS. However, this process does not physically remove the data from the disk; the data remains intact until new data is written to the same clusters, overwriting the old data.

As the disk's lifespan 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 free clusters scattered throughout the disk. This process of splitting a file into non-contiguous clusters is known as external fragmentation, which can significantly degrade the performance of a Hard Disk Drive (HDD).

2.5.1 Hard Disk Drive

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

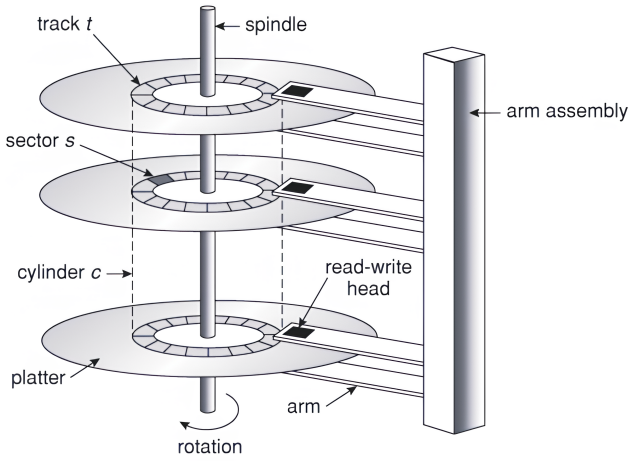


Figure 2.9: HDD structure

Externally, HDDs present a multitude of blocks. Each sector write is indivisible and includes a header along with an error correction code. However, writing multiple sectors can be interrupted, resulting in a torn write scenario, where only part 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 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 HDD, 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. 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.

Standard disk interfaces The standard disk interfaces are:

- *ST-506*: ancient standard characterized by commands and addresses in stored in device registers.
- *Advanced Technology Attachment* (ATA): standardized as Integrated Drive Electronics (IDE).
- *Integrated Drive Electronics* (IDE): used primarily in older systems.
- *Serial Advanced Technology Attachment* (SATA): current standard for connecting storage devices.
- *SCSI*: a packet-based interface with devices translating LBA to internal formats. It is transport-independent and utilized in various devices including USB drives.
- *iSCSI*: a variant of SCSI over TCP/IP and Ethernet.

Transfer time With HDD we can have four types of delay:

- *Rotational delay*: time it takes for the desired sector to rotate under the read head. It's influenced by the disk's rotational speed, measured in RPM. The full rotation delay is calculated as:
$$R = \frac{1}{RPM_{\text{Disk}}}$$
While the average rotation time is typically half of the full rotation delay, given by:
$$T_R = \frac{60 \cdot R}{2}$$
in seconds.
- *Seek 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 $T_{S_{avg}}$ is one-third of the maximum seek time $T_{S_{\max}}$.
- *Transfer time*: time required to read or write bytes of data from or to the disk.
- *Controller overhead*: 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 input output time, is determined by summing up various elements:
$$T_{I/O} = T_S + T_R + T_T + T_C$$

Example:
Let's consider a hard disk with the following specifications: a read and 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:

$$T_{I/O} = T_S + T_R + T_T + T_C = 6 + T_R + T_T + 0.2$$

Given:

$$T_R = 60 \cdot \frac{1000}{2 \cdot RPM_{\text{Disk}}} = 60 \cdot \frac{1000}{2 \cdot 10000} = 3.0 \text{ ms}$$

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

Hence:

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

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:

$$T_{I/0} = (1 - D_L)(T_S + T_R) + T_T + T_C$$

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

$$T_{I/0} = (1 - D_L)(T_S + T_R) + T_T + T_C = (0.25)(6 + 3) + 0.01 + 0.2 = 2.46 \text{ ms}$$

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:

- *OS scheduling*: requests are reordered based on 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.

2.5.2 Solid State Drive

Solid State Drive (SSD), in contrast to traditional HDD, 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. In particular, the memory is structured into pages and blocks. Each page contains several LBA, while a block generally comprises multiple pages, collectively holding approximately 128 KB and 256 KB of data:

- *Blocks*: smallest units that can be erased, typically consisting of multiple pages.
- *Pages*: these are the smallest units that can be read or written. 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 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 five pages, drive size of one 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 OS 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.

The elements in an SSD are:

- *Flash Transition Layer* (FTL): emulate the functionality of a traditional HDD for the OS. 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 collector*: recycle pages containing old data (marked as dirty or 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.
- *Mapping Table*: 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 FTL manages two distinct tables: log blocks for page-level mapping, and 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 SSD memory's erase and write cycle is limited, leading to asymmetry 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 block an erase and write cycle counter, ensuring that the difference between the maximum and minimum erase and write cycles remains below a certain threshold, denoted as e.

Summary 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 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.

2.5.3 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, 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

Storage systems can be classified into three main categories:

- *Direct Attached Storage* (DAS): these storage systems are directly linked to a server or workstation, appearing as disks within the client operating system.
- *Network Attached Storage* (NAS):a computer connected to a network, offering file-based data storage services to other devices on the network.
- *Storage Area Networks* (SAN): remote storage units connected to servers through specific networking technologies.

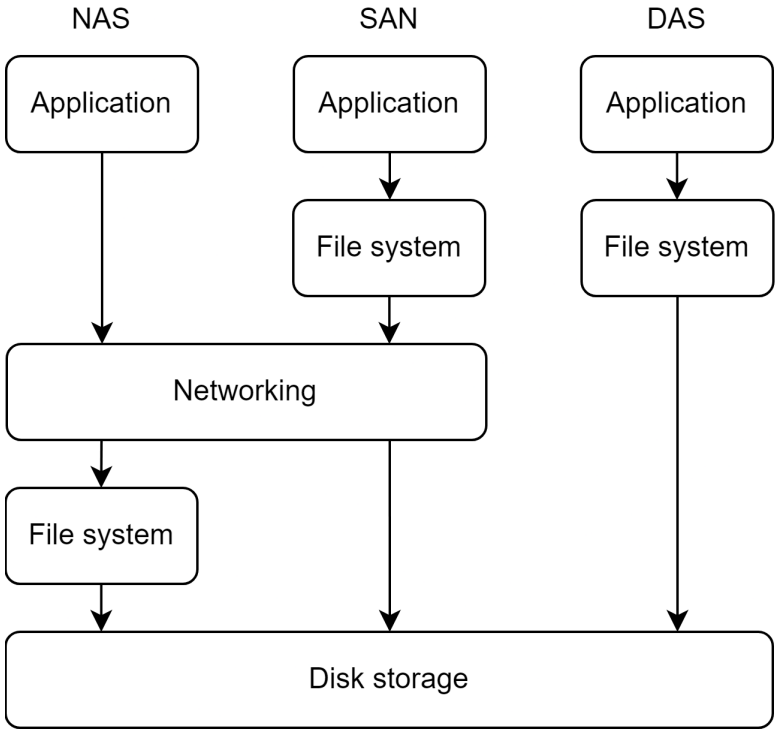


Figure 2.10: Storage system classification

2.6.1 Direct Attached Storage

Direct Attached Storage (DAS) refers to storage systems directly connected to a server or workstation. 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; any external disks connected via a point-to-point protocol to a PC can be classified as DAS.

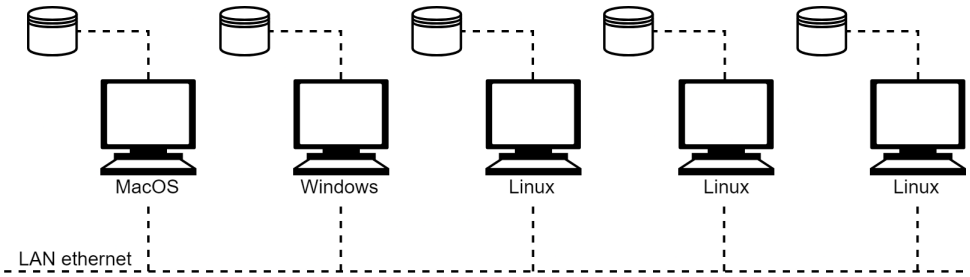


Figure 2.11: DAS architecture

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.

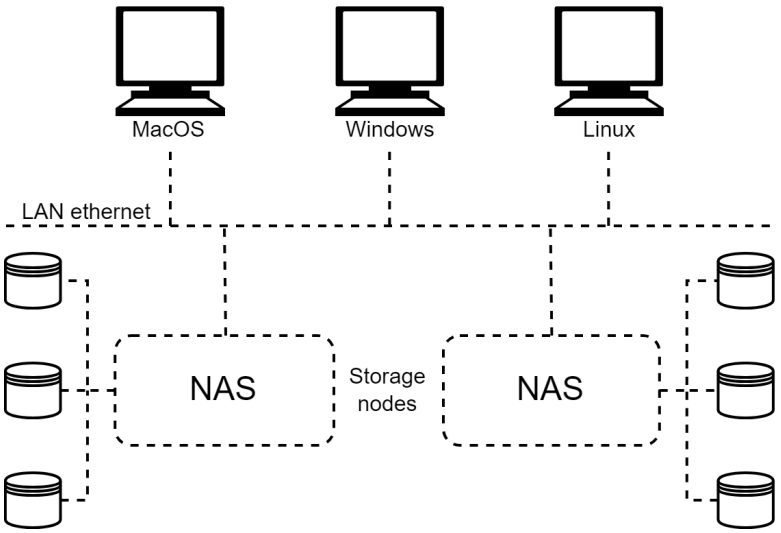


Figure 2.12: NAS architecture

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.

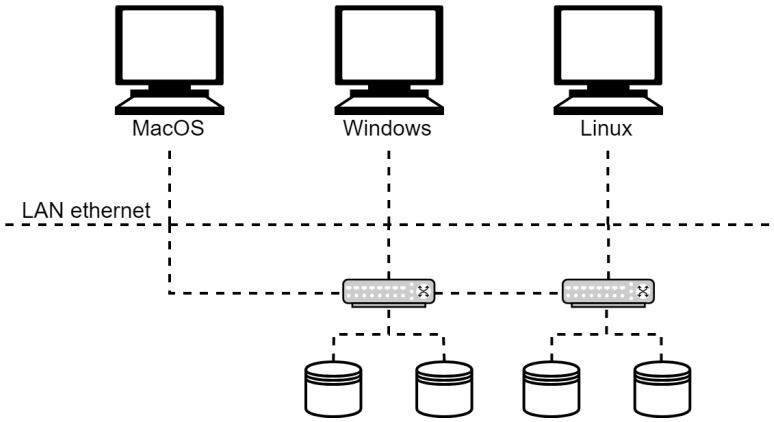


Figure 2.13: Storage Area Network architecture

2.6.4 Summary

NAS and DAS The primary distinctions between NAS and DAS lie in their design and functionality. DAS operates as an extension of an existing server and is not necessarily networked. It is directly connected to a server or workstation, appearing as local disks or volumes within the client operating system. In contrast, NAS is intentionally designed as a convenient, self-contained solution for sharing files across a network. It connects to the network, offering file-based data storage services to multiple devices. The performance of NAS is primarily dependent on the network's speed and congestion levels.

NAS and SAN NAS and SAN differ fundamentally in their storage and file system provisions. NAS offers both storage and a file system, presenting itself to the client OS as a file server. This allows clients to map network drives to shares on the NAS server. On the other hand, SAN provides only block-based storage, leaving file system management to the client side. A disk accessed through SAN appears to the client OS as a local disk, visible in disk and volume management utilities and available for formatting with a file system.

Traditionally, NAS is used for low-volume access to a large amount of storage by many users, making it suitable for file storage and sharing in big data applications. In contrast, SAN is designed for high-performance environments, such as database management systems (DBMS) and virtual environments, where petabytes of storage and simultaneous access to files are required, such as in streaming audio and video.

	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 Redundant Array of Independent Disks

Redundant Arrays of Independent Disks (RAID) is a storage technology introduced in the 1980s by Patterson. Its main goal is to enhance the performance, capacity, and reliability of storage systems by combining multiple independent disks into a single logical unit. RAID contrasts with Just a Bunch of Disks (JBOD), where each disk functions as a separate entity with its own mount point.

RAID utilizes two primary techniques: data striping to improve performance and redundancy to enhance reliability. These methods work together to provide superior storage capabilities for various computing needs.

RAID systems distribute data through input output virtualization, which means that data is transparently spread across the disks, requiring no action from users.

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

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

There are several benefits to data striping:

- Multiple independent input output requests can be executed in parallel by several disks.
- Single multiple-block input output requests can be executed by multiple disks in parallel.

Redundancy Redundancy in RAID protects against data loss due to disk failure by duplicating or distributing data across multiple disks. This ensures that 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.

Performance In redundancy-based RAID configurations, error-correcting codes are computed and stored on disks separate from those holding the primary data. These codes allow for data recovery in the event of disk failures. However, redundancy introduces performance overhead, particularly during write operations, as updates must also be made to the redundant information.

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

2.7.1 RAID 0

In RAID 0, data is inscribed onto a single 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, requiring a minimum of two drives. It is cost-effective as it does not use redundancy, meaning error-correcting codes are neither calculated nor 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.

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

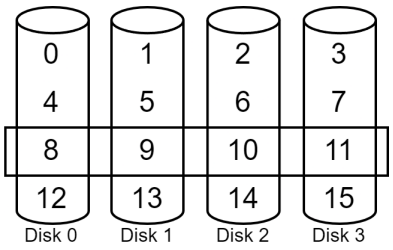


Figure 2.14: Data striping

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

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

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

Chunk size influences array performance: smaller chunks lead to increased parallelism, while larger chunks result in reduced seek times.

Capacity	N	
Reliability	0	
Sequential reads and writes	$N \cdot S$	$N \cdot S$
Random reads and writes	$N \cdot R$	$N \cdot R$
Mean Time To Failure	$\text{MTTF}_{\text{RAID 0}} = \frac{\text{MTTF}}{N}$	

Table 2.1: RAID 0

2.7.2 RAID 1

RAID 1 is a data storage configuration where data is mirrored to a second disk. This provides high reliability, as the second copy can be used if a disk fails. However, writes are slower than on standard disks due to duplication, and using half of the capacity results in higher costs.

RAID 1 can theoretically mirror content over more than one disk, enhancing resiliency even if multiple disks fail. Additionally, with a voting mechanism, it can identify errors not reported by the disk controller. However, this is rarely used due to high overhead and costs.

Mirroring Mirroring involves making two copies of all data, providing redundancy so that if one copy is lost or corrupted, the other can be used to recover the data. This approach offers both high performance and data protection.

Capacity	$\frac{N}{2}$	
Reliability	$\frac{N}{2}$	
Sequential reads and writes	$\frac{N}{2} \cdot S$	$\frac{N}{2} \cdot S$
Random reads and writes	$N \cdot R$	$\frac{N}{2} \cdot R$
Mean Time To Failure	$\text{MTTF}_{\text{RAID 1}} = \frac{\text{MTTF}^2}{2 \cdot \text{MTTR}}$	

Table 2.2: RAID 1

2.7.3 RAID 01

RAID 01 is a RAID 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 in the RAID 0 stripe, the data remains available on the other drives, and the RAID 1 mirror provides an additional copy for protection. The minimum number of drives required for this configuration is four.

2.7.4 RAID 10

RAID 10 combines the mirroring of RAID 1 with the striping of RAID 0, offering both data redundancy and improved performance. In a RAID 10 setup, data is first mirrored across two or more drives (RAID 1) and then the mirrored data is striped across additional drives (RAID 0). This configuration provides high reliability and performance for both read and write operations, making it ideal for databases and other applications with high input output demands. RAID 10 requires a minimum of four drives.

RAID 10 and RAID 01 Both RAID 10 and RAID 01 offer the same performance and storage capacity. The critical difference lies in their fault tolerance: RAID 10 is more fault-tolerant than RAID 01. In RAID 10, the system can sustain multiple disk failures as long as no mirror pair loses all its drives. In contrast, RAID 01 has a higher risk of total data loss if a disk failure occurs within the striped sets.

2.7.5 RAID 4

In RAID 4, one disk in the array 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 during writes:

- *Additive parity*: calculates the new parity value by XORing all the recomputed block parity values with the new data value.
- *Subtractive parity*: calculates the new parity value by XORing the old parity value with the new data value.

Reads Reads in RAID 4 are efficient because data is evenly distributed across all non-parity disks. This ensures parallel read operations without significant performance degradation. RAID 4 performs well with sequential reads and writes, leveraging parallelization across non-parity blocks. However, all writes on the same stripe update the parity drive once.

Random writes Random writes can cause a bottleneck in RAID 4 due to the need to update the parity drive, potentially causing serialization. Writing to a RAID 4 array involves reading the target block and the parity block, calculating the new parity block, and then writing both the target and new parity blocks. This can degrade performance, especially if the parity drive is slow or has limited bandwidth.

Capacity	$N-1$	
Reliability	1	
Sequential reads and writes	$(N-1)\cdot S$	$(N-1)\cdot S$
Random reads and writes	$(N-1)\cdot R$	$\frac{R}{2}$
Mean Time To Failure	–	

Table 2.3: RAID 4

2.7.6 RAID 5

In RAID 5, 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:

1. Reading the target block and the parity block.
2. Calculating the new parity block by subtracting the old parity block from the target block.
3. Writing the target block and the new parity block.

This process involves four operations (two reads and two writes) distributed evenly across all drives.

Capacity	$N-1$	
Reliability	1	
Sequential reads and writes	$(N-1)\cdot S$	$(N-1)\cdot S$
Random reads and writes	$N\cdot R$	$\frac{N}{4}\cdot R$
Mean Time To Failure	$\mathrm{MTTF}_{\mathrm{RAID\ 5}} = \frac{\mathrm{MTTF}^2}{N(N-1)\cdot \mathrm{MTTR}}$	

Table 2.4: RAID 5

2.7.7 RAID 6

RAID 6 provides more fault tolerance than RAID 5, allowing for the concurrent failure of two disks. RAID 6 uses Solomon-Reeds codes with two redundancy schemes (P+Q), distributing parity blocks across all disks. This configuration requires a minimum of four data disks and two parity disks, resulting in six disk accesses for each write operation. The MTTF is calculated as:

$$\mathrm{MTTF}_{\mathrm{RAID\ 6}} = \frac{2\cdot \mathrm{MTTF}^3}{N(N-1)(N-2)\cdot \mathrm{MTTR}^2}$$

2.7.8 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

Hot spares Many RAID systems include a hot spare, an idle, unused disk installed in the system. In the event of a drive failure, the array is immediately rebuilt using the hot spare, minimizing downtime.

Implementation RAID can be implemented in either hardware or software.

Hardware RAID uses specialized controllers built into the motherboard or a separate expansion card, offering faster and more reliable performance by offloading RAID management to dedicated hardware. However, migrating a hardware RAID array to a different controller can be challenging.

Software RAID uses software drivers running on the computer's CPU. It is simpler to migrate and cheaper than hardware RAID, but it generally has lower performance and reliability due to the consistent update problem, which can occur if the CPU is not powerful enough or is occupied with other processes.

2.8 Networking

There are several network architectures, including:

1. *Monolithic app*: uses minimal network resources and proprietary protocols.
2. *Client-server*: involves high network demands within enterprise applications.
3. *Web applications*: utilizes ubiquitous TCP/IP, accessible from anywhere.
4. *Microservices*: cloud providers segment servers into microservices.

Data center applications may include cloud computing, cloud storage, and web services, which help consolidate computation and network resources.

As server performance improves, the demand for inter-server bandwidth naturally increases. Doubling the number of compute or storage elements can easily double the aggregate compute capacity or storage. However, networking presents challenges for straightforward horizontal scaling. While doubling leaf bandwidth is straightforward, increasing bisection bandwidth is more complex, especially when 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.

Bisection bandwidth characterizes network capacity, representing the ability of randomly communicating processors to transmit data across the network's central portion. Ensuring every server can communicate with every other server necessitates doubling not only the leaf bandwidth but also the bisection bandwidth.

To simplify scaling, several networking architectures can be employed:

- *Switch-centric*: utilizes switches for packet forwarding tasks.
- *Server-centric*: employs servers equipped with multiple Network Interface Cards (NICs) to serve as switches alongside their computational functions.
- *Hybrid*: combines elements of both switch-centric and server-centric architectures.

2.8.1 Switch-centric architecture

In switch-centric architectures, traffic is divided into two main flows:

- *Northbound-southbound*: Direct connection between the internet and servers.

- *East-west*: Inter-server communication within a data center, often for storage replication and VM migration.

East-west traffic typically surpasses northbound-southbound traffic in volume and significance within the data center environment.

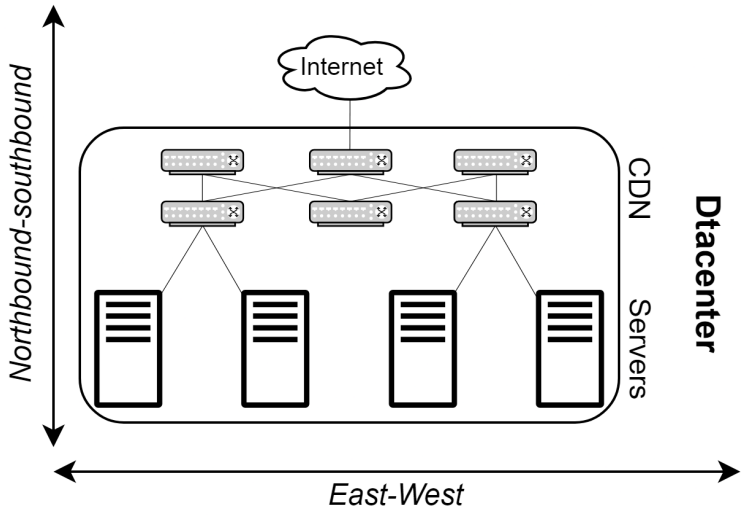


Figure 2.15: Switch-centric architecture

Classical 3-tier architecture The classical 3-tier switch-centric architecture is structured into three distinct layers:

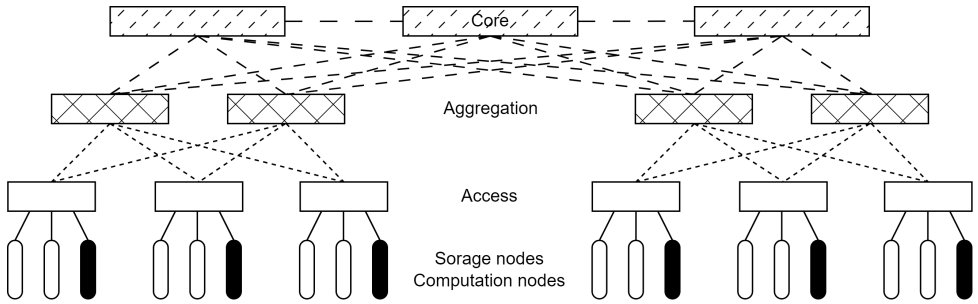


Figure 2.16: Classical 3-tier architecture

In this configuration, servers interface with the Data Center Network (DCN) via access switches.

Switches can be categorized based on their position relative to the server racks:

- *Top-of-Rack* (ToR): all servers in a rack are linked to a ToR access switch. Aggregation switches are housed either in dedicated racks or shared racks alongside other ToR switches and servers, resulting in simpler cabling and lower costs due to the limited number of ports per switch. However, this configuration has limited scalability and increased switch management complexity.

- *End-of-Row* (EoR): aggregation switches are placed at the end of each corridor, serving a row of racks. Servers connect directly to the aggregation switch in another rack. This arrangement requires more ports on the aggregation switches, more intricate cabling, and

longer, costlier cables. Typically, a patch panel facilitates connections between servers and the aggregation switch. Despite more complex cabling, this setup offers simpler switch management.

Boosting bandwidth can be achieved through several methods. Increasing the number of switches at the core and aggregation layers enhances bandwidth. Additionally, using routing protocols like Equal Cost Multiple Path (ECMP) enables equitable traffic distribution across various routes, further enhancing bandwidth capabilities.

While this solution is straightforward, it can be costly in large data centers for several reasons. The upper layers require faster, more expensive network equipment, and each layer is serviced by different types of switches, increasing acquisition costs and complicating management, spare-part stocking, and energy consumption.

Leaf-spine architecture The leaf-spine switch-centric architecture consists of two distinct layers:

- *Leaf*: ToR switch.
- *Spine*: dedicated aggregation switches.

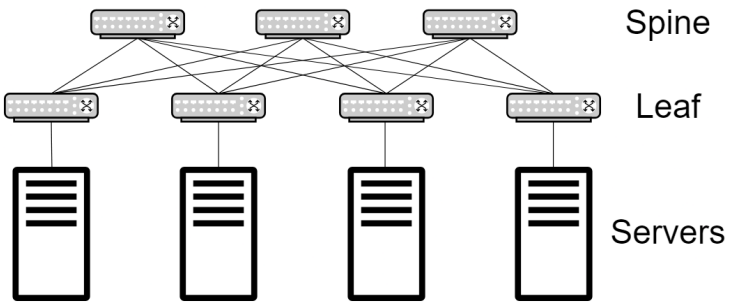


Figure 2.17: Leaf-spine architecture

Leaf-spine topologies are inspired by telephony networks, featuring a non-folded Clos structure. This architecture ensures that each stage is fully interconnected, with every matrix in one stage linked to every matrix in the subsequent stage.

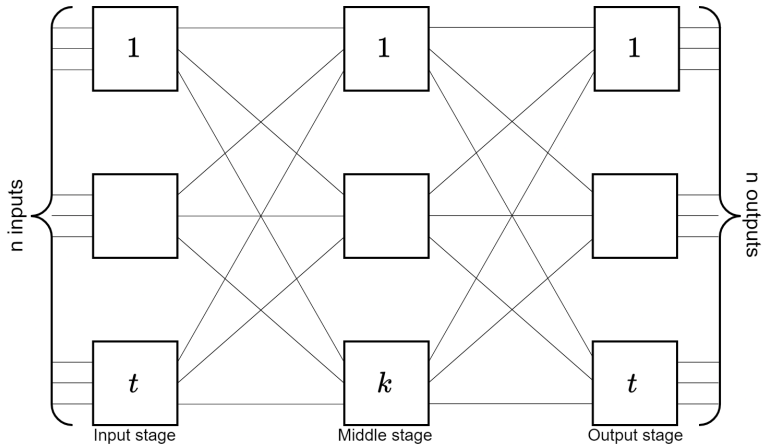


Figure 2.18: Clos network

When $k \geq n$, communication can always be reorganized to create a path between any pair of inactive input/output channels. If $k \geq 2n - 1$, a path between any pair of idle input/output channels is guaranteed to be available. The parameter t is flexible in this topology, and when $n = t = k$, each switching module functions unidirectionally. In the leaf-spine topology, every switching module operates bidirectionally:

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

The benefits of the leaf-spine architecture include:

- 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 Multiple Path (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.

To scale, a two-tier network structure can be expanded by introducing an additional row of switches or by transforming each spine-leaf group into a POD (Point of Delivery) and incorporating a super spine tier. This architecture represents a highly scalable and cost-effective DCN design.

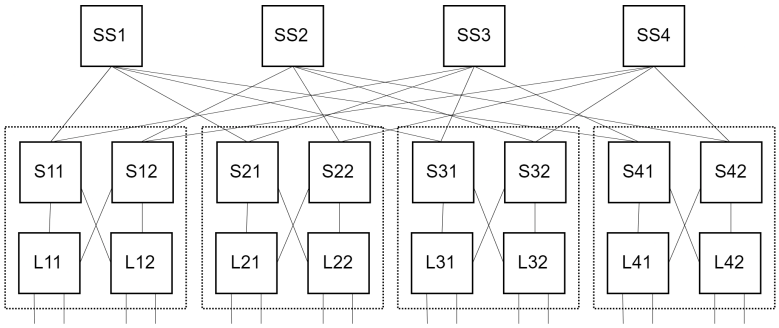


Figure 2.19: Pod-based three-tier Clos

A POD is a module or cluster comprising network, compute, storage, and application elements working together to provide a network service. The POD is a standardized and replicable pattern, enhancing the modularity, scalability, and manageability of data infrastructure. A leaf switch with $2k^2$ bidirectional ports is configured with k ports connecting to servers and k ports linking to the DCN. The POD is replicated P times, each consisting of k servers, $2P$ switches with $2k$ ports, and k switches with P ports.

Fat three architecture In a fat tree design, choosing $P = 2k$, there are $2k^3$ servers, $5k^2$ switches with $2k$ ports each, and $2k$ PODs at the edge layer, each containing $2k^2$ servers. In the edge layer, each edge switch is connected directly to k servers in a POD and k aggregation switches. Each aggregation switch is connected to k core switches, highlighting partial connectivity at the switch level.

VL2 architecture The VL2 architecture is an economical hierarchical fat-tree-based DCN architecture designed to provide high bisection bandwidth. It employs three types of switches: intermediate, aggregation, and ToR switches. A key feature of the VL2 Network is its use of a load-balancing technique called Valiant Load Balancing (VLB).

2.8.2 Server-centric 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 NICs to assemble a 3D torus network, as well as the presence of lengthy paths and heightened routing complexity.

2.9 Datacenter structure

In a typical data center building, only about half of the available space is used for actual servers. The remaining space is dedicated to cooling systems and energy management.

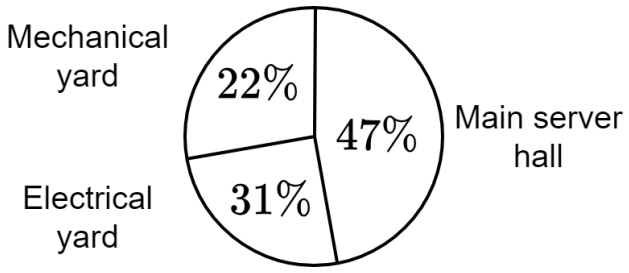


Figure 2.20: Usual building usage

A WSC includes critical components beyond servers, such as power delivery systems, cooling infrastructure, and building facilities, all of which are essential for its operation.

2.9.1 Power system

To protect against power failures, data centers use battery systems and diesel generators to back up the external power supply. A typical Uninterruptible Power Supply (UPS) integrates three functions: energy storage, switch for active power input, and elimination of voltage spikes.

Data center power consumption can reach several megawatts, with cooling systems typically consuming about half of the total energy used by IT equipment (servers, network infrastructure, and storage). Globally, data centers account for approximately 3% of the world's electricity supply and contribute about 2% of total greenhouse gas emissions. This highlights the urgent need for energy-efficient solutions and sustainable practices in the data center industry.

Power usage effectiveness Power Usage Effectiveness (PUE) measures the energy efficiency of a data center:

$$\text{PUE} = \frac{\text{Total facility power}}{\text{IT equipment power}}$$

Total facility power includes the energy consumption of IT systems (servers, network devices, storage units) and other equipment like cooling systems, UPS, switchgear, generators, lighting, and fans. Data Center Infrastructure Efficiency (DCIE) is the reciprocal of PUE.

2.9.2 Cooling systems

IT equipment generates substantial heat, requiring an expensive and efficient cooling system. This system typically includes coolers, heat exchangers, and cold water tanks. The main cooling system types are:

- *Open loop*: utilizes cold outside air to produce chilled water or directly cool servers. It is cost-effective compared to traditional chillers.
- *Closed loop*: common on the data center floor, this system extracts and expels heat from servers, directing it to a heat exchanger. Cooled air is then recirculated to the servers.
- *Two loop*: involves primary and secondary loops. The primary loop includes airflow from the underfloor plenum, through the racks, and back to the Computer Room Air Conditioning (CRAC) unit. The secondary loop circulates liquid coolant from the CRAC to external heat exchangers, which release heat into the environment.
- *Three loop*: involves primary air circuit, secondary liquid cooling loop, and tertiary heat rejection loop. This setup manages the thermal environment effectively.

A water-cooled chiller operates similarly to a water-cooled air conditioner, using cooling towers to cool a water stream by evaporating a portion into the atmosphere. In-rack and in-row coolers place air-to-water heat exchangers near the servers to shorten the distance for heat dissipation.

Direct cooling of server components can be achieved with cold plates, which are liquid-cooled heat sinks. These are used for components with the highest power dissipation, while other components rely on air cooling. The liquid carries heat to a liquid-to-air or liquid-to-liquid heat exchanger, which can be located near the tray or rack, or integrated into the data center infrastructure, such as a cooling tower.

2.9.3 Datacenter taxonomy

Datacenter availability is categorized into four distinct tier levels, each with specific 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 99.995% uptime per year Less than 26.3 minutes of downtime per year

CHAPTER 3

Software infrastructures

3.1 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.

3.1.1 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.

3.2 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.

3.2.1 Cloud computing services

Cloud Computing offers a diverse array of services:

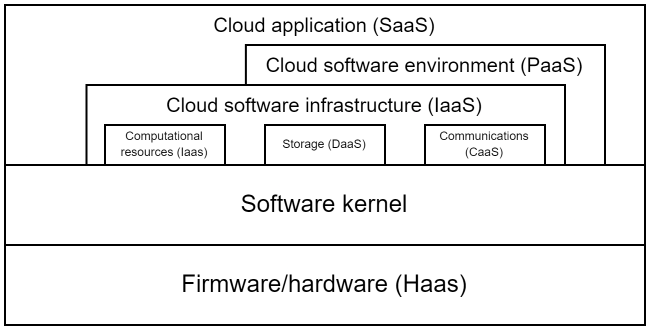


Figure 3.1: Cloud Computing services

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 ML 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.

3.2.2 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.

3.2.3 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.

3.3 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.

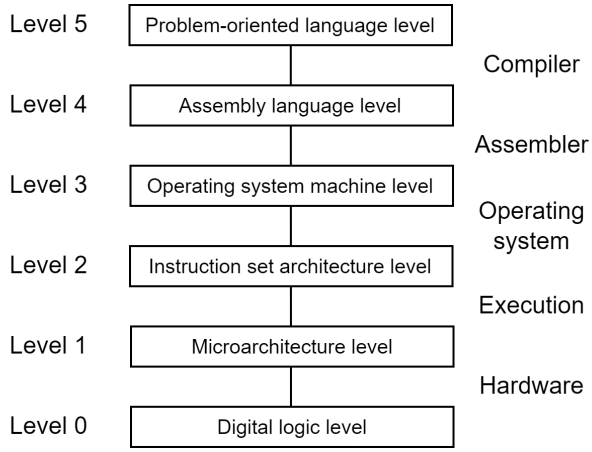


Figure 3.2: Machine's architectures levels

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.

3.3.1 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.

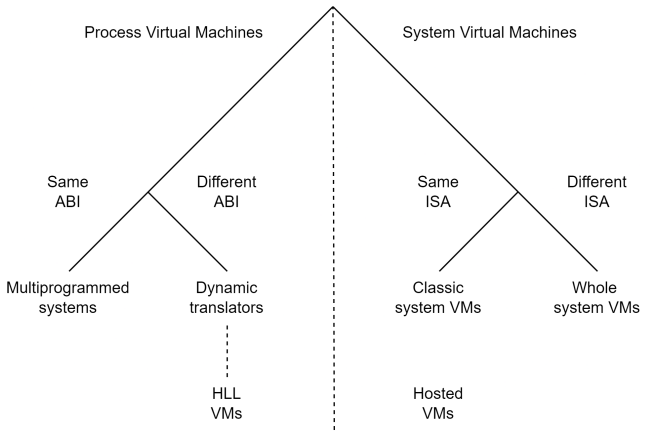


Figure 3.3: Virtual Machines taxonomy

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.

3.3.2 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.

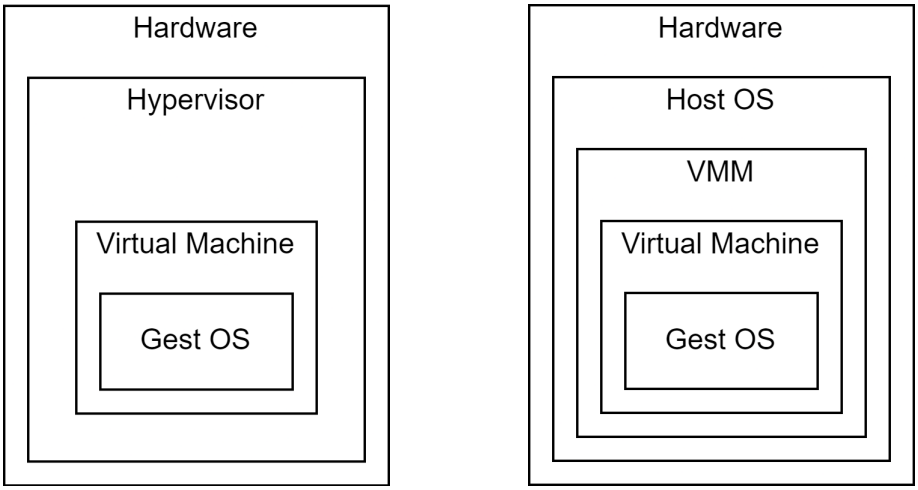
3.3.3 Virtual Machines properties

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.

3.3.4 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.

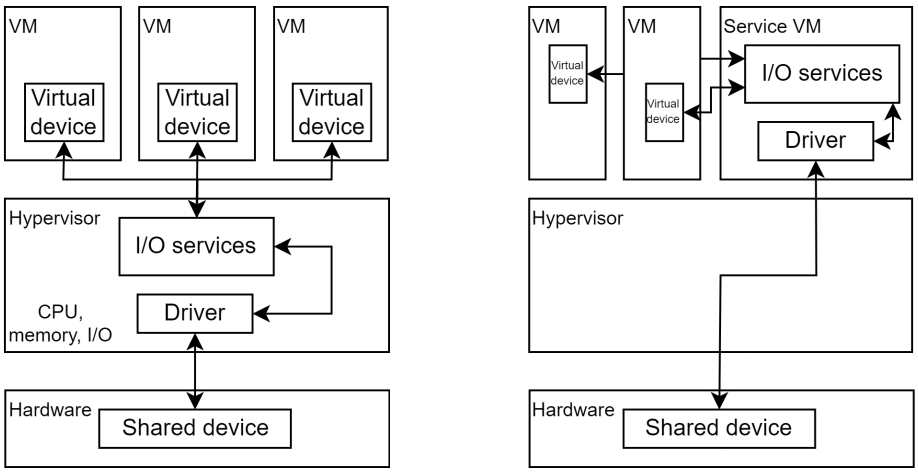


(a) Hypervisor type one (b) Hypervisor type two

Figure 3.4: 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

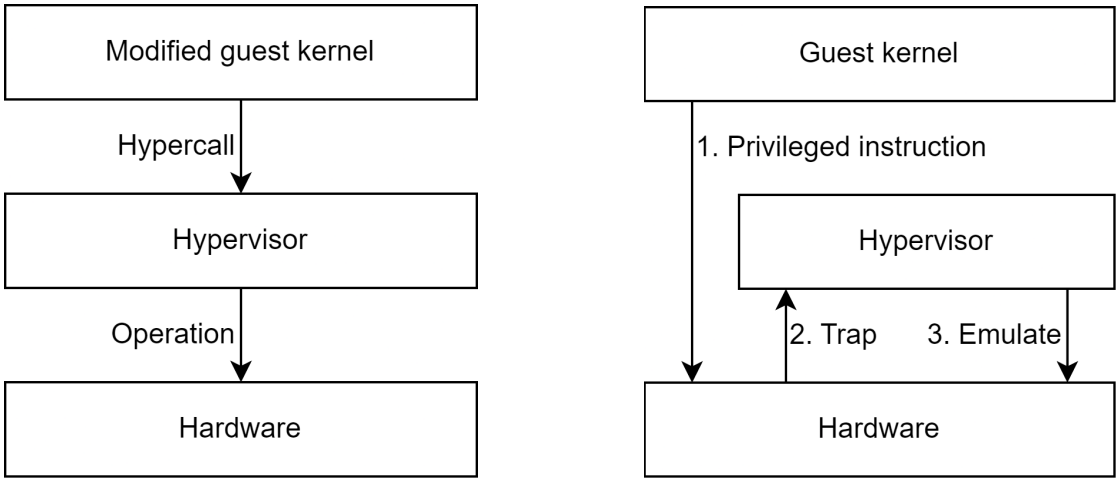
Figure 3.5: Type one Hypervisor architectures

- *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.

3.3.5 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 3.6: System level virtualization techniques

3.4 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.

3.4.1 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.

3.4.2 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.

CHAPTER 4

Dependability

4.1 Introduction

Definition (*Dependability*)**.** Dependability is a measure of the trust we place in a system.

Dependability also encompasses the system's ability to perform its intended functions, characterized by the following attributes:

- *Reliability*: the continuous delivery of correct service.
- *Availability*: the readiness for correct service.
- *Maintainability*: the ease with which maintenance can be performed.
- *Safety*: the prevention of catastrophic consequences.
- *Security*: the protection of data confidentiality and integrity.

Substantial efforts are often devoted to functional verification to ensure the system's implementation meets specifications, fulfills requirements, adheres to constraints, and optimizes selected parameters. Despite addressing these aspects, system failures can still occur, usually due to some form of breakdown.

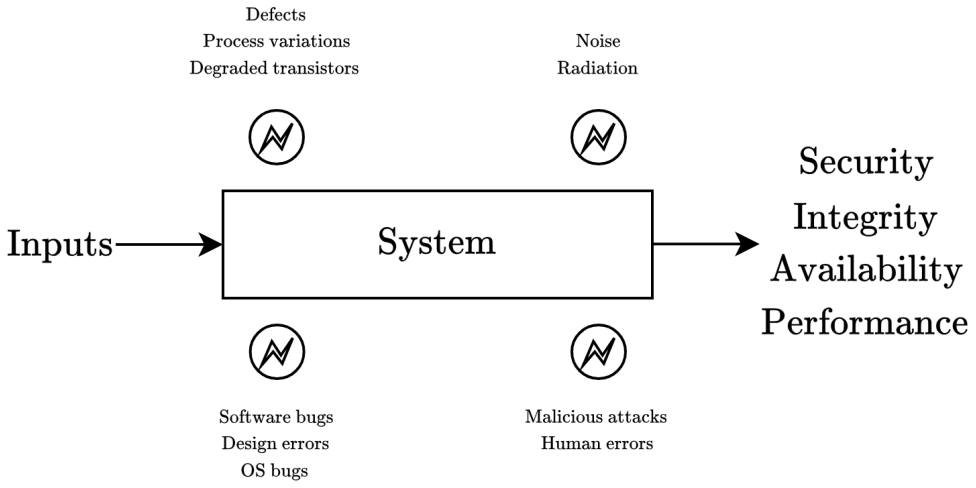


Figure 4.1: Dependability

A single system failure can impact a large number of individuals and lead to significant costs, particularly in terms of economic loss or physical damage. Systems that lack dependability are less likely to be adopted or trusted. Additionally, undependable systems can result in information loss, leading to substantial recovery expenses.

4.2 Dependability principles

Dependability is crucial both during the design phase and throughout 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 enhance dependability.

During runtime, the focus shifts to:

- Detecting any malfunctions or failures.
- Investigating and understanding the root causes of these issues.
- Implementing appropriate measures to mitigate the impact of these malfunctions.

Failures are common in both development and operational stages. While development-stage failures should be avoided, operational failures are inevitable due to the nature of system components and must be managed effectively. Additionally, the effects of these failures should be predictable and deterministic rather than catastrophic.

Historically, dependability was primarily a concern for safety-critical and mission-critical applications, such as space exploration, nuclear facilities, and avionics, due to the high costs associated with ensuring dependability. This high cost was justified only in situations where it was absolutely necessary.

In non-critical systems, operational failures can lead to economic losses and reputational damage, as seen in consumer products. However, in mission-critical systems, operational failures can have serious or irreversible consequences for the mission. In safety-critical systems, failures pose a direct threat to human life if they occur during operation.

4.3 Datacenters dependability

In data centers, downtime is a significant concern. Key components requiring dependability include computing systems, sensors, and actuators (collectively referred to as nodes), network infrastructure for communication, and cloud services for data storage and processing. The seamless operation of these elements is crucial for the system's overall functionality.

4.3.1 Dependability requirements

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

1. Employing a conservative design approach.

2. Thoroughly validating the design.
3. Conducting detailed testing of both hardware and software components.
4. Implementing an infant mortality screen to identify and address early failures.
5. Focusing on error avoidance strategies.
6. Incorporating mechanisms for error detection and masking during system operation.
7. Utilizing online monitoring tools.
8. Implementing diagnostics for identifying issues.
9. Enabling self-recovery and self-repair capabilities.

In practice, dependable systems can be achieved through robust design (emphasizing error-free processes and design practices) and robust operation (employing fault-tolerant measures such as monitoring, detection, and mitigation).

4.3.2 Dependability in practice

Dependability can be enhanced at various levels:

- *Technology*: incorporating reliable and robust components during design and manufacturing.
- *Architecture*: integrating standard components with solutions designed to handle potential failures effectively.
- *Application*: developing algorithms or operating systems that can mask and recover from failures.

However, enhancing dependability typically entails increased costs and potential reductions 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 depends on factors such as the application field, working scenario, employed technologies, algorithms, and applications.

Dependability often involves trade-offs with performance and power consumption, necessitating careful consideration of these trade-offs.

4.4 Reliability and availability

Dependability can be subdivided into the following components:

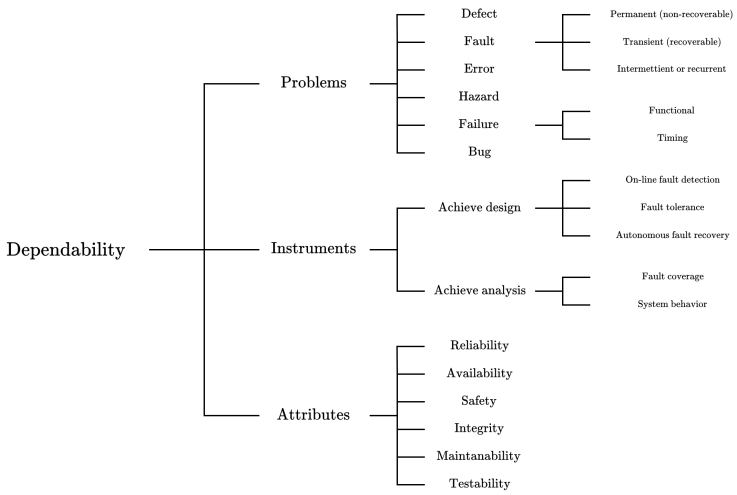


Figure 4.2: Dependability components

4.4.1 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.

Reliability is denoted as the probability that the system operates correctly in a given environment until time t, expressed as:

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

Reliability is a monotonically decreasing function ranging from one to zero. This metric is often applied to systems where even brief periods of malfunction are intolerable or those that are difficult or impossible to repair.

Unreliability is the complementary measure to reliability, calculated as:

$$U_R(t) = 1 - R(t)$$

4.4.2 Availability

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

Availability is denoted as the probability that the system will be operational at time t:

$$A(t) = \Pr(\text{not failed at time } t) = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

Unavailability is the complementary measure to availability, calculated as:

$$U_A(t) = 1 - A(t)$$

4.4.3 Other indices

Definition (*Repairable system*)**.** A system is said to be repairable if $A(t) > R(t)$.

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

Definition (*Mean Time To Failure*)**.** The Mean Time To Failure (MTTF) represents the average duration before any failure occurs.

MTTF can also be calculated as the integral of reliability:

$$\text{MTTF} = \int_0^\infty R(t)dt = \frac{\text{total operating time}}{\text{number of failures}}$$

Definition (*Mean Time Between Failures*)**.** The Mean Time Between Failures (MTBF) denotes the average duration between two consecutive failures.

Definition (*Failures In Time*)**.** Failures In Time (FIT) refer to the reciprocal of the MTBF:

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

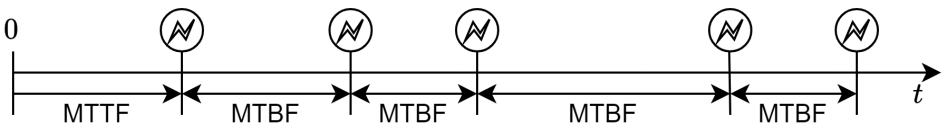


Figure 4.3: MTTF and MTBF

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

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.

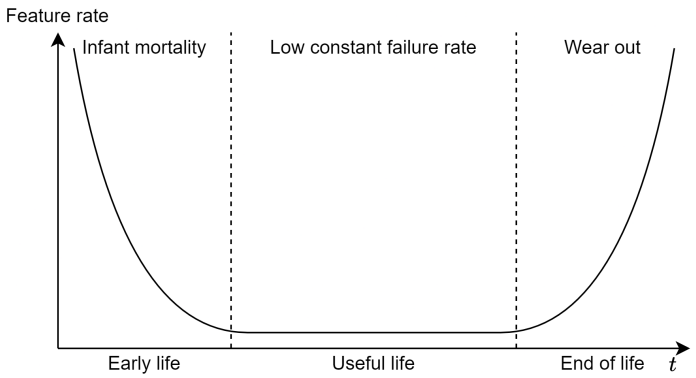


Figure 4.4: Product reliability curve

4.4.4 Defect identification

To identify defective products and determine the 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. However, ensuring high reliability in systems can pose greater challenges.

Using information from the reliability function allows for the computation of a complex system's reliability over time, representing its expected lifetime.

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

Definition (*Error*)**.** An error is 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.

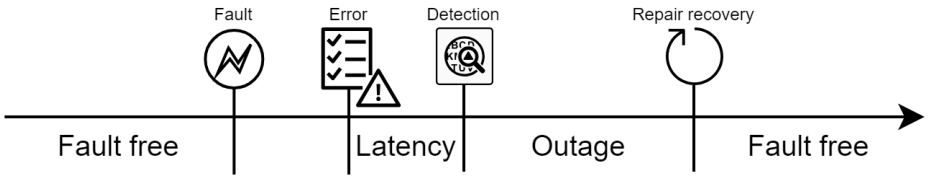


Figure 4.5: Reliability measures

4.4.5 Block Diagrams

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

Reliability Block Diagram The block reliability can be defined as:

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

Reliability Block Diagrams (RBD) provide an approach to compute the reliability of a system based on the reliability of its components.

In RBD, the elements can be connected in two ways:

- *Series*: all components must be operational for the system to function properly. For systems with n blocks in series, the total reliability is:

$$R_S(t) = \prod_{i=1}^{n} R_i(t) = \prod_{i=1}^{n} e^{-\lambda_i t}$$

- *Parallel*: at least one component is operational, the system functions properly. For systems with n blocks in parallel, the total reliability is:

$$R_S(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) = 1 - \prod_{i=1}^{n} \left(1 - e^{-\lambda_i t}\right)$$

Availability Block Diagram Availability Block Diagrams, the elements can be connected in two ways:

- *Series*: all components must be operational for the system to function properly. For systems with n blocks in series, the total availability is:

$$A_S(t) = \prod_{i=1}^{n} A_i(t) = \prod_{i=1}^{n} \frac{\text{MTTF}_i}{\text{MTTF}_i + \text{MTTR}_i}$$

- *Parallel*: if at least one component is operational, the system functions properly. For systems with n blocks in parallel, the total availability is:

$$A_S(t) = 1 - \prod_{i=1}^{n} (1 - A_i(t)) = 1 - \prod_{i=1}^{n} \left(1 - \frac{\text{MTTF}_i}{\text{MTTF}_i + \text{MTTR}_i}\right)$$

4.4.6 Redundancy

A system may consist of two parallel replicas, where the primary replica operates continuously, and the redundant replica is activated only when the primary replica fails.

For operational redundancy, the system requires 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. A system with one primary replica and n redundant replicas (identical and perfectly switchable) can be represented by the reliability function:

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

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

$$R_S(t) = R_V(t) \sum_{i=r}^{n} R(t)^i (1 - R(t))^{n-i} \binom{n}{i}$$

Here, $R(t)$ denotes the component reliability, and $R_V(t)$ signifies the switching reliability.

CHAPTER 5

Performance

5.1 Introduction

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

Computer performance can be evaluated by measuring the amount of useful work a computer system or network accomplishes within a specific time frame and resource allocation.

Systems are often validated primarily against functional requirements, with less emphasis on quality requirements early in the development process. This is partly due to limited initial information about system quality. However, understanding quality is crucial for managing both cost and performance throughout the system's lifecycle. This understanding is essential not only during the design and sizing phases but also as the system evolves.

5.1.1 System quality evaluation

System quality evaluation involves both intuition and trend extrapolation, allowing for quick and flexible assessments based on gut feelings and projected patterns. However, experts with sufficient experience in these areas are scarce, and the accuracy of their predictions can sometimes be questionable.

Experimentation is a highly beneficial and often preferred method for evaluating alternatives. While experiments provide detailed insights into system behavior under specific conditions, they can be costly and labor-intensive, lacking broader generalizations. Despite these limitations, experiments typically offer high accuracy.

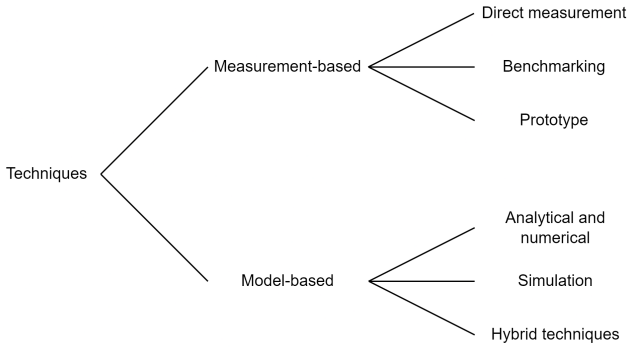


Figure 5.1: Quality evaluation techniques

5.1.2 Model-based approach

Due to the complexity of systems, abstraction through modeling is necessary.

Definition (*Model*)**.** A model is a simplified representation of a system that captures its essential features while being less complex than the actual system.

Models are valuable for predicting and evaluating system behavior.

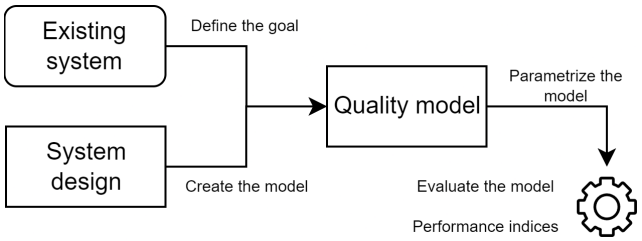


Figure 5.2: Model definition

The main model-based techniques include:

- *Analytical and numerical methods*: these rely on mathematical techniques, often utilizing probability theory and stochastic processes. They are highly efficient and precise but tend to be very specific.
- *Simulation*: this involves replicating the behavior of the model. Simulations are versatile but may lack accuracy in scenarios involving rare events. Achieving high accuracy can also result in lengthy solution times.
- *Hybrid methods*: these 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 studies the behavior of systems with numerous tasks and limited resources, leading to queues and delays. It models computer systems as networks of queues.

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

In computer systems, queues are prevalent:

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

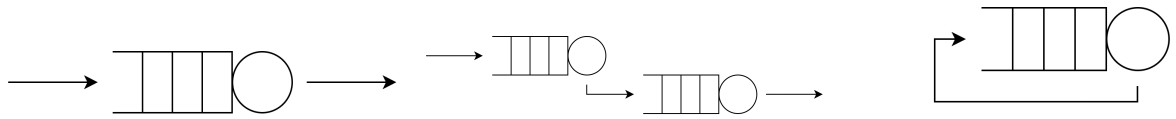
Queueing theory aids in performance prediction, particularly for capacity planning, and relies on stochastic modeling and analysis. The success of queueing networks lies in their ability to abstract low-level system details, focusing instead on high-level performance characteristics.

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

The elements of queueing models are:

- *Arrival of customers*: customer arrivals represent jobs entering the system, detailing their frequency, speed, and types.



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

Figure 5.3: Types of data arrival

- *Service*: he duration a job spends being served, indicating the time a server dedicates to satisfying a customer. The key characteristics of service time include its average duration and distribution function. Possible scenarios include:
  - *Single server*: the service facility can handle only one customer at a time. Waiting customers remain in the queue until selected for service, with the selection process depending on the service discipline.
  - *Multiple servers*: a fixed number of servers, each capable of serving a customer simultaneously. If the number of customers is less than or equal to the number of servers, there is no queueing, and each customer has direct access to a server. Otherwise, additional customers must wait in the queue.
  - *Infinite servers*: there are always enough servers available for every arriving customer, eliminating queues.

- *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, suspending arrivals until capacity is available, or a customer leaves.
  - Arrivals continue, but incoming customers are turned away until capacity is available again.

  When a job in service leaves the system, a job from the queue can enter the now vacant service center. The 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 FIFO, LIFO, random selection, and priority.
- *Population*: the characteristic of interest regarding the population is typically its size. If the population size is fixed, 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, it is assumed to be infinite. Users can be categorized based on their behaviors, with different classes differing in characteristics such as arrival rate or service demand.
- *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. The main routing algorithms include probabilistic (each path is assigned a probability), round robin (job rotation among all paths), and shortest queue (path with the least occupied queue).

For many systems, the system can be conceptualized 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 corresponding to the customer's requirements.

5.2.2 Taxonomy

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

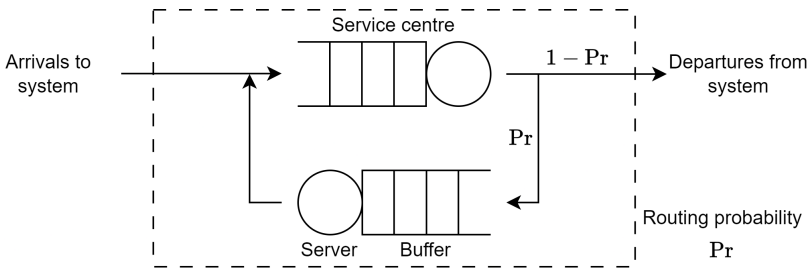


Figure 5.4: Queueing network representation

A network may be:

- *Open*: customers may arrive from or depart to some external environment.
- *Closed*: a fixed population of customers remains within the system.
- *Mixed*: classes of customers within the system exhibit both open and closed patterns of behavior.

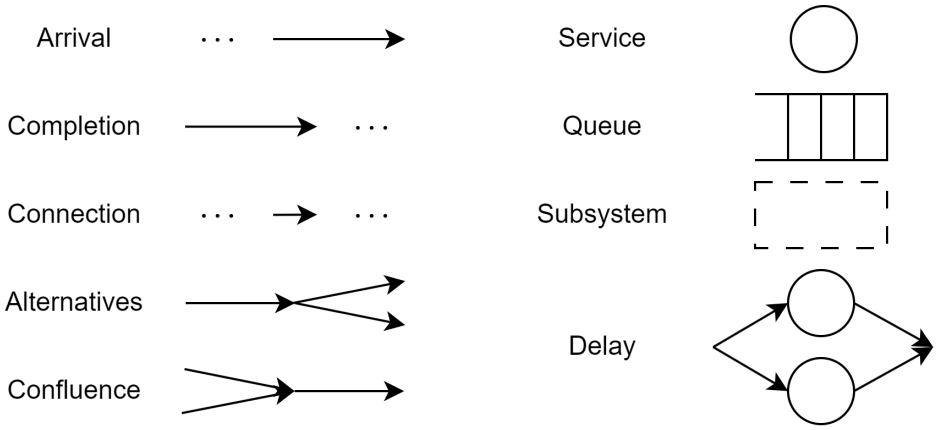


Figure 5.5: Graphical notation

5.3 Operational laws

Operational laws provide a model for understanding the average behavior of various systems. These laws are highly general and impose minimal assumptions about the behavior of the random variables that characterize the system. They are based on observable variables, which are values derived from observing a system over a finite period. The underlying assumptions include:

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

T	Length of time we observe the system
A	Number of request arrivals we observe
C	Number of request completions we observe
B	Total amount of time during which the system is busy
N	Average number of jobs in the system

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

$\lambda = \frac{A}{T}$	Arrival rate
$X = \frac{C}{T}$	Throughput
$U = \frac{B}{T}$	Utilization
$S = \frac{B}{C}$	Mean service time per completed job

We will assume that the number of arrivals equals the number of completions during an observation period:

$$A = C$$

If the system is job flow balanced, the arrival rate will be the same as the completion rate:

$$\lambda = X$$

A system can be viewed as comprising several devices or resources, each of which can be analyzed individually as a system under 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.

Resource-based variables We define the following variables for each resource k:

A_k	Number of request arrivals we observe for resource k
C_k	Number of request completions we observe at resource k
B_k	Total amount of time during which resource k is busy
N_k	Average number of jobs in resource k

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

$\lambda = \frac{A_k}{T}$	Arrival rate
$X = \frac{C_k}{T}$	Throughput
$U = \frac{B_k}{T}$	Utilization
$S = \frac{B_k}{C_k}$	Mean service time per completed job

5.3.1 Utilization law

Using the previously determined observed values, we find the utilization law:

$$U = XS$$

This can also be used in resource-based analysis:

$$U_k = X_k S_k$$

5.3.2 Little's law

We can derive Little's law as:

$$N = XR$$

Little's law can be applied to the entire system as well as to subsystems:

$$N_k = X_k R_k$$

Derivation Let's denote W as the accumulated time in the system (in jobs per second). Then, we can express:

$$N = \frac{W}{T} \qquad R = \frac{W}{C}$$

Thus, we can write:

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

5.3.3 Interactive response time law

In interactive systems, the think time Z is the time where the system is waiting for processing. 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:

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

5.3.4 Forced flow law

During an observation interval, we can compute the number of completions at each resource within the system. 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.
- 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:

$$X_k = V_k X$$

5.3.5 Demand law

If we know the processing required by each job at a resource, we can calculate the resource's utilization. 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:

$$D_k = S_k V_k$$

5.3.6 General response time law

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.
- *Residence time* R_k: this accounts for the average time spent by a job at station k during its stay in the system.

The relation between residence time and response time is similar to that between demand and service time:

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

For single queue open systems, $V_k = 1$, implying that the average service time and service demand are equal, and response time and residence time are identical.

If the mean number of jobs in the system or the system-level throughput are not known, an alternative method can be used. Applying Little's Law to the k-th resource, we find that:

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

From the forced flow law, 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. The general response time law is:

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

he 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.

5.3.7 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. Their simplicity allows for quick and easy application, making them valuable tools in system analysis and modeling.

5.4 Performance bound analysis

Performance bounds provide 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. Additionally, performance bounds allow for the simultaneous treatment of multiple alternatives, offering a comprehensive understanding of system performance.

Bounding analysis focuses on single-class systems and aims to establish asymptotic bounds, both upper and lower, on performance indices such as throughput and response time. By employing bounding analysis, critical influences of system bottlenecks can be highlighted and quantified, offering valuable insights into system performance.

Definition (*Bottleneck*)**.** A bottleneck refers to the resource within a system that experiences the highest service demand.

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 enables the evaluation of numerous candidate configurations, focusing on the dominant resource. By treating multiple configurations as a single alternative, bounding analysis streamlines decision-making based on preliminary estimates.

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 and the lower limit for system response time.
- *Pessimistic bounds*: represent the lower limit for system throughput and the upper limit for system response time.

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.

5.4.1 Open models asymptotic bounds

Throughput Throughput represents the maximum arrival rate that the system can effectively process. If the arrival rate exceeds this bound, the system becomes saturated, leading to indefinite wait times for new jobs. The throughput bound is calculated as the reciprocal of the maximum service demand $D_{\max}$:

$$X \leq \frac{1}{D_{\max}}$$

Response time Response time refers to the largest and smallest possible response times experienced at a given arrival rate. These bounds are explored only when the arrival rate is less than the saturation arrival rate, as the system becomes unstable otherwise. Two extreme situations are considered:

1. When no customers interfere with each other, resulting in response time equal to the sum of all service demands.

2. When there is no pessimistic bound on response times due to batch arrivals. As the batch size increases, more customers wait increasingly longer times, leading to no pessimistic bound on response times regardless of how small the arrival rate might be.

In the general case we have:

$$R \geq \sum_k D_k$$

5.4.2 Closed models asymptotic bounds

Throughput With multiple customers in the system, the system throughput can be expressed as:

$$X = \frac{N}{N \sum_k D_k + Z}$$

The largest throughput is achieved when jobs consistently find the queue empty, and service begins immediately:

$$X = \frac{N}{\sum_k D_k + Z}$$

The system's utilization at each resource is constrained to be less than or equal to one. Since the bottleneck resource is the first to saturate, the throughput has this constraint:

$$X \leq \frac{1}{D_{\max}}$$

Thus, the bounds for the throughput are:

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

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

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

Response time From the throughput bounds, we obtain the following:

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

5.4.3 What-if analysis

What-if analysis is a decision-making technique that explores potential outcomes by varying one or more input variables within a model or system. This method allows users to evaluate the impact of different decisions or changes by simulating their effects on the system's results. By adjusting parameters and observing the corresponding changes, decision-makers can gain valuable insights into potential risks, opportunities, and optimal strategies.