Cloud Computing

Hui Zeng *

Summer Semester 2021

*All notes are summarized from the lecture and tutorial materials provided by Prof. Michael Gerndt and his team. Images are retrieved from the lecture as well as tutorial slides.

Contents

1 Introduction

Different IT-trends boosts the need for cloud computing:

- Outsourcing, either infrastructure or management
- IT as a service: pay per use
- Re-centralization of data: similar to data centers, cloud be provided as a central place for data storage.
- Resource sharing instead of over-provisioning: same resource can be used for multiple purposes
- Server consolidation: instead of having multiple physical servers, with each dedicated to a certain service, servers are virtualized and put on one/reduced number of physical machines.
- Scalable computing
- Application dynamism: amount of request on web changes over time.
- Green computing, big data, stream processing, IoT, machine learning, etc.

Cloud Computing the definition is mainly divided by

- ubiquitous, convenient, on-demand network access to a **shared pool of configurable computing resources** (eg: networks, servers, storage, applications, services)
- resources can be **rapidly provisioned** and released with **minimal management effort** or service provider interaction
- cloud model is composed of
  - 3 service models
  - 4 deployment models
  - 5 essential characteristics

1.1 3 Service Models: IaaS, PaaS and SaaS

Three service models, ranking from outsourcing the least to the most: IaaS → PaaS → SaaS.

1.1.1 IaaS: Infrastructure as a Service

- Offering: provision processing, storage, networks, other fundamental computing resources
- Rights as consumer:
  - deploy and run **arbitrary** software, including **operating systems and applications**

- control over OS, storage, deployed applications
- limited control of select networking components

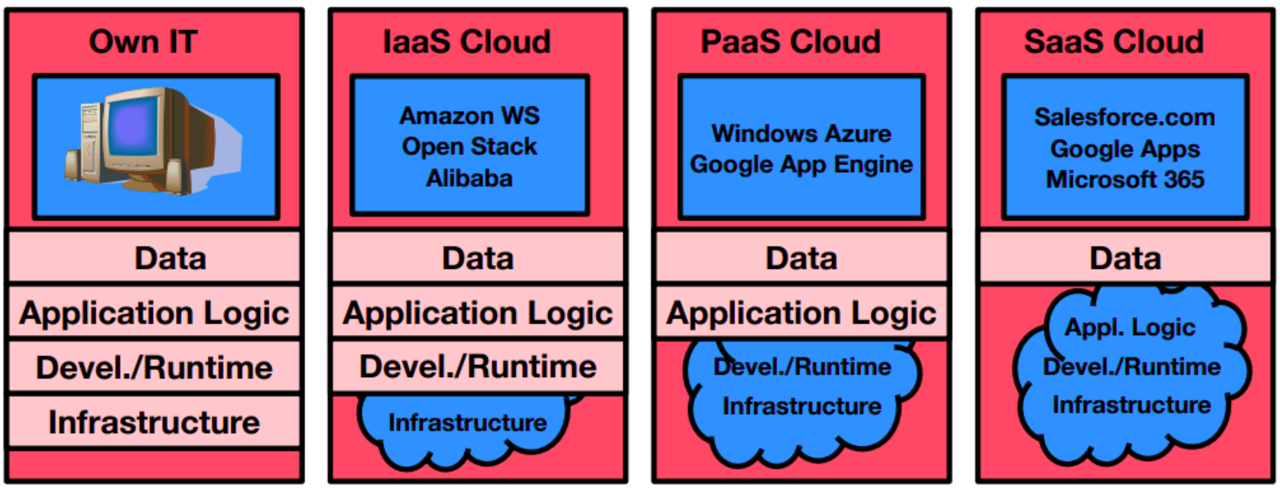
- No control as consumer:
  - underlying cloud infrastructure

1.1.2 PaaS: Platform as a Service

- Offering: application infrastructure services(eg: development platforms, libraries, tools, databases) through client interface
- Rights as consumer:
  - limited user-specific application configuration settings
- No control as consumer:
  - underlying cloud infrastructure
  - network, servers, storage, OS
  - individual application capabilities
- Example: MS Azure, Amazon FaaS, Google application engine

1.1.3 SaaS: Software as a Service

- Offering: provider's applications on cloud through client interface
- Rights as consumer:
  - limited user-specific application configuration settings
- No control as consumer:
  - underlying cloud infrastructure
  - network, servers, OS, storage
  - individual application capabilities



Service Model	IaaS	PaaS	SaaS
Service category	VM rental, online storage	online operating environment, online database, online message queues	application and software rental
Service customization	server template	Logic resource template	application template
Service provisioning	automation	automation	automation
Service accessing and using	remote console, web services	online development and debugging, integration of offline development tools and the cloud	Browser, web service interfaces, SDKs, apps

Service Model	IaaS	PaaS	SaaS
Service monitoring	physical resource monitoring	logic resource monitoring	application monitoring
Service level management	dynamic orchestration of physical resources	dynamic orchestration of logic resources	dynamic orchestration of applications
Service accounting	physical resource metering	logic resource usage metering	application usage metering
security	storage encryption and isolation, VM isolation, VLAN, SSL/SSH	data isolation, operating environment isolation, SSL	data isolation, application isolation, SSL, Web authentication and authorization

1.2 4 Deployment Models: Private, Community, Public, Hybrid

- Private Cloud:
  - service offered **via private network** for **single client**.
- Community Cloud:
  - service offered to **a specific group of clients**.
- Public Cloud:
  - service offered **over Internet via Web-application** or third-party provider for **everyone**.
- Hybrid Cloud: combination of public and private cloud.

1.3 5 Essential Characteristics

- **on-demand self-service**:
  - able to **provision computing capabilities** unilaterally(no interaction required with provider).
- **broad network access**:
  - capabilities can be available and accessed through by **diversely thin or thick client platforms** (mobile, tablets, cable, etc.)
- **resource pooling**:
  - **multi-tenant model** is used, multiple customers shares the computing capabilities at the same time, according to their self-customized demand. Specification of resource location can be possible at higher abstraction level.
- **rapid elasticity**:
  - computing capabilities can be **elastically provisioned and released** in any quantity at any time. The process can be automated or scaled according to dynamic demand.
- **measured service**:
  - automatically control and optimize resource use by **leveraging a metering capability**. Resource usage can be monitored, controlled and reported.

1.4 Pros & Cons of Clouds

- Advantages:
  - scalability, elasticity
  - rapid deployment
  - no capital investment for physical resources
  - outsourcing of infrastructure management
  - limited access to on-premise servers
  - fault tolerance: multiple servers have data replicas, if one node fails, other nodes will replace.
  - collaboration
- Disadvantages:
  - no control over security, based on ”trust”.
  - no control over hardware/infrastructure
  - vendor lockin: service is not standardized, not compatible to other vendors.
  - cost on monthly fees: if demand for same computational power is constant, fee may be higher than building own hardware. Only recommendable for dynamic demand.
  - breaking SLAs: your performance may be influenced by other tenants(multi-tenant model).

2 Base Technologies of Clouds

2.1 Process Technology

Production the processors are produced from semi-conductor materials. It's primarily produced on a **waver** consisting of a lot of chips. Later the waver is cut after the production process. Individual chips will be packaged into the system.

Transistor

- traditional 2D planar transistor: a 2D-planar structure, the gate controls how much current flows from source through the drain.
- 3D tri-gate transistor **FinFET**: conducting channels on 3 sides with a **vertical fin structure**. The width of a Fin is **10nm**, and it keeps shrinking.

The smaller the structure gets, the more transistors fit on the same space, the faster the transistor gets.

2.2 Processor Architecture

2.2.1 CPU

- current state of CPU development:
  - increase in transistors: up to 10nm or even 5nm.
  - stop in increase of clock speed: up to 4GHz. Limitation: cooling. (the faster the clock speed, the more energy consumed, the hotter)
  - continuous need of performance improvement: parallelism on the chip, since halt in clock speed.
- trends in CPU development:
  - multi-core processors: parallelism
  - SIMD support (Single Instruction, Multiple Data): parallelism inside of a single instruction, computation of vectors of values in parallel.
  - combination of core private and shared caches:
    - data saved in cache for repeated operations
    - with multiple caches, cores can communicate. However, this may disturb the usage of cache.
  - hardware support for energy control: **dynamic voltage and frequency scaling**
    - chips work in a dynamic frequency controlled by hardware according to the need of the running software.
    - It checks whether operations are memory-bound or compute-bound.
  - 64-bit architectures
- challenge: the **memory hierarchy**

- access to the main memory is slow, involving several hundreds of cycles for CPU, while each of these cycles can be responsible for multiple operations.

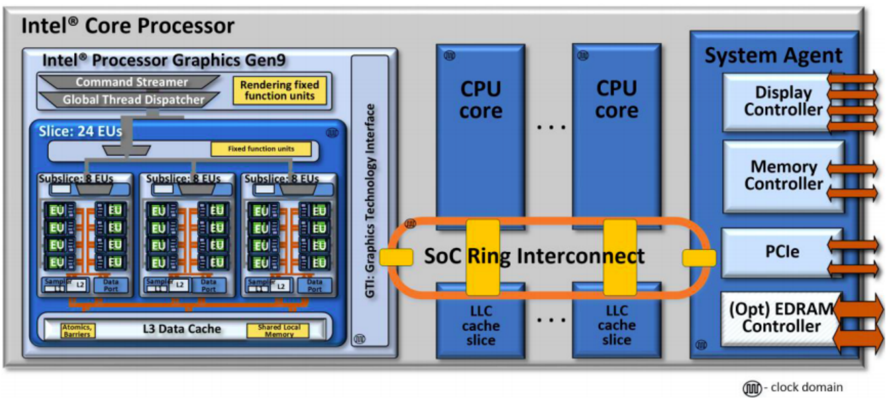
  → save such cycles for accessing data but keep the **data near to execution**.
- **Level-1 Instruction & Data Cache**: fastest, but too slow to feed all data in large size. (64 Bytes: 8 double precision values, 64 bit long values)
- **Level-2 Unified Cache**: not separated, access better.
- **Translation Lookaside Buffer (TLB)**: translates virtual addresses into physical addresses and loads into main memory, it only stores the **most recent translation**, no need to lookup constantly.

Core Cache Size/Latency/Bandwidth

Metric	Nehalem	Sandy Bridge	Haswell
L1 Instruction Cache	32K, 4-way	32K, 8-way	32K, 8-way
L1 Data Cache	32K, 8-way	32K, 8-way	32K, 8-way
Fastest Load-to-use	4 cycles	4 cycles	4 cycles
Load bandwidth	16 Bytes/cycle	32 Bytes/cycle (banked)	64 Bytes/cycle
Store bandwidth	16 Bytes/cycle	16 Bytes/cycle	32 Bytes/cycle
L2 Unified Cache	256K, 8-way	256K, 8-way	256K, 8-way
Fastest load-to-use	10 cycles	11 cycles	11 cycles
Bandwidth to L1	32 Bytes/cycle	32 Bytes/cycle	64 Bytes/cycle
L1 Instruction TLB	4K: 128, 4-way 2M/4M: 7/thread	4K: 128, 4-way 2M/4M: 8/thread	4K: 128, 4-way 2M/4M: 8/thread
L1 Data TLB	4K: 64, 4-way 2M/4M: 32, 4-way 1G: fractured	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way
L2 Unified TLB	4K: 512, 4-way	4K: 512, 4-way	4K+2M shared: 1024, 8-way

All caches use 64-byte lines

2.2.2 Skylake Architecture



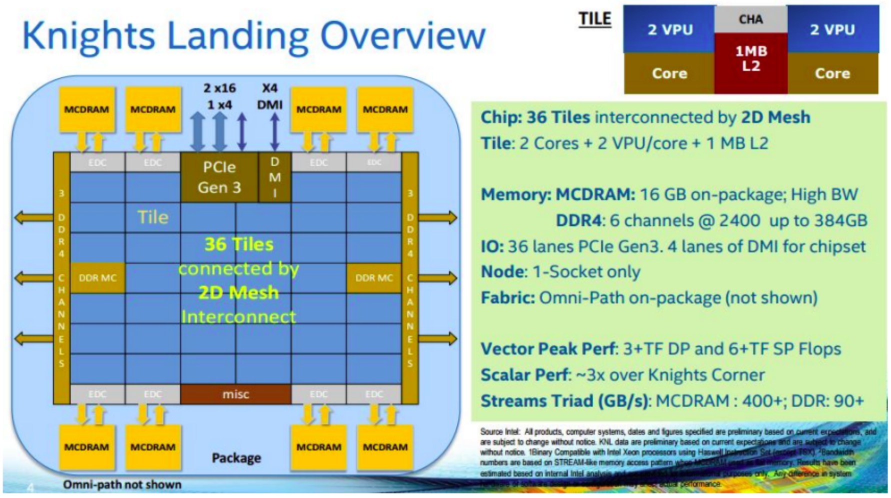
The archiecture of a processor (one chip)

- graphic processor: accelerator for specified computation

- system agent: support structures (eg: display controller, memory controller, PCIe for I/O, EDRAM controller)
- CPU cores: homogeneous cores with a **private cache each**.
- LLC cache slice: each slice cache is associated to each CPU core.
  - If core misses information in its private cache: through interconnect, it checks which slice cache contains the info, then it propagates to the cache associated the CPU core and returns the info back to the CPU.
- SoC Ring interconnect: all parts are connected by a ring bust.
  - if CPU writes to I/O device – PCIe, it first puts information onto the bust, then it propagates into the PCIe and is written to the disk.

→ the **interconnect ring** is the **bottleneck** for increasing the cores.
→ alternatives: Xeon Phi

2.2.3 Xeon Phi Architecture



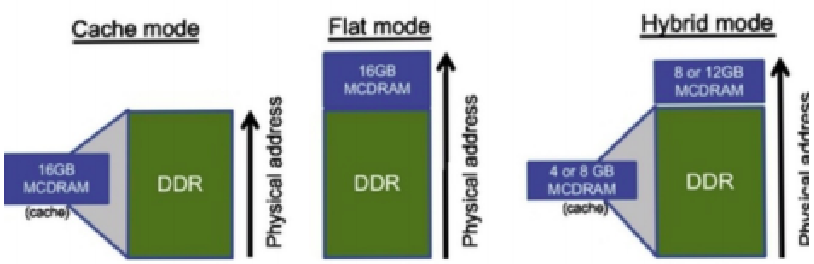
- Goal: allows significantly **more cores** in a single processor, in a single CPU die.
- Idea:
  - the die is organized into a **tile-architecture**.
  - 36 compute tiles are connected through a **2D mesh network** → connection between tiles in both x- and y-direction.
  - each tile has 2 cores → **72 cores** in total
- Tile structure:
  - 2 cores
  - 2 VPUs (Vector Processing Unit) for each core → 4 in total per tile.

- L2 cache: shared between the cores, but as a private cache for each tile.

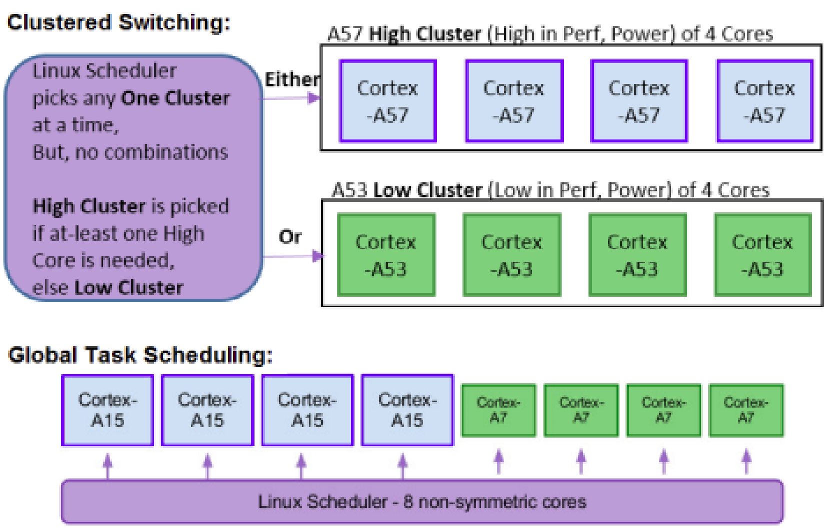
  → multiple copies of an address can be in different private L2 caches of different tiles, which **must be coherent**.
- CHA (Caching Hold Agent): responsible for the **coherence**. It's connected to each tile, keeping track of the status of the copies by implementing a **coherence protocol**.

• Memory:
- DDR4 memory
- **MCDRAM**– Multi-channel DRAM, **high bandwidth**(450 GB/s)

  Memory modes:
  * flat mode: all MCDRAM is used as **physical address**. Data structure can explicitly choose between MCDRAM or DDR.
  * cache mode: all MCDARM is used as **L3 cache**. physical addresses only on DDR. If data is being processed from DDR, it's mapped to MCDRAM.
  * hybrid mode: combination of flat and cache mode. Part of MCDRAM is used as **L3 cache**, the other as **physical addresses**.



2.2.4 Processors for mobile devices: ARM



- **Big Little Principle**
  - Combination of **high clusters** and **low clusters**, controlled by clustered switching.
    - high cluster: high in performance
    - low cluster: low in performance, but energy efficient
    - only **one cluster** at a time, **no combinations**.
  - Global task scheduling: tasks are scheduled according to the requirement between 2 clusters.
- Use-cases: Apple Processor A14 (2 high performance Firestorm, 2 energy-efficient Icestorm)

2.3 Accelerator Programming

- Motivation:
  - increase computational speed and reduce energy consumption

    → achieved by **specialization** in operations/on-chip communication/memory accesses

    → **accelerator**
- Types:
  - GPGPU (General Purpose Graphic Processors)
  - FPGA
  - standard cores
- Designs:
  - CPU with accelerators attached: computation can be offloaded onto the accelerator.
  - accelerators-only design
  - accelerator booster: a collection of accelerators as a separate part from the whole system. Jobs can be computed by these accelerators when necessary. Accelerator booster can be shared among parallel jobs.

2.3.1 Graphic Processing Units (GPU)

- Usage:
  - visualization
  - **general processing** (NVIDIA)
- Parallelism: multi-threading, MIMD, SIMD
- Challenges:

  - a **specialized programming interface** for the GPGPU needed (eg: CUDA from NVIDIA)
  - **scheduling coordination** on system processor and GPU
  - **transfer of data** between system memory and GPU memory
- example: NVIDIA Tesla P100

NVIDIA Tesla P100

- GP100 (GPU):
  - L2 cache: shared among all compute units – streaming multi-processor
  - NVLink: able to connect multiple GPGPUs together
  - memory controller: access to high bandwidth memory
  - 6 Graphic Processing Clusters(GPC)
    * 10 Streaming Multi-Processor each GPC, 60 in total
    * 5 Textural Processing Clusters (TPC), 1 for 2 SM, 30 in total
- High Bandwidth Memory (HBM)
  - **vertical stacks** of memory dies connected by microscopic wires
    → near and tight connection between memories
  - 180 GB/s per stack bandwidth

→ good for data parallel processing like vector processing.

2.3.2 Field Programmable Gate Arrays (FPGA)

- hardware which is programmable
- Consist of:
  - **array of logic gates** to implement **hardware-programmed special functions**
  - **specialized functional units** (eg: signal processors, multipliers)
  - static **memory**
- programmed in VHDL: the program describe functions to be executed, it will then be translated into look-up tables, which are put into the logic gates
- Use-case:
  - as **accelerator** for specialized computations
  - **filtering** for databases
  - as **switches, routers** for communication
  - **preprocessing**: I/O hardware of FPGA accesses the DDR, local computation can be organized in the pipeline of FPGA. Local preprocessing will be done on FPGA and the output will be sent ot external processor via PCIe.
- examples: Altera, Xilinx

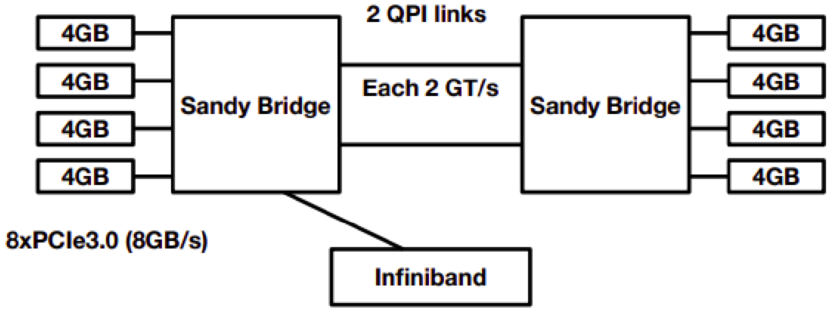
2.4 Architecture for Parallelism: Shared Memory Systems

- Idea: 2 architectures to **achieve parallelism**, which is combining multiple processors together for computation.
  - shared memory system
  - distributed memory system
- **Non-Uniform Memory Access (NUMA)**:
  - **multiple CPU** with multiple cores are connected through **single physical address** space.

    → access time depends on the **distance to the physical address** (memory)

    → CPU accesses either local(near) or remote memory → locate the memory near for fast access.

  Example: SuperMUC
  - 2 multi-core processors(Sandy Bridge) with 32GB memory in total
  - each processor can access to all 32GB memory, **however access time differs**
  - **Latency**
    * local: ∼50ns, ∼135 cycles
    * remote: ∼90ns, ∼240 cycles



- Programming interfaces for Shared Memory Systems
  - explicit threading
  - automatic parallelization: sequential code is given to compiler, which **automatically** parallelize the work among available CPUs.
  - OpenMP: directive-based parallel programming, parallel computations are **explicitly expressed**.
- Challenges in parallel computing:
  - explicit synchronization needed.
  - **cocurrency bugs**: the outcome of the computation depends on the speed of access to the memory → non-deterministic results possible.
  - control of **data locality**

2.5 Architecture for Parallelism: Distributed Memory Systems

- Characteristics
  - **Coupling** of individual nodes via network: processor only have access to the memory in node.
  - **no shared** physical address space
  - communication between nodes: transfer of **messages**
- Programming in Distributed Memory Systems: **more difficult** than shared memory systems.
  - + : **rare race conditions** → cocurrency bugs low.
  - – : Message Passing Interface(MPI), have to explicitly decompose or insert message passing → more difficult to program.
  - Process to Process communication and collective operations
- Challenges:
  - **more difficult** to programm than shared memory systems
  - **expensive communication**, much slower than access to memory.

$$t(message) = \text{startup time} + \frac{\text{message size}}{\text{bandwidth}}$$

    * communication with one large message is more efficient than multiple small messages (startup time)
    * mapping onto processors has performance impact. Communication may not need to go through the entire but locally. (eg: 2D mesh network)

2.6 Data Center Networks

Inside the data center, the **servers** are connected by **LAN**– Local Area Network. Each **server** is connected to a **switch**. The **switches** are connected, which allows **communication between the servers**.

Multiple **LANs** are possible in a data center. The **LANs** can be connected through a **router**. A router is based on a IP-address, it can make decisions depending on the IP-address (eg: receiver of message).

The **routers** in the data center can be connected to a **WAN**– Wide Area Network, the internet. The **WAN** can be connected to a **local router**, which a client has access to . This is the **last mile** to reach the client, frequently **radio network** is used – WLAN or GSM.

2.6.1 Different Networks

- Types:
  - WAN – Wide Area Networks

    * homogeneous base technology (opto-electronic)
  - LAN – Local Area Networks and Cluster Networks
    * non-shared Ethernet
    * Infiniband
  - Last Mile
    * heterogeneous base technology (Radio, TV cables, etc.)

- Performance Metrics:
  - **Latency**: transport time of a message
    * physical delay: time needed to go through the links, limited by speed of light, **not optimizable**.
    * protocol delay: time needed to execute protocol operations. **compensated by increase of CPU performance.**
    * line waiting time: time to wait until the link is available. negligible up to 10% utilization. **reduced by increasing bandwidth**.
    * transmission time: time needed to send certain amount of data over the link. **reduced by increasing bandwidth**.

$$\text{transmission time} = \frac{\text{message size}}{bandwidth}$$

  - **Bandwidth** (byte/sec): the speed transporting a message

2.6.2 Local Area Network

Ethernet

- first implementation based on a **shared cable**: all computer are connected through one cable. Only one computer can transmit message at a time.
- now **switched Ethernet**: each computer is connected to a switch. Switches are connected together to enable communication. A switch replicates all packets to all ports.
- Speed: 10 Mbit/s, 100 Mbit/s or 1000 Mbit/s

VLAN – Virtual Local Area Network

- Characteristics:
  - a single LAN is **partitioned** into **multiple virtual LANs**.
    → direct traffic or for security reasons.
  - each virtual LAN is a **single broadcast domain**
  - **communication** between VLANs only through **router**
- **Port-based VLAN**

  - The **ports** of a switch are **specifically assigned** to a **VLAN**.
  - servers of a VLAN from two switches are communicated through a **link**.
  - # links = # VLANs

- **Tagged VLAN**
  - one port of a switch is not connected to server. This port is connected to other switches through a **link**. This link manages all packets.
  - **Tag**: packets are **only forwarded to ports with the same tag**.
  - # links = 1

Infiniband

- Characteristics:
  - low latency, high bandwidth
  - speed: 25 Gbit/s
  - RDMA access: **direct access of memory of other computer**. Instead of packing data into a message an sending to the operation system and then taking it out, RDMA can **directly fetch** the data from memory and forward it to the requester.

    No protocol overhead or handling of message → **faster**.
  - based on **Virtual Interface Adapter**: data transfer don't require operation system support.
- Use-case: clusters and servers

2.6.3 Software-defined Networks

- Motivation:
  - higher speed
  - automated network configurations
  - security
  - adaptation to performance variations
- Current network management:
  - Management plane –> Control plane –> Data/Forwarding plane
- Software Defined Networking:
  - allows network administrators to **programmatically initialize, control, change and manage the network behavior dynamically** via open interfaces
  - Protocol: OpenFlow
    * enables an open interface to **interact** with networking devices (machines, switches from different providers)

    * network layers on top of L3
    * SDN controllers communicate to L3 switches using openFlow protocols.

2.6.4 Last Mile Networks

Wireless Local Area Network – WLAN

- consists of **clients** and **access points** as routers.
- modes:
  - **infrastructure**: clients connect to the access point
  - **ad hoc**: clients communicate with each other
- Security:
  - Wireless Equivalent Privacy (WEP)
  - Wi-Fi Protected Access (WPA1, WPA2)
- Speed:
  - 802.11 n: 800 Mbit/s, 70m
  - 802.11 ac: 1733 Mbit/s, 35m

Digital Subscriber Line – DSL

- transmission of data **over telephone lines**, share with telephone service (different frequency)
- Speed:
  - asymmetric DSL: upstream bandwidth **much lower** than downstream
  - downstream: 256 Kbit/s to 100 Mbit/s

Very-high-bit-rate Digital Subscriber Line – VDSL

- higher speed than DSL:
  - different versions: VDSL, VDSL2, VDSL2-Vplus
  - VDSL: up to 55Mbit/s downstream, 16 Mbit/s upstream
- VDSL with **vectoring**: reduce crosstalk between different lines. (Crosstalk: communication on one line influences the communciation on other lines)

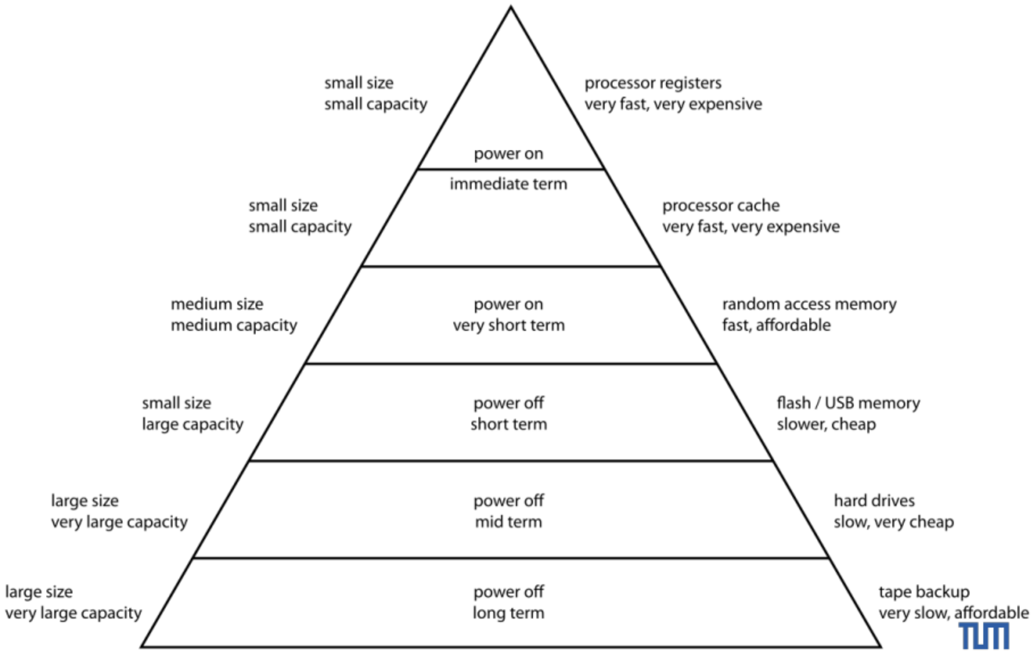
  → special encoding of neighbouring lines: a provider needs to have **access to all lines in a bundle**

  → current implementation difficult

Global System for Mobile Communication –GSM

- network for **mobile phones**, 3G, 4G, 5G
- access to the network using SIM card

2.7 Storage Technologies



2.7.1 Local Storage

Redundant Array of Independent Disks – RAID

- Goal: increase reliability or bandwidth
- RAID 0: **distribute** blocks over **2 disks**, if **write both blocks on two disks** at the same time, it gets **double bandwidth** to the disk.

  → **higher bandwidth**
- RAID 1: **replicates** blocks on **2 disks**. If one fails, we still have the information on the other disk.

  → **higher reliability**
- RAID 5: **distribute** blocks over **4 disks**, while one disk saves the **parity information**. If one block fails, the parity information enables **reconstruction**. Parity information of blocks is not written on the same disk, but **distributed** over all disks.

  → **higher bandwidth and reliability**

Flash

- non-volatile memory, retains stored information even after power is removed.
- Write operation: tunnel injection, a high positive voltage between control gate and source **pushes electrons into the floating gate**. It stays/saves in the floating gate as stored information.

- Read operation: a higher voltage is required at the drain to make the channel conduct, the electrons move from source to drain.
- Increasing storage density: **increase floating gates** in a flash cell
  - Single Level Cells (SLC): stores **1 bit** of information
  - Multiple Level Cells (MLC): stores **2 bits** of information
  - # floating gates ↑, cost per bit ↓, storage density ↑, program-erase cycles ↓, write/reading speed ↓
- Use-case: USB-disks, SD-cards, mobile phone storage, built in SSD

SSD

- Comparison with hard disks:
  - lower latency, random access
  - smaller storage capacity
  - less power hungry
  - faster read/write speed

2.7.2 Data Center Storage

- storage comparison: €/IOPS
- only based on flash storage or SSDs, combined with RAID, with special controllers optimized for SSDs.

  → IOPS ↑, latency ↓, bandwidth ↑, cost ↑

2.7.3 Provisioning of Storage

- 3 ways to provide storage:
  - Direct Attached Storage: storage devices are attached to the individual computer
  - Storage Area Network
  - Network Attached Storage

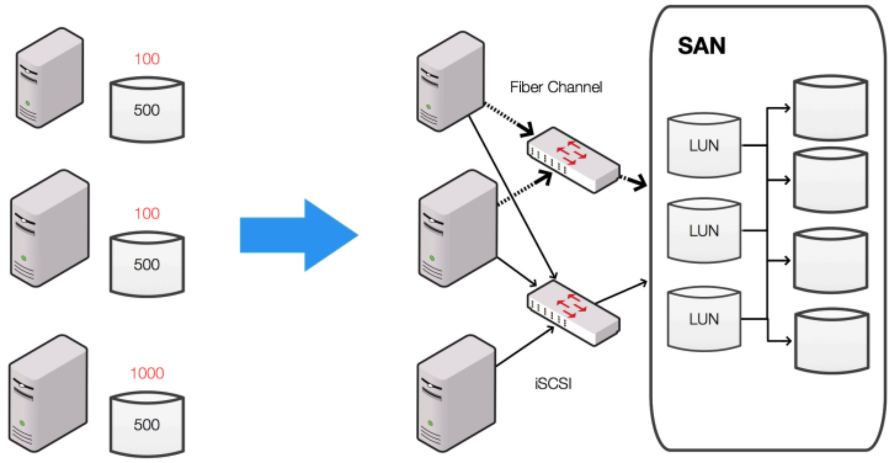
Storage Area Network – SAN

- access to **block level** data storage
- a **specialized network** connecting the servers which separates from LAN
- **no pre-existing file system**, the server can define its own file system according to needs
- a shared pool of spare resources, which allows **flexible allocation of spare storage**
- Advantages:

- **flexible distribution** of devices between clients, reconfiguration of distribution in software instead of adaptation of cabling.
- easy replacement of faulty servers
- back-up can be done centrally, easier disaster protection
- no pre-existing file system, allows **customization** according to needs.

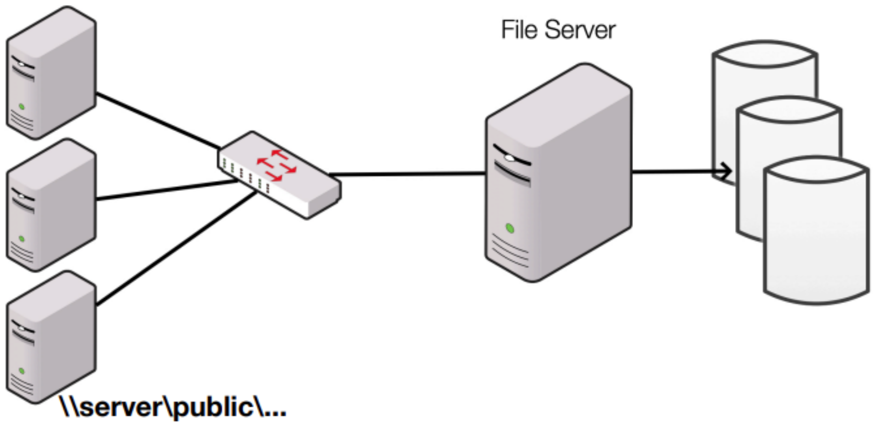
Disadvantages:

- shared network bandwidth
- shared performance of storage devices



Network Attached Storage – NAS

- storage devices(disks) are connected to a **file server**
- computers **go over the network** and **access the file system** on file server
- an **existing file system**.
- network file sharing protocols: NFS
- storage devices: RAID to increase bandwidth and reliability
- Use-case: streaming contents(movies, images) to home network. If connected to home WiFi, then access to local storage. Access out of home using VPN and public IP



2.7.4 Storage Virtualization

- Goal: location transparency
- Process: allocate a **virtual disk**, which will be **mapped to a real physical hard-disk** in the Storage Area Network. The **client won't know about which hard-disk is allocated** for the virtual disk.

Block Virtualization

- the **mapping** of a **virtual disk and block number** to a **physical disk and block number**.
- Use-case: SAN, flexible mapping, disk expansion and shrinking
- implementation:
  - host based: host runs virtualization software
  - storage device based: disk array provides a virtualization level
  - **network based**: virtualization device is in LAN and connected to a SAN, **most frequent implementation**
    * **in-band**: client sends request for certain blocks to the controller in the network, the controller fetches the data and returns the data to the client.
    * **out-of-band**: host contacts the controller, gets the mapping information and accesses the data in SAN directly without the help of controller.

File Virtualization

- Virtualization on **file level**
- Use-case: **Distributed File System**
  - allows transparent access to multiple NAS server. Files located on multiple NAS servers **appears as if on a single NAS**, the client doesn't know on which server the file exists.

3 Virtualization

- Idea: resource usage by a single user is **under-utilized**, the **efficiency** of usage of resource is **low**.

  → own a machine in a shared manner → **virtualization**

- Definition: Virtualization is a computer architecture technology where **multiple virtual machines** are **multiplexed** in the **same hardware**. (all VMs are connected and are owners of the hardware)

- Goal:
  - enhance resource sharing, **improve** machine **efficiency**
  - able to replace and upgrade hardware **on the fly**, **without interrupting the running program or rebooting**.
  - reduce down time
  - **faster provisioning** of multiple machines

- Modes of operation:
  - **kernel mode**: higher privilege
    * OS allows execution of **all CPU instructions**
    * kernel codes **don't execute in user mode**.
    * execute in **superuser/supervisor** privilege.
  - **user mode**:
    * OS allows execution of **few CPU instructions**
    * if user applications have to execute **privileged instructions**, they **ask kernel** to execute.
    * execute in user privilege.

3.1 Technology for Virtualization: Hypervisors

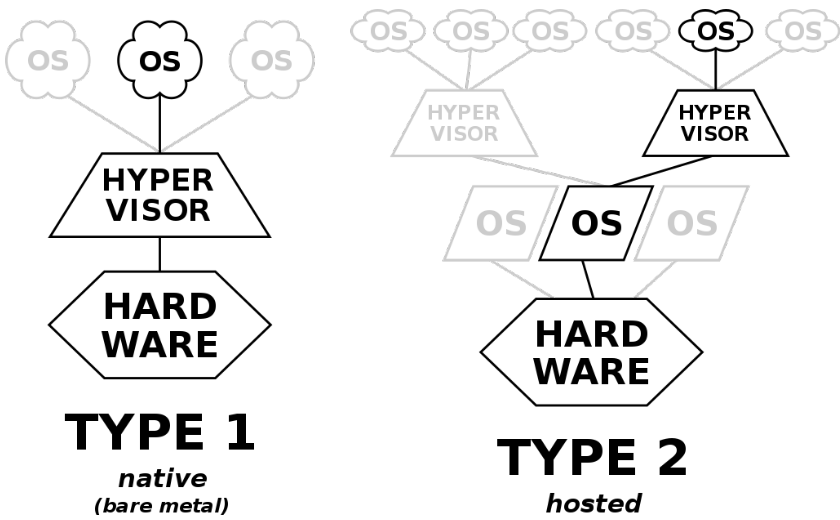
A **hypervisor** (or virtual machine monitor, VMM) is a kind of emulator; it is computer software, firmware or hardware that **creates and runs virtual machines**. A **computer on which a hypervisor runs** one or more virtual machines is called a **host machine**, and **each virtual machine** is called a **guest machine**.
The hypervisor **presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems**. Multiple instances of a variety of operating systems may **share the virtualized hardware resources**: for example, Linux, Windows, and macOS instances can all run on a single physical x86 machine.
This contrasts with operating-system-level virtualization, where all instances (usually called containers) must share a single kernel, though the guest operating systems can differ in user space, such as different Linux distributions with the same kernel.

3.1.1 Bare-Metal Hypervisor

- runs **directly** on **host's hardware** – Ring 0
- Guest OS operates at Ring 1
- provides **almost full isolation** to the users. All Guest OSs are **independent** and are owner of the hardware. VMM **controls the hardware and to manages resource allocation to guest OSs**.
- Use-case: Hyper-V, ESX, Xen

3.1.2 Hosted Hypervisor

- **runs on a conventional OS** just as other computer programs
- abstracts guest OSs from the host OS.
- Use-case: VMware player, VMware workstation, VirtualBox



3.2 Full Virtualization

- for **any guest OS without any modifications**
- VMM works in Ring 0, guest OSs work in Ring 1 – Bare-metal hypervisor
- Problem: **execution error** of privilege instructions from guest OS being in **lower ring** (Ring 1)
  - Solution: VMM intercepts such error and **emulates** the instructions **on the fly**

    → not all instructions are trapped.
  - Solution: a **binary translator**, which overrides these privileged instructions and places them in translation cache.
- Consequences:

- system calls: takes **10 times more cycles** compared to no hypervisor, because **fault message** are issued, translated and executed **for every system call**.
- I/O virtualization: major issue. **more I/Os** due to more CPUs (guest computers) but I/O chipset **can't be easily extended**
- memory virtualization: 2-stage mapping process with VMM.
  * program's memory addresses → virtual physical memory (on VMM) → real physical memory
  * VMM maintains a **shadow page table**, this process takes **3 to 400 more cycles** than without VMM.

3.3 Paravirtualization

- guest OS needs to be **modified at source code level**. Information is given in prior in code, that if such instruction is being executed, it needs to go directly to the VMM instead of the hardware.

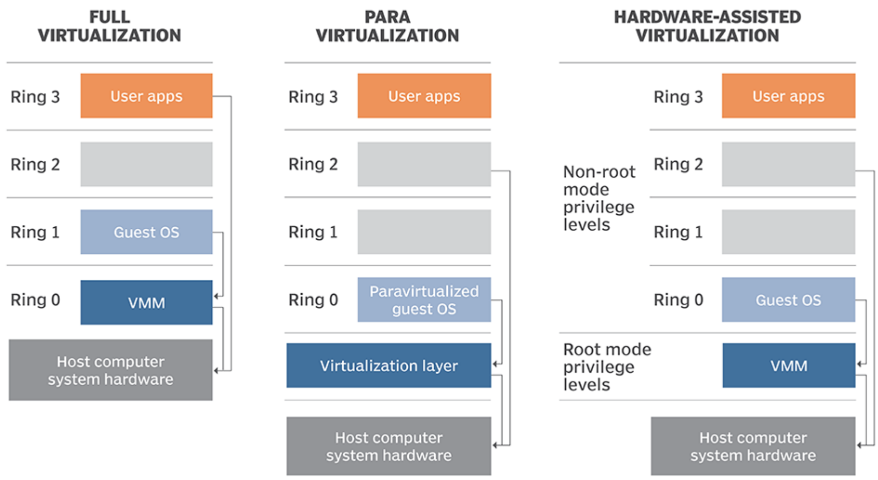
  → no execution error because of ring privilege. → no trapping or binary translation needed
- Hypervisor provides **interface** to accommodate critical kernel operations (memory management, interrupt handling)
- Advantages:
  - runtime changes are avoided. (no on-the-fly modification)
  - unnecessary trapping of critical instruction avoided (modified in code)
  - lower virtualization overhead

  Disadvantages:
  - a modified guest OS needed when changing machines.

3.4 Hardware-assisted Virtualization

- Idea: enables efficient full virtualization using **help from hardware capabilities**, primarily from the host processors.

  → a processor extension is introduced in a **higher priority layer** – Root mode privilege level with VMM.
- Guest OS now operates at **Ring 0** and can execute all critical instructions



3.5 OS-Level Virtualization

- Idea:
  - disadvantages of hypervisor-based virtualization: OS dependent, not scalable, not portable, slow deployment.
  - higher scalability required in application.

  → **OS-Level virtualization**

- VMM is **on top of host OS**

- building blocks: linux kernel features
  - **namespaces**: **limit the views**, wrap a group of resources for limited access, managed using APIs
    * PID: create another set of PID
    * cgroup: new views to root directories
    * network: new views to network resources
  - **cgroups**: control groups, **limits the applications to a specific set of resources**
    * memory cgroup: memory resource controller, isolates the memory behavior of a group of task from others. when memory exceeds, running processes will be killed.
    * others: cpu, blkio, cpuset, devices, freezer cgroup

- **Container**:
  - allows multiple **isolated** linux systems of same kind on a single host OS.
  - resources are **isolated in a container** for each user.
  - it utilizes **cgroups and namespaces** to limit the views and resources of each container.
  - Use-case: **Docker**

- Advantages:
  * runtime isolation: isolates different runtime environments based on application requirement
  * cost-efficiency: no creation of entire virtual OS for each user.
  * easy portability of containers
  * high scalability, easy replication of containers across environments
  * faster deployment: layer-concept, only new layers/updated layers are built.