Contents

Chapter 1

Cloud Computing Basics

Cloud computing is the practice of using remote servers to process and store data, instead of a local server or personal machine. With that being said, enjoy!

1.1 Servers and Clients

A **server** is a computer/software that processes requests, that is, receives data from a client or another server and delivers data based on those requests. A **client** is a computer/software that accesses a service made available by a server.

1.2 Server Hardware and Cloud Servers

Server hardware runs the server (duh), which is expected to have no downtime. The standard practice for enterprise companies is to have data centers with air-conditioned server racks full of computers, which also provide as backups for all possible contingencies. For a poor individual however, the best solution is to throw a dedicated computer with a network cable in a closet.

§ **Note:** The unit of measure for the height of one rack frame is a rack unit (U), and a typical server rack is 42U tall.

The alternative is the **cloud computing model**, where server hardware is rented out from a large service provider. The provider purchases their own hardware, builds/maintains the data center, and handles the networking. There are various financial models for users purchasing server resources: pay-as-you-go (charging based on actual consumption), pure subscription to a set amount of resources, etc.

The main advantage of the cloud computing model is that it allows companies to save on operating expenses (opex), and not focus on hiring IT professionals or managing hardware.

1.3 Public vs. Private Cloud

The **public cloud** is the cloud computing model where the service provider handles all of the maintenance of the server hardware – none of it is owned by the user purchasing server space. The resources are usually given on shared hardware, but the good news is that the buyer doesn't have to worry about any of the finer details. This model is the one most people understand about cloud computing.

The **private cloud** is where services are provided and deployed on private infrastructure, that is, the company manages the hardware themselves. The advantage of this model is that the company gets control of the environment and management of cloud resources. This model is less the cloud computing model described above and more just the company owning their own server hardware (literally).

1.4 Infrastructure as a Service (IaaS)

1.4.1 About

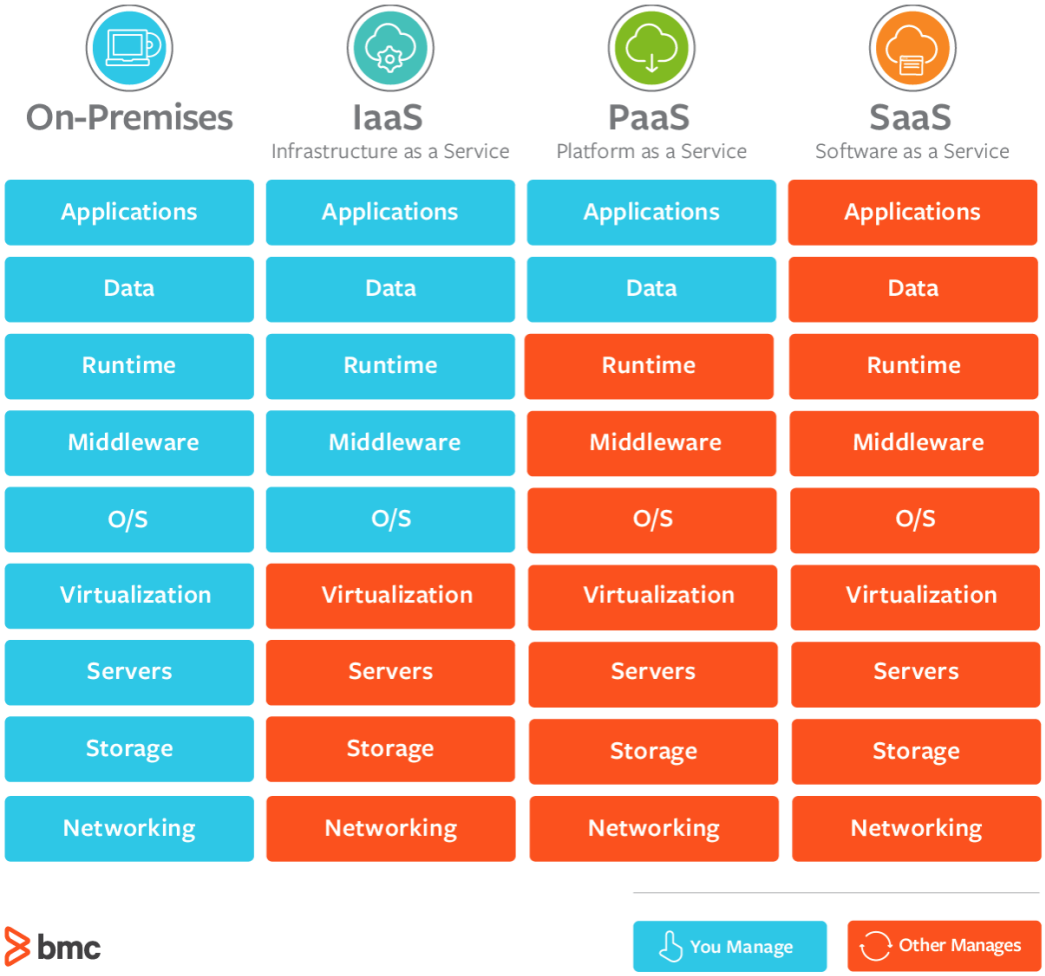
Infrastructure as a Service is the cloud computing model as we know it – the computing resources are given by the provider, and the buyer has self-service to the given resources (monitoring computers, changing networking, storage). Some popular examples are Google Cloud, Amazon AWS, and Microsoft Azure.

1.4.2 Difference between On-Premises, Iaas, PaaS, SaaS

A quick overview of the other different service offerings:

- On-Premises sounds exactly like what it's supposed to be: the computers and software are installed on the premise of the organization using the software.
- With Platform as a Service, users are mainly concerned with development and deployment of their application, as the server, storage, and networking is covered by the provider. Two common examples are Heroku and Amazon Elastic Beanstalk.
- With Software as a Service, users don't have to do anything but use the software provided. The creator of the software (the seller) has to maintain the application and the server running it. The most well-known examples are Google Apps, Dropbox, and Salesforce.

Below is a diagram that illustrates the various control levels of the different types of offerings (provided by `bmc.com`):



1.5 Cloud Resource Management

1.5.1 Security Groups

Security groups are a way of filtering the traffic that comes in and out of a cloud instance. Typically, a security group will allow you to specify what protocol, port range, source, and direction (inbound/outbound) packets can pass through. Packets which don't match a rule will be dropped.

Chapter 2

Networking

2.1 IP Addresses

Internet Protocol (IP) addresses are used to identify devices in a network (both local and wide).

2.1.1 IPv4

An IPv4 (the 4th version of IP) address consists of 4 bytes (32 bits), delimited by a period/dot. Specifically, an IPv4 address is of the form `xxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx` or is translated into some decimal form like `192.168.0.1`.

§ **Note:** Subnetting, or netmasking, is a technique to divide a network for performance/management. A **subnet mask** is a bitmask which will provide the network address when a logical AND is applied to the IP address. A subnet mask is written the same way as an IP address – its shorthand, represented as `/xx` and appended to the end of an IP, shows how many of the first `xx` bits of the mask are 1s. For example, with the IP address `192.168.0.1/24`, the 24 means the first 24 bits of the subnet mask are ones, which can be written as `255.255.255.0`.

IPv4 address are split into 5 distinct classes:

- A: The first byte value ranges from 0 to 127 with a subnet mask of 8 (first byte for network address, remaining three for host addresses).
- B: The first byte value ranges from 128 to 191 with a subnet mask of 16 (first two bytes for network address, remaining two for host addresses).
- C: The first byte value ranges from 192 to 223 with a subnet mask of 24 (first three bytes for network address, remaining one for host addresses).
- D: The first byte value ranges from 224 to 239, which is used for multicast addresses (whose packets then get distributed to multiple recipients, per the name).
- E: The first byte value ranges from 240 to 255, and these IPs are used for experimental purposes.

One of the main issues with IPv4 is that its range of addresses have all been used (4 billion addresses, in Sep. 2015), and that broadcasting (sending packets to all clients on the LAN) clogs the network. The solutions to the problems listed are the introduction of IPv6 and NAT, and a router (respectively).

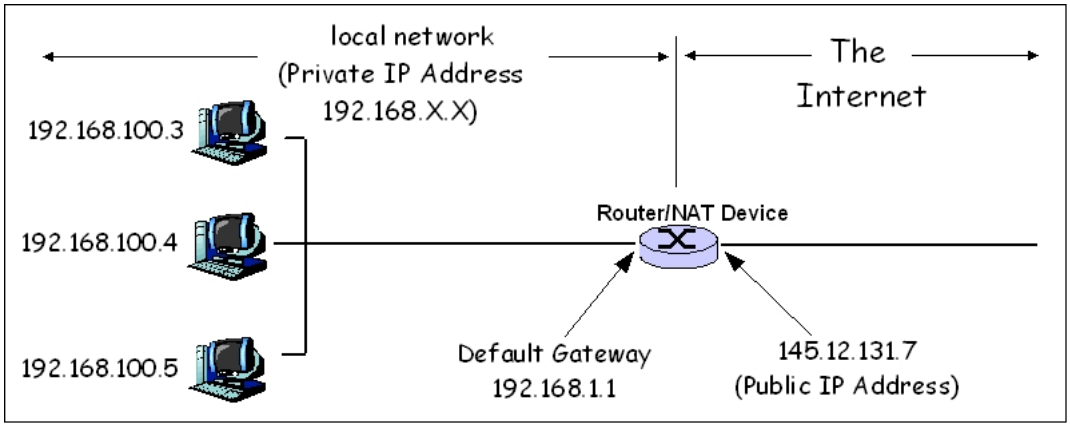
2.1.2 IPv6

IPv6 is the latest iteration of IP, which is 128 bits long. Other than a wider range of addresses, its benefits include:

- Authentication and access control
- Smaller header fields (half the size)
- 64-bit alignment (faster processing)
- Scalable and efficient routing

2.2 Network Address Translation (NAT)

Network Address Translation (NAT) is one of the other techniques used to circumvent the lack of addresses in IPv4, as well as provide security to local networks. NAT uses private address ranges, where a NAT device (e.g., router) maps a single IP address to the entire private network. The devices in the private network are then given a private IP, and attempts to directly connect to a device using a private IP from a device outside the network will be dropped.



Example of a NAT (Source: `wikipedia`)

§ **Note:** SNAT = Source NAT, DNAT = Destination NAT: two things the gateway is in charge of handling

§ **Note:** 192.168.x.x is the typical private network for homes

2.3 Domain Name System

Domain Name System, or DNS, is the phonebook of the internet. It takes a domain name and translates it into an IP so browsers can load internet resources.

2.4 Floating IP

A **floating IP** is a public, routable IP address that is obtained from a pool of IPs that a system administrator or cloud provider configures and provides as a server resource.

Chapter 3

Web Development

3.1 Overview

When a browser (the client) makes a request for a webpage, the workflow looks like this:

1. The client makes an async request to the server via HTTP, which looks like this: `GET / HTTP/1.1`
2. The server receives the request and finds the corresponding HTML file based on given parameters.
3. The client receives the HTML file, and as the HTML is read, it makes subsequent requests to the server (or an external one) for additional needed files like CSS or JS files. While the requests are being made (or are complete), the browser is rendering the page.

3.2 History

Originally, websites were not dynamic – maps were used for hyperlinking images, table design was popular, and responsive design did not exist (adjusting viewport and display by device).

Now, all the cool and modern websites have a multitude of CSS libraries (Bootstrap, Material) as well as templating libraries/frameworks (React, Vue) which can generate dynamic content from templates.

3.3 HTTP

3.3.1 About

Hypertext Transfer Protocol (HTTP) is the main protocol for the Worldwide Web (WWW), and is created to be human-readable format between the client and server (is also very inefficient). It supports a variety of methods, most commonly GET and POST (for receiving and sending data). HTTP requests come with a variety of request/response headers, which can be viewed in the browser's Developer Tools (in the Network tab).

3.3.2 URLs

A **uniform resource locator** (also called a web address) is a reference to a web resource that specifies its location on a network and the mechanism for retrieving it. A sample URL is shown below:

`https://www.facebook.com:443/users.html?user=ferdman&date=2019#bio`

Each piece can be categorized:

1. `https://` **Schema**: the protocol being used
2. `www.facebook.com` **Server Address**
3. `:443` **Port**: defaulted to 80 for http, and 443 for https
4. `/users.html` **Path**: doesn't have to include the `.html`, `index.html` of the given path will be defaulted
5. `?user=ferdman?date=2019` **Query String**: parameters for the server, which start with `?` and is of the format `k=v`, delimited by `?`
6. `#bio` **Anchor**: doesn't get sent to the server, is handled by the browser for navigation (typically)

§ **Note:** Non-printable characters in the query string are encoded, e.g., space = `%20`

URLs are technically a subset of URIs, because they are more specific locators (URL) for a generic identifier (URI).

3.3.3 HTTP Response Codes

For every HTTP response sent back from the server, a three digit number usually comes with it to indicate the status of the response:

- 1xx Informational (Continue, Processing, ...)
- 2xx Success (OK, Created, ...)
- 3xx Redirection (Page Moved, Redirect, ...)
- 4xx Client Error (Bad Request, Unauthorized, ...)
- 5xx Server Error (Service Unavailable, HTTP Version not supported, ...)

The most common codes are 200 (OK), and 404 (Not Found).

3.4 XHR, Fetch, and AJAX

XMLHttpRequest, or **XHR**, is a browser object that allows HTTP requests to be made in JavaScript. Despite its name, it support many datatypes (JSON, SOAP).

Fetch is a new browser method for sending network requests. Fetch code is much more concise, and supports new functionality (although it lacks some of XHR's capabilities).

Asynchronous JavaScript and XML, or **AJAX**, is a general technique for sending asynchronous JavaScript requests. Both XHR and Fetch can be used to achieve this.

3.5 REST

REpresentational State Transfer, or REST, is a design pattern for APIs. In essences, a server with a RESTful API will send to the client a *representation of the state* of the requested resource. For example, if we wanted to fetch a specific user (e.g., Ferdman), the API will return the state of the user, including their name, number of posts, friends, etc.

The need for a RESTful API arised when traditional webpages had servers send back snippets of HTML to render, but the client had no control over the actual handling of the data.

3.6 Message Formats

JavaScript Object Notation, or JSON, is the most well-known format. Use it. The main advantages is its readability, as well as its well defined structure and native support in JavaScript.

Older formats include XML and SOAP, which were different standards of representing data. Both formats failed to see widespread success, and WSDL, an extension of SOAP was created. It allowed users to see what function calls were being made, and it contributed to the now rising SOAP, which sees a lot of use in industry.

3.7 Comet and Websockets

Comet is a shitty model for HTTP client/server connections – it is unidirectional, where the server sends the client data. The main issue with comet is that the client has to repeatedly open new connections with the server to send it more information. The upside is that the server only needs to maintain one connection to send information, which is less expensive.

In comes **websockets**, where a two-way connection is maintained between client and server. This is a persistent, low latency connection which removes the problem of long-polling (keeping HTTP connection alive until server has data, which had the overhead of HTTP).

3.8 Cookies, Web Sessions, and JWT

Cookies are a small piece of data sent from the server to the client, used to record client state, which is accessed by the web server/client computer for various reasons (authentication, metrics, stalking).

Cookies are sent through requests by setting the SET-COOKIE header, and one of its main uses is to send stateful information, which avoids leaking information in the query string. Modern implementations of cookie in a client/server setting will involve signing the information in the cookie to prevent tampering.

However, as the traditional use of cookies aren't completely secure, **sessions** are a model where the server sends a key which is stored in a cookie. That key is related to some data which is associated server-side.

JWT, which was not covered in class (feel free to skip), is a web standard for encoding a JSON payload alongside a secret. It works almost like a session ID, but encoded and also has other info.

3.9 Speeding Up Requests

Successive HTML requests cause slow loading. How can we speed it up?

3.9.1 Caching

We can store content from previous HTTP requests, so that we don't have to make requests for resources we already have. Stored content will have expirations and can be replaced if the content is updated before expiration.

3.9.2 Pipelining

Just send a bunch of requests in a pipelined manner.

3.9.3 Content Delivery Network (CDN)

A CDN is a system of distributed servers which deliver content to a user based on their geographic location. This technique reduces latency as much as possible; the CDN is responsible for hosting your files on all their machines to deliver your resources quickly.

3.10 Proxies

A HTTP proxy is a middleman which routes client requests from a browser to the Internet, while potentially providing features like caching and security. It is usually implemented in the application layer.

Chapter 4

Databases

4.1 Overview

A database is a piece of software designed to store data reliably and consistently. We do not want to expose the database (duh), which means we should never open up a port to the public – hackers will scan ports like 3306 (MySQL default) to dig out data.

4.2 SQL and Related Syntax

A database is considered a SQL database if the data is defined by schemas and retrieved using Structured Query Language (SQL). The main advantages of using SQL is that you get referential integrity, structured data, and data guarantee.

Useful components of SQL (ordered by relevance):

- DML/CRUD: INSERT, SELECT, UPDATE, DELETE
- Data-definition Language (DDL): CREATE, ALTER, DROP
- Data-control language (DCL): GRANT, REVOKE for permissioning

4.2.1 MySQL

Used to be incredibly fast (thus popular), then it got caught up in a bunch of features (triggers).

4.3 NoSQL

A **NoSQL** database is schemaless: for the most part, we can describe these databases as a key-value store. The advantage of this is that adding new "columns" is supposedly faster, since all that has to be done is add more properties to a given object (thus tables aren't locked during column updates). Which means developers can release new software with just some property changes.

Some "variants" (buzzwords) of NoSQL databases include:

- Document systems: databases made to store larger data, such as a pdf scan (not just a JSON string).
- Column system: for any given key, you have a set of "columns".

And basically the only use case is if you don't care about your data (jk, if your columns are constantly being changed, this is good too).

Chapter 5

Virtualization

Virtualization is the act of virtualizing computer hardware and operating system by creating a virtual machine. A virtual machine uses the same underlying hardware resources as the host machine, and a single host machine can host multiple virtual machines.

A hypervisor is a piece of software that runs virtual machines. It exposes a virtual operating platform for guest machines that has hardware abstractions and any other abstractions that operating systems require to run.

5.1 History

Fundamentally emulators are just software that runs instructions, it was initially super slow because emulating commands (say Assembly commands like ADD/SUB) required the overhead of doing `x` with the registers. This initially meant it would take too many commands to run just one – and that added up. There would be long switch statements to figure out which command to run (same thing – still ADD, JMP, ...).

There were two major developments that sped up emulation:

1. **Hypervisor**: Type 1 hypervisors (bare metal), run directly on the hardware and makes it possible to use more of the system resources (run commands on physical hardware). The problem is that emulators are still filled with switch statements...
2. **Xen**: A type 1 hypervisor that supports concurrent execution on computer hardware through virtualization (virtual disk, etc). The better way to think about it is that Xen is a adapter for hardware. Later, Intel/AMD added more hardware support, adding control structures (VMCS) which defined the virtual machines and added support for `vmenter` and `vmexit` commands, when the machine needed help from the hypervisor.

Most VMs are run this way now, and we can't even tell it's slow.

5.2 Cloud Provider Instances

When creating an instance on a cloud, cloud providers offer a **flavor** selection, which is a combination of the CPU, memory and disk space of an instance. Additional network options can be configured, such as latency, bandwidth and aggregate transfer.

5.3 Definitions

Some useful definitions for this section – measurement unit in parenthesis:

- **Instance**: a virtual machine (VM) running on a server
- **Image**: system/software preinstalled into instance
- **Latency** (ms): amount of time it takes to transfer data. Limited by distance (solvable with a CDN).
- **Bandwidth** (GB/s): amount of data that can be transferred per unit of time
- **Aggregate Transfer** (GB): total amount of data transferred
- **Input/Output Operations per Second** (**IOPS**) (k (thousands)): it's in the name

5.4 Disk

Your given storage mechanism – here we care about:

- Latency
- Bandwidth
- IOPS

An average benchmark for some types of disks:

	Range	SSD	HDD
Latency	1 to 20ms	1ms	10ms
Bandwidth	100 to 1000MB/s	1GB/s	100MB/s
IOPs	200k	200k	20k
Capacity	4 to 12000GB	1TB	12TB
Capacity/$		10GB/$	33GB/$

5.5 Networking

Your given internet connection – the main metrics you need to care about for performance are:

- Latency
- Bandwidth
- Aggregate Transfer
- Jitter – the latency variability of a network (this one is less important)

Most of the times VMs don't use the full bandwidth given to them which is why cloud providers don't give you the information up front. They want to tell you that you have a good network, and then sometimes give you a good transfer in the middle of the night.

Another reason why cloud providers don't provide how much network you're getting is because of torrents and porn – two activities which take up bandwidth. It lets providers charge you extra for all the network you're using.

5.6 Memory

Hardware that stores information temporarily for immediate use, memory also has similar key characteristics:

- Latency (ns)
- Bandwidth
- Capacity

When determining what memory to use, latency and bandwidth are disregarded because typical CPUs cannot keep up with the memory (limited by the computing power and bandwidth of the machine). The capacity is actually the most expensive component of a server and one of the key bottlenecks.

One of memory's primary functions is to store disk cache – this is because retrieval from memory is much faster than disks (even SSD).

We say memory is oversubscribed when VMs are assigned more resources than are actually available (performance issue). Storing disk cache actually takes up a good chunk of the memory (which causes it to run out), and if the OS runs into this problem, it can:

- Throw out the disk cache as needed (the information is still on disk anyways)
- Use balloon software – that is, pretend more memory has been used to free up space.

5.7 Pricing

Most providers will charge based on compute and storage resources (basically flavor), as well as the aggregate transfer.

§ **Note:** "Serverless" servers is a model where providers dynamically allocate machine resources, and charge based on amount of resources consumed. Essentially they give you containers (see next section), instead of a cloud instance. It can be much cheaper!

5.8 Containers

A **container** is a packaged application such that it can be run in an isolated environment. That is, it is bundled with its dependencies, system tools, and runtime.

§ **Note:** In the case of an example like Docker, container images become containers at runtime.

The main advantage of containers is that they're easy to bring up as well as destroy, which lets us experiment much quicker.

5.9 Init

Init is a system used to bootstrap the user space and manager user processes; it contains utilities to initialize cloud machines.

Chapter 6

Scaling

How to start from a baby instance to big enterprise clusters: here are the terms and ideas you need to know.

6.1 Migration

Migration is the core to growing our servers: it involves copying over and well, migrating data over.

Migration involves moving an instance from one physical host to another physical host by using cloning. **Cloning** is creating a new instance that is a copy of the source instance (i.e., a snapshot). A live migration is a migration where new data written to the original instance is kept track of, and then replicated when the cloning is finished. Usually it also involves switching the floating IP to point to the new instance as well, making the change seamless and unnoticeable to the end user.

6.2 Actual Scaling

We understand scaling in two ways:

- **Scale Up**: Improving machine resources; more CPU, memory, etc. Also known as vertical scaling.
- **Scale Out**: Adding more instances to separate workload. Also known as horizontal scaling.

Scaling up is fairly obvious to do. Scaling out is a little more delicate, and we need to understand the concept of sharding and replication.

Sharding involves splitting up a database into many distinct, smaller databases. An example is splitting up a database by user's full name alphabetically (one database for A-M, another for N-Z). As a result, data interaction becomes much more responsive.

Replication is a different technique, where multiple databases which are all identical are hosted on many instances. Updates to one database gets sent to one "master" database, which propagates the change to all the other "slave" databases. With enough time, all the databases will reach **eventual consistency**, where all the data across the databases are the same (if no new updates are made to a given item).

6.3 Service-oriented Architecture

One of the most popular ways that companies structure their services is by following a **service-oriented architecture** (SOA). What this means is that the functionality of the service is split up into various decoupled services which all perform a set of tasks. This helps with cleanliness and debugging, and allows teams to work on one small service at a time. **Microservices** are actually the same as SoA, but slightly nuanced in that the services are much more granular, performing a single task. For the most part, these two are referring to the same thing.

Chapter 7

Development Tools

7.1 Scripting Languages

The most overpowered things to hit the computer science industry, scripting languages allow us to develop quicker. The language itself has a lot of built-in functionality and is easy to get demos started. Notable examples include Python and JavaScript.

7.2 Message Brokers

A **message broker** is software that receives, validates, transforms (message protocols), and routes messages across services. This pattern essentially allows for the decoupling of services. RabbitMQ is an example of one of the most popular message broker services available.

7.3 Others

There is also Docker, Kubernetes, Ansible, and other homework items that are valuable development tools. I won't write them up because it's too late.

This page intentionally left blank to signal divide between midterm 1 and midterm 2 content.

Chapter 8

Development Practices

8.1 Choosing a DB or File System

There are many data storage tools available; in order to effectively pick a tool, we must consider the **CAP theorem**. The CAP theorem is a concept that states a distributed DB system can only have two of the three:

- **C**onsistency: All nodes should see the same data at the same time. To achieve this, all nodes will need time to propagate updates, and thus will not always be *available*.
- **A**vailability: The system should remain operational 100% of the time.
- **P**artition Tolerance: A single node failure should not bring down the entire network; data records are sufficiently replicated across networks. With modern systems, partition tolerance is a necessity, so the tradeoff is usually between consistency and availability.

§ **Note:** Picking a **centralized** (easy to setup but hard to scale) versus a **decentralized** (harder to setup but more effective) is also an important decision!

The main key-value (?) distributed file systems available today are **HDFS** (Hadoop Distributed File System) and **Ceph**. Their main functions include:

- Support for large objects/items
- Sharding and Replication out of the box (can easily do 1000+ instances)

Their main differences lie in their implementation and how they're built:

For HDFS (Centralized – has something called a NameNode):

- Data locality: moving computation closer to data rather than moving data closer to computation, which results in faster execution

For Ceph (Decentralized?):

- No data locality: no coupling of compute and storage
- Hard to optimize – have no idea which instance your data is in

8.1.1 Storing Images and Other Media

The best place to store large media files is usually NoSQL databases or distributed file systems – they come with out of the box sharding/replication features that make development much easier. In addition, there may be libraries to assist in chunking large data so that large files could be loaded incrementally.

SQL databases are only used if the files are absolutely critical.

For the middleware/backend, large files should be stored in memory for performance, and disk if security/risk is more important.

8.2 Writing Logs

Logs are a record of events that occur in an operating system or software – they're the backbone of debugging.

It's tempting to log everything possible, but writing logs is performance intensive as it consumes a lot of RAM. To better identify logs in a distributed system, we have to have 2 things: what module produced the message and the severity of the message.

8.2.1 Log Severity

Severity in logs is usually classified by the following (in a rough ordering from least to most urgent):

1. Verbose (VERB)
2. Very Verbose (VVERB)
3. INFO
4. DEBUG
5. WARN
6. ERROR
7. CRITICAL
8. FATAL

The usual process is to split the severities into separate files. Some systems will actually duplicate messages across files depending on relevancy (maybe we want to see messages WARN-level and lower).

8.2.2 Syslogd

Syslogd is a UNIX utility that specifies which messages go into which log files.

One of its primary functions is to perform a **log rotation**, which is a mechanism that prevents you from filling your entire disk space with logs. In a log rotation, we specify:

- The length of the rotation: how many days worth of logs are being kept. Other outdated files will be deleted.
- Which files should be compressed (e.g., compress all log files 4 days before today).

Another function of syslogd is to manage log filenames (duh). For each new day, the log files are suffixed with `.1`. If a log file already exists with that suffix, the offending log file becomes `.2`, and the cycle contains. Essentially, it stands for days elapsed.

8.3 Consistency in Logs

To ensure consistency in log files, synchronizing the time throughout all the instances is important. Otherwise, instances in different time zones (and even the same actually) will output log files with conflicting times. To remedy this, it is recommended that all instances use **UTC** (Coordinated Universal Time).

The recommended software package for clock synchronization is **NTP (Network Time Protocol)**; more specifically, it is a networking protocol. The way it works is that the NTP client makes a time-request exchange to some NTP server, where it calculates link delay, local time offset, and its local clock to match the clock at the server's computer.

8.4 Searching Logs

There are two recommended software (packages) for ingesting and searching logs, and those are:

- Splunk
- ELK: Elasticsearch, Logstash, Kibana

Chapter 9

A Study of Search Engines

9.1 How does Google rank its pages?

For a query through Google, it looks at matches in the HTML document (specifically the description tag in the header), location, and **page reputation**.

Page reputation considers a variety of factors, namely:

- Amount of traffic received
- Number of links to it from other reputable webpages
- Avoidance from blacklists

9.2 Precision and Recall

Some useful definitions:

- **True Positive**: Correct results that are returned
- **True Negative**: Incorrect results that are not returned
- **False Positive**: Incorrect results that are returned
- **False Negative**: Correct results that are not returned

When evaluating the quality of a search engine, we look at the proportion of the above result types. Specifically, we talk about 2 things:

- **Precision**: the fraction of relevant results versus displayed results (less false negatives and false positives)
- **Recall**: the fraction of relevant results versus all the potential relevant results (amount of false positives in particular)

Displayed as mathematical expressions:

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Actual Results}}$$

§ **Note:** Google indexes way more sites than Bing, but since Bing indexes less sites, their first page likely has more relevant results (better recall)! To be honest, Google has multiple copies of all sites on the Internet...

9.3 Creating a Search Engine

A way to efficiently index webpages is to store them in an **inverted index** data structure. What that is, is just a semantic: ultimately it maps key words to documents, rather than a document mapping to its list of words (which is a **forward index**).

We can also improve performance by **stemming**: removing "ing" or "s" from words, and by removing **stop words**: commonly used words like "the" or "a". However, when ranking, search engines may or may not take into account stop words. Similarly, search engines may also help perform spell-check on your words.

And so when a user has a multi-word query, we grab all the documents for each keyword and do a union of the set to maximize recall. This process is costly, and we have to scale out the system:

9.3.1 Scaling out the Search Engine

Instantly, we turn to sharding, because we want to split up what is indexed (of course, the shards are then replicated). The current standard for sharding is to shard by URL: queries will need to grab from multiple machines for a keyword, but currently this is the best complexity-wise.

§ **Note:** An alternative is to shard by keyword, but 2 main problems is that shards have no idea where the other documents/webpages are during multi-keyword searches (adds complexity when piecing results together), and that recrawling websites will mean multiple shards are going to be updated. Also, the shards are going to be extremely large.

9.4 Extras

`Robots.txt` is a document you can include in your website to prevent crawling of certain urls – Google abides by it but many crawlers don't.

Snippets are the summaries of the search results page – on Google, they consist of the page title, the URL of the web resource, and a *query-based summary* of the web document.

Chapter 10

Performance

We preface this chapter with two common ways to improve performance: **caching**, which typically using in-memory storages since they're the fastest, and **queueing writes**, which means we stick the message in a queue and return OK (we cheat by updating the GUI anyways, and we're fucked if the machine handling request goes down).

Otherwise, we scale up/out:

10.1 Bottlenecks

A **bottleneck** is a process in a chain of processes whose capacity reduces the throughput of the entire chain (i.e., a point of congestion). Every system will have a bottleneck – if you improve the bottleneck, it will appear elsewhere. That's why it's also known as "shifting the bottleneck". There are two main types of bottlenecks, hardware and artificial bottlenecks.

10.1.1 Hardware Bottlenecks

Hardware bottlenecks have to do with memory, CPU, network, or disk usage. We can go over each shortly:

- Memory: Not enough RAM
- CPU: Not enough free cores / Cores are always busy (never idle)
- Network: Internet speeds too slow (almost never a problem)
- Disk: Not enough drive space, usually due to logging

10.1.2 Artificial Bottlenecks

Artificial bottlenecks have to do with the code quality itself, whether it's a bad framework or inefficient algorithms. It might even be the lack of indexes in your database or timeouts.

That is, maybe we are using an exponential time algorithm for a task when it could be linear time. Or using React, Angular, and Vue all at the same time when we only need one frontend framework/library.

Artificial bottlenecks can also deal with the framework itself – Node.js only runs on one core, so scaling up the machine Node is on won't help (there may be libraries to help though).

10.2 Identifying Bottlenecks (top/htop)

top is a command line utility that shows a real-time summary of the system. Referencing the image below, we can describe what all the mumbo-jumbo means:

- Line 1 (`top - 06:27:01 up 2 days`): The current UTC time and how long the system has been up.
- Line 1 (`load average: 0.04, 0.05, 0.02`): Measure of how many processes are ready to run (running/queued) at a given time, calculated over 1, 5, and 15 minutes, respectively. Reasonable numbers should match of be less than the number of VCPUs; any more and we have overloading.
- Line 2: Self-explanatory – the numbers don't add up to the total because `top` doesn't report every status. For reference, processes usually do nothing (sleep) until an interrupt triggered by either a time event or I/O. Zombie tasks are those not reaped by their parent.
- Line 3: Total CPU usage: `us` = user processes, `sy` = kernel processes, `id` = idle. A general rule of thumb is to keep idle time above 40% – otherwise, your system might be too busy.
- Line 4: Physical memory: Actual RAM use
- Line 5: Virtual memory: Essentially a partition of your hard drive used as extra RAM

And for the table, which is a list of processes:

- PID: Process ID
- PR: Priority
- VIRT: Virtual Memory (KiB) – virtual memory usage (usually how much memory the program has asked for)
- RES: Resident Memory (KiB) – the non-swapped physical memory usage (usually how much memory is actually used)
- S: Status
- %CPU: CPU usage as a percentage of a single CPU – if the process used 3 cores at 50%, it would show 150%.

```
top - 06:27:01 up 2 days,  7:40,  1 user,  load average: 0.04, 0.05, 0.02
Tasks: 108 total,   1 running,  59 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.2 st
KiB Mem :  7772624 total,  3961472 free,   123092 used,  3688060 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  7340416 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
24666 ubuntu    20   0   44532   4212   3588 R   0.7  0.1   0:00.27 top
23983 root      20   0       0      0      0 I   0.3  0.0   0:00.03 kworker/u8:2
    1 root      20   0  225412   9312   6836 S   0.0  0.1   0:14.40 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.03 kthreadd
    3 root      20   0       0      0      0 I   0.0  0.0   0:05.68 kworker/0:0
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.03 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   0:04.42 rcu_sched
```

Shown above: sample `top` output

And as an extra note, `htop` is a more colorful and slightly more informative version of `top`.

10.3 Checking Disc I/O

`vmstat` is the main linux utility for checking virtual memory statistics. The main columns to look at are BI and BO, which represent blocks in (received) and blocks out, respectively.

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 3960964  57344 3631260    0    0     1    19   10    6  1  0 98  0  0
```

Shown above: sample `vmstat` output

10.4 Approach to Improving Performance

Usually the first thing to look at is your artificial bottlenecks: if your software is poorly written, no amount of sharding and replication will save your application.

Afterwards, we can move onto checking logs and using `top`. It is highly recommended to gather all your statistics and logs in one place for better analytics (see Section 8.2/12.1 or Chapter 13).

Once the bottleneck is found, the things to do are (in a rough order): check your app's configurations, review code (if applicable), and scale up (easier thing to do) or scale out.

In practice we care a lot about our **tail latencies**, that is, the higher percentiles. There's always going to be something that is slow, but we want to reduce the worst-case times for users.

Chapter 11

Machine Learning

11.1 Why use it?

We use machine learning (ML) because we are incapable of coming up with an algorithm for the problem. The most popular uses for ML are: computer vision (CV), natural language processing (NLP), and recommendation systems.

11.2 ML Types and Models

11.2.1 Supervised vs. Unsupervised

Supervised learning is where the model is fed output that is labelled with the correct answers. An example is a training model where objects in an image are already tagged – the machine then learns the characteristics of the tagged objects and can identify future objects using their characteristics. Common uses for supervised learning are classification (more later) and regression (prediction and forecasting).

Unsupervised learning is the exact opposite: none of the outputs are labeled, and the machine must act on the information without guidance. An example is a model that is trained on images of oak and willow trees – the model can categorize the two based on their characteristics and differentiate them. Common uses for unsupervised learning are clustering (finding groupings in the data) and association (finding rules that describe data, e.g., people who play X tend to play Y).

11.2.2 Neural Networks

Neural networks are a general classification for all machine learning algorithms where input data is sent to a "neuron" that does a dot product of the data with weights (which then forwards it to neurons of the next layer, ..., until we reach the output). **Artificial neural network** is the more official term, differentiating the ML side from an actual biological neural network (our brains).

In the case of a **convolutional neural network** (CNN), each weight is represented by a matrix, which is then used to produce a pixel. Specifically, CNNs slide a kernel across dimensions to produce an image. We use an image to explain CNNs because it is most commonly applied to visual imagery. We use this method because in a way it allows the model to also pick up on the surroundings of an image as well.

Long short-term memory (LSTM) is a neural network that introduces a memory unit called the **cell**, which stores data useful for computations across all units (which may alter the memory). A popular use of an LSTM is in **transformers**, which translates one sequence to another. An example of a transformer is for language translation – using an LSTM, the neural net is able to use past words to give context to future ones.

11.2.3 Classification, Detection, Segmentation

All techniques listed above fall under computer vision, which we describe below:

- **Classification**: labelling what is in the video (doesn't say where it is, just if it exists)
- **Detection**: distinguishing distinct objects in an image/video, separating them using bounding boxes
- **Segmentation**: providing an exact outline of each object that is classified

11.3 ML Techniques

11.3.1 Embedding

Embedding is the process of projecting an input into a better representation space. In this class, we specifically refer to the transformation of words into numbers for easier processing.

11.4 Miscellaneous Terms

- **Overfitting**: Training a model too closely to a particular set of data, such that it is unable to generalize and is unable to fit additional data points.
- **Hidden Layer**: Essentially any layer in an artificial neural network that isn't the input or output (anything in between). It takes the weighted inputs and produces output using an activation function.

Chapter 12

Monitoring and Security

12.1 Introduction to Monitoring

Outside of `top`, the most popular (and scalable) method of discovering bottlenecks, bugs, and any issues is to use a monitoring service. Most monitoring services are **agent-based**, where agents report metrics to a main server, which aggregates all the data.

§ **Note:** Generally, an agent is usually software that runs in the background for *another user or program*, which differentiates it from a daemon.

12.2 Security Practices

Here we talk about best practices for security:

12.2.1 Password Hashing

Password hashing is the act of performing a one-way transformation on a password, such that it is impossible to turn the hashed password back to its original. That way, if a collection of password were somehow compromised, hackers still wouldn't be able to log into the system since they can't reverse it.

12.3 Security Vulnerabilities

In this section, we talk about various vulnerabilities possible in any system:

12.3.1 Server-side Code Injection

In a **server-side injection attack**, the attacker submits malicious input which gets executed on the server. The input could be submitted in various ways, such as web forms or URL parameters.

One of the prerequisites for a server-side attack is to gain an understanding of the back-end application – a common method is to use an information disclosure attack, but we won't go into that here.

Some examples of a server-side attack: SQL injection, command injection (like shell?), and buffer overflow attacks.

12.3.2 Client-side/Cross-site Code Injection

Cross-site (XSS) injections work similarly to server-side ones; instead of SQL or shell commands, script tags are entered into web forms (or anything that allows info to be saved into the DB). Once the script tags are in the database, any user who queries that script tag will run malicious code which can send private data to the attacker's private server (or worse). This kind of attack is usually referred to as a *stored* XSS attack.

Another kind of XSS is a *reflected* attack, here the attacker sends a malicious link/URL to a user, which contains a script tag in its query parameters. This attack usually involves more social engineering or mass-sending of emails to work, as well as a lack of data escaping of the insecure website which allows script tags as query parameters.

Chapter 13

Analytics

13.1 MapReduce

MapReduce is a processing technique/algorithm for processing/generating huge datasets on a cluster. The procedure is composed of 3 parts:

1. Map: Performs the conversion of data – more specifically, it takes key/value pairs and produces another set of intermediate key/value pairs
2. Shuffle: Said to be part of the Reduce phase, shuffling is where intermediate data from mappers are transferred to reducers
3. Reduce: Performs a summary operation to create a smaller set of tuples (producing zero or more outputs)

13.2 Apache Hadoop

Apache Hadoop is a collection of open-source software utilities (library) that allows for distributed processing of large datasets across clusters of computers. Its main strengths is its ability to work efficiently on parallel and on cheap hardware. It is comprised of 4 main modules:

- Hadoop Common: libraries and modules used by all other Hadoop modules
- HDFS: distributed file-system that stores data, providing high bandwidth
- Hadoop YARN: platform responsible for managing computing resources in clusters and using data to schedule applications
- Hadoop MapReduce: implementation of MapReduce programming model for large-scale data processing

Hadoop is good for long-running batch jobs – however, for real-time use, not so much.

13.3 Apache Spark

Apache Spark is another popular distributed computing framework. Spark is better for real-time processing, but is highly dependent on doing stuff from memory – reading stuff from disk is going to kill it. Its more likely to be seen in a smaller startup, maybe with less data.