Put Back-of-the-envelope Numbers in Perspective

Learn to use appropriate numbers in back-of-the-envelope calculations.

We'll cover the following

Why do we use back-of-the-envelope calculations?

A distributed system has compute nodes connected via a network. There's a wide variety of available compute nodes and they can be connected in many different ways. Back-of-the-envelope calculations help us ignore the nitty-gritty details of the system (at least at the design level) and focus on more important aspects.

Some examples of a back-of-the-envelope calculation could be:

- The number of concurrent TCP connections a server can support.
- The number of requests per second (RPS) a web, database, or cache server can handle.
- The storage requirements of a service.

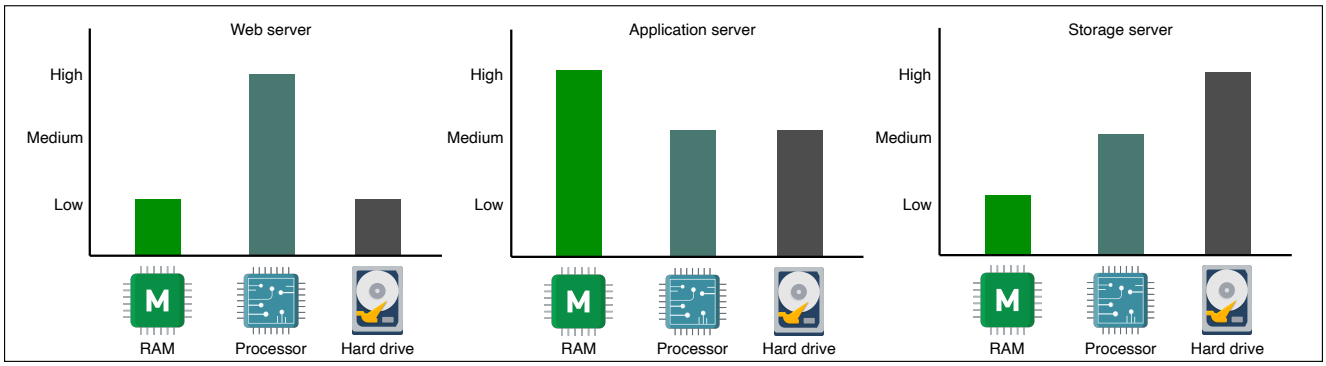
Choosing an unreasonable number for such calculations can lead to a flawed design. Since we need good estimations in many design problems, we'll discuss

all the relevant concepts in detail in this lesson. These concepts include:

- The types of data center servers.
- The realistic access latencies of different components.
- The estimation of RPS that a server can handle.
- Examples of bandwidth, servers, and storage estimation.

Types of data center servers

Data centers don't have a single type of server. Enterprise solutions use commodity hardware to save cost and develop scalable solutions. Below, we discuss the types of servers that are commonly used within a data center to handle different workloads.



An approximation of the resources required on the web, application, and storage layer of the server. The Y-axis is a categorical axis with data points for low, medium, and high

Web servers

For scalability, the web servers are decoupled from the application servers. **Web servers** are the first point of contact after load balancers. Data centers have racks full of web servers that usually handle API calls from the clients. Depending on the service that's offered, the memory and storage resources in web servers can be small to medium. However, such servers require good computational resources. For example, Facebook has used a web server with 32 GB of RAM and 500 GB of storage space. But for its high-end computational needs, it partnered with Intel to build a custom 16-core processor.

> **Note**: Many numbers quoted in this lesson are obtained from the data

> center design that Facebook open-sourced in 2011. Due to the slowing of Moore's law-induced performance circa 2004, the numbers are not stale.

Application servers

Application servers run the core application software and business logic. The difference between web servers and application servers is somewhat fuzzy. Application servers primarily provide dynamic content, whereas web servers mostly serve static content to the client, which is mostly a web browser. They can require extensive computational and storage resources. Storage resources can be volatile and non-volatile. Facebook has used application servers with a RAM of up to 256 GB and two types of storage—traditional rotating disks and flash—with a capacity of up to 6.5 TB.

Storage servers

With the explosive growth of Internet users, the amount of data stored by giant services has multiplied. Additionally, various types of data are now being stored in different storage units. For instance, YouTube uses the following datastores:

1. **Blob storage** for its encoded videos.
2. A **temporary processing queue storage** that can hold a few hundred hours of video content uploaded daily to YouTube for processing.
3. Specialized storage called **Bigtable** for storing a large number of thumbnails of videos.
4. **Relational database management system (RDBMS)** for users and videos metadata (comments, likes, user channels, and so on.

Other data stores are still used for analytics—for example, Hadoop's HDFS. Storage servers mainly include structured (for example, SQL) and non-structured (NoSQL) data management systems.

Coming back to the example of Facebook, we know that they've used servers with a storage capacity of up to 120 TB. With the number of servers in use, Facebook is able to house exabytes of storage. One exabyte is 10^{18} Bytes. By convention, we measure storage and network bandwidth in base 10, and not

base 2. However, the RAM of these servers is only 32 GB.

> **Note:** The servers described above are not the only types of servers in a data center. Organizations also require servers for services like configuration, monitoring, load balancing, analytics, accounting, caching, and so on.

The numbers open-sourced by Facebook are outdated as of now. In the table below, we depict the capabilities of a server that can be used in the data centers of today:

Typical Server Specifications

Component	Count
Number of sockets	2
Processor	Intel Xeon X2686
Number of cores	36 cores (72 hardware threads)
RAM	256 GB
Cache (L3)	45 MB
Storage capacity	15 TB

> **Note**: Hardware threads were originally called simultaneous multithreading. Later, Intel rebranded it as hyper threading.

The numbers above are inspired by the Amazon bare-metal server, but there can be more or less powerful machines supporting much higher RAM (up to 8 TB), disk storage (up to 24 disks with up to 20 TB each, circa 2021), and cache memory

(up to 120 MB).

Standard numbers to remember

A lot of effort goes into the planning and implementation of a service. But without any basic knowledge of the kind of workloads machines can handle, this planning isn't possible. Latencies play an important role in deciding the amount of workload a machine can handle. The table below depicts some of the important numbers system designers should know in order to perform resource estimation.

Important Latencies

Component	Time (nanoseconds)
L1 cache reference	0.9
L2 cache reference	2.8
L3 cache reference	12.9
Main memory reference	100
Compress 1KB with Snzip	3,000 (3 microseconds)
Read 1 MB sequentially from memory	9,000 (9 microseconds)
Read 1 MB sequentially from SSD	200,000 (200 microseconds)
Round trip within same datacenter	500,000 (500 microseconds)
Read 1 MB sequentially from SSD with speed ~1GB/sec SSD	1,000,000 (1 milliseconds)
Disk seek	4,000,000 (4 milliseconds)
Read 1 MB sequentially from disk	2,000,000 (2 milliseconds)
Send packet SF->NYC	71,000,000 (71 milliseconds)

Apart from the latencies listed above, there are also throughput numbers measured as queries per second (QPS) that a typical single-server datastore can handle.

Important Rates

QPS handled by MySQL	1000

QPS handled by key-value store	10,000
QPS handled by cache server	100,000–1 M

The numbers above are approximations and vary greatly depending on a number of reasons like the type of query (**point↴** and ***range↴***), the specification of the machine, the design of the database, the indexing, and so on.

Requests estimation

This section discusses the number of requests a typical server can handle in a second. Within a server, there are limited resources and depending on the type of client requests, different resources can become a bottleneck. Let's understand two types of requests.

- **CPU-bound requests:** These are the type of requests where the limiting factor is the CPU.
- **Memory-bound requests:** These are the types of requests that are limited by the amount of memory a machine has.

Let's approximate the RPS for each type of request. But before that, we need to assume the following:

- Our server has the specifications of the typical server that we defined in the table above.
- Operating systems and other auxiliary processes have consumed a total of 16 GB of RAM.
- Each worker consumes 300 MBs of RAM storage to complete a request.
- For simplicity, we assume that the CPU obtains data from the RAM. Therefore, a caching system ensures that all the required content is

available for serving, without there being a need to access the storage layer.
- Each CPU-bound request takes 200 milliseconds, whereas a memory-bound request takes 50 milliseconds to complete.

Let's do the computation for each type of request.

CPU bound: A simple formula used to calculate the RPS for CPU-bound requests is:

$$RPS_{CPU} = Num_{CPU} \text{ x } \frac{1}{Task_{time}}$$

In this calculation, we use these terms:

RPS_{CPU} : CPU bound RPS

Num_{CPU} : Number of CPU threads, which are also called hardware threads

$Task_{time}$: The time each task takes to complete

> Then, RPS_{CPU} = 72 x $\frac{1}{200ms}$ = 360 RPS.

The rationale for the calculation shown above is that we can visualize one second as a box and we calculate how many mini-boxes (tasks) can fit inside the big box—that is, the number of tasks that can be completed in one second by a number of CPUs. So, a higher number of CPUs/threads will result in a higher RPS.

Memory-bound requests: For memory-bound requests, we use the following formula:

$$RPS_{memory} = \frac{RAM_{size}}{Worker_{memory}} \text{ x } \frac{1}{Task_{time}}$$

In this calculation, we use these terms:

RPS_{memory} : Memory bound RPS

RAM_{size} : Total size of RAM

$Worker_{memory}$: A worker in memory that manages a request

> Then, RPS_{memory} = $\frac{240GB}{300MB}$ x $\frac{1}{50ms}$ = 16,000 RPS

Continuing our box analogy from the explanation of CPU-bound processes, here we first calculate the number of boxes there are (how many memory-bound processes a server can host) and then how many mini-boxes (tasks) we can fit in each of the bigger boxes.

A service receives both the CPU-bound and memory-bound requests. Considering the case that half the requests are CPU-bound and the other half memory-bound, we can handle a total of $\frac{360}{2} + \frac{16{,}000}{2} = 8,180 \approx 8,000$ RPS

The calculations above are only an approximation for developing an understanding of the basic factors involved in estimating RPS. In reality, a lot of other factors come into play. For instance, latency is required to do a disk seek in case the data is not readily available in RAM or if a request is made to the database server, which will also include the database and network latency. Additionally, the type of query also matters. Of course, faults, bugs in code, node failures, power outages, network disruptions, and more are inevitable factors.

On a typical day, various types of requests arrive, and a powerful server that only serves static content from the RAM might handle as many as 500k RPS. On the other end of the spectrum, computational-intensive tasks like image processing may only allow a maximum of 50 RPS.

> **Note:** In reality, capacity estimation is a hard problem, and organizations learn how to improve it over the years. A monitoring system keeps an eye on all parts of our infrastructure to give us early warnings about overloading servers.

Examples of Resource Estimation

Try your hand at some of the back-of-the-envelope numbers.

We'll cover the following

Introduction

Now that we've set the foundation for resource estimation, let's make use of our background knowledge to estimate resources like servers, storage, and bandwidth. Below, we consider a scenario and a service, make assumptions, and based on those assumptions, we make estimations. Let's jump right in!

Number of servers required

Let's make the following assumptions about a Twitter-like service.

Assumptions:

- There are 500 million (M) daily active users (DAU).
- A single user makes 20 requests per day on average.
- Recall that a single server can handle 8,000 RPS.

Estimating the Number of Servers

Daily active users (DAU)	500	M
Requests on average / day	20	
Total requests / day	f 10	Billion
Total requests / second	f 115	K
Total servers required	f 15	

Hide Calculations

- Total requests per second $\frac{500M \times 20}{86400} = 115K$ requests per second which is the RPS for our system.
- For the number of servers, $\frac{Number\ of\ Requests/sec}{RPS\ of\ server} = \frac{115K}{8k} \approx 15$ servers to handle all requests.

Indeed, the number above doesn't seem right. If we only need 15 commodity servers to serve 500M daily users, then why do big services use millions of servers in a data center? The primary reason for this is that the RPS is not enough to estimate the number of servers required to provide a service. Also, we made some underlying assumptions in the calculations above. One of the assumptions was that a request is handled by one server only. In reality, requests go through to web servers that may interact with application servers that may also request data from storage servers. Each server may take a different amount of time to handle each request. Furthermore, each request may be handled differently depending upon the state of the data center, the application, and the request itself. Remember that we have a variety of servers for providing various

services within a data center.

We have established that:

1. Finding accurate capacity estimations is challenging due to many factors—for example, each service being designed differently, having different fan-out rules and hardware, server responsibilities changing over time, etc.
2. At the design level, a coarse-grained estimation is appropriate because the purpose is to come up with reasonable upper bounds on the required resources. We can also use advanced methods from the queuing theory and operations research to make better estimates. However, an interview or initial design is not an appropriate time to use such a strategy. Real systems use various methods (back-of-the-envelope calculations, simulations, prototyping, and monitoring) to improve on their initial (potentially sloppy) estimates gradually.

Therefore, we approximate the number of servers by depicting how many clients a server handles on a given day:

$$\frac{Number\ of\ daily\ active\ users}{RPS\ of\ a\ server}$$

To have a better approximation, let's use the following two assumptions:

1. We assume that each daily active user (DAU) of the service sends just one request per day.

$$\frac{Number\ of\ daily\ active\ users}{RPS\ of\ a\ server} = \frac{Number\ of\ requests/day}{RPS\ of\ server} \quad (1)$$

2. We assume that there is a specific second in the day when all the requests of all the users arrive at the service simultaneously. We use it to get the capacity estimation for a peak load. To do better, we will need request and response distributions, which might be available at the prototyping level. We might assume that distributions follow a particular type of distribution,

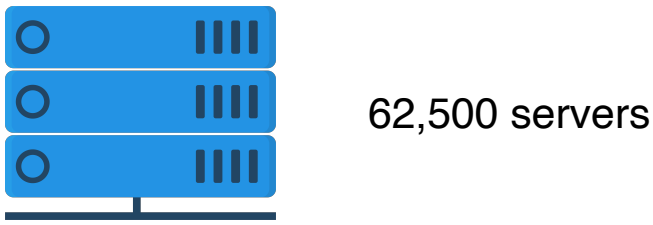
for example, the Poisson distribution. By using DAU as a proxy for peak load for a *specific second*, we have avoided difficulties finding the arrival rates of requests and responses.

Therefore, the DAU will then become the number of requests per second. We already have RPS for a server; therefore, equation (1) becomes:

$$\frac{Number\ of\ requests/second}{RPS\ of\ server}$$

By inserting values in the equation above, we yield the following number of servers:

$$\frac{Number\ of\ requests/second}{RPS\ of\ server} = \frac{500\ M}{8000} = 62,500$$



Number of servers required for a Twitter-like service

> **Note:** Our calculations are based on an approximation that might not give us a tight bound on the number of servers, but still it's a realistic one. Therefore, we use this approach in estimating the number of servers in our design problems. Informally, the equation given above assumes that one server can handle 8,000 users per second. We use this reference in the rest of the course as well.

Storage requirements

In this section, we attempt to understand how storage estimation is done by using Twitter as an example. We estimate the amount of storage space required by Twitter for new tweets in a year. Let's make the following assumptions to begin with:

- We have a total of 250 M daily active users.
- Each user posts three tweets in a day.
- Ten percent of the tweets contain images, whereas five percent of the tweets contain a video. Any tweet containing a video will not contain an image and vice versa.
- Assume that an image is 200 KB and a video is 3 MB in size on average.
- The tweet text and its metadata require a total of 250 Bytes of storage in the database.

Then, the following storage space will be required:

Estimating Storage Requirements

Daily active users (DAU)	250	M
Daily tweets	3	
Total requests / day	*f* 750	M
Storage required per tweet	250	B
Storage required per image	200	KB
Storage required per video	3	MB
Storage for tweets	*f* 187.5	GB
Storage for images	*f* 15	TB
Storage for videos	*f* 112.5	TB
Total storage	*f* 128	TB

Hide Calculations

- Total tweets: $250M \times 3 = 750 \times 10^6$ tweets per day.
- Storage required for tweets in one day:
  $750 \times 10^6 \times 250B = 187.5\ GB$
- Storage required for 10 percent images for one day:
  $\frac{750 \times 10^6 \times 10}{100} \times 200 \times 10^3 B = 15\ TB$
- Storage required for 5 percent video content for one day:
  $\frac{750 \times 10^6 \times 5}{100} \times 3 \times 10^6 B = 112.5\ TB$

> - Total storage required for one day =
> $0.1875TB + 15TB + 112.5TB \approx 128TB$
> - Storage required for one year = $365 \times 128TB = 46.72PB$

The total storage required by Twitter in a year

Bandwidth requirements

In order to estimate the bandwidth requirements for a service, we use the following steps:

1. Estimate the daily amount of incoming data to the service.
2. Estimate the daily amount of outgoing data from the service.
3. Estimate the bandwidth in Gbps (gigabits per second) by dividing the incoming and outgoing data by the number of seconds in a day.

Incoming traffic: Let's continue from our previous example of Twitter, which requires 128 TBs of storage each day. Therefore, the incoming traffic should support the following bandwidth per second:

$$\frac{128 \times 10^{12}}{86400} \times 8 \approx 12 Gbps$$

> **Note:** We multiply by 8 in order to convert Bytes(B) into bits(b) because bandwidth is measured in bits per second.

Outgoing traffic: Assume that a single user views 50 tweets in a day. Considering the same ratio of five percent and 10 percent for videos and images, respectively, for the 50 tweets, 2.5 tweets will contain video content whereas five tweets will contain an image. Considering that there are 250 M active daily users, we come to the following estimations:

Estimating Bandwidth Requirements

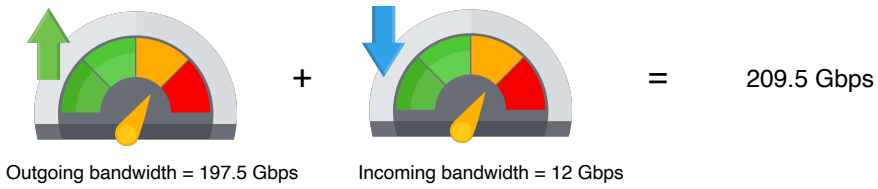
Daily active users (DAU)	250	M
Daily tweets viewed	50	per user
Tweets viewed / second	*f* 145	K
Bandwidth required for tweets	*f* 0.3	Gbps
Bandwidth required for images	*f* 23.2	Gbps
Bandwidth required for videos	*f* 174	Gbps
Total bandwidth	*f* 197.5	Gbps

Hide Calculations

- $250M \times 50\ tweets = 12.5$ billion tweets are viewed per day
- Tweets viewed per second: $\frac{12.5}{86400} = 145 \times 10^3$ tweets per second
- Bandwidth for tweet: $145 \times 10^3 \times 250 \times 8\ bits \approx 0.3\ Gbps$ (One tweet is equal to 250 bytes).
- Bandwidth of 10 percent images in tweets in a second: $145 \times 10^3 \times \frac{10}{100} \times 200 \times 10^3 \times 8\ bits = 23.2\ Gbps$
- Bandwidth for 5 percent video tweets in a second: $145 \times 10^3 \times \frac{5}{100} \times 3 \times 10^6 \times 8\ bits = 174\ Gbps$

The total outgoing traffic required will be equal to:
$0.3 + 23.2 + 174 \approx 197.5\ Gbps$.

> Twitter will need a total of $12\ Gbps$ of incoming traffic and $197.5\ Gbps$ of outgoing, assuming that the uploaded content is not compressed. Total bandwidth requirements = $12 + 197.5 = 209.5\ Gbps$.



The total bandwidth required by Twitter

Back
Mark As Completed
Next
Put Back-of-the-envelope Numbers ...
Introduction to Building Blocks for M...