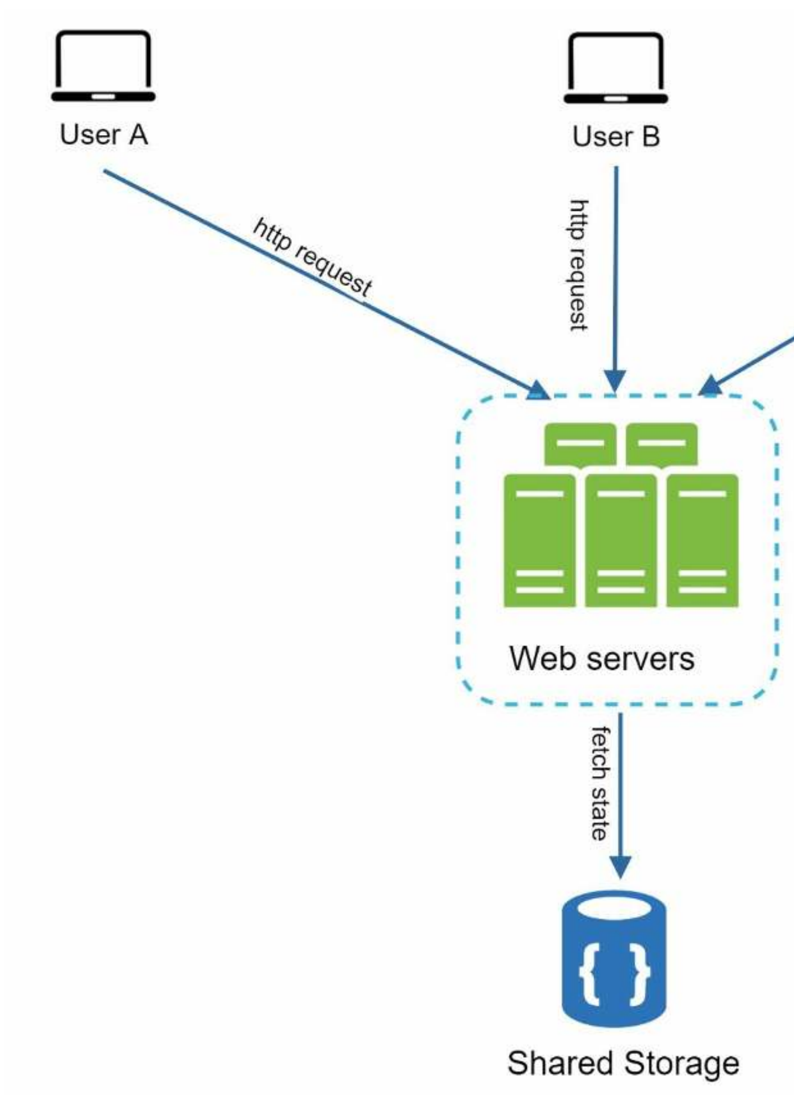


Figure 1-13

In this stateless architecture, HTTP requests from users can be sent to any web servers, which fetch state data from a shared data store. State data is stored in a shared data store and kept out of web servers. A stateless system is simpler, more robust, and scalable.

Figure 1-14 shows the updated design with a stateless web tier.

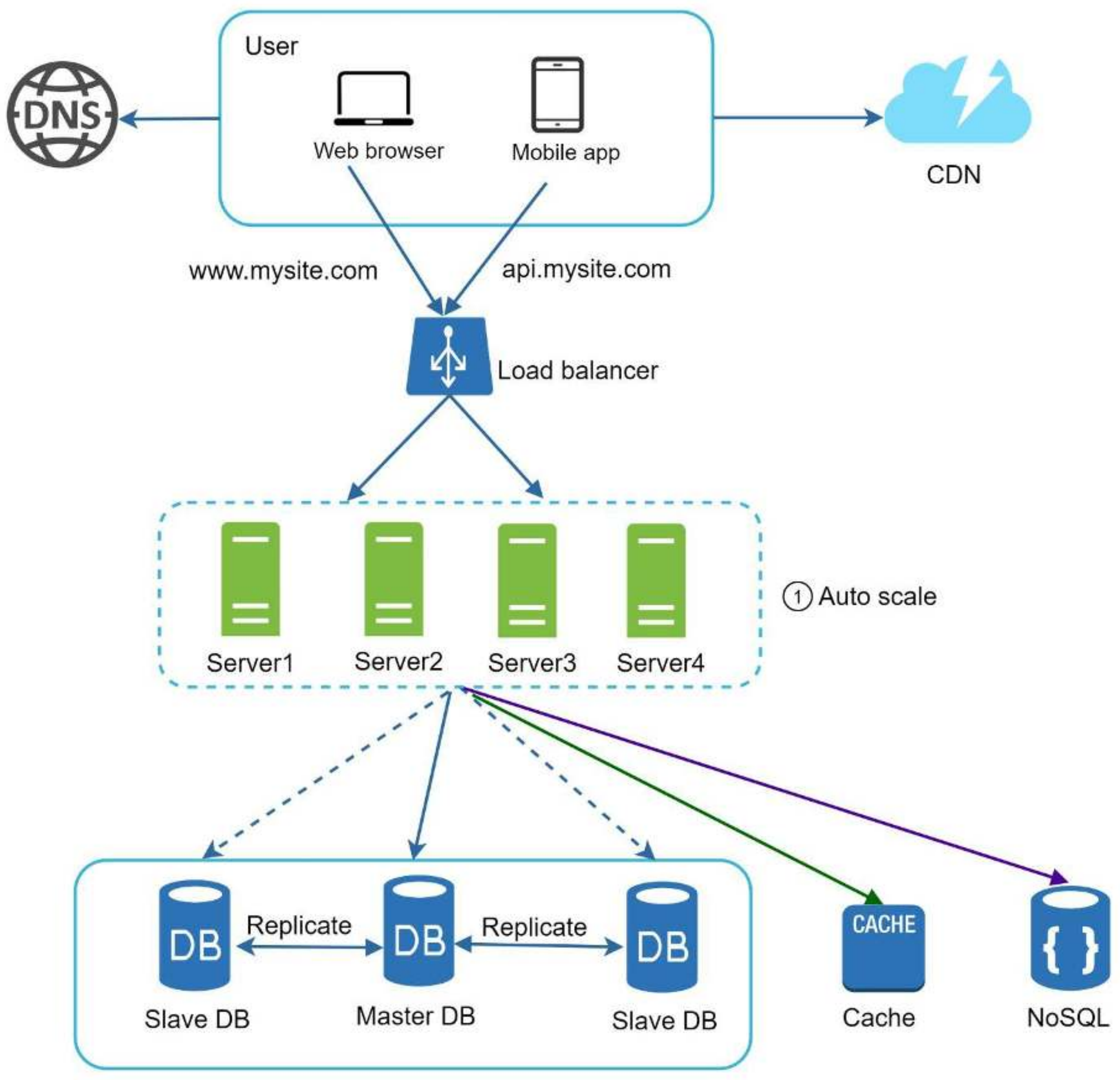


Figure 1-14

In Figure 1-14, we move the session data out of the web tier and store them in the persistent data store. The shared data store could be a relational database, Memcached/Redis, NoSQL, etc. The NoSQL data store is chosen as it is easy to scale. Autoscaling means adding or removing web servers automatically based on the traffic load. After the state data is removed out of web servers, auto-scaling of the web tier is easily achieved by adding or removing servers based on traffic load.

Your website grows rapidly and attracts a significant number of users internationally. To improve availability and provide a better user experience across wider geographical areas, supporting multiple data centers is crucial.

Data centers

Figure 1-15 shows an example setup with two data centers. In normal operation, users are geoDNS-routed, also known as geo-routed, to the closest data center, with a split traffic of *x%* in US-East and *(100 – x)%* in US-West. geoDNS is a DNS service that allows domain names to be resolved to IP addresses based on the location of a user.

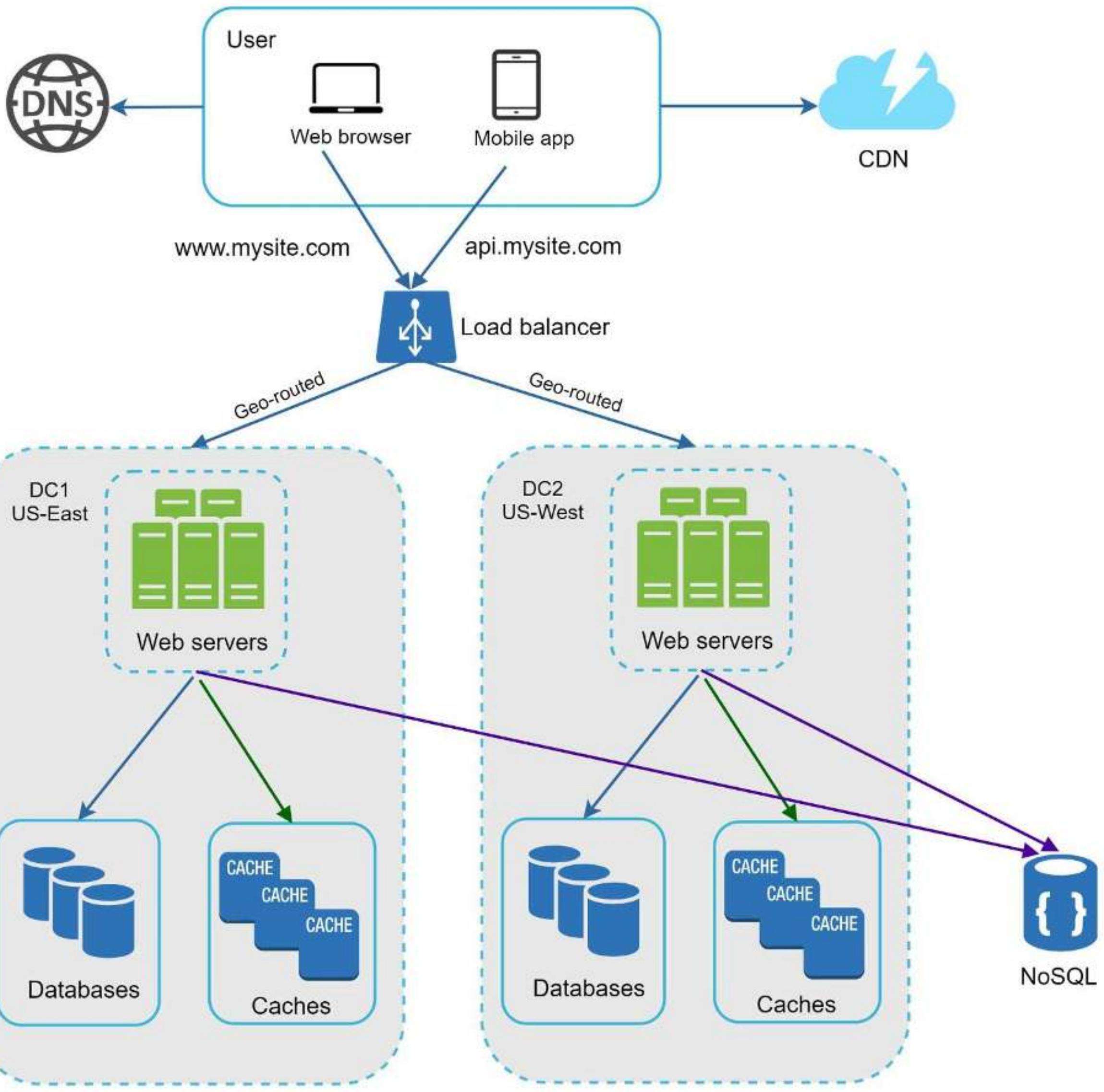


Figure 1-15

In the event of any significant data center outage, we direct all traffic to a healthy data center. In Figure 1-16, data center 2 (US-West) is offline, and 100% of the traffic is routed to data center 1 (US-East).

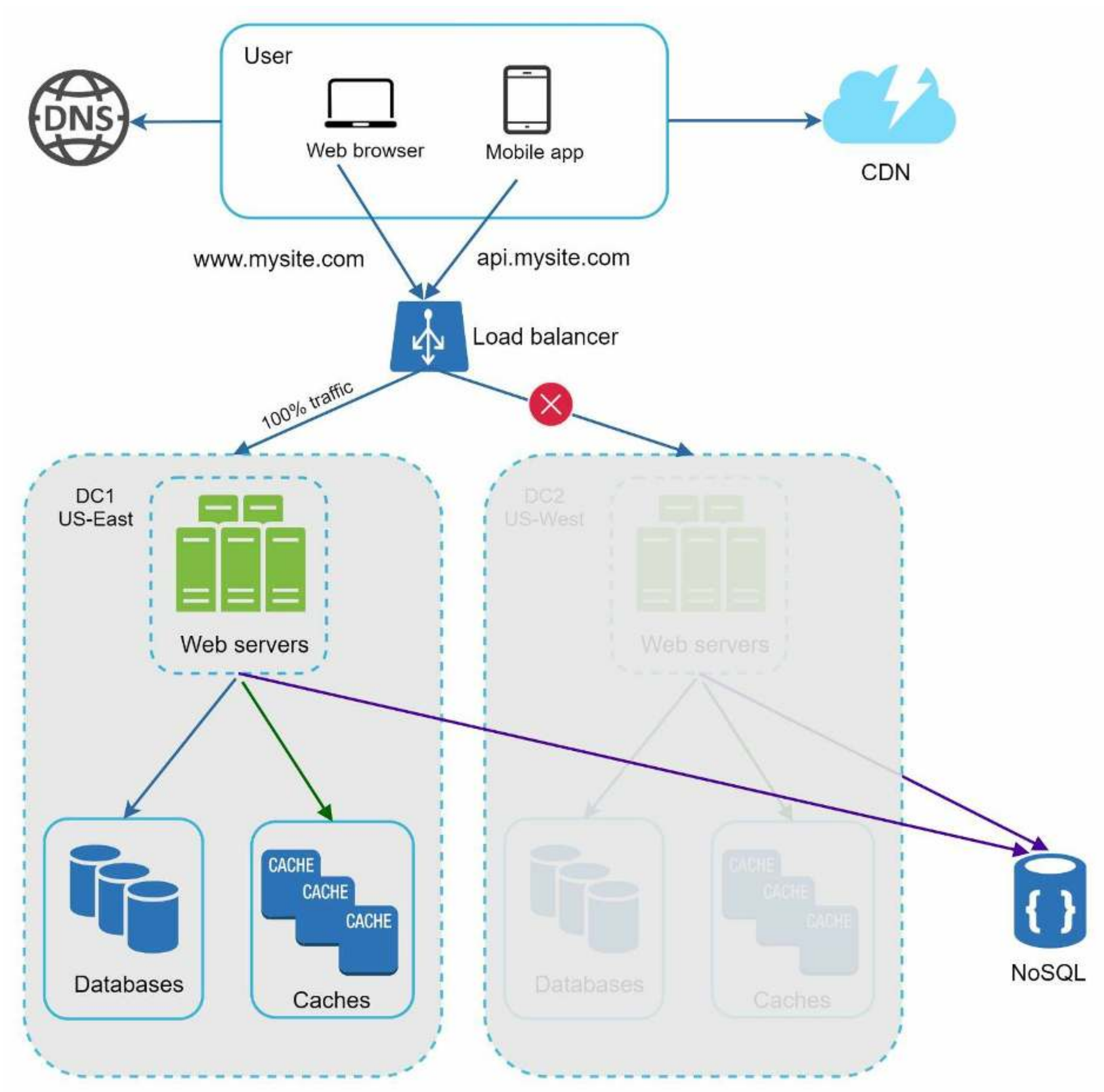


Figure 1-16

Several technical challenges must be resolved to achieve multi-data center setup:

• Traffic redirection: Effective tools are needed to direct traffic to the correct data center. GeoDNS can be used to direct traffic to the nearest data center depending on where a user is located.

• Data synchronization: Users from different regions could use different local databases or caches. In failover cases, traffic might be routed to a data center where data is unavailable. A common strategy is to replicate data across multiple data centers. A previous study shows how Netflix implements asynchronous multi-data center replication [11].

• Test and deployment: With multi-data center setup, it is important to test your website/application at different locations. Automated deployment tools are vital to keep services consistent through all the data centers [11].

To further scale our system, we need to decouple different components of the system so they can be scaled independently. Messaging queue is a key strategy employed by many real-world distributed systems to solve this problem.

Message queue

A message queue is a durable component, stored in memory, that supports asynchronous communication. It serves as a buffer and distributes asynchronous requests. The basic architecture of a message queue is simple. Input services, called producers/publishers, create messages, and publish them to a message queue. Other services or servers, called consumers/subscribers, connect to the queue, and perform actions defined by the messages. The model is shown in Figure 1-17.

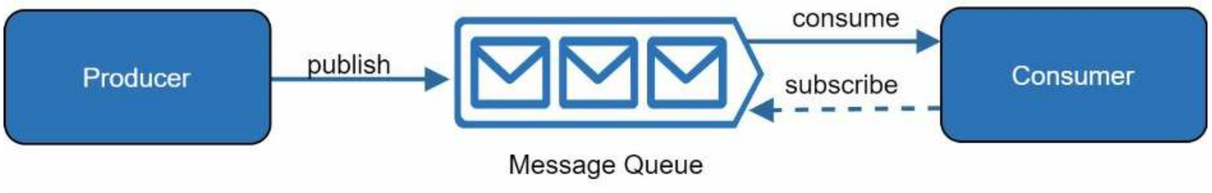


Figure 1-17

Decoupling makes the message queue a preferred architecture for building a scalable and reliable application. With the message queue, the producer can post a message to the queue when the consumer is unavailable to process it. The consumer can read messages from the queue even when the producer is unavailable.

Consider the following use case: your application supports photo customization, including cropping, sharpening, blurring, etc. Those customization tasks take time to complete. In Figure 1-18, web servers publish photo processing jobs to the message queue. Photo processing workers pick up jobs from the message queue and asynchronously perform photo customization tasks. The producer and the consumer can be scaled independently. When the size of the queue becomes large, more workers are added to reduce the processing time. However, if the queue is empty most of the time, the number of workers can be reduced.

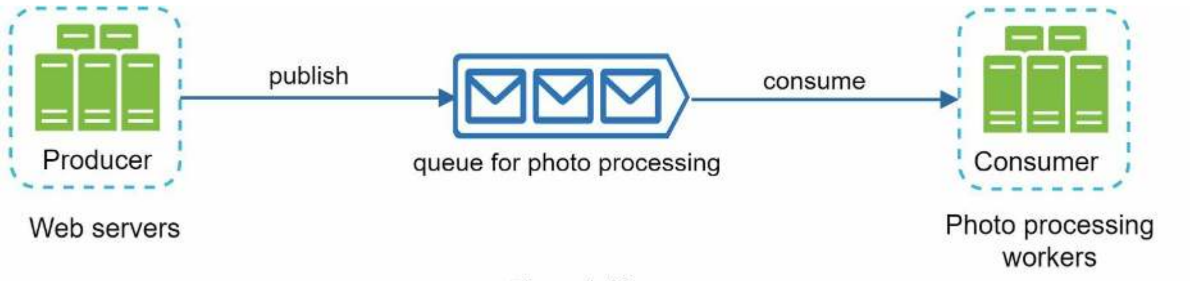


Figure 1-18

Logging, metrics, automation

When working with a small website that runs on a few servers, logging, metrics, and automation support are good practices but not a necessity. However, now that your site has grown to serve a large business, investing in those tools is essential.

Logging: Monitoring error logs is important because it helps to identify errors and problems in the system. You can monitor error logs at per server level or use tools to aggregate them to a centralized service for easy search and viewing.

Metrics: Collecting different types of metrics help us to gain business insights and understand the health status of the system. Some of the following metrics are useful:

- Host level metrics: CPU, Memory, disk I/O, etc.
- Aggregated level metrics: for example, the performance of the entire database tier, cache tier, etc.
- Key business metrics: daily active users, retention, revenue, etc.

Automation: When a system gets big and complex, we need to build or leverage automation tools to improve productivity. Continuous integration is a good practice, in which each code check-in is verified through automation, allowing teams to detect problems early. Besides, automating your build, test, deploy process, etc. could improve developer productivity significantly.

Adding message queues and different tools

Figure 1-19 shows the updated design. Due to the space constraint, only one data center is shown in the figure.

1. The design includes a message queue, which helps to make the system more loosely coupled and failure resilient.
2. Logging, monitoring, metrics, and automation tools are included.

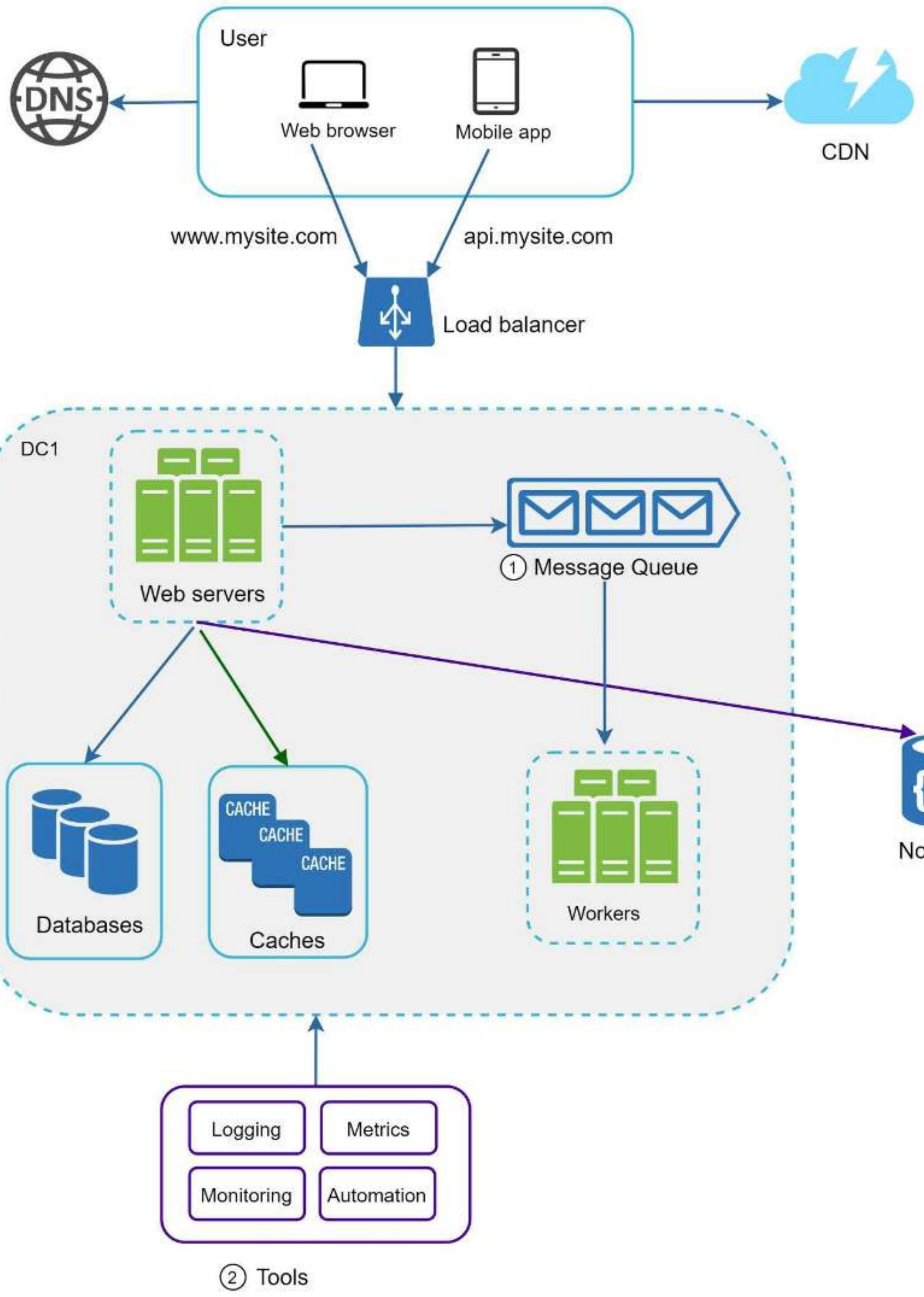


Figure 1-19

As the data grows every day, your database gets more overloaded. It is time to scale the data tier.

Database scaling

There are two broad approaches for database scaling: vertical scaling and horizontal scaling.

Vertical scaling

Vertical scaling, also known as scaling up, is the scaling by adding more power (CPU, RAM, DISK, etc.) to an existing machine. There are some powerful database servers. According to Amazon Relational Database Service (RDS) [12], you can get a database server with 24 TB of RAM. This kind of powerful database server could store and handle lots of data. For example, stackoverflow.com in 2013 had over 10 million monthly unique visitors, but it only had 1 master database [13]. However, vertical scaling comes with some serious drawbacks:

- You can add more CPU, RAM, etc. to your database server, but there are hardware limits. If you have a large user base, a single server is not enough.
- Greater risk of single point of failures.
- The overall cost of vertical scaling is high. Powerful servers are much more expensive.

Horizontal scaling

Horizontal scaling, also known as sharding, is the practice of adding more servers. Figure 1-20 compares vertical scaling with horizontal scaling.

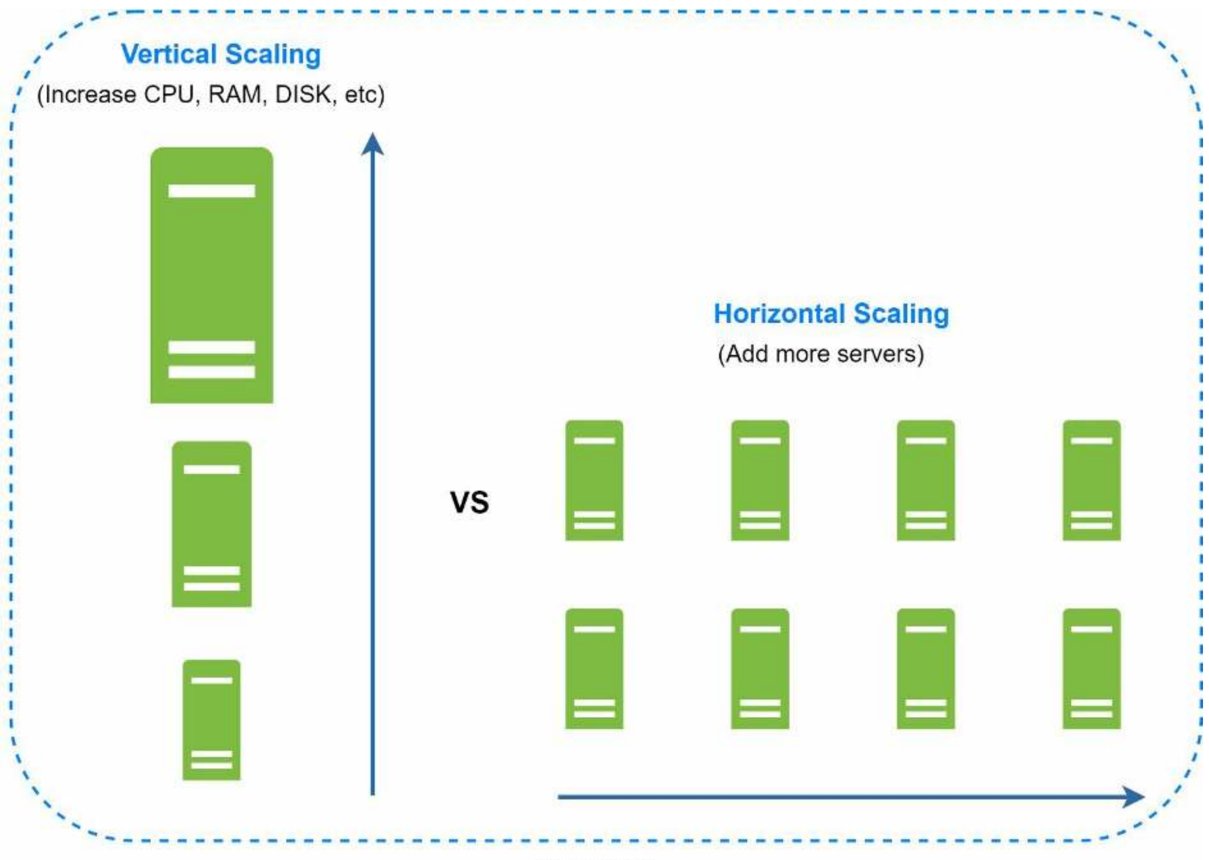


Figure 1-20

Sharding separates large databases into smaller, more easily managed parts called shards. Each shard shares the same schema, though the actual data on each shard is unique to the shard.

Figure 1-21 shows an example of sharded databases. User data is allocated to a database server based on user IDs. Anytime you access data, a hash function is used to find the corresponding shard. In our example, *user_id % 4* is used as the hash function. If the result equals to 0, shard 0 is used to store and fetch data. If the result equals to 1, shard 1 is used. The same logic applies to other shards.

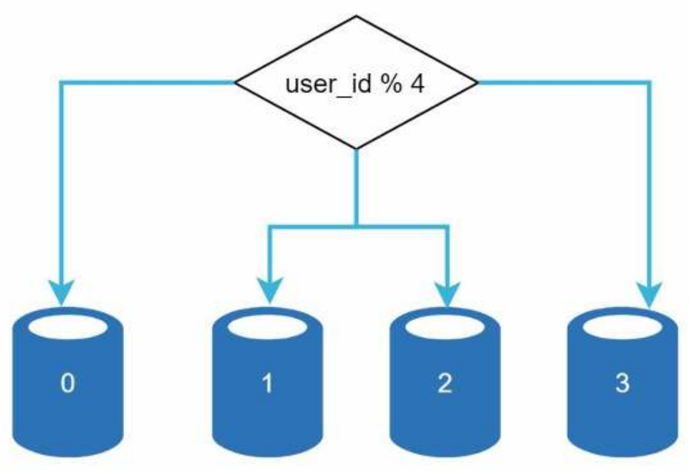


Figure 1-21

Figure 1-22 shows the user table in sharded databases.

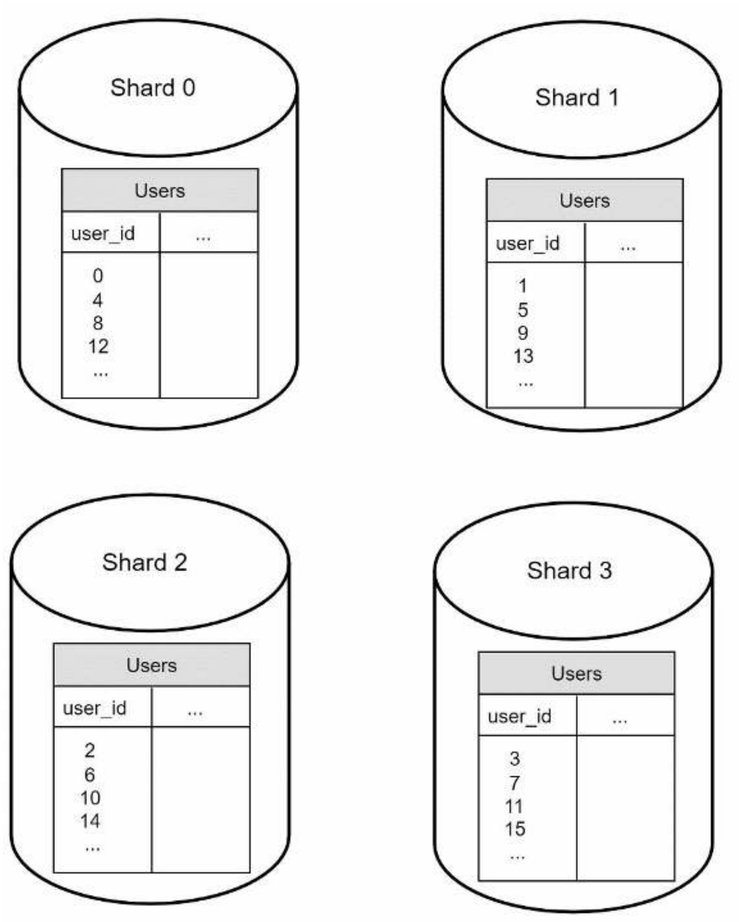


Figure 1-22

The most important factor to consider when implementing a sharding strategy is the choice of the sharding key. Sharding key (known as a partition key) consists of one or more columns that determine how data is distributed. As shown in Figure 1-22, *"user_id"* is the sharding key. A sharding key allows you to retrieve and modify data efficiently by routing database queries to the correct database. When choosing a sharding key, one of the most important criteria is to choose a key that can evenly distributed data.

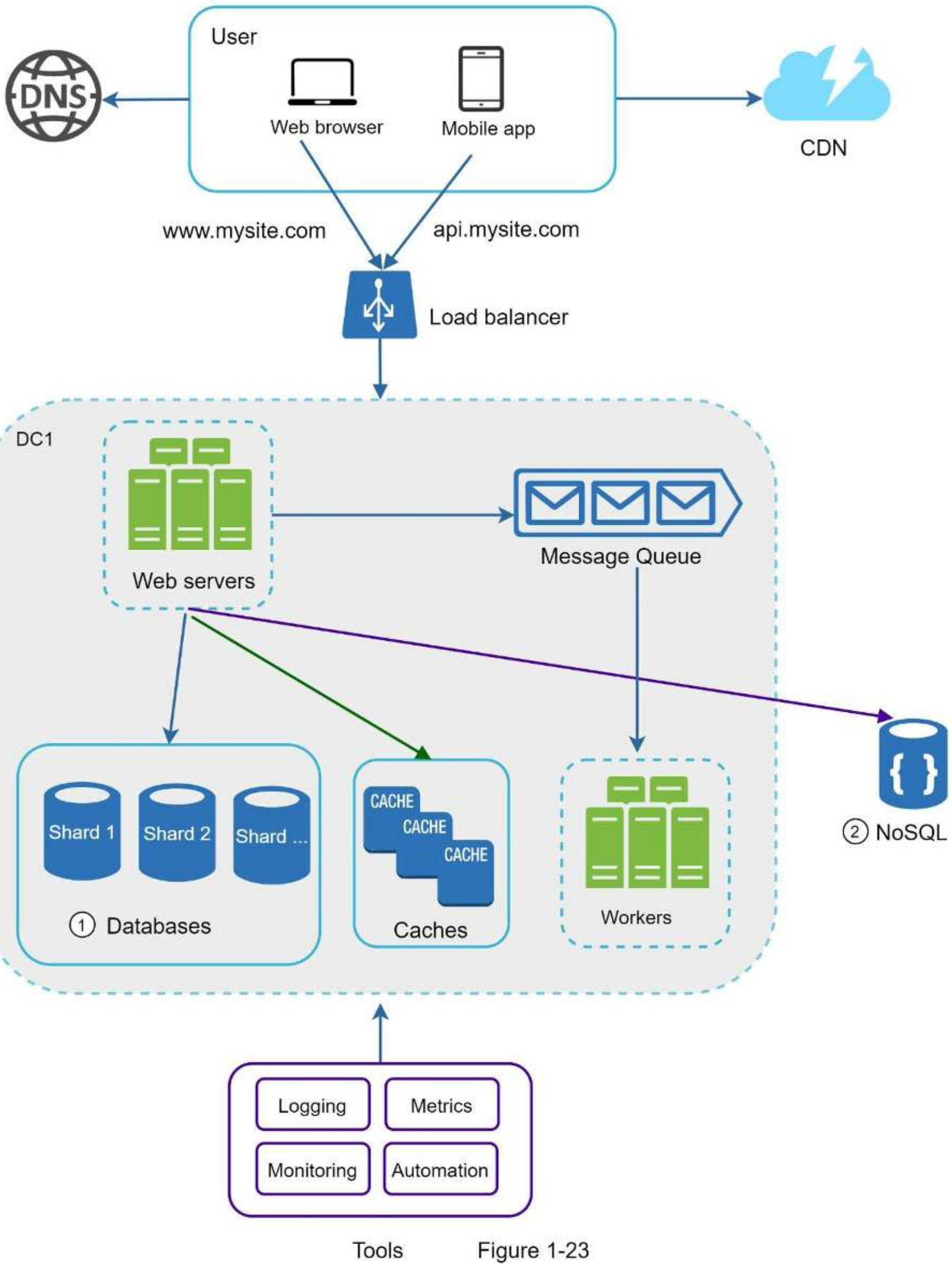
Sharding is a great technique to scale the database but it is far from a perfect solution. It introduces complexities and new challenges to the system:

Resharding data: Resharding data is needed when 1) a single shard could no longer hold more data due to rapid growth. 2) Certain shards might experience shard exhaustion faster than others due to uneven data distribution. When shard exhaustion happens, it requires updating the sharding function and moving data around. Consistent hashing, which will be discussed in Chapter 5, is a commonly used technique to solve this problem.

Celebrity problem: This is also called a hotspot key problem. Excessive access to a specific shard could cause server overload. Imagine data for Katy Perry, Justin Bieber, and Lady Gaga all end up on the same shard. For social applications, that shard will be overwhelmed with read operations. To solve this problem, we may need to allocate a shard for each celebrity. Each shard might even require further partition.

Join and de-normalization: Once a database has been sharded across multiple servers, it is hard to perform join operations across database shards. A common workaround is to de-normalize the database so that queries can be performed in a single table.

In Figure 1-23, we shard databases to support rapidly increasing data traffic. At the same time, some of the non-relational functionalities are moved to a NoSQL data store to reduce the database load. Here is an article that covers many use cases of NoSQL [14].



User
Web browser
Mobile app
DNS
CDN
www.mysite.com
api.mysite.com
Load balancer
DC1
Web servers
Message Queue
Shard 1
Shard 2
Shard ...
① Databases
CACHE
CACHE
CACHE
Caches
Workers
② NoSQL
Logging
Metrics
Monitoring
Automation
Tools

Figure 1-23

Millions of users and beyond

Scaling a system is an iterative process. Iterating on what we have learned in this chapter could get us far. More fine-tuning and new strategies are needed to scale beyond millions of users. For example, you might need to optimize your system and decouple the system to even smaller services. All the techniques learned in this chapter should provide a good foundation to tackle new challenges. To conclude this chapter, we provide a summary of how we scale our system to support millions of users:

- Keep web tier stateless
- Build redundancy at every tier
- Cache data as much as you can
- Support multiple data centers
- Host static assets in CDN
- Scale your data tier by sharding
- Split tiers into individual services
- Monitor your system and use automation tools

Congratulations on getting this far! Now give yourself a pat on the back. Good job!

Reference materials

[1] Hypertext Transfer Protocol: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[2] Should you go Beyond Relational Databases?: https://blog.teamtreehouse.com/should-you-go-beyond-relational-databases

[3] Replication: https://en.wikipedia.org/wiki/Replication_(computing)

[4] Multi-master replication: https://en.wikipedia.org/wiki/Multi-master_replication

[5] NDB Cluster Replication: Multi-Master and Circular Replication: https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-replication-multi-master.html

[6] Caching Strategies and How to Choose the Right One: https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/

[7] R. Nishtala, "Facebook, Scaling Memcache at," 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’13).

[8] Single point of failure: https://en.wikipedia.org/wiki/Single_point_of_failure

[9] Amazon CloudFront Dynamic Content Delivery: https://aws.amazon.com/cloudfront/dynamic-content/

[10] Configure Sticky Sessions for Your Classic Load Balancer: https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-sticky-sessions.html

[11] Active-Active for Multi-Regional Resiliency: https://netflixtechblog.com/active-active-for-multi-regional-resiliency-c47719f6685b

[12] Amazon EC2 High Memory Instances: https://aws.amazon.com/ec2/instance-types/high-memory/

[13] What it takes to run Stack Overflow: http://nickcraver.com/blog/2013/11/22/what-it-takes-to-run-stack-overflow

[14] What The Heck Are You Actually Using NoSQL For: http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html

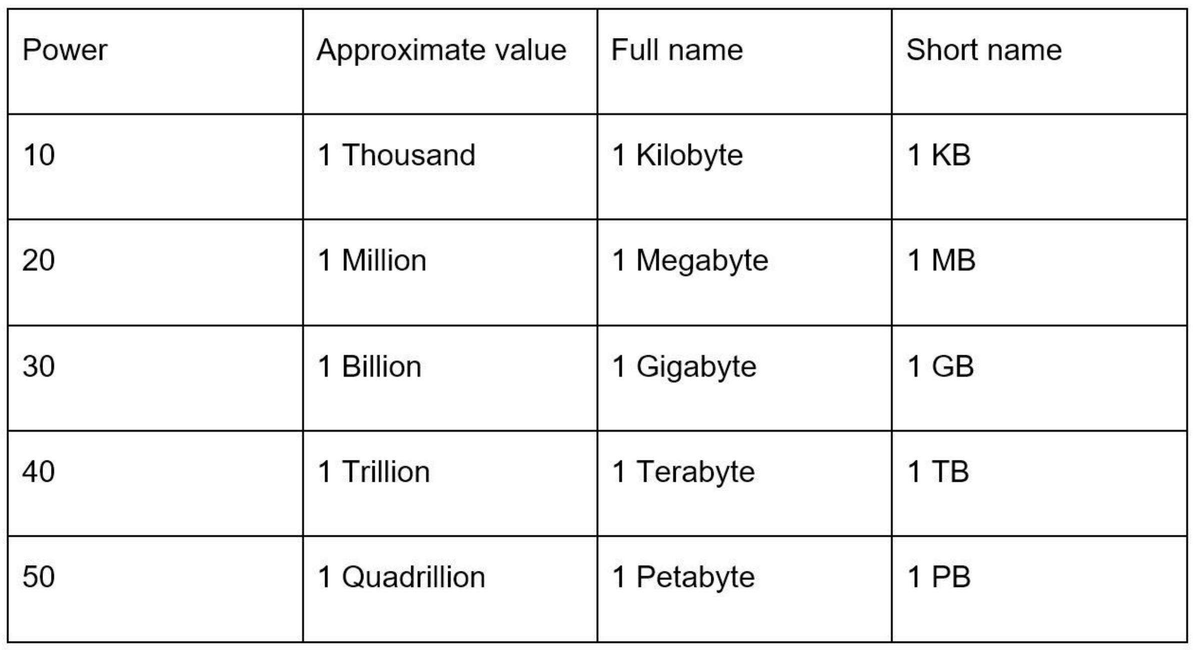
CHAPTER 2: BACK-OF-THE-ENVELOPE ESTIMATION

In a system design interview, sometimes you are asked to estimate system capacity or performance requirements using a back-of-the-envelope estimation. According to Jeff Dean, Google Senior Fellow, “back-of-the-envelope calculations are estimates you create using a combination of thought experiments and common performance numbers to get a good feel for which designs will meet your requirements” [1].

You need to have a good sense of scalability basics to effectively carry out back-of-the-envelope estimation. The following concepts should be well understood: power of two [2], latency numbers every programmer should know, and availability numbers.

Power of two

Although data volume can become enormous when dealing with distributed systems, calculation all boils down to the basics. To obtain correct calculations, it is critical to know the data volume unit using the power of 2. A byte is a sequence of 8 bits. An ASCII character uses one byte of memory (8 bits). Below is a table explaining the data volume unit (Table 2-1).

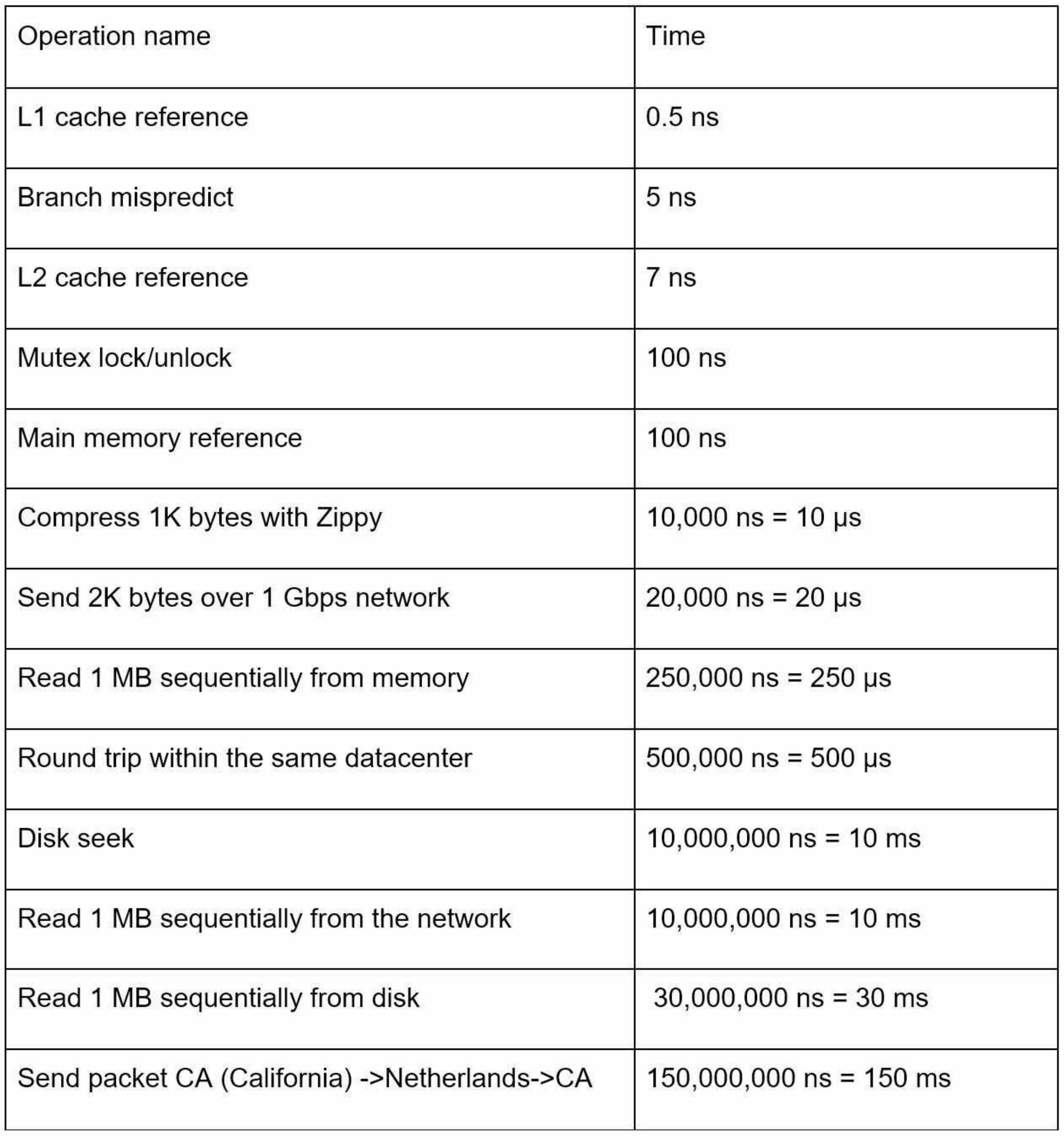


Power	Approximate value	Full name	Short name
10	1 Thousand	1 Kilobyte	1 KB
20	1 Million	1 Megabyte	1 MB
30	1 Billion	1 Gigabyte	1 GB
40	1 Trillion	1 Terabyte	1 TB
50	1 Quadrillion	1 Petabyte	1 PB

Table 2-1

Latency numbers every programmer should know

Dr. Dean from Google reveals the length of typical computer operations in 2010 [1]. Some numbers are outdated as computers become faster and more powerful. However, those numbers should still be able to give us an idea of the fastness and slowness of different computer operations.



Operation name	Time
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns = 10 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within the same datacenter	500,000 ns = 500 µs
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from the network	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	30,000,000 ns = 30 ms
Send packet CA (California) ->Netherlands->CA	150,000,000 ns = 150 ms

Table 2-2

Notes

----

ns = nanosecond, µs = microsecond, ms = millisecond
1 ns = 10^-9 seconds
1 µs= 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 µs = 1,000,000 ns

A Google software engineer built a tool to visualize Dr. Dean’s numbers. The tool also takes the time factor into consideration. Figures 2-1 shows the visualized latency numbers as of 2020 (source of figures: reference material [3]).

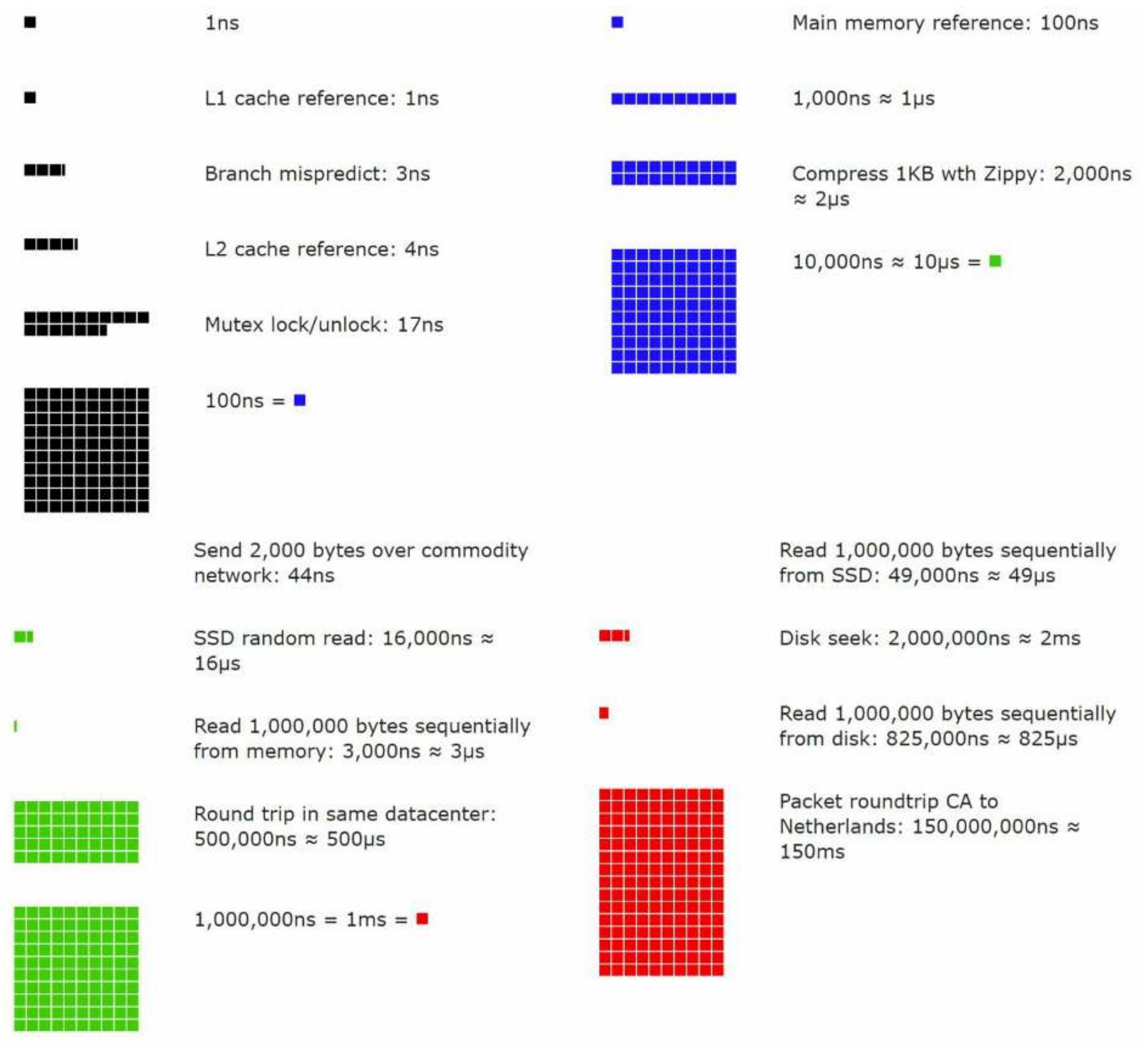


Figure 2-1

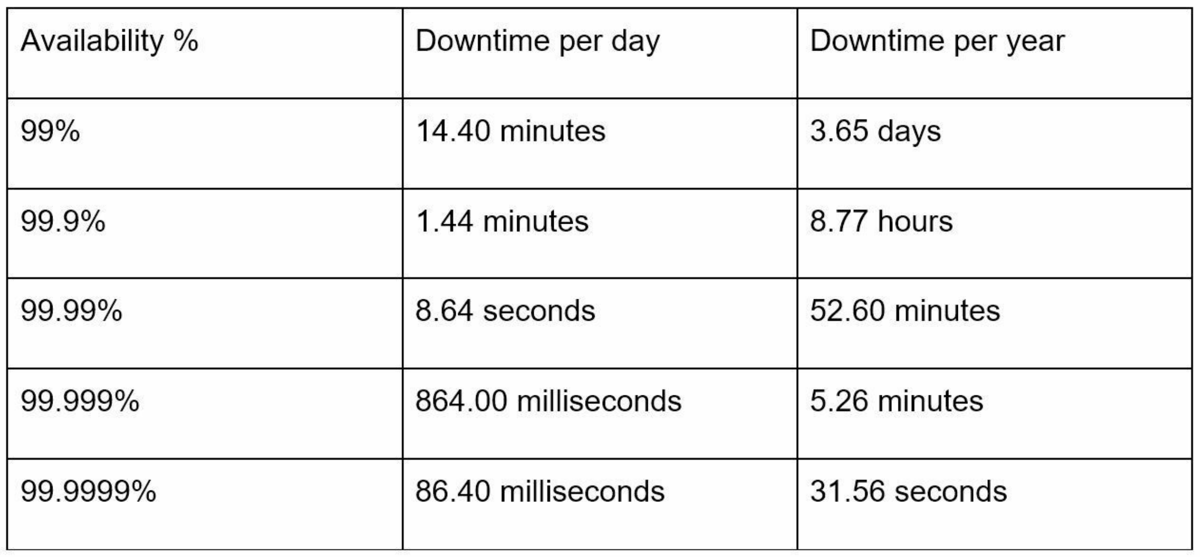
By analyzing the numbers in Figure 2-1, we get the following conclusions:

- Memory is fast but the disk is slow.
- Avoid disk seeks if possible.
- Simple compression algorithms are fast.
- Compress data before sending it over the internet if possible.
- Data centers are usually in different regions, and it takes time to send data between them.

Availability numbers

High availability is the ability of a system to be continuously operational for a desirably long period of time. High availability is measured as a percentage, with 100% means a service that has 0 downtime. Most services fall between 99% and 100%.

A service level agreement (SLA) is a commonly used term for service providers. This is an agreement between you (the service provider) and your customer, and this agreement formally defines the level of uptime your service will deliver. Cloud providers Amazon [4], Google [5] and Microsoft [6] set their SLAs at 99.9% or above. Uptime is traditionally measured in nines. The more the nines, the better. As shown in Table 2-3, the number of nines correlate to the expected system downtime.



Availability %	Downtime per day	Downtime per year
99%	14.40 minutes	3.65 days
99.9%	1.44 minutes	8.77 hours
99.99%	8.64 seconds	52.60 minutes
99.999%	864.00 milliseconds	5.26 minutes
99.9999%	86.40 milliseconds	31.56 seconds

Table 2-3

Example: Estimate Twitter QPS and storage requirements

Please note the following numbers are for this exercise only as they are not real numbers from Twitter.

Assumptions:

- 300 million monthly active users.
- 50% of users use Twitter daily.
- Users post 2 tweets per day on average.
- 10% of tweets contain media.
- Data is stored for 5 years.

Estimations:

Query per second (QPS) estimate:

- Daily active users (DAU) = 300 million * 50% = 150 million
- Tweets QPS = 150 million * 2 tweets / 24 hour / 3600 seconds = ~3500
- Peek QPS = 2 * QPS = ~7000

We will only estimate media storage here.

- Average tweet size:
  - tweet_id   64 bytes
  - text   140 bytes
  - media   1 MB
- Media storage: 150 million * 2 * 10% * 1 MB = 30 TB per day
- 5-year media storage: 30 TB * 365 * 5 = ~55 PB

Tips

Back-of-the-envelope estimation is all about the process. Solving the problem is more important than obtaining results. Interviewers may test your problem-solving skills. Here are a few tips to follow:

- Rounding and Approximation. It is difficult to perform complicated math operations during the interview. For example, what is the result of “99987 / 9.1”? There is no need to spend valuable time to solve complicated math problems. Precision is not expected. Use round numbers and approximation to your advantage. The division question can be simplified as follows: “100,000 / 10”.
- Write down your assumptions. It is a good idea to write down your assumptions to be referenced later.
- Label your units. When you write down “5”, does it mean 5 KB or 5 MB? You might confuse yourself with this. Write down the units because “5 MB” helps to remove ambiguity.
- Commonly asked back-of-the-envelope estimations: QPS, peak QPS, storage, cache, number of servers, etc. You can practice these calculations when preparing for an interview. Practice makes perfect.

Congratulations on getting this far! Now give yourself a pat on the back. Good job!