Reference materials

[1] J. Dean.Google Pro Tip: Use Back-Of-The-Envelope-Calculations To Choose The Best Design:

http://highscalability.com/blog/2011/1/26/google-pro-tip-use-back-of-the-envelope-calculations-to-choo.html

[2] System design primer: https://github.com/donnemartin/system-design-primer

[3] Latency Numbers Every Programmer Should Know:

https://colin-scott.github.io/personal_website/research/interactive_latency.html

[4] Amazon Compute Service Level Agreement:

https://aws.amazon.com/compute/sla/

[5] Compute Engine Service Level Agreement (SLA):

https://cloud.google.com/compute/sla

[6] SLA summary for Azure services: https://azure.microsoft.com/en-us/support/legal/sla/summary/

CHAPTER 3: A FRAMEWORK FOR SYSTEM DESIGN INTERVIEWS

You have just landed a coveted on-site interview at your dream company. The hiring coordinator sends you a schedule for that day. Scanning down the list, you feel pretty good about it until your eyes land on this interview session - System Design Interview.

System design interviews are often intimidating. It could be as vague as “designing a well-known product X?”. The questions are ambiguous and seem unreasonably broad. Your weariness is understandable. After all, how could anyone design a popular product in an hour that has taken hundreds if not thousands of engineers to build?

The good news is that no one expects you to. Real-world system design is extremely complicated. For example, Google search is deceptively simple; however, the amount of technology that underpins that simplicity is truly astonishing. If no one expects you to design a real-world system in an hour, what is the benefit of a system design interview?

The system design interview simulates real-life problem solving where two co-workers collaborate on an ambiguous problem and come up with a solution that meets their goals. The problem is open-ended, and there is no perfect answer. The final design is less important compared to the work you put in the design process. This allows you to demonstrate your design skill, defend your design choices, and respond to feedback in a constructive manner.

Let us flip the table and consider what goes through the interviewer’s head as she walks into the conference room to meet you. The primary goal of the interviewer is to accurately assess your abilities. The last thing she wants is to give an inconclusive evaluation because the session has gone poorly and there are not enough signals. What is an interviewer looking for in a system design interview?

Many think that system design interview is all about a person's technical design skills. It is much more than that. An effective system design interview gives strong signals about a person's ability to collaborate, to work under pressure, and to resolve ambiguity constructively. The ability to ask good questions is also an essential skill, and many interviewers specifically look for this skill.

A good interviewer also looks for red flags. Over-engineering is a real disease of many engineers as they delight in design purity and ignore tradeoffs. They are often unaware of the compounding costs of over-engineered systems, and many companies pay a high price for that ignorance. You certainly do not want to demonstrate this tendency in a system design interview. Other red flags include narrow mindedness, stubbornness, etc.

In this chapter, we will go over some useful tips and introduce a simple and effective framework to solve system design interview problems.

A 4-step process for effective system design interview

Every system design interview is different. A great system design interview is open-ended and there is no one-size-fits-all solution. However, there are steps and common ground to cover in every system design interview.

Step 1 - Understand the problem and establish design scope

"Why did the tiger roar?"

A hand shot up in the back of the class.

"Yes, Jimmy?", the teacher responded.

"Because he was HUNGRY".

"Very good Jimmy."

Throughout his childhood, Jimmy has always been the first to answer questions in the class. Whenever the teacher asks a question, there is always a kid in the classroom who loves to take a crack at the question, no matter if he knows the answer or not. That is Jimmy.

Jimmy is an ace student. He takes pride in knowing all the answers fast. In exams, he is usually the first person to finish the questions. He is a teacher's top choice for any academic competition.

DON'T be like Jimmy.

In a system design interview, giving out an answer quickly without thinking gives you no bonus points. Answering without a thorough understanding of the requirements is a huge red flag as the interview is not a trivia contest. There is no right answer.

So, do not jump right in to give a solution. Slow down. Think deeply and ask questions to clarify requirements and assumptions. This is extremely important.

As an engineer, we like to solve hard problems and jump into the final design; however, this approach is likely to lead you to design the wrong system. One of the most important skills as an engineer is to ask the right questions, make the proper assumptions, and gather all the information needed to build a system. So, do not be afraid to ask questions.

When you ask a question, the interviewer either answers your question directly or asks you to make your assumptions. If the latter happens, write down your assumptions on the whiteboard or paper. You might need them later.

What kind of questions to ask? Ask questions to understand the exact requirements. Here is a list of questions to help you get started:

- What specific features are we going to build?
- How many users does the product have?
- How fast does the company anticipate to scale up? What are the anticipated scales in 3 months, 6 months, and a year?
- What is the company’s technology stack? What existing services you might leverage to simplify the design?

Example

If you are asked to design a news feed system, you want to ask questions that help you clarify the requirements. The conversation between you and the interviewer might look like this:

Candidate: Is this a mobile app? Or a web app? Or both?

Interviewer: Both.

Candidate: What are the most important features for the product?
Interviewer: Ability to make a post and see friends' news feed.

Candidate: Is the news feed sorted in reverse chronological order or a particular order? The particular order means each post is given a different weight. For instance, posts from your close friends are more important than posts from a group.
Interviewer: To keep things simple, let us assume the feed is sorted by reverse chronological order.

Candidate: How many friends can a user have?
Interviewer: 5000

Candidate: What is the traffic volume?
Interviewer: 10 million daily active users (DAU)

Candidate: Can feed contain images, videos, or just text?
Interviewer: It can contain media files, including both images and videos.

Above are some sample questions that you can ask your interviewer. It is important to understand the requirements and clarify ambiguities

Step 2 - Propose high-level design and get buy-in

In this step, we aim to develop a high-level design and reach an agreement with the interviewer on the design. It is a great idea to collaborate with the interviewer during the process.

- Come up with an initial blueprint for the design. Ask for feedback. Treat your interviewer as a teammate and work together. Many good interviewers love to talk and get involved.
- Draw box diagrams with key components on the whiteboard or paper. This might include clients (mobile/web), APIs, web servers, data stores, cache, CDN, message queue, etc.
- Do back-of-the-envelope calculations to evaluate if your blueprint fits the scale constraints. Think out loud. Communicate with your interviewer if back-of-the-envelope is necessary before diving into it.

If possible, go through a few concrete use cases. This will help you frame the high-level design. It is also likely that the use cases would help you discover edge cases you have not yet considered.

Should we include API endpoints and database schema here? This depends on the problem. For large design problems like "Design Google search engine", this is a bit of too low level. For a problem like designing the backend for a multi-player poker game, this is a fair game. Communicate with your interviewer.

Example

Let us use "Design a news feed system" to demonstrate how to approach the high-level design. Here you are not required to understand how the system actually works. All the details will be explained in Chapter 11.

At the high level, the design is divided into two flows: feed publishing and news feed building.

- Feed publishing: when a user publishes a post, corresponding data is written into cache/database, and the post will be populated into friends' news feed.

• Newsfeed building: the news feed is built by aggregating friends' posts in a reverse chronological order.

Figure 3-1 and Figure 3-2 present high-level designs for feed publishing and news feed building flows, respectively.

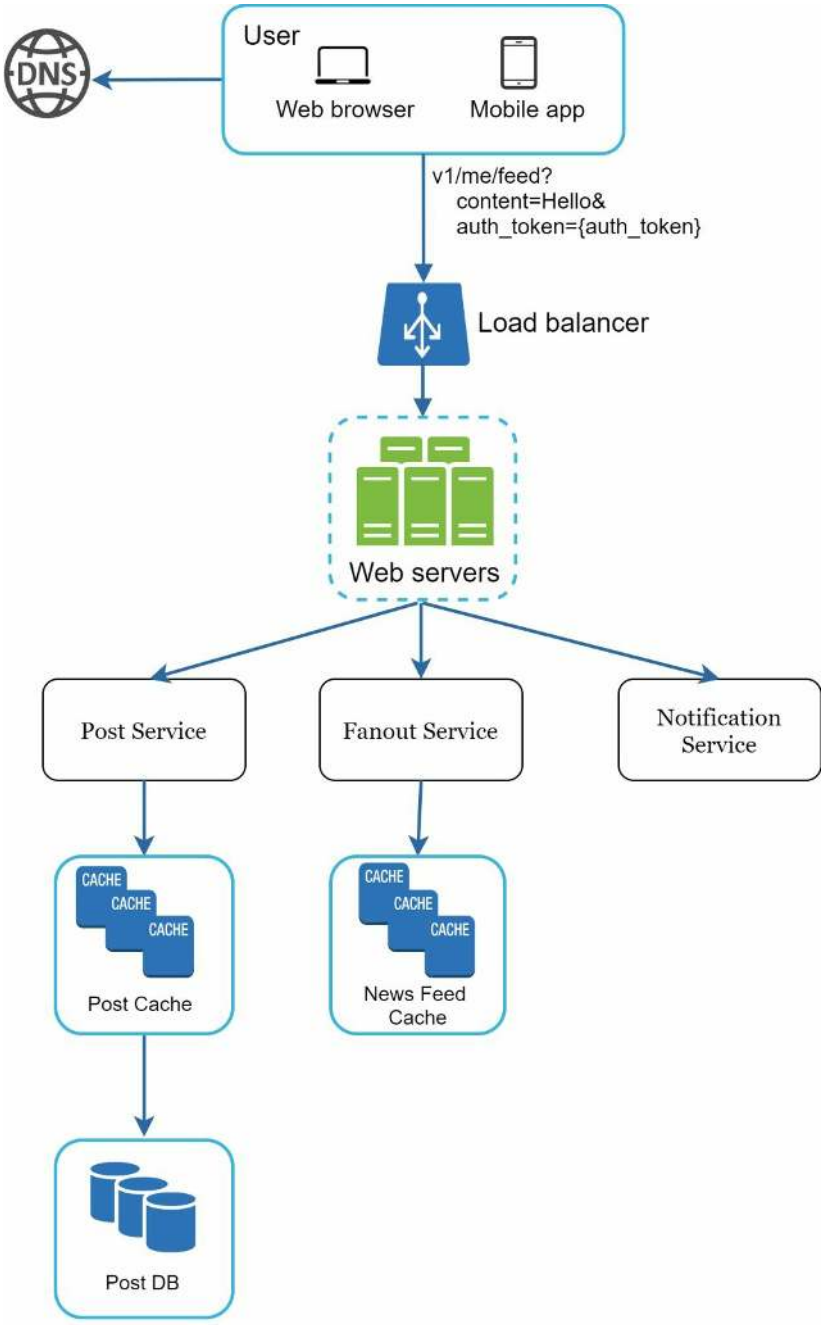

Figure 3-1

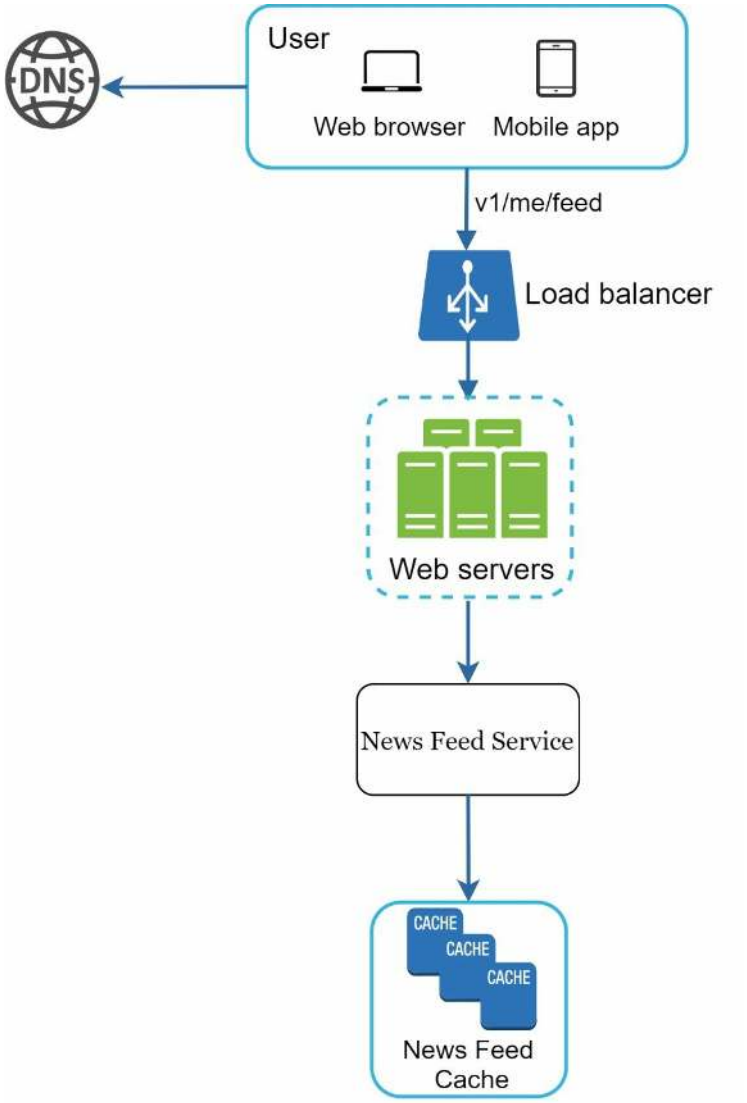

Figure 3-2

Step 3 - Design deep dive

At this step, you and your interviewer should have already achieved the following objectives:

- Agreed on the overall goals and feature scope
- Sketched out a high-level blueprint for the overall design
- Obtained feedback from your interviewer on the high-level design
- Had some initial ideas about areas to focus on in deep dive based on her feedback

You shall work with the interviewer to identify and prioritize components in the architecture. It is worth stressing that every interview is different. Sometimes, the interviewer may give off hints that she likes focusing on high-level design. Sometimes, for a senior candidate interview, the discussion could be on the system performance characteristics, likely focusing on the bottlenecks and resource estimations. In most cases, the interviewer may want you to dig into details of some system components. For URL shortener, it is interesting to dive into the hash function design that converts a long URL to a short one. For a chat system, how to reduce latency and how to support online/offline status are two interesting topics.

Time management is essential as it is easy to get carried away with minute details that do not demonstrate your abilities. You must be armed with signals to show your interviewer. Try not

to get into unnecessary details. For example, talking about the EdgeRank algorithm of Facebook feed ranking in detail is not ideal during a system design interview as this takes much precious time and does not prove your ability in designing a scalable system.

Example

At this point, we have discussed the high-level design for a news feed system, and the interviewer is happy with your proposal. Next, we will investigate two of the most important use cases:

1. Feed publishing
2. News feed retrieval

Figure 3-3 and Figure 3-4 show the detailed design for the two use cases, which will be explained in detail in Chapter 11.

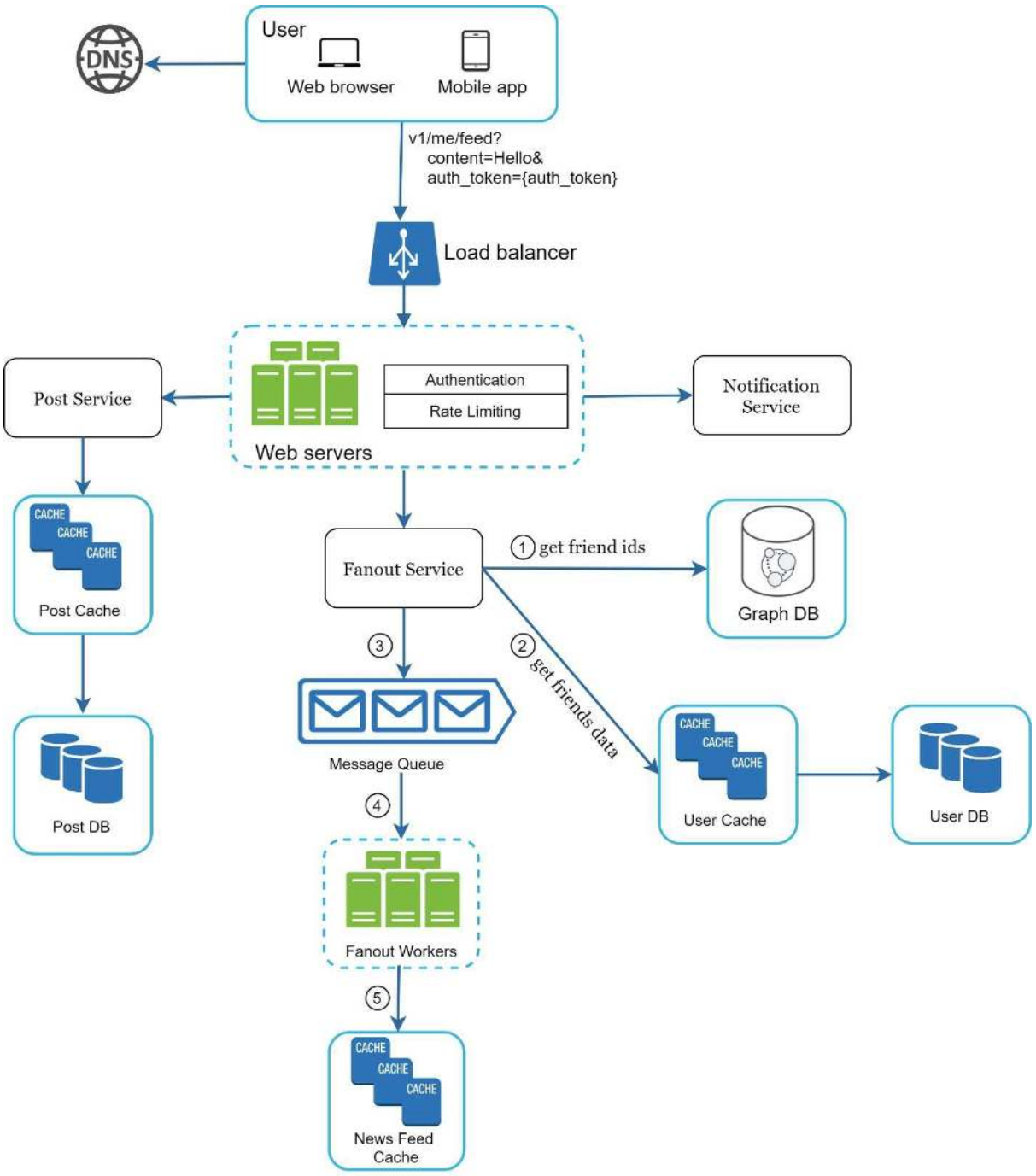
User
Web browser
Mobile app
DNS
v1/me/feed?
content=Hello&
auth_token={auth_token}
Load balancer
Authentication
Rate Limiting
Web servers
Post Service
Notification Service
CACHE
Post Cache
Post DB
Fanout Service
1 get friend ids
Graph DB
2 get friends data
3
Message Queue
4
Fanout Workers
5
User Cache
User DB
News Feed Cache

Figure 3-3

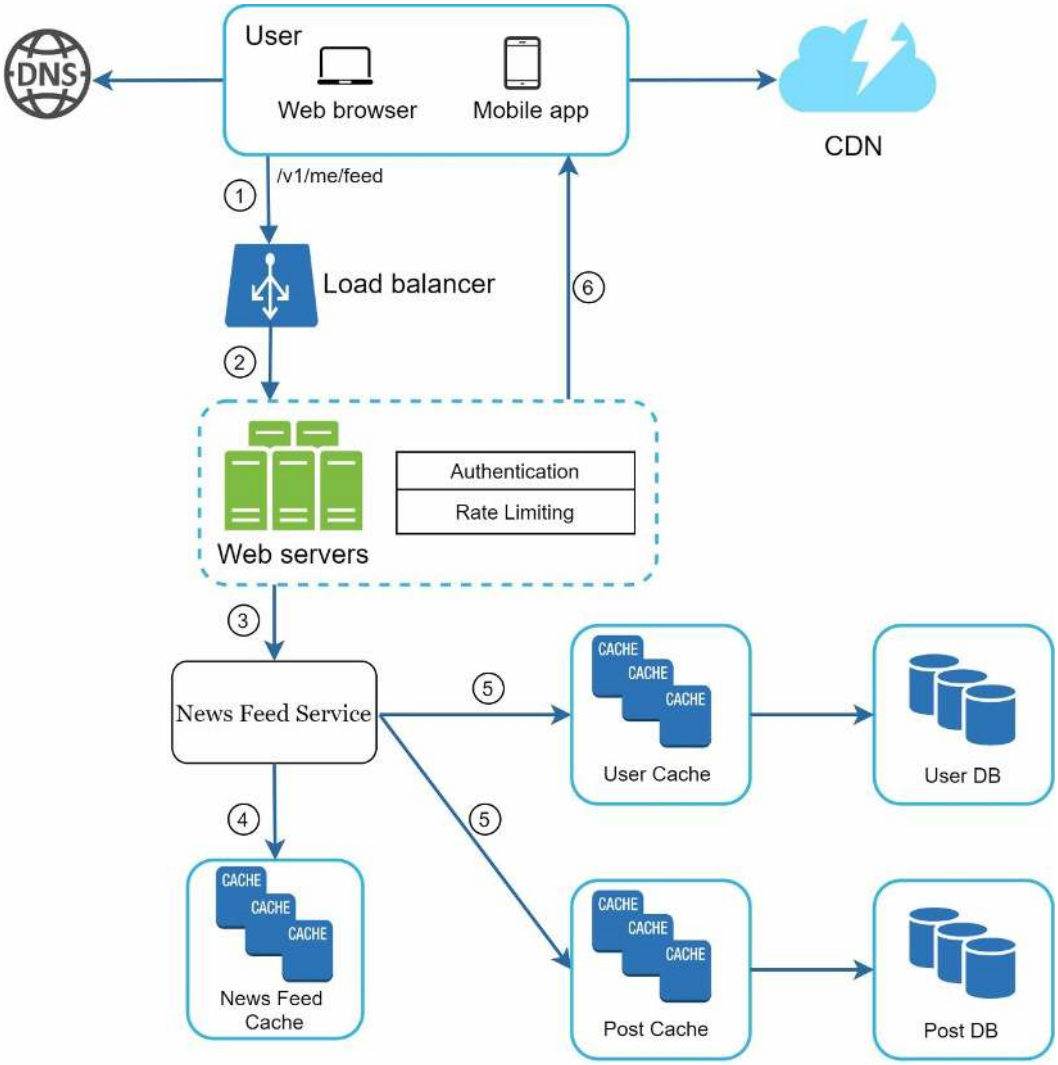

Figure 3-4

Step 4 - Wrap up

In this final step, the interviewer might ask you a few follow-up questions or give you the freedom to discuss other additional points. Here are a few directions to follow:

- The interviewer might want you to identify the system bottlenecks and discuss potential improvements. Never say your design is perfect and nothing can be improved. There is always something to improve upon. This is a great opportunity to show your critical thinking and leave a good final impression.
- It could be useful to give the interviewer a recap of your design. This is particularly important if you suggested a few solutions. Refreshing your interviewer's memory can be helpful after a long session.
- Error cases (server failure, network loss, etc.) are interesting to talk about.
- Operation issues are worth mentioning. How do you monitor metrics and error logs? How to roll out the system?
- How to handle the next scale curve is also an interesting topic. For example, if your current design supports 1 million users, what changes do you need to make to support 10 million users?
- Propose other refinements you need if you had more time.

To wrap up, we summarize a list of the Dos and Don’ts.

Dos

- Always ask for clarification. Do not assume your assumption is correct.
- Understand the requirements of the problem.
- There is neither the right answer nor the best answer. A solution designed to solve the problems of a young startup is different from that of an established company with millions of users. Make sure you understand the requirements.
- Let the interviewer know what you are thinking. Communicate with your interview.
- Suggest multiple approaches if possible.
- Once you agree with your interviewer on the blueprint, go into details on each component. Design the most critical components first.
- Bounce ideas off the interviewer. A good interviewer works with you as a teammate.
- Never give up.

Don’ts

- Don't be unprepared for typical interview questions.
- Don’t jump into a solution without clarifying the requirements and assumptions.
- Don’t go into too much detail on a single component in the beginning. Give the high-level design first then drills down.
- If you get stuck, don't hesitate to ask for hints.
- Again, communicate. Don't think in silence.
- Don’t think your interview is done once you give the design. You are not done until your interviewer says you are done. Ask for feedback early and often.

Time allocation on each step

System design interview questions are usually very broad, and 45 minutes or an hour is not enough to cover the entire design. Time management is essential. How much time should you spend on each step? The following is a very rough guide on distributing your time in a 45-minute interview session. Please remember this is a rough estimate, and the actual time distribution depends on the scope of the problem and the requirements from the interviewer.

Step 1 Understand the problem and establish design scope: 3 - 10 minutes

Step 2 Propose high-level design and get buy-in: 10 - 15 minutes

Step 3 Design deep dive: 10 - 25 minutes

Step 4 Wrap: 3 - 5 minutes

CHAPTER 4: DESIGN A RATE LIMITER

In a network system, a rate limiter is used to control the rate of traffic sent by a client or a service. In the HTTP world, a rate limiter limits the number of client requests allowed to be sent over a specified period. If the API request count exceeds the threshold defined by the rate limiter, all the excess calls are blocked. Here are a few examples:

- A user can write no more than 2 posts per second.
- You can create a maximum of 10 accounts per day from the same IP address.
- You can claim rewards no more than 5 times per week from the same device.

In this chapter, you are asked to design a rate limiter. Before starting the design, we first look at the benefits of using an API rate limiter:

- Prevent resource starvation caused by Denial of Service (DoS) attack [1]. Almost all APIs published by large tech companies enforce some form of rate limiting. For example, Twitter limits the number of tweets to 300 per 3 hours [2]. Google docs APIs have the following default limit: 300 per user per 60 seconds for read requests [3]. A rate limiter prevents DoS attacks, either intentional or unintentional, by blocking the excess calls.
- Reduce cost. Limiting excess requests means fewer servers and allocating more resources to high priority APIs. Rate limiting is extremely important for companies that use paid third party APIs. For example, you are charged on a per-call basis for the following external APIs: check credit, make a payment, retrieve health records, etc. Limiting the number of calls is essential to reduce costs.
- Prevent servers from being overloaded. To reduce server load, a rate limiter is used to filter out excess requests caused by bots or users' misbehavior.

Step 1 - Understand the problem and establish design scope

Rate limiting can be implemented using different algorithms, each with its pros and cons. The interactions between an interviewer and a candidate help to clarify the type of rate limiters we are trying to build.

Candidate: What kind of rate limiter are we going to design? Is it a client-side rate limiter or server-side API rate limiter?
Interviewer: Great question. We focus on the server-side API rate limiter.

Candidate: Does the rate limiter throttle API requests based on IP, the user ID, or other properties?
Interviewer: The rate limiter should be flexible enough to support different sets of throttle rules.

Candidate: What is the scale of the system? Is it built for a startup or a big company with a large user base?
Interviewer: The system must be able to handle a large number of requests.

Candidate: Will the system work in a distributed environment?
Interviewer: Yes.

Candidate: Is the rate limiter a separate service or should it be implemented in application code?
Interviewer: It is a design decision up to you.

Candidate: Do we need to inform users who are throttled?
Interviewer: Yes.

Requirements

Here is a summary of the requirements for the system:

- Accurately limit excessive requests.
- Low latency. The rate limiter should not slow down HTTP response time.
- Use as little memory as possible.
- Distributed rate limiting. The rate limiter can be shared across multiple servers or processes.
- Exception handling. Show clear exceptions to users when their requests are throttled.
- High fault tolerance. If there are any problems with the rate limiter (for example, a cache server goes offline), it does not affect the entire system.

Step 2 - Propose high-level design and get buy-in

Let us keep things simple and use a basic client and server model for communication.

Where to put the rate limiter?

Intuitively, you can implement a rate limiter at either the client or server-side.

- Client-side implementation. Generally speaking, client is an unreliable place to enforce rate limiting because client requests can easily be forged by malicious actors. Moreover, we might not have control over the client implementation.
- Server-side implementation. Figure 4-1 shows a rate limiter that is placed on the server-side.

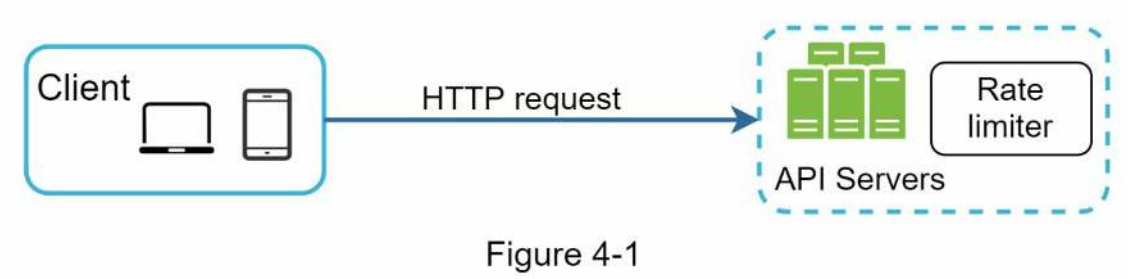

Figure 4-1

Besides the client and server-side implementations, there is an alternative way. Instead of putting a rate limiter at the API servers, we create a rate limiter middleware, which throttles requests to your APIs as shown in Figure 4-2.

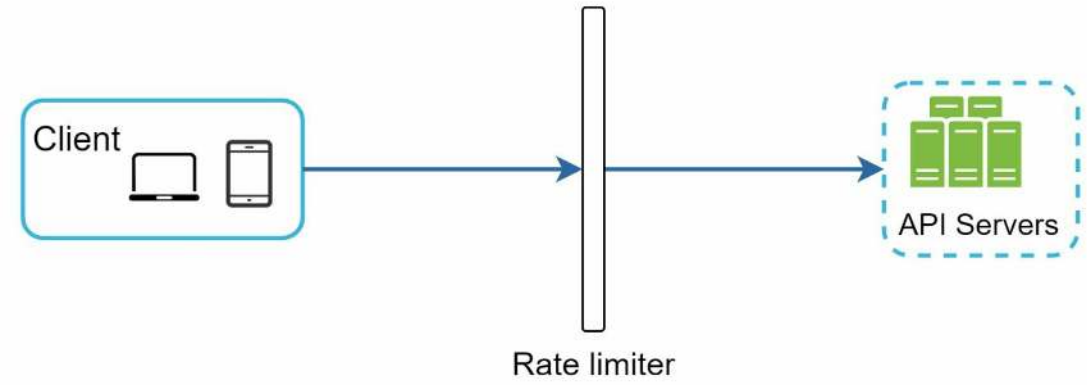

Figure 4-2

Let us use an example in Figure 4-3 to illustrate how rate limiting works in this design. Assume our API allows 2 requests per second, and a client sends 3 requests to the server within a second. The first two requests are routed to API servers. However, the rate limiter middleware throttles the third request and returns a HTTP status code 429. The HTTP 429 response status code indicates a user has sent too many requests.

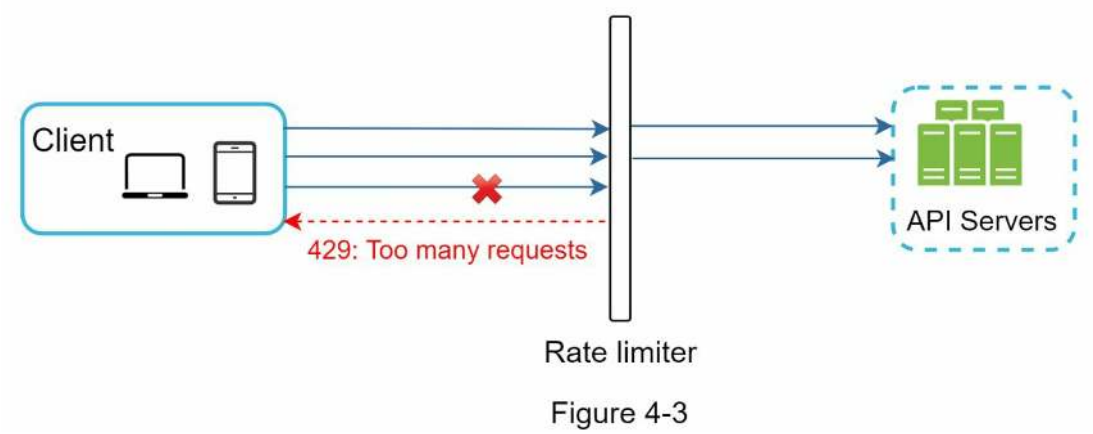

Figure 4-3

Cloud microservices [4] have become widely popular and rate limiting is usually implemented within a component called API gateway. API gateway is a fully managed service that supports rate limiting, SSL termination, authentication, IP whitelisting, servicing static content, etc. For now, we only need to know that the API gateway is a middleware that supports rate limiting.

While designing a rate limiter, an important question to ask ourselves is: where should the rater limiter be implemented, on the server-side or in a gateway? There is no absolute answer. It depends on your company's current technology stack, engineering resources, priorities, goals, etc. Here are a few general guidelines:

- Evaluate your current technology stack, such as programming language, cache service, etc. Make sure your current programming language is efficient to implement rate limiting on the server-side.
- Identify the rate limiting algorithm that fits your business needs. When you implement everything on the server-side, you have full control of the algorithm. However, your choice might be limited if you use a third-party gateway.
- If you have already used microservice architecture and included an API gateway in the design to perform authentication, IP whitelisting, etc., you may add a rate limiter to the API gateway.
- Building your own rate limiting service takes time. If you do not have enough engineering resources to implement a rate limiter, a commercial API gateway is a better option.

Algorithms for rate limiting

Rate limiting can be implemented using different algorithms, and each of them has distinct pros and cons. Even though this chapter does not focus on algorithms, understanding them at high-level helps to choose the right algorithm or combination of algorithms to fit our use cases. Here is a list of popular algorithms:

- Token bucket
- Leaking bucket
- Fixed window counter
- Sliding window log
- Sliding window counter

Token bucket algorithm

The token bucket algorithm is widely used for rate limiting. It is simple, well understood and

commonly used by internet companies. Both Amazon [5] and Stripe [6] use this algorithm to throttle their API requests.

The token bucket algorithm work as follows:

- A token bucket is a container that has pre-defined capacity. Tokens are put in the bucket at preset rates periodically. Once the bucket is full, no more tokens are added. As shown in Figure 4-4, the token bucket capacity is 4. The refiller puts 2 tokens into the bucket every second. Once the bucket is full, extra tokens will overflow.

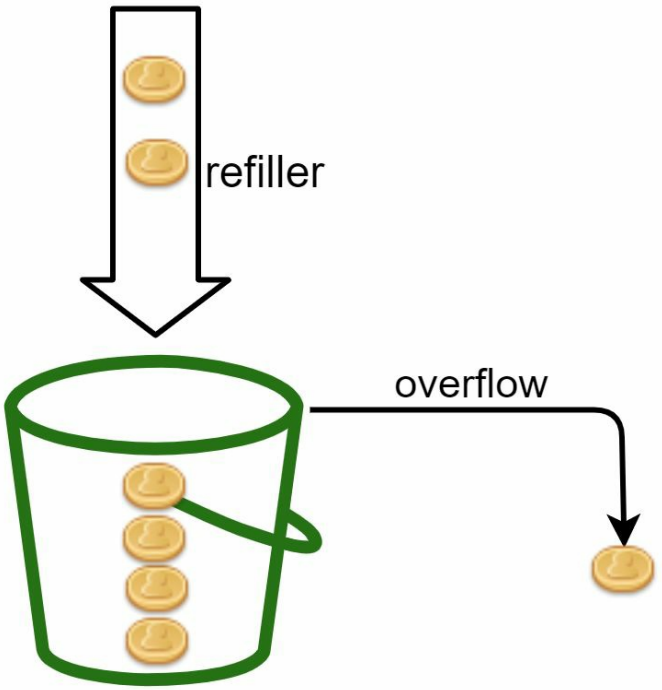

Figure 4-4

- Each request consumes one token. When a request arrives, we check if there are enough tokens in the bucket. Figure 4-5 explains how it works.
  - If there are enough tokens, we take one token out for each request, and the request goes through.
  - If there are not enough tokens, the request is dropped.

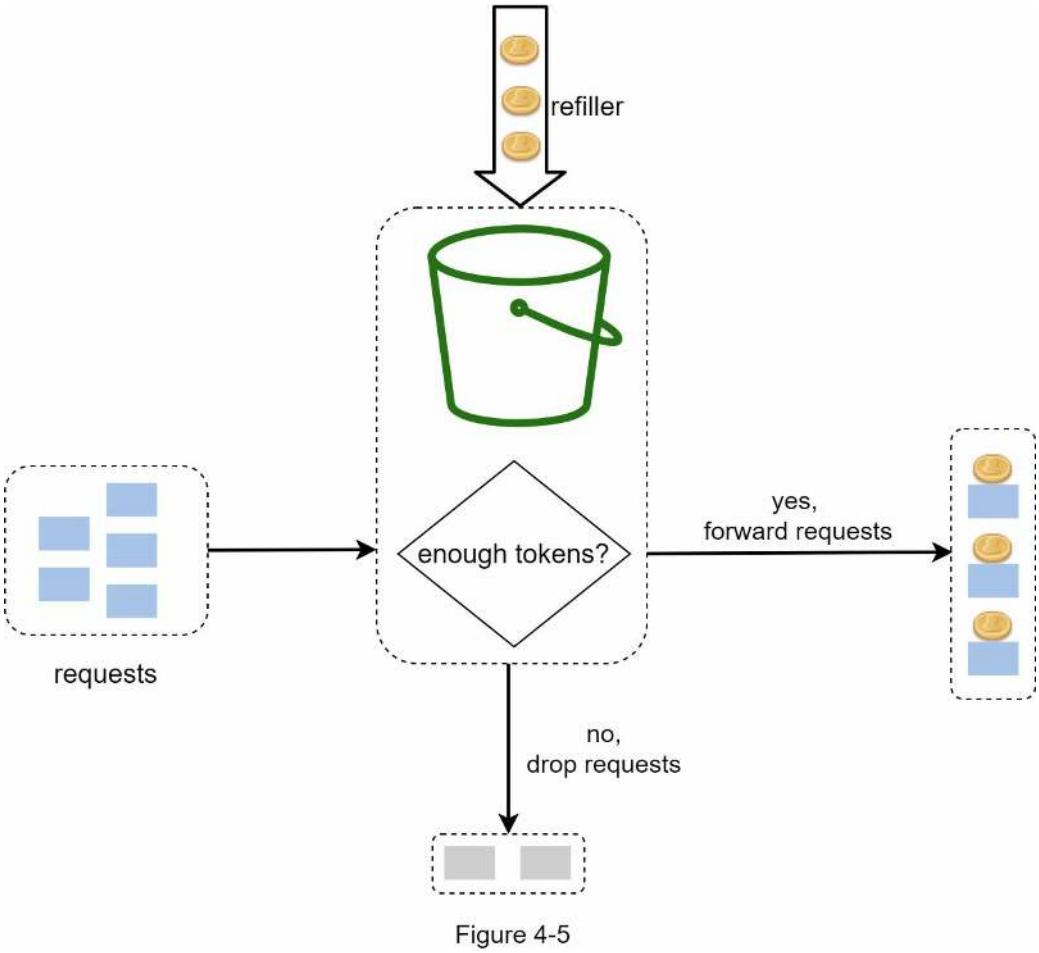

Figure 4-5

Figure 4-6 illustrates how token consumption, refill, and rate limiting logic work. In this example, the token bucket size is 4, and the refill rate is 4 per 1 minute.

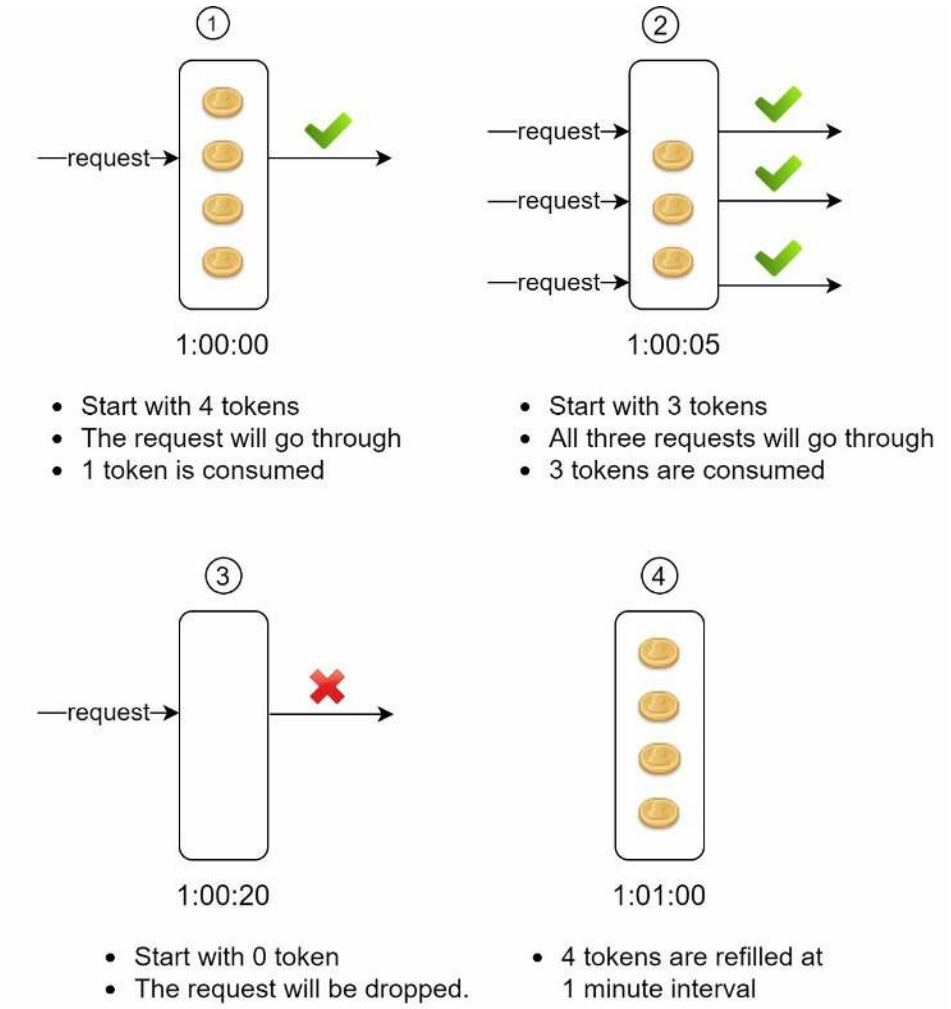

Figure 4-6

The token bucket algorithm takes two parameters:

- Bucket size: the maximum number of tokens allowed in the bucket
- Refill rate: number of tokens put into the bucket every second

How many buckets do we need? This varies, and it depends on the rate-limiting rules. Here are a few examples.

- It is usually necessary to have different buckets for different API endpoints. For instance, if a user is allowed to make 1 post per second, add 150 friends per day, and like 5 posts per second, 3 buckets are required for each user.
- If we need to throttle requests based on IP addresses, each IP address requires a bucket.
- If the system allows a maximum of 10,000 requests per second, it makes sense to have a global bucket shared by all requests.

Pros:

- The algorithm is easy to implement.
- Memory efficient.
- Token bucket allows a burst of traffic for short periods. A request can go through as long as there are tokens left.

Cons:

- Two parameters in the algorithm are bucket size and token refill rate. However, it might

be challenging to tune them properly.

Leaking bucket algorithm

The leaking bucket algorithm is similar to the token bucket except that requests are processed at a fixed rate. It is usually implemented with a first-in-first-out (FIFO) queue. The algorithm works as follows:

- When a request arrives, the system checks if the queue is full. If it is not full, the request is added to the queue.
- Otherwise, the request is dropped.
- Requests are pulled from the queue and processed at regular intervals.

Figure 4-7 explains how the algorithm works.

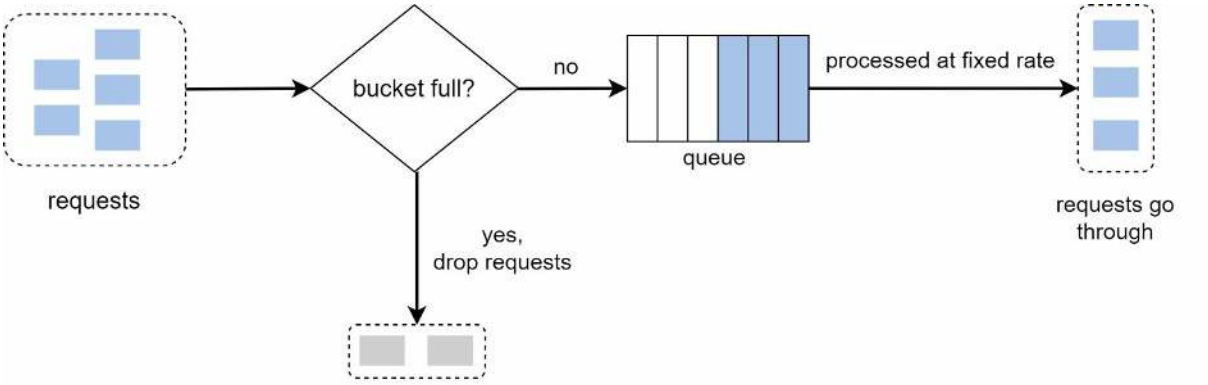

Figure 4-7

Leaking bucket algorithm takes the following two parameters:

- Bucket size: it is equal to the queue size. The queue holds the requests to be processed at a fixed rate.
- Outflow rate: it defines how many requests can be processed at a fixed rate, usually in seconds.

Shopify, an ecommerce company, uses leaky buckets for rate-limiting [7].

Pros:

- Memory efficient given the limited queue size.
- Requests are processed at a fixed rate therefore it is suitable for use cases that a stable outflow rate is needed.

Cons:

- A burst of traffic fills up the queue with old requests, and if they are not processed in time, recent requests will be rate limited.
- There are two parameters in the algorithm. It might not be easy to tune them properly.

Fixed window counter algorithm

Fixed window counter algorithm works as follows:

- The algorithm divides the timeline into fix-sized time windows and assign a counter for each window.
- Each request increments the counter by one.
- Once the counter reaches the pre-defined threshold, new requests are dropped until a new time window starts.

Let us use a concrete example to see how it works. In Figure 4-8, the time unit is 1 second and the system allows a maximum of 3 requests per second. In each second window, if more than 3 requests are received, extra requests are dropped as shown in Figure 4-8.

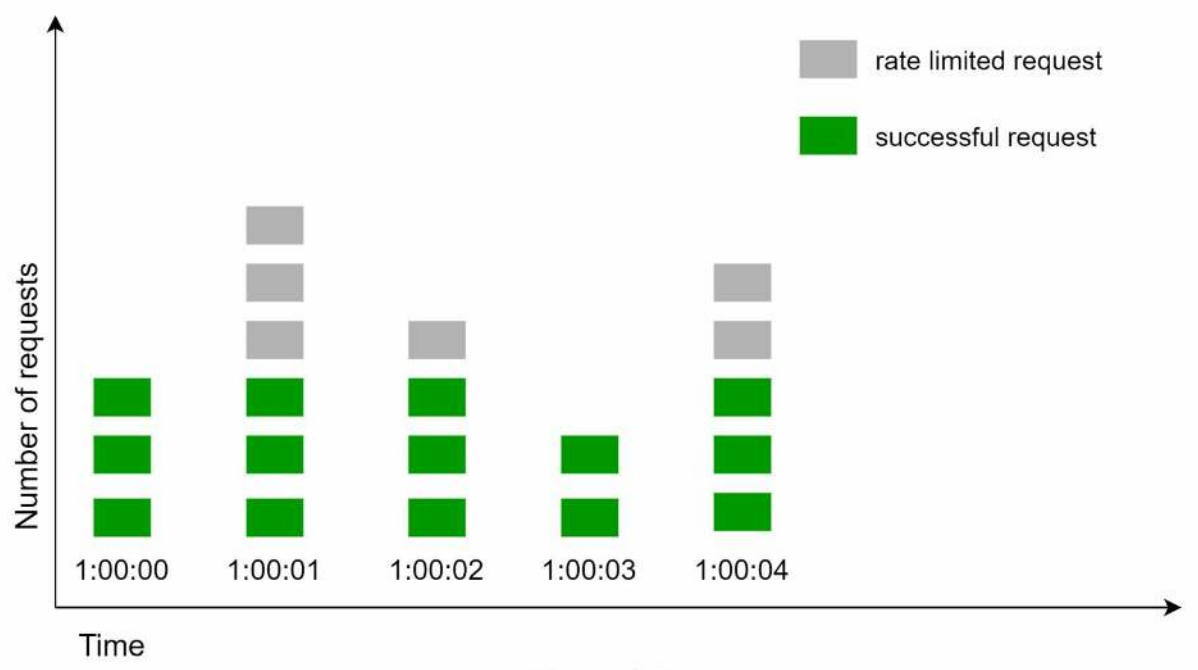

Figure 4-8

A major problem with this algorithm is that a burst of traffic at the edges of time windows could cause more requests than allowed quota to go through. Consider the following case:

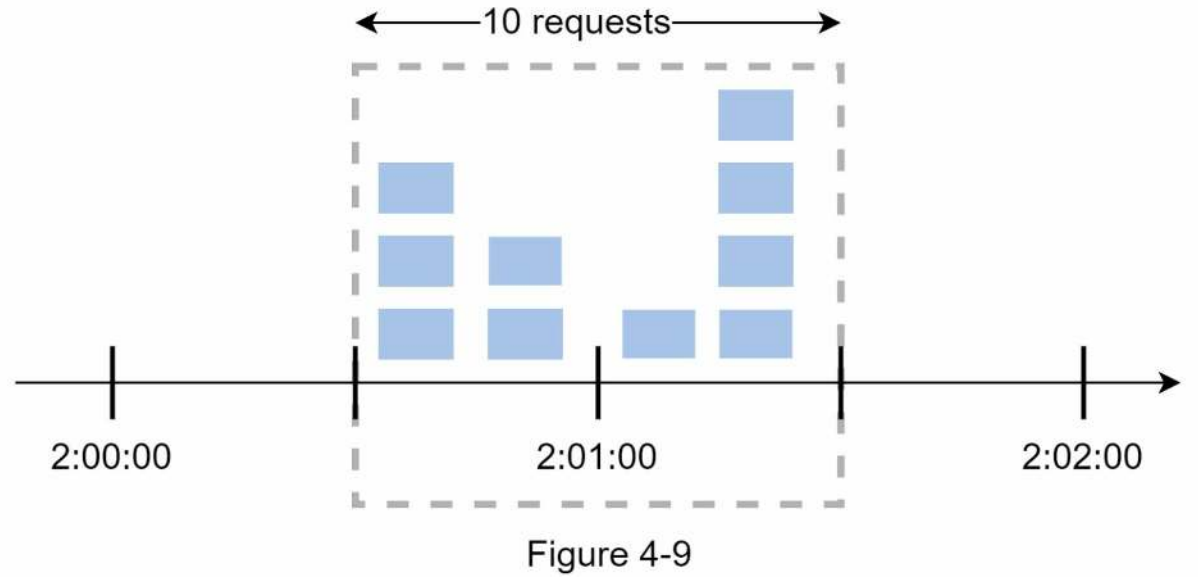

Figure 4-9

In Figure 4-9, the system allows a maximum of 5 requests per minute, and the available quota resets at the human-friendly round minute. As seen, there are five requests between 2:00:00 and 2:01:00 and five more requests between 2:01:00 and 2:02:00. For the one-minute window between 2:00:30 and 2:01:30, 10 requests go through. That is twice as many as allowed requests.

Pros:

- Memory efficient.

• Easy to understand.

• Resetting available quota at the end of a unit time window fits certain use cases.

Cons:

• Spike in traffic at the edges of a window could cause more requests than the allowed quota to go through.

Sliding window log algorithm

As discussed previously, the fixed window counter algorithm has a major issue: it allows more requests to go through at the edges of a window. The sliding window log algorithm fixes the issue. It works as follows:

• The algorithm keeps track of request timestamps. Timestamp data is usually kept in cache, such as sorted sets of Redis [8].

• When a new request comes in, remove all the outdated timestamps. Outdated timestamps are defined as those older than the start of the current time window.

• Add timestamp of the new request to the log.

• If the log size is the same or lower than the allowed count, a request is accepted. Otherwise, it is rejected.

We explain the algorithm with an example as revealed in Figure 4-10.

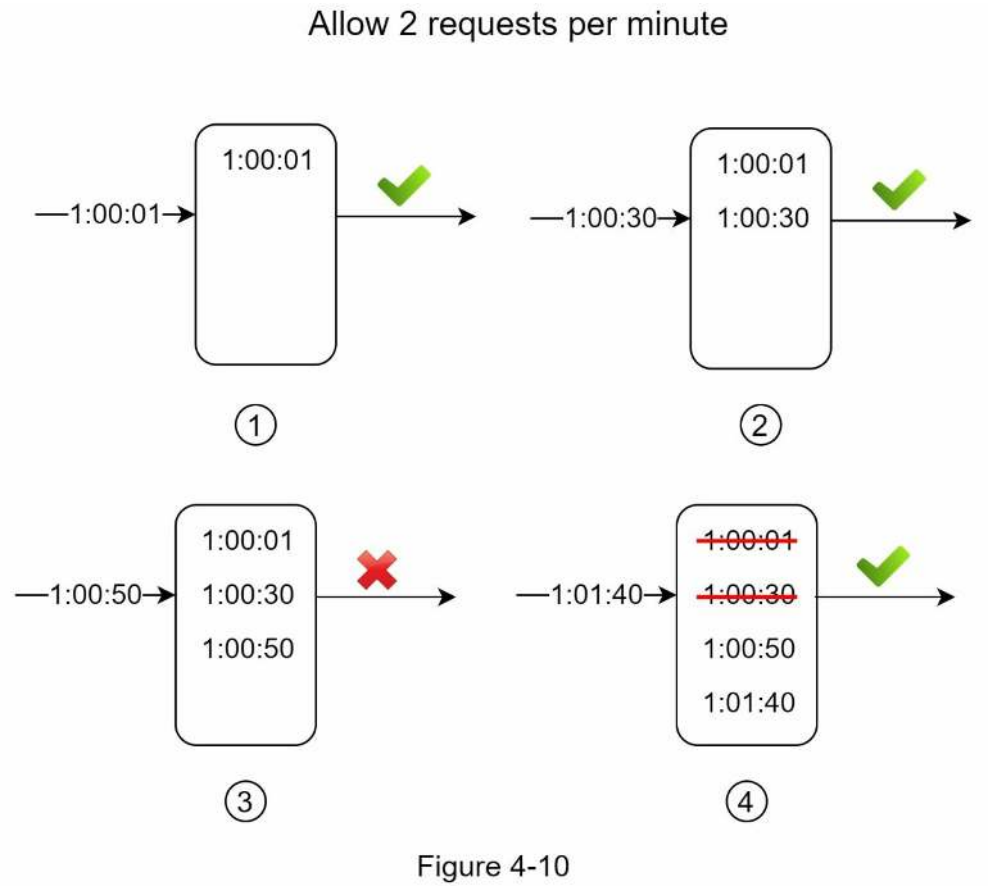

Figure 4-10

In this example, the rate limiter allows 2 requests per minute. Usually, Linux timestamps are stored in the log. However, human-readable representation of time is used in our example for better readability.

• The log is empty when a new request arrives at 1:00:01. Thus, the request is allowed.