Jahong Liu

System Design Basics 1: Application Architecture

Application Architecture

Within a production application architecture, various components work together to create a robust system.

A developer's perspective

Developers write code that is deployed to a server. Simple definition of a **server**: a computer that handles requests from another computer. This server also requires **persistent** storage to store the application's data. A server may have built-in storage, but that has its limitations in terms of size. As such, a server may talk to an external storage system (database, cloud etc). This storage may not be part of the same server, and is instead connected through a **network**.

A user's perspective

A user is someone who makes a request from the server, usually through a web browser. In this case, the web browser is the client to whom the server responds to.

If a user wanted to use a front-end feature, the server will respond with necessary JavaScript/HTML/CSS code, compiled to display what the user requested. But, what if we have a ot of users and the single server cannot handle all of the requests on its own? There is bound to be a bottleneck, either through our RAM or out CPU. To maintain performance while dealing with multiple users, we will need to scale our server.

Scaling our server

To handle multiple requests, it might be a good idea to add more RAM or upgrade to a CPU with more cores and higher clocking speed. However, every computer has a limitation in terms of upgrades. Upgrading components within the same computer is referred as **vertical scaling**.

We can also have multiple servers running our code, and we can distribute the user requests among these servers. This way, not all users are taling to one server, which ensures that the speed of each server remains intact. This also ensures that if one server were to go down, we can direct our traffic to one of our other servers. This is known as **horizontal scaling**.

Generally, in large systems, we prefer horizontal scaling, as it is much more powerful, and can be achieved with commodity hardware (i.e., relatively inexpensive, standard hardware). However, it also requires much more engineering effort, as we need to ensure that the servers are communicating with each other, and that the user requests are being distributed evenly.

For simple applications however, vertical scaling may be sufficient and the easier solution to implement. Even some services within Amazon Prime Video were recently migrated from a microservice architecture to a monolithic architecture.

But with multiple servers, what determines which requests go to which server? This achieved through a **load balancer**. A load balancer will evenly distribute the incoming requests across a group of servers.

It's also important to remember that servers don't exist in isolation. It is highly likely that servers are interacting with external servers, through APIs. For example, a website interacts with other services like Stripe, Mapbox through an API.

HDDs are mechanical, and have read/write head. The older they get, the more wear and tear they collect which slows them down overtime. SSDs are significantly faster because they don't have moving parts, and reply on reading and writing data electronically (similar to RAM)

Logging and Metrics

Servers also have logging services, which gives the developer a log of all the activity that happened. Logs can be written to the same server, but for better reliability they are commonly written to another external server.

This gives developers insight into how the requests went, if any errors occured, or what happened before a server crashed. However, logs don't provide the complete picture. If our RAM has become the bottleneck of our server, or our CPU resources are restricting the requests being handled efficiently, we require a **metrics** service. A metric service will collect data from different sources within our server environment, such as CPU usage, network traffic etc. This allows developers to gain insights into server's behavior and identify potential bottlenecks.

Alerts

As developers, we wouldn't want to keep checking metrics to see if any unexpected behavior exhibits itself. This would be like checking your phone every 55 minutes for a notification. It is more ideal to receive a push notification. We can program alerts so that whenever a certain metric fails to meet the target, the developers receive a push notification. For example, if 100%100% of the user requests receive successful responses, we could set an alert to be notified if this metric dips under 95%95%

Caches

Most CPUs have an L1, L2, and L3 cache, which are physical components that are much faster than RAM, but they only stores data on the order of KBs or tens of MBs

Whenever a read operation is requested, the cache is checked before the RAM and the disk. If the data requested is in the cache, and is unchanged since the last time it was accessed, it will be fetched from the cache, and not the RAM. Reading and writing to the cache is a lot faster than RAM and disk. It is up to the operating system to decide what gets stored in the cache.

Caching is an important concept applied in *many* areas beyond computer architecture. For example, web browsers use cache to keep track of frequently accessed web pages to load them faster. This stored data might include HTML, CSS, JavaScript, and images, among other things. If the data is still valid from the time the page was cached, it will load faster. But in this case, the browser is using the disk as cache, because making internet requests is a lot slower than reading from disk.

The cache being part of the CPU is only part of the reason why it is faster than RAM. Cache is what's known as SRAM. If you are interested, you might view this resource from **MIT,** which gives a gentle introduction.

Closing Notes

All three memory types have their own use. If you are writing an essay, and the computer were to shut down, you would want it stored on a disk drive. While multi-tasking, you would want the programs opened to be stored in the RAM.