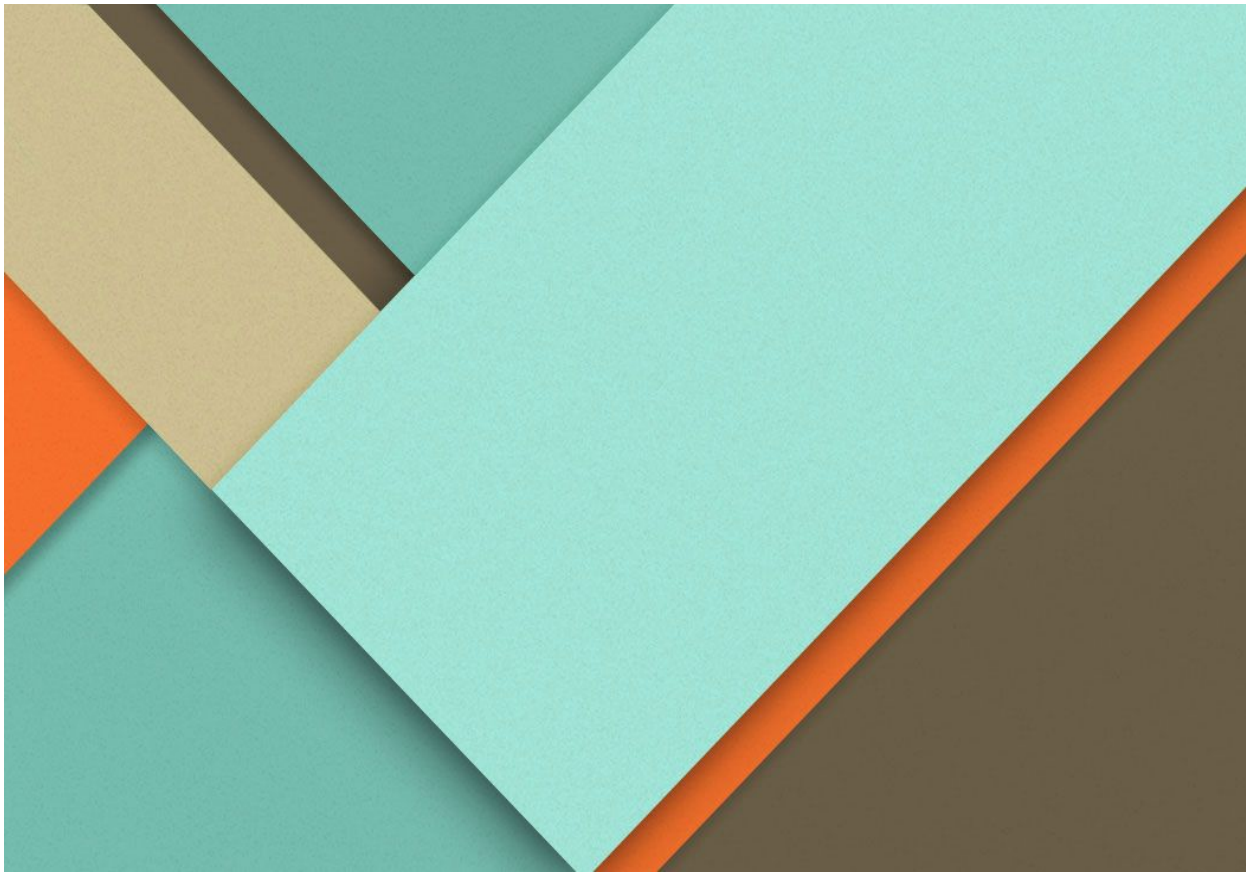

Distributed System

Dealing with large and changing number of client requests

Group 9
Zeyu Yao
Tao Yu
Morgan Greer
Jeremy Wong

Why is this necessary?

It is not hard to imagine a web-based system will server hundreds of thousands user requests when it becomes more and more popular. Clearly, such a large number of concurrent requests can crash a single server because of its limited processing capability.

What we need to do is strengthen our system with the capability to handle large and changing number of users.

Server-side optimization

If we think from the perspective of users, we want to reduce users waiting time as possible as we can (optimising use experience), especially when lots of clients accessing the system.

To do this, the problem domain contains these aspects:

- Data processing time on the server side
- Data transfer time on the network
- Data processing time on the client side (local)

What we have been focused on is optimizing the processing power of the entire system. A typical way to do that is the distributed system architecture.

Implementation

We have designed a very simple distributed system architecture. As shown in figure 1, the system has been separated into three roles:

- Management server (reverse proxy & load balancing)
- Web server (multiple)
- Database server

The idea is to hide multiple web servers behind a management server (reverse proxy) and let it distribute client requests across these web servers (load balancing). And all the web servers could access to a same remote database server which keeps data consistency among web servers.

Considering etherpad (the web-based system), we selected NGINX to build load balancing and reverse proxy by adding configuration files. And using MySQL server & MySQL client to separate database away.

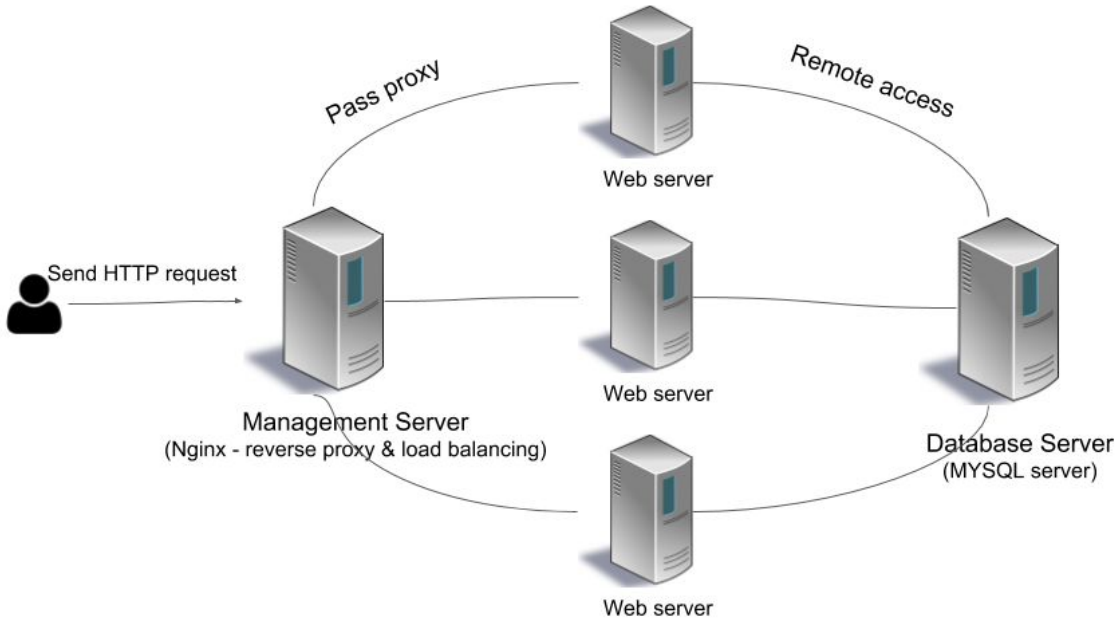

Figure-1: Distributed System Architecture

Other strategies

Besides implementing load balancing and a reverse proxy, there are other strategies available to optimize the entire system.

If we focus on the database for example, a distributed database is a traditional approach to 'scale-out' data into multiple database servers when under the scale of hundreds of millions. A single database server may not support intensive read requests, and performance may be problematic. The specific implementation approach is known as horizontal scale and read&write separation.

Do it in DevOps approach

In DevOps, automation is the key, we create scripts to deploy our web-based system easier. These simple automated scripts and tasks replaced traditional manual methods. In the continuous deployment strategy, we can execute automated scripts repeatedly, which eliminates the possibility of human error and reduces operational costs. Thus, organizations could deliver complex services faster and more efficiently.

Challenges

Although distributed system architecture has improved system concurrent processing capability and system availability, there are still several challenges. When we design a system architecture, we should think of these:

1. Scalability

   The system should be designed so that as the business volume increases, the corresponding system must also be able to expand to provide the corresponding services.

2. Security

   Encryption is used to provide proper protection for shared resources, and all sensitive information passed on the network needs to be encrypted. A denial of service attack is still a problem to be solved.

3. Consistency

   Data is scattered or copied to different machines. How to ensure the consistency of data between hosts will become a difficult point.