Final Design of Quora

Learn about the limitations of Quora's design and improve the design.

We'll cover the following

- Limitations of the proposed design
- Detailed design of Quora
  - Service hosts
  - Vertical sharding of MySQL
  - MyRocks
  - Kafka
  - Technology usage

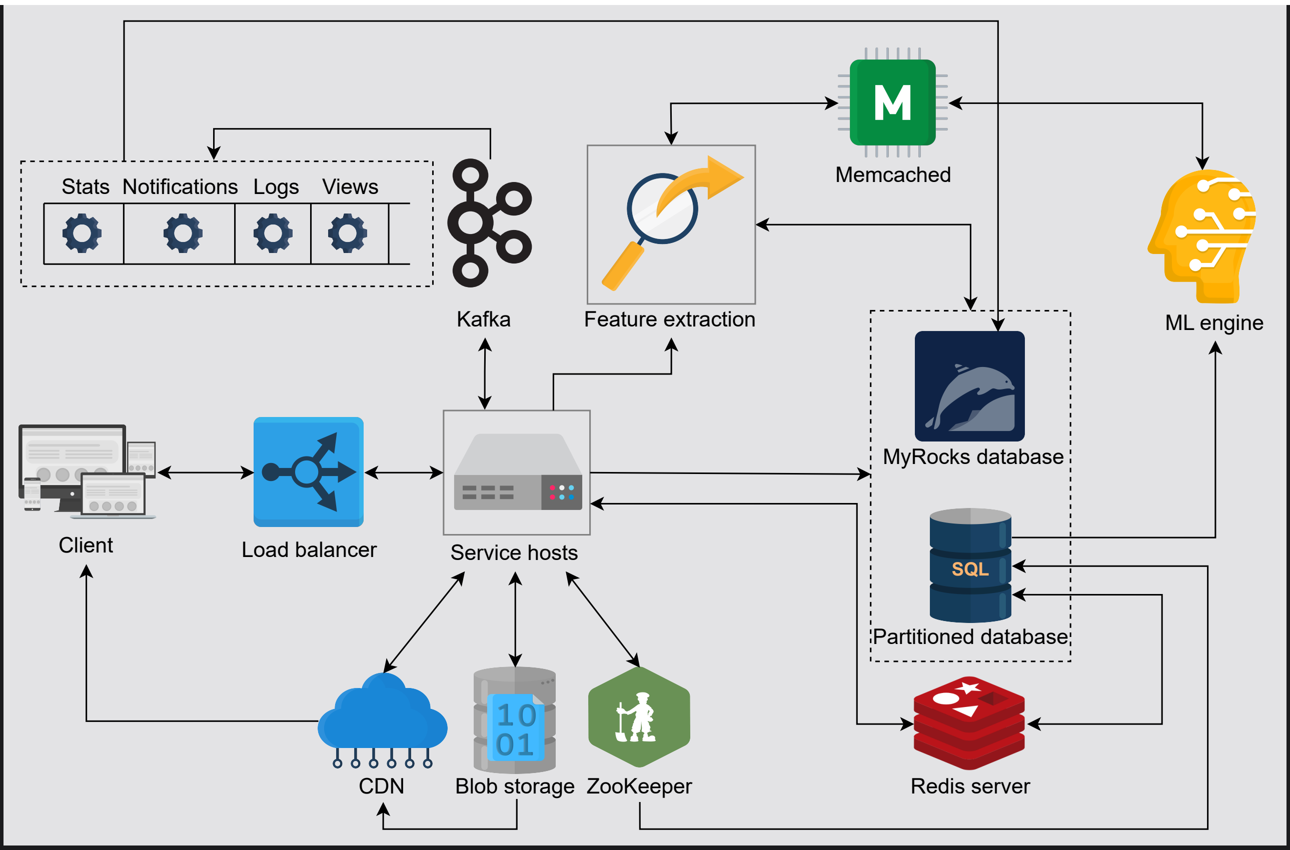
Limitations of the proposed design

The proposed design serves all the functional requirements. However, it has a number of serious drawbacks that emerge as we scale. This means that we are unable to fulfill the non-functional requirements. Let's explore the main shortcomings below:

- **Limitations of web and application servers**: To entertain the user's request, payloads are transferred between web and application servers, which increases latency because of network I/O between these two types of servers. Even if we achieve parallel computation by separating the we from application servers (that is, the manager and worker processes), the added latency due to an additional network link erodes a user's experienc Apart from data transfer, control communication between the router library with manager and worker processes also imposes additional performance penalties.

- **In-memory queue failure**: The internal architecture of application servers log tasks and forward them to the in-memory queues, which serve them to the workers. These in-memory queues of different priorities can be subject to failures. For instance, if a queue gets lost, all the tasks in that queue are lost as well, and manual engineering is required to recover those tasks. This greatly reduces the performance of the system. On the other hand, replicating these queues requires increasing RAM size. Also, with the number of features (functional requirements) that our system offers, many tasks can get assembled, which results in insufficient memory. At the same time, it is not desirable to choke application servers with not-so-urgent tasks. For example, application servers should not be burdened with tasks like storing view counts for answers, adding statistics to the database for later analysis, and so on.
- **Increasing QPS on MySQL**: Because we have a higher number of features offered by our system, few MySQL tables receive a lot of user queries. This results in a higher number of QPS on certain MySQL servers, which can result in higher latency. Furthermore, there is no scheme defined for disaster recovery management in our design.
- **Latency of HBase:** Even though HBase allows high real-time throughput, its P99 latency is not among the best. A number of Quora features require the ML engine that has a latency of its own. Due to the addition of the higher latency of HBase, the overall performance of the system degrades over time.

The issues highlighted above require changes to the earlier proposed design. Therefore, we'll make the following adjustments and update our design:



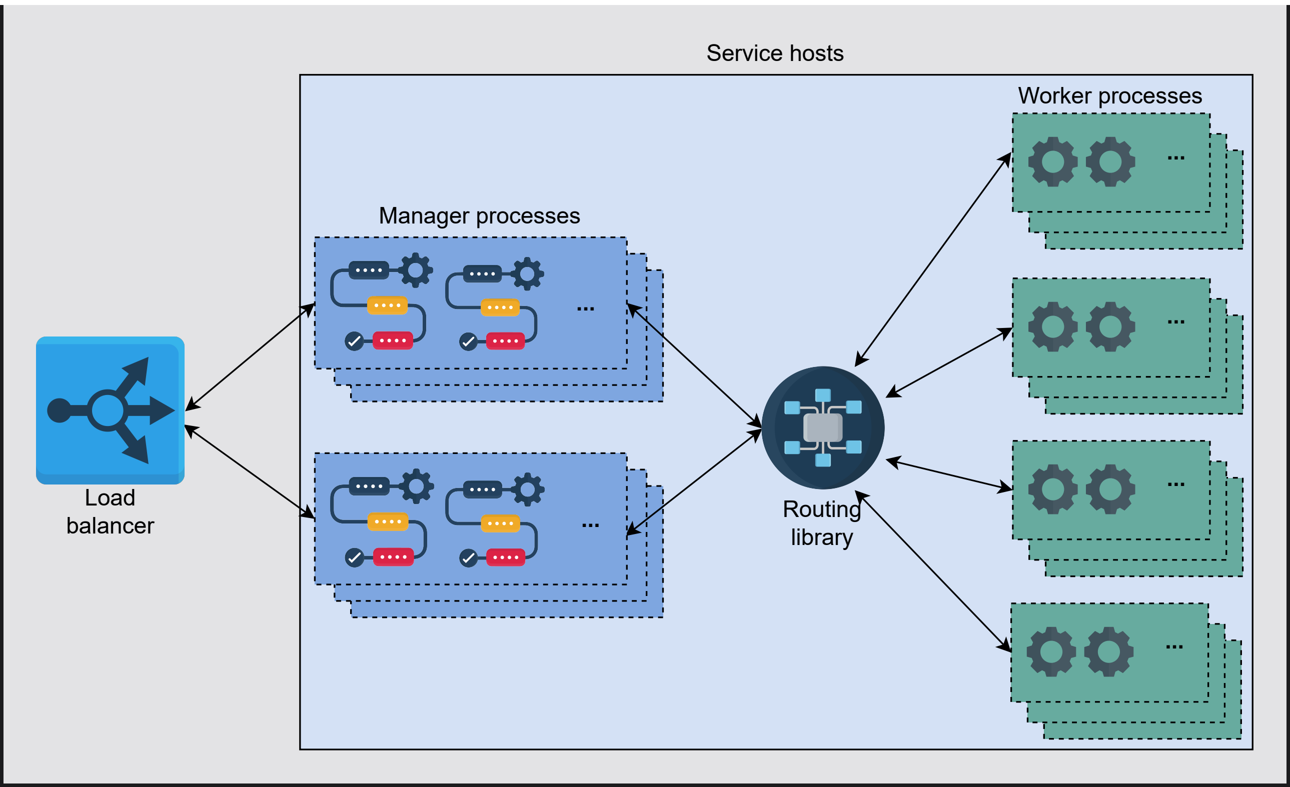
Detailed design of Quora

Detailed design of Quora

Let’s understand the improvements in our design:

Service hosts

We combine the web and application servers within a single powerful machine that can handle all the processes at once. This technique eliminates the network I/O and the latency introduced due to the network hops required between the manager, worker, and routing library processes. The illustration below provides an abstract view of the updated web server architecture:



The updated design, where web and application servers are combined in the service host

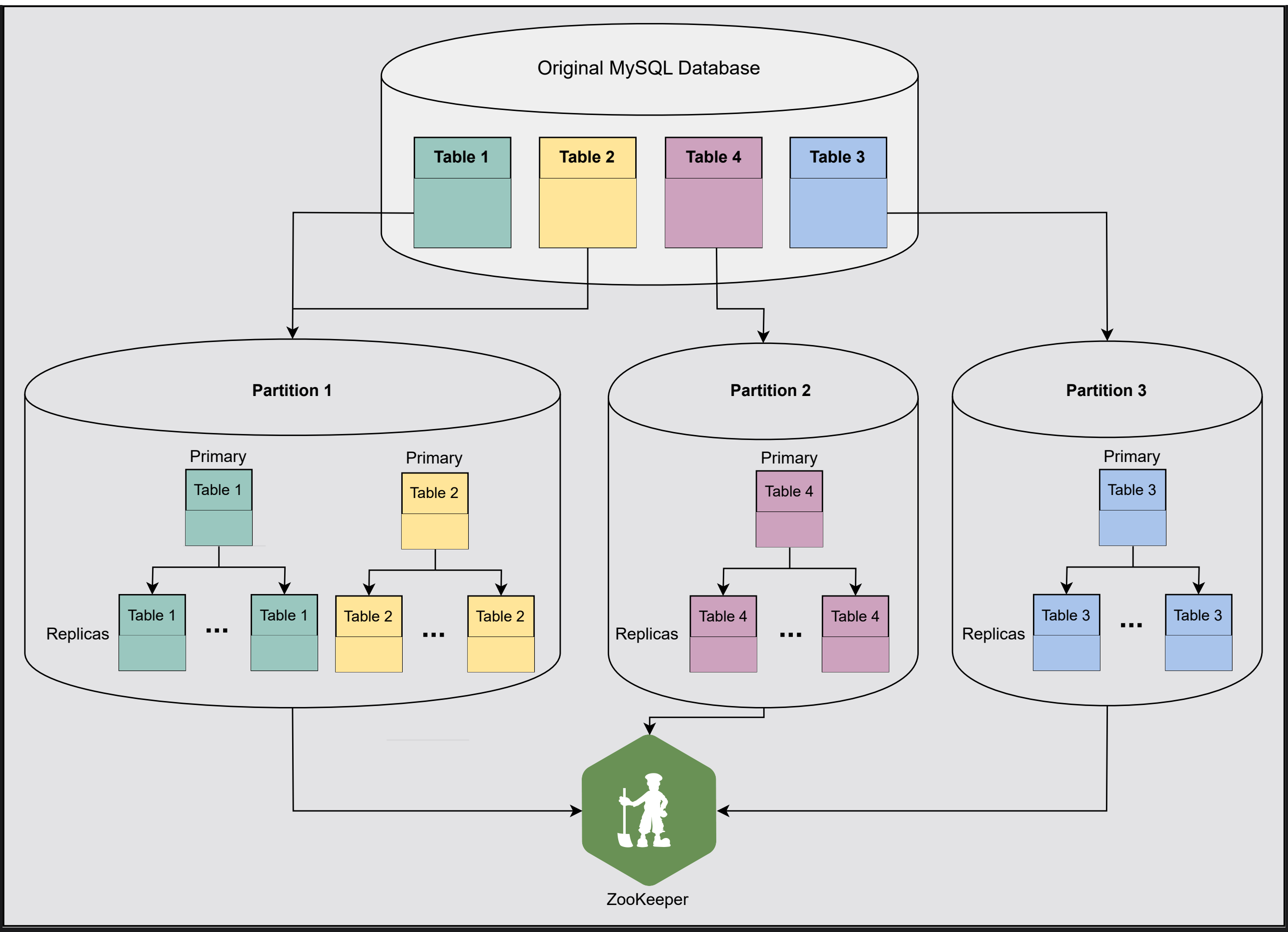
Vertical sharding of MySQL

Tables in the MySQL server are converted to separate shards that we refer to as **partitions**. A partition has a single primary server and multiple replica servers.

The goal is to improve performance and reduce the load due to an increasing number of queries on a single database table. To achieve that, we do vertical sharding in two ways:

1. We split tables of a single database into multiple partitions. The concept is depicted in Partitions 2 and 3, which embed Tables 4 and 3, respectively.
2. We combine multiple tables into a single partition, where `join` operations are anticipated. The concept is depicted in Partition 1, which embeds Tables 1 and 2.

Therefore, we are able to co-locate related data and reduce traffic on hot data. The illustration below depicts vertical sharding at Quora.



The architecture of vertical sharding at Quora

After we complete the partitioning, we require two types of mappings or metadata to complete our scaling process:

1. Which partitions contain which tables and columns?
2. Which hosts are primary and replicas of a particular partition?

Both of these mappings are maintained by a service like ZooKeeper.

The sharded design above ensures scalability because we are able to locate related data in a single partition, and therefore it eliminates the need for querying data from multiple shards. Also, the number of read-replicas can be increased for hot shards, or further sharding may be performed. For edge cases where joining may be needed, we can perform it at the application level.

> **Note:** Vertical sharding is of particular interest in Quora's design because horizontal sharding is more common in the database community. The main idea behind vertical sharding is to achieve

> scalability by carefully dividing or re-locating tables and eliminating join operations across different shards. Nevertheless, a vertically sharded partition or table can grow horizontally to an extent where horizontal sharding will be necessary to retain acceptable performance.

MyRocks

The new design embeds MyRocks as the key-value store instead of HBase. We use the MyRocks version of RocksDB for two main reasons:

1. MyRocks has a lower p99 latency instead of HBase. Quora claims to have reduced P99 latency from 80 ms to 4 ms using MyRocks.
2. There are operational tools that can transfer data between MyRocks and MySQL.

> **Note:** Quora serves the ML compute engine by extracting features from questions and answers stored in MySQL. In this case, the operational tools come in handy to transfer data between MyRocks and MySQL.

Kafka

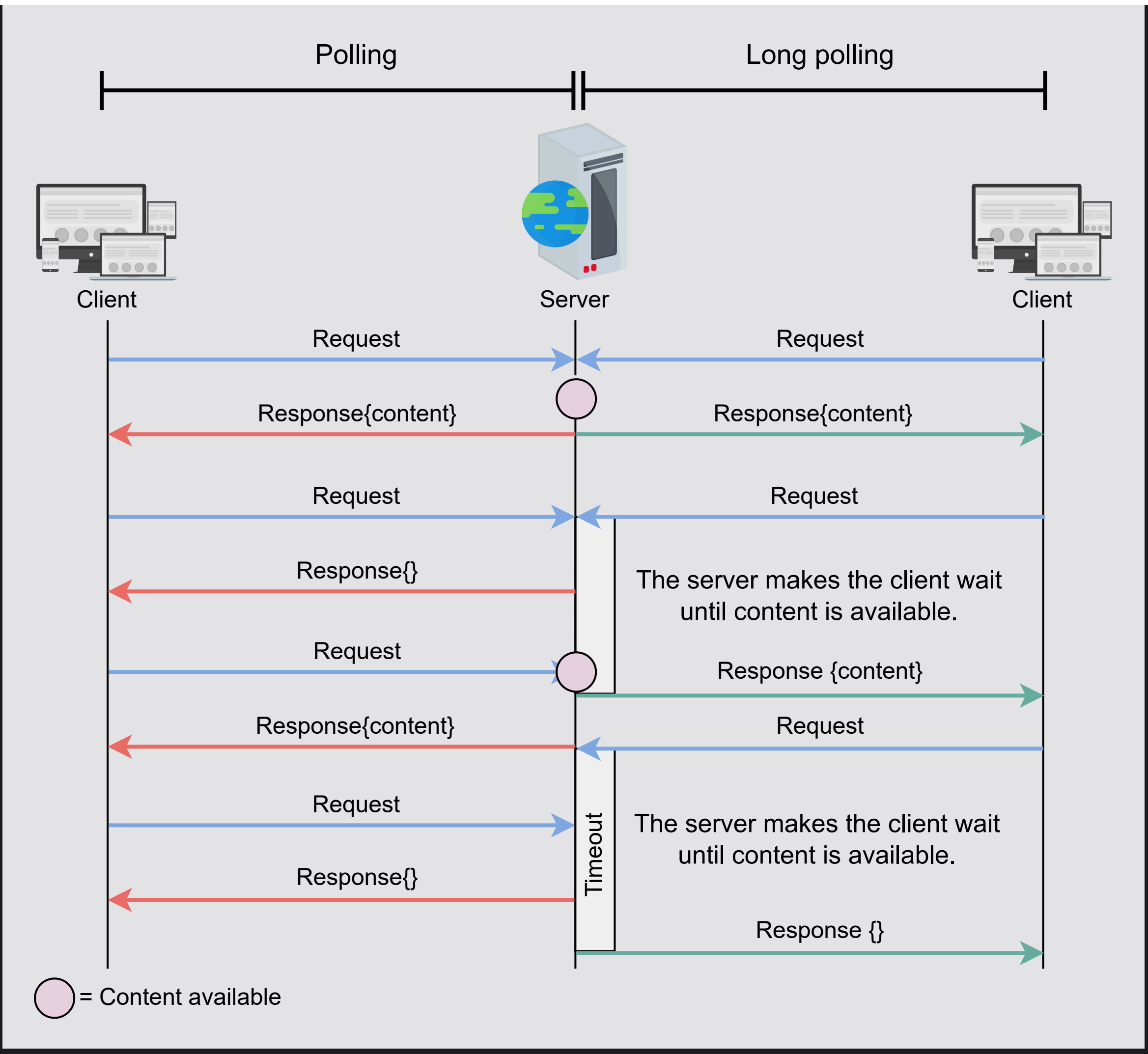
Our updated design reduces the request load on service hosts by separating not-so-urgent tasks from the regular API calls. For this purpose, we use Kafka, which can disseminate jobs among various queues for tasks such as the view counter (see Sharded Counters), notification system, analytics, and highlight topics to the user. Each of these jobs is executed through cron jobs.

Technology usage

Services that scale quickly have little time to develop new features and handle an increasing number of requests from users. Such services employ cloud infrastructure to handle spikes in traffic. Also, the choice of programming language is important. Just like we mentioned that YouTube chose Python for faster programming, we can apply the same logic to Quora. In fact, Quora uses the Python Paste web framework.

It is desirable to use a faster programming language like C++ to develop the feature extraction service. For online recommendation services through a ML engine, feature extraction service should be quick, to enable the ML engine to accomplish accurate recommendations. Not only that, but reducing the latency burden on the ML engine allows it to provide a larger set of services. We can employ the Thrift service to support interoperability between programming languages within different components.

Features like comments, upvotes, and downvotes require frequent page updates from the client side. **Polling** is a technique where the client (browser) frequently requests the server for new updates. The server may or may not have any updates but still responds to the client. Therefore, the server may get uselessly overburdened. To resolve this issue, Quora uses a technique called **long polling**, where if a client requests for an update, the server may not respond for as long as 60 seconds if there are no updates. However, if there is an update, the server will reply immediately and allow the client to make new requests.


Polling
Long polling
Client
Server
Client
Request
Request
Response{content}
Response{content}
Request
Request
Response{}
The server makes the client wait until content is available.
Request
Response {content}
Response{content}
Request
Request
Timeout
The server makes the client wait until content is available.
Response{}
Response {}
= Content available