**Quora Design**

1. Solidify the requirements – both Functional and Non-Functional

* Functional Requirements:
  + **Questions and answers:** Users can ask questions and give answers. Questions and answers can include images and videos.
  + **Upvote/downvote and comment:** It is possible for users to upvote, downvote, and comment on answers.
  + **Search:** Users should have a search feature to find questions already asked on the platform by other users.
  + **Recommendation system:** A user can view their feed, which includes topics they’re interested in. The feed can also include questions that need answers or answers that interest the reader. The system should facilitate user discovery with a recommender system.
  + **Ranking answers:** We enhance user experience by ranking answers according to their usefulness. The most helpful answer will be ranked highest and listed at the top.
* Non-Functional Requirements:
* **Scalability**: The system should scale well as the number of features and users grow with time. It means that the performance and usability should not be impacted by an increasing number of users.
* **Consistency**: The design should ensure that different users’ views of the same content should be consistent. In particular, critical content like questions and answers should be the same for any collection of viewers. However, it is not necessary that all users of Quora see a newly posted question, answer, or comment right away.
* **Availability**: The system should have high availability. This applies to cases where servers receive a large number of concurrent requests.
* **Performance**: The system should provide a smooth experience to the user without a noticeable delay.

1. Scope the Problem

* What kind of clients? Mobile Apps, Web Browsers, Smart TVs?

1. Capacity/ Resource Estimation
2. Traffic Estimates
3. DAU – 300M
4. Total Users – 1B
5. Image/ Video Ratio – 15% has images, 5% has videos
6. Image Size – 250KBs
7. Video Size – 5MBs
8. Request per day by each user - 20
9. RPS of a server - 8000
   * URLShortenings/ Sec – [Skipped]
   * URLRedirections/ Sec – [Skipped]
   * Total Servers Required – (300M x 20)/ 8000 = 37500
10. Storage Estimates
11. Time Duration for which objects are required to be stored – [Skipped]
12. Usage/ day. URLShortenings/Month – [Skipped]
13. File Size Requirements – [Skipped]

* Total Requests for 5 years = [Skipped]
* Total Storage Required = [Skipped]

1. Bandwidth Estimates

* Incoming Data – [Skipped]
* Outgoing Data – [Skipped]

1. Memory (Cache) Estimates
2. 80/20 Rule? Yes

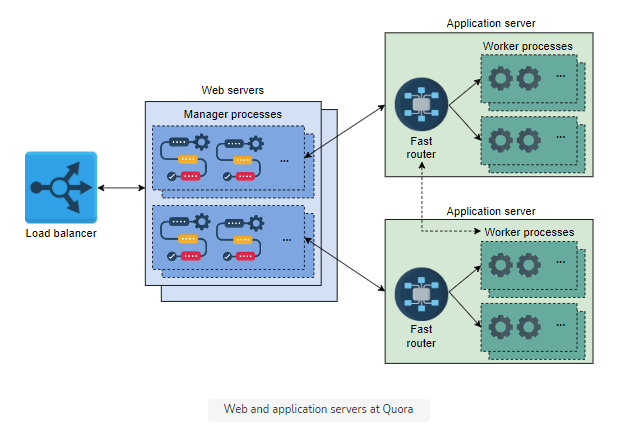
* Cache Size = [Skipped]

1. System API’s

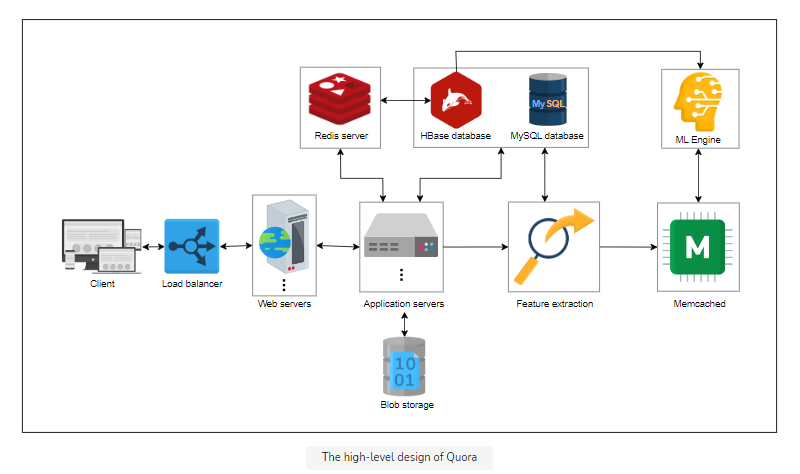
* postQuestion(user\_id, question, description, topic\_label, video, image)
* postAnswer(user\_id, question\_id, answer\_text, video, image)
* upvote(user\_id, question\_id, answer\_id)
* comment(user\_id, answer\_id, comment\_text)
* search(user\_id, search\_text)
* **We use a sequencer to generate the different IDs mentioned in the API calls.**

1. Database Design
2. Present the building blocks of the Design
3. Web and application servers:

* The web and application servers maintain various processes to generate a webpage. The web servers have manager processes and the application servers have worker processes for handling various requests.
* The manager processes distribute work among the worker processes using a **router library.**
* The router library is enqueued with tasks by the manager processes and dequeued by worker processes. Each application server maintains several in-memory queues to handle different user requests. The following illustration provides an abstract view of web and application servers



1. Datastores: essential for storing all sorts of data, such as user questions and answers, comments, and likes and dislikes. Also, user data will be stored in the databases.
   1. We can use critical data like questions, answers, comments, and upvotes/downvotes in a relational database like MySQL because it offers a higher degree of consistency.
   2. NoSQL databases like HBase can be used to store the number of views of a page, scores used to rank answers, and the extracted features from data to be used for recommendations later on.
   3. Because recomputing features is an expensive operation, HBase can be a good option to store and retrieve data at high bandwidth. We require high read/write throughput because big data processing systems use high parallelism to efficiently get the required statistics.
2. Load Balancers: will be used to divide the traffic load among the service hosts.
3. Blob Store: blob storage is required to store videos and images posted in questions and answers.
4. Distributed Caching System: will be used to store frequently accessed data. We can also use caching to store our view counters for different questions.
   1. For performance improvement, two distributed cache systems are used: Memcached and Redis.
      1. Memcached is primarily used to store frequently accessed critical data that is otherwise stored in MySQL.
      2. Redis is mainly used to store an online view counter of answers because it allows in-store increments.
5. CDNs:
6. Compute servers: A set of compute servers are required to facilitate features like recommendations and ranking based on a set of attributes. These features can be computed in online or offline mode. The compute servers use machine learning (ML) technology to provide effective recommendations. Naturally, these compute servers have a substantially high amount of RAM and processing power.
7. Propose a Design Diagram and get an agreement

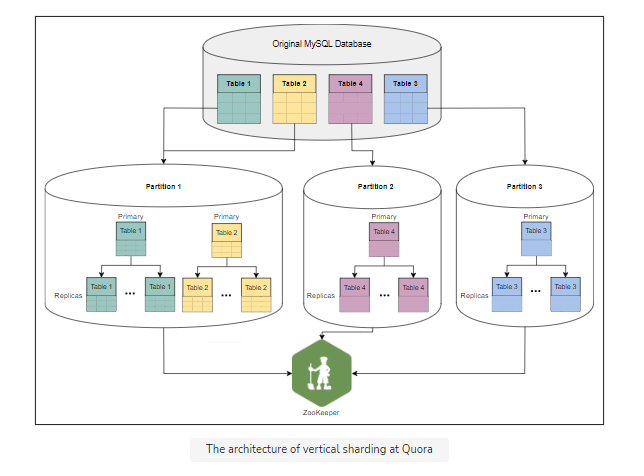


**Limitations of Proposed Design**

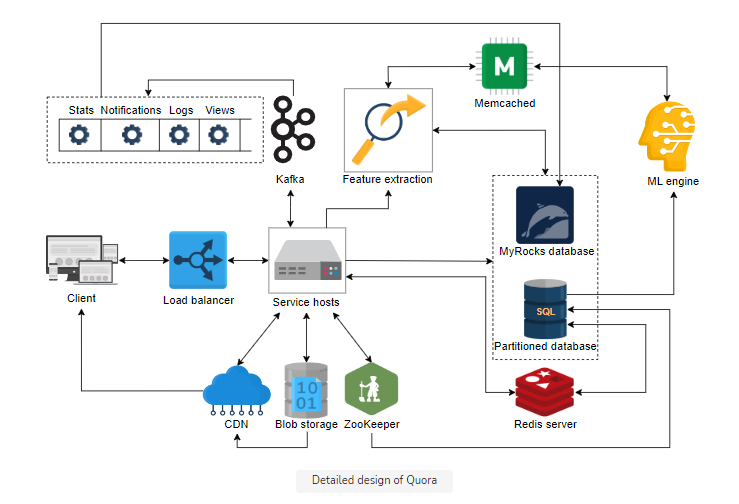
1. Limitations of web and application server:
   * Payloads are transferred between web and application servers, which increases latency because of network I/O between these two types of servers.
   * Apart from data transfer, control communication between the router library with manager and worker processes also imposes additional performance penalties.
2. 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.
   * 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.
3. Increasing QPS on MySQL (which results in higher latency):
   * 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.
4. Latency of HBase:
   * Even though HBase allows high real-time throughput, its P99 latency is not among the best.
   * Due to the addition of the higher latency of HBase, the overall performance of the system degrades over time.

**Detailed Design of Quora**

1. 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.
2. 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.
   * 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.

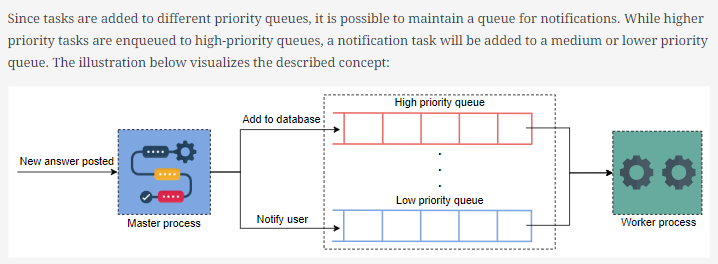


1. MyRocks
   * The new design embeds MyRocks as the key-value store instead of HBase. We use the MyRocks version of RocksDB for two main reasons:
     1. MyRocks has a lower p99 latency instead of HBase. Quora claims to have reduced P99 latency from 80 ms to 4 ms using MyRocks.
     2. There are operational tools that can transfer data between MyRocks and MySQL.
2. 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.

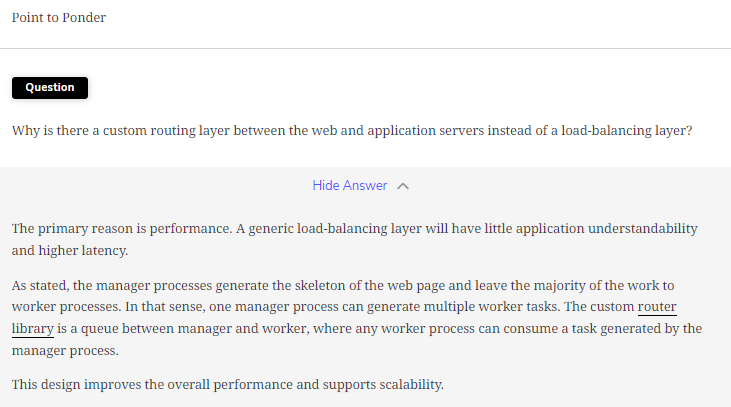


8. Workflow

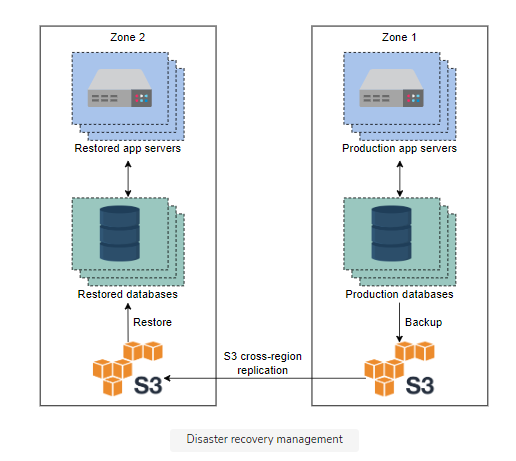
1. **Posting question, answers, comments:** 
   * The web servers receive user requests through the load balancer and direct them to the application servers.
   * The questions and answers data is stored in a MySQL database, whereas any videos and images are stored in the blob storage. A similar approach is used to post comments and upvote or downvote answers.
   * Task prioritization is performed by employing different queues for different tasks. We perform prioritization because certain tasks require immediate attention—for example, fetching data from the database for a user request—while others are not so urgent—for example, sending a weekly email digest. The worker processes will perform tasks by fetching from these queues.



1. **Answer ranking system** 
   * Answers to questions can be sorted based on date. Although it is convenient to develop a ranking system on the basis of date (using time stamps), users prefer to see the most appropriate answer at the top. Therefore, Quora uses ML to rank answers.
   * Different features are extracted over time and stored in the HBase for each type of question. These features are forwarded to the ML engine to rank the most useful answer at the top.
   * We cannot use the number of upvotes as the only metric for ranking answers because a good number of answers can be jokes—and such answers also get a lot of upvotes.
   * It is good to implement the ranking system offline because good answers get upvotes and views over time.
   * Also, the offline mode poses a lesser burden on the infrastructure. Implementing the ranking system offline and the need for special ML hardware makes it suitable to use some public cloud elastic services.
2. **Recommendation system** 
   * The recommendation system is responsible for several features. For example, we might need to develop a user feed, find related questions and ads, recommend questions to potential respondents, and even highlight duplicate content and content in violation of the service’s terms of use.
   * Unlike the answer ranking system, the recommendation system must provide both online and offline services. This system receives requests from the application server and forwards selected features to the ML engine.
3. **Search feature**
   * Over time, as questions and answers are fed to the Quora system, it is possible to build an index in the HBase. User search queries are matched against the index, and related content is suggested to the user.
   * Frequently accessed indexes can be served from cache for low latency.
   * The index can be constructed from questions, answers, topics labels, and usernames.



1. Specific Design Components:
2. Design Evaluation:
3. Availability
   * **Isolation between different components**
   * **CDN**
   * **Redundant instances**
   * **GSLB**
   * **Amazon S3 Storage for Backup data during the Disaster Recovery**
     + The first and foremost approach of handling a disaster is frequent backups. The frequency of backups depends on the size of the data. Daily backups are suitable for our design because we can backup individual data stores and shards without any hassle.
     + The approach is fairly straightforward. The data, application servers, and configurations are backed up in the Amazon S3 storage service in the same zone. Zonal replication between S3 storage facilitates transfer to another zone. Later, the application and database servers can be restored from the S3 storage in another zone.

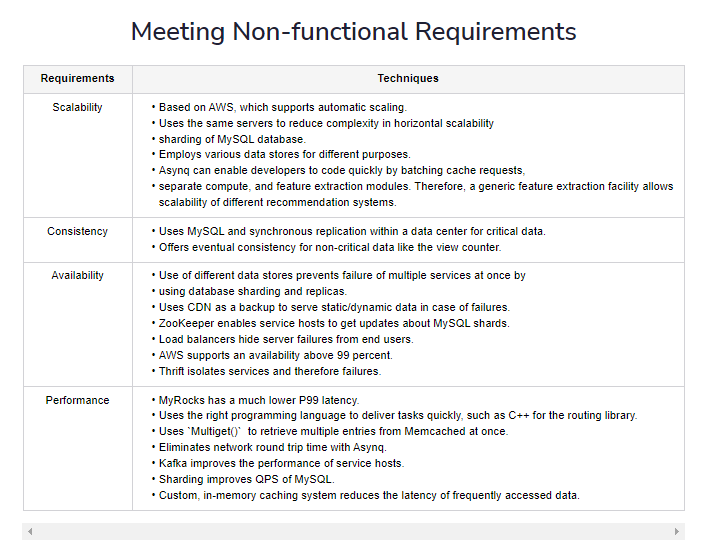


1. Scalability

* **Powerful and homogeneous service hosts:** Quora uses powerful machines because service hosts use an in-memory cache, some level of queueing, maintain manager, worker, and routing library. The horizontal scaling of these service hosts is convenient because they are homogeneous.

1. Consistency
   * **Different consistency schemes** may be selected for different types of data. For example, certain critical data like questions and answers should be stored synchronously.
   * Other data like view counts may not necessarily be stored synchronously because it is not a goal of the Quora service to ensure that all users see the same number of views as soon as the question is posted.
2. Performance

* **Several Datastores for different reasons**
* **Distributed Caches**
* **Kafka** to queue similar tasks and assign them to cron jobs that otherwise take a long time if executed via API calls.



11. Additional Details:

1. Load Balancers:
   * We could use a simple Round Robin approach that distributes incoming requests equally among backend servers.
   * A problem with Round Robin LB is that we do not consider the server load. As a result, if a server is overloaded or slow, the LB will not stop sending new requests to that server. To handle this, a more intelligent LB solution can be placed that periodically queries the backend server about its load and adjusts traffic based on that.
2. Caching:

* **Which Cache**: We can use any off-the-shelf solution like Memcached, which can store full URLs with their respective hashes.
* **How much cache memory should we have?** We can start with 20% of daily traffic and, based on clients’ usage patterns, we can adjust how many cache servers we need.
* **Which cache eviction policy would best fit our needs?** When the cache is full, and we want to replace a link with a newer/hotter URL, how would we choose? Least Recently Used (LRU) can be a reasonable policy for our system.