Evaluation of Quora’s Design

Learn how the proposed design fulfills the non-functional requirements.

We'll cover the following

Fulfilling requirements

We have used various techniques to fulfill our functional requirements. However, we need to determine if we have fulfilled the non-functional requirements. We’ll highlight some of the mechanisms we have utilized to address the non-functional requirements:

- **Scalability:** Our system is highly scalable for several reasons. The updated design uses 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.

  On the database end, our design shards the MySQL databases vertically, which avoids issues in scalability because of overloaded MySQL servers. To reduce complex `join` queries, tables anticipating `join` operations are placed in the same shard or partition.

> **Note:** As mentioned earlier, vertical sharding may not be enough because each shard can grow large horizontally. For large MySQL tables, writing becomes a bottleneck. Therefore, our design may have to adhere to horizontal sharding, a well-known practice in database scaling.

- **Consistency:** Due to the variety of functionalities offered by Quora, different consistency schemes may be selected for different types of data. For example, certain critical data like questions and answers should be stored synchronously. In this case, performance can take a hit because users don't expect instantaneous responses to their questions. It means that a user may get a reply in five minutes, one hour, one day, or no response at all, depending on the user's question and the availability of would-be respondents.

  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. For such cases, eventual consistency is favored for improved performance.

> **Note:** In general, our design is equipped with strong techniques to reduce the user-perceived latency as a whole.

- **Availability:** Some of the main ideas to improve availability include isolation between different components, keeping redundant instances, using CDN, using configuration services like ZooKeeper, and load balancers to hide failures from users.

Why Isolate Services?

Our design, however, lacks any disaster recovery management, which we'll explore in the next section.

- **Performance:** This design has a strong performance because we have employed the right technology for the right feature. For example, we have used several datastores for different reasons. On top of that, we used different distributed caches depending upon the use case and access frequency. Also, we employed Kafka to queue similar tasks and assign them to cron jobs that otherwise take a long time if executed via API calls.

> **Note:** Quora claims that using its custom queuing solution, it can handle roughly 15,000 tasks per second.

Fun Facts

Meeting Non-functional Requirements

Requirements	Techniques
Scalability	• Based on AWS, which supports automatic scaling. • Uses the same servers to reduce complexity in horizontal sca • sharding of MySQL database. • Employs various data stores for different purposes. • Asynq can enable developers to code quickly by batching cache • separate compute, and feature extraction modules. Therefore, a generic f allows scalability of different recommendation syste
Consistency	• Uses MySQL and synchronous replication within a data center • Offers eventual consistency for non-critical data like the view c
Availability	• Use of different data stores prevents failure of multiple servic • using database sharding and replicas.

	• Uses CDN as a backup to serve static/dynamic data in case of • ZooKeeper enables service hosts to get updates about MySQL • Load balancers hide server failures from end users. • AWS supports an availability above 99 percent. • Thrift isolates services and therefore failures.
Performance	• MyRocks has a much lower P99 latency. • Uses the right programming language to deliver tasks quickly, such as C++ • Uses `Multiget()` to retrieve multiple entries from Memcached • Eliminates network round trip time with Asynq. • Kafka improves the performance of service hosts. • Sharding improves QPS of MySQL. • Custom, in-memory caching system reduces the latency of frequently

Disaster recovery

Our proposed and detailed design does not cater to the situation of natural disasters. While we have met other non-functional requirements, durability, fault-tolerance, and availability are incomplete without a disaster recovery management plan. This section will explore some mechanisms that provide resilience against disasters.

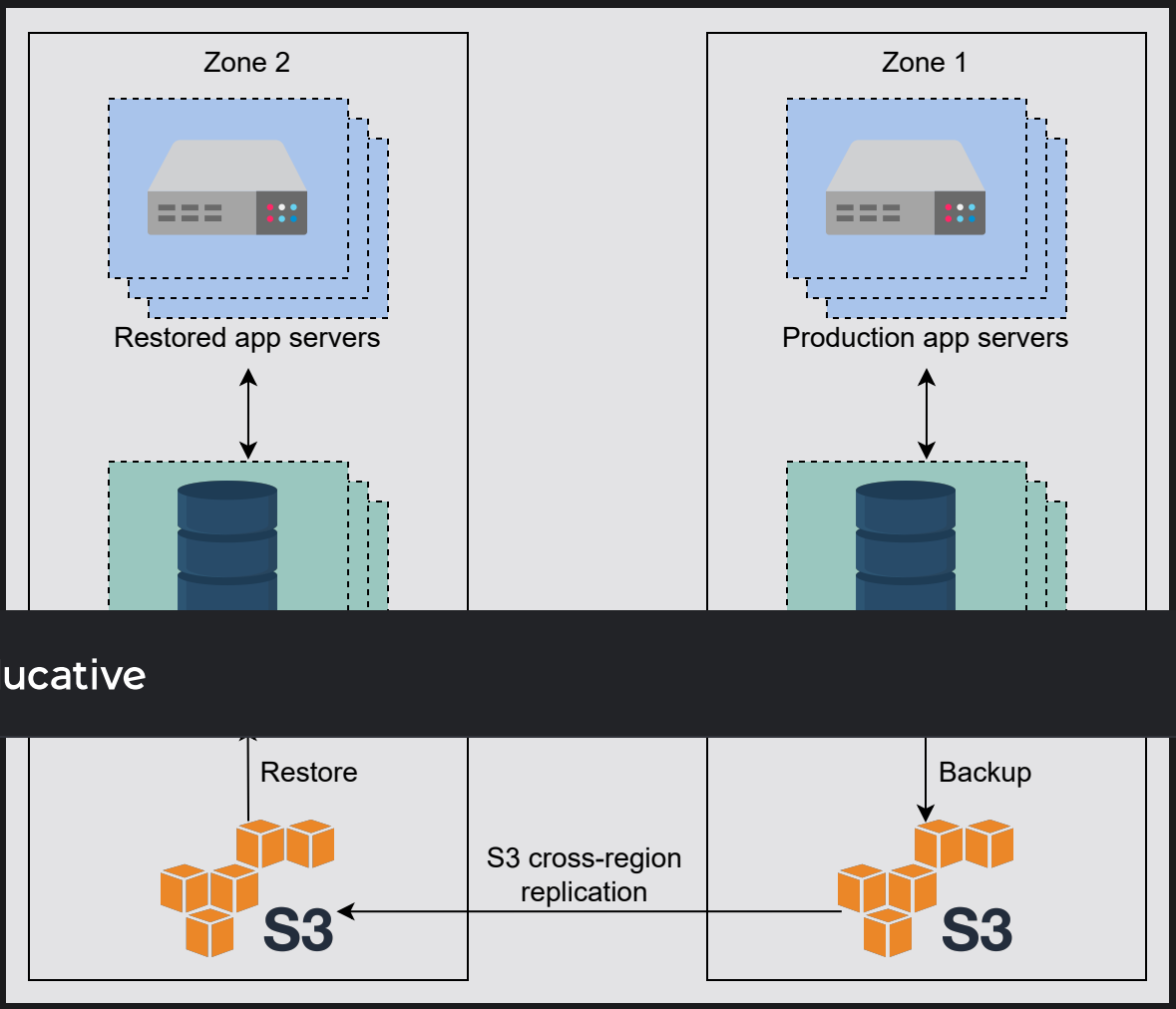
The first and foremost approach of handling a disaster is frequent backups. The frequency of backups depends on the size of the data. Daily backups are suitable for our design because we can backup individual data stores and shards without any hassle. Of course, backups will be stored at remote destinations because natural disasters can wipe out the entire facility at a location.

> **Note:** Taking regular backup at remote locations is not enough. Quick restoration of backed-up data in a timely manner completes a disaster recovery plan.

The following are important questions for designing a disaster recovery plan:

- What data and systems are considered critical to recover from disasters?
- How fast is the restoration from the backup facility?
- Can all systems be recovered through backups?
- How can we deal with potential loss of data that we couldn't replicate before the disaster hit?

The illustration below shows a simple architecture of how a disaster recovery scheme works:



Disaster recovery management

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.

The restoration strategy is simple and effective but it has a couple of drawbacks:

- We might lose some data that is not backed up since we take backups on a daily basis. However, this issue can be mitigated if we do synchronous replication across regions.
- Restoration can take a long time (a few hours), and most databases do not serve queries while data is being recovered.

> **Note:** In general, Amazon provides service with high reliability and availability. For instance, S3 service reports 99.999999999% durability and 99.9% availability over a year.

Conclusion

Throughout this design, we learned how Quora is able to scale its services as the number of users increases. One interesting aspect of the design includes the vertical sharding of the MySQL database. Apart from that, the Quora design discusses a variety of techniques to meet the functional and non-functional requirements. However, our scope did not include the usage of techniques like natural language processing (NLP) to remove spelling mistakes in user's questions or typeahead services during the search.

Point to Ponder

Question

How can using different data stores for different types of data be beneficial in disaster recovery?

Show Answer

Back

Mark As Completed

Next

Final Design of Quora

System Design: Google Maps