System Design (/courses/system-design)
/ Storage Scalability (/courses/system-design/topics/storage-scalability/)
/ Highly Available Database

Highly Available Database

Bookmark

Design a distributed key value store which is highly available and is network partition tolerant

Features:

> *“ This is the first part of any system design interview, coming up with the features which the system should support. As an interviewee, you should try to list down all the features you can think of which our system should support. Try to spend around 2 minutes for this section in the interview. You can use the notes section alongside to remember what you wrote. ”*

Q: What is the amount of data that we need to store?
A: Let's assume a few 100 TB.

Q: Do we need to support updates?
A: Yes.

Q: Can the size of the value for a key increase with updates?
A: Yes. In other words, its possible a sequence of keys could co-exist on one server previously, but with time, they grew to a size where all of them don't fit on a single machine.

Q: Can a value be so big that it does not fit on a single machine?
A: No. Let's assume that there is an upper cap of 1GB to the size of the value.

Q: What would the estimated QPS be for this DB?
A: Let's assume around 100k.

Estimation:

> *This is usually the second part of a design interview, coming up with the estimated numbers of how scalable our system should be. Important parameters to remember for this section is the number of queries per second and the data which the system will be required to handle.*
> *Try to spend around 5 minutes for this section in the interview.*

Total storage size : 100 TB as estimated earlier
Total estimated QPS : Around 100k

Q: What is the minimum number of machines required to store the data?

A: Assuming a machine has 10TB of hard disk, we would need minimum of 100TB / 10 TB = 10 machines to store the said data. Do note that this is bare minimum. The actual number might be higher if we decide to have replication or more machines incase we need more shards to lower the QPS load on every shard.

()

Design Goals:

> “
> **Latency** - Is this problem very latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?). For example, search typeahead suggestions are useless if they take more than a second.
> **Consistency** - Does this problem require tight consistency? Or is it okay if things are eventually consistent?
> **Availability** - Does this problem require 100% availability?
>
> *There could be more goals depending on the problem. It's possible that all parameters might be important, and some of them might conflict. In that case, you’d need to prioritize one over the other.* ”

? ◀ **Q:** Is Latency a very important metric for us?

A: Since we want to be available all the time, we should try to have lower latency.

💬 ()

? ◀ **Q:** Consistency vs Availability?

A: As the question states, we need good availability and partition tolerance.
Going by the CAP theorem (Nicely explained at http://robertgreiner.com/2014/08/cap-theorem-revisited/ (http://robertgreiner.com/2014/08/cap-theorem-revisited/)), we would need to compromise with consistency if we have availability and partition tolerance.
We can however aim at having eventual consistency. As is the case with any storage system, data loss is not acceptable.

💬 ()

● Deep Dive:

> *“ Lets dig deeper into every component one by one. Discussion for this section will take majority of the interview time(20-30 minutes). ”*

> “ Note : In questions like these, the interviewer is looking at how you approach designing a solution. So, saying that I’ll use a NoSQL DB like Cassandra is not an ideal answer. It is okay to discuss the architecture of Cassandra for example with rationale around why some components were designed the way they were..”

Q: Is sharding required?

A: Lets look at our earlier estimate about the data to be stored. 100TB of data can’t be stored on a single machine.
Lets say that we somehow have a really beefy machine which can store that amount of data, that machine would have to handle all of the queries (All of the load) which could lead to a significant performance hit.

> “ Tip: You could argue that there can be multiple copies of the same machine, but this would not scale in the future. As my data grows, its possible that I might not find a big beefy enough machine to fit my data. ”

So, the best course of action would be to shard the data and distribute the load amongst multiple machines.

()

Q: Should the data stored be normalized?
(http://www.studytonight.com/dbms/database-normalization.php
(http://www.studytonight.com/dbms/database-normalization.php))

A: If the data is normalized, then we need to join across tables and across rows to fetch data. If the data is already sharded across machine, any join across machines is highly undesirable (High latency, Less indexing support).
With storing denormalized information however, we would be storing the same fields at more than one place. However, all information related to a

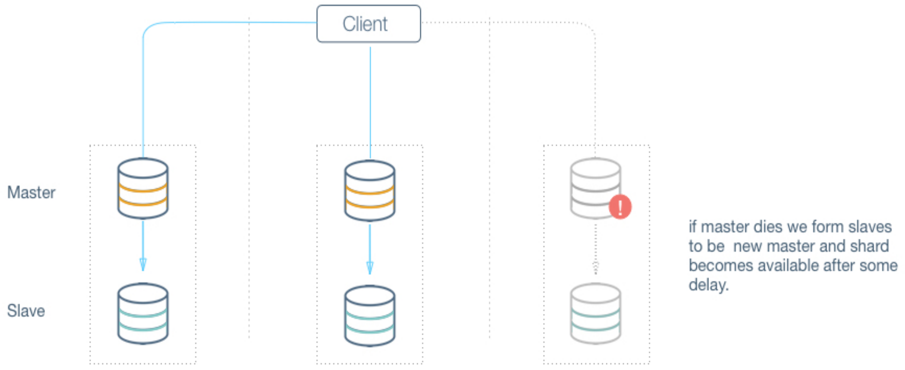
row (or a key) would be on the same machine. This would lead to lower latency.
However, if the sharding criteria is not chosen properly, it could lead to consistency concerns (After all, we are storing the same data at multiple places). However, for this case, we are more concerned with availability and ready to compromise on consistency as long as things become eventually consistent.
In this case, it seems like having denormalized rows makes sharding easier for us and suits our use case better.

💬 ()

❓ ◀ **Q:** How many machines per shard ? How does a read / write look in every shard?

A: Going back to our design goals, low latency and high availability are our design goals.
Lets assume we have somehow sharded the rows into shards. Hence, lets first look at how the architecture might look at within a shard.

Master Slave



if master dies we form slaves to be new master and shard becomes available after some delay.

One simple solution might be to have a master node in each shard which has a slave node which reads all new updates from master and keeps updating itself (The slave in this case might not have the same view as master and would lag a little bit). Clients can read from either the master or the slave depending on which responds earlier (or being slightly more intelligent with the reads to give more preference to the master, and fallback to slave after the read call to master). That could lead to inconsistent views on newer entries across master and client, but would ensure high read availability.
However, what happens to writes when the master goes down. The writes

start failing since only master was taking up the writes.
We can argue that we can have the slave become the new master in such a case. However, even that implies unavailability for the period of failover from master to the slave as new master.
Also, if the slave is lagging a bit, and then the master has a hardware failure, we run the risk of losing data.

This means that we definitely need more than one machine taking the write traffic if we are to be available all the time.

Multi Master

Lets say we modify the previous design where both machines accept write AND read traffic. Lets name the machine m1 and m2.
If m1 accepts write without depending on m2, then it is possible m1 is doing write on a row state which is not the same as m2. That could lead to huge consistency concerns and the DB might become forever inconsistent. DBs might get operations out of order and it can cause eventual inconsistency if the order of operation matters (double the column value and then increment it vs increment the column value and then double it).

From above examples we see that any system with a *master* node will not be highly available, therefore we move to peer to peer systems.

()

Q: Can a peer to peer system be highly available in case of a DB machine dying?

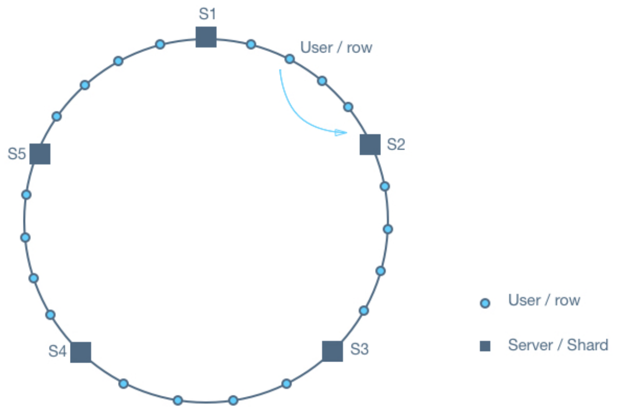
Hint: Single point of failure!

A: We define a peer to peer system where every node is equally privileged and any two nodes can communicate. Yes, since we don't have a single point of failure anymore, therefore our system can theoretically be available even in presence of dying DB machines. Dynamo and Cassandra are examples of examples of such systems, both of them lack the *master* node and therefore have no single point of failure. Our highly available datastore will be highly based on Dynamo and Cassandra, as a reader you don't need to know about them.

()

Q: How will we exactly shard data for a peer to peer system?

A: Refer to https://www.interviewbit.com/problems/sharding-a-database/ (https://www.interviewbit.com/problems/sharding-a-database/) for a detailed answer.

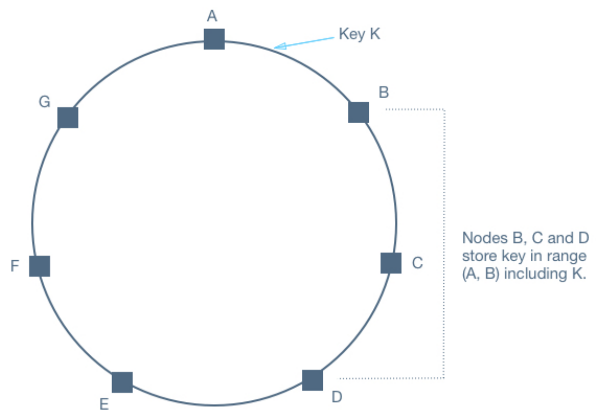


()

Q: How do we store redundant data?

A: We will need to store duplicate data to prevent data loss in case of some of our DB machines getting corrupted. To store the data we can use consistent hashing which assigns every data to a particular node on the ring. Let's call *P* as our replication factor(we will store *P* copies of data). Now for a data D, we have to choose *P* nodes where we will store copies of D.
Now, how do we choose these P nodes? We will choose the *P* clockwise consecutive nodes starting from the node where D is mapped by our hashing function.
An important point to dicuss here is that even though any data might be mapped to a particular virtual node, the virtual node is not the master node for this data for either read or right. A client can request to read or write a data from any node they want. This is essential in creating a

highly available system.



()

Q: How does a write/read happen in our system?

A:
Write request:
A client can contact any node in the ring with a put() request to write data, this node acts as a coordinating node for this request. Coordinating node then forwards the request to the mapping nodes for the data(hashing) and waits for acknowledgement from them. When it receives **W**(explained later) acknowledgements, it returns a write-accepted message to the client.

Read request:
To perform a get() request, client can connect to any node in the ring which becomes the coordinating node for this request. The coordinating node then asks all replica nodes for this data and returns consolidated data to the client when **R** of them replies back.

Read and Write consistency:
W and **R** are called write and read consistency number respectively. To recap, **W** is the minimum number of nodes from which the coordinating node should get an ack before making a write successful and **R** is the minimum number of nodes from which the coordinating node should get back read values to return them back to the client.
R, W together forms quorum of the system. For a read to be consistent(return the latest write), we need to keep **W + R > P**.
Depending on the feature requirement W and R can be adjusted, for

example to have very fast writes we can keep **W = 1** and **R = P**. If our system is read heavy we can keep **R = 1** and **W = P**. If read and write are equally distributed, we can keep both R and W as (P+1)/2.

()

Q: What if a machine goes down?

A: Since no node is only responsible for a piece of data, it's going down won't really affect our writes/reads. As long as **W** out of P nodes are available for some key, it can be updated(similarily for read).

Note that in case of less than **W** nodes available to write for some data, we can relax our write consistency(sacrificing data consistency for availability).

()

Q: What kind of consistency can we provide?

A: If we keep **W = P**, we can provide strong consistency but we won't be available for some writes even if one of our DB machine dies.

Earlier we saw in master-master configuration that in network partition cases, our masters may diverge to provide availability. In our current system, essentially all of our nodes are master and the point that they will diverge should be taken for granted and we should build our system considering it.

Therefore we should build for the case where **W** is less than P, hence our writes will be propagated i.e. some machines will have an updated view of data and some will not, therefore they are not consistent. The best we can guarentee here is eventual consistency, that is in due time, all the changes will be applied to every server and in the end they will all have a consistent view of the data.

To achieve eventual consistency, we need to be able to resolve differences between data on two servers. There are a couple of detect and resolve data conflicts that may arise.

First, if data(key, value) we store is such that value is just a single column, we can use a simple criteria of LWW(last write wins) to resolve conflicts. So if two servers have different view of a key, in the resolve step we can update the server with the stale with the new data and therefore become consistent.

The other way is to store augmented data for each row indicating all the coordinating nodes for the row till now. Now, to detect and understand conflict we can compare the augmented data. If one is a subset of the other(all the writes seen by one of the row has been seen by the other row) we can safely ignore the one with smaller augmented data. Otherwise, we have a conflit for our row and need application level logic to resolve it. This way is usually required when our value if composed of more than one independent column.

💬 ()

🏆 You have now mastered this problem!

Discussion

Post a comment (https://discuss.interviewbit.com/session/sso?return_path=https://d

mitesh_joshi almost 2 years ago

How exactly do peer-to-peer systems work? Doesn't there still have to be a high-l 3

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/how-exactly-do-peer-to-peer-systems-work-doesnt-there-still-have-to-be-a-high-l)

sumit_007 almost 4 years ago

For a read to be consistent(return the latest write), we need to keep W + R &gt; 2

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/for-a-read-to-be-consistent-return-the-latest-write-we-need-to-keep-w-r-gt) sunnyk over 3 years ago

One thing I don't get is previously it mentioned 10K QPS and the size of one valu 2

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/one-thing-i-dont-get-is-previously-it-mentioned-10k-qps-and-the-size-of-one-valu)

tusharbisht almost 4 years ago

Does P, W and R predefined ? Or they change dynamically as per new machines addit 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/does-p-w-and-r-predefined-or-they-change-dynamically-as-per-new-machines-addit)

guest_user_251 over 3 years ago

You say " we would need to compromise with consistency if we have availability an 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/you-say-we-would-need-to-compromise-with-consistency-if-we-have-availability-an)

henry_henry almost 4 years ago

I see, I was originally a bit confused about consistent hashing using peer-peer s 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/i-see-i-was-originally-a-bit-confused-about-consistent-hashing-using-peer-peer-s)

ran_yu about 3 years ago

Nice course. I only think it is a little bit long 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/nice-course-i-only-think-it-is-a-little-bit-long)

mitesh_joshi almost 2 years ago

We are writing multiple copies of data on different nodes and since we are doing 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/we-are-writing-multiple-copies-of-data-on-different-nodes-and-since-we-are-doing)

DroidX86 10 months ago

Size of the value for a key can increase as mentioned earlier. But the solution w 1

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/size-of-the-value-for-a-key-can-increase-as-mentioned-earlier-but-the-solution-w)

aman-dubey_976 11 months ago

Master Slave Architecture 0

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/master-slave-architecture)

subithal-subbaiah 3 months ago

Multi masters in storage system 0

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/multi-masters-in-storage-system)

Blog (https://blog.interviewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |

Contact Us (/pages/contact_us/) | Terms (/pages/terms/) | Privacy Policy (/pages/privacy/)

Scaler Academy Review (/scaler-academy-review) |

System Design Interview Questions (/courses/system-design/) |

Google Interview Questions (/google-interview-questions/) |

Facebook Interview Questions (/facebook-interview-questions/) |

Amazon Interview Questions (/amazon-interview-questions/) |

Microsoft Interview Questions (/microsoft-interview-questions/) |

SQL Interview Questions (/sql-interview-questions/) |

Python Interview Questions (/python-interview-questions/) |

Javascript Interview Questions (/javascript-interview-questions/) |

MVC Interview Questions (/mvc-interview-questions/) |

React Interview Questions (/react-interview-questions/) |

Angular Interview Questions (/angular-interview-questions/) |

Directi Interview Questions (/directi-interview-questions/) |

Yahoo Interview Questions (/yahoo-interview-questions/) |

LinkedIn Interview Questions (/linkedin-interview-questions/) |

VMware Interview Questions (/vmware-interview-questions/) |

eBay Interview Questions (/ebay-interview-questions/) |

Flipkart Interview Questions (/flipkart-interview-questions/)

Like Us (https://www.facebook.com/interviewbit) Follow Us (https://twitter.com/interview_bit)

Email (mailto:hello@interviewbit.com)

Got suggestions ? We would love to hear your feedback.