Highly Consistent Database

Bookmark

Design a distributed key value store which is highly consistent 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?
Anwer: 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 10M

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.

2 ()

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: No, but it would be good to have a lower latency.

0 ()

Q: Consistency vs Availability?

A: As the question states, we need tight consistency and partitioning. 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 availability if we have tight consistency and partitioning. As is the case with any storage system, data loss is not acceptable.

0 ()

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 HBase is not an ideal answer. It is okay to discuss the architecture of HBase 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.
Let's 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.

0 ()

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

Q: Can I shard the data so that all the data required for answering my most frequent queries live on a single machine?

4 ()

A: Most applications are built to store data for a user (consider messaging for example. Every user has his / her own mailbox). As such, if you shard based on every user as a row, its okay to store data in a denormalized fashion so that you won't have to query information across users. In this case, lets say we go with storing data in denormalized fashion.

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).

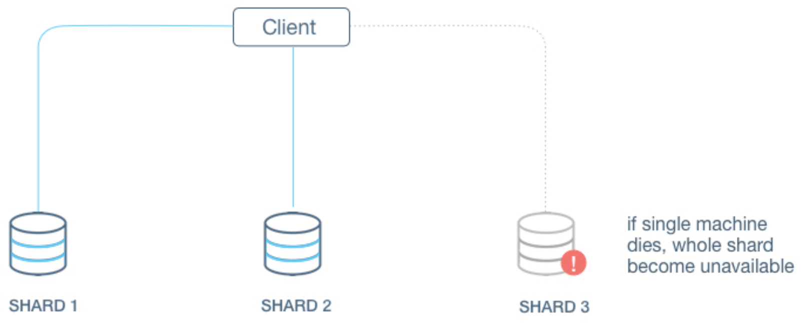
3 ()

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

Q: Can we keep just one copy of data?

1 ()

A: Since there is only one copy of the data, reading it should be consistent. As long as there are enough shard to ensure a reasonable load on each shard, latency should be acceptable as well. Reads and writes would work exactly how they work with a single DB just that there would be a row -> shard -> machine IP (Given a row, tell me the shard it belongs to and then given the shard, give me the machine I should be querying / writing to) resolution layer in between.
There is just one tiny problem with this model. What if the machine in the shard goes down? Our shard will be unavailable (which is fine as governed by the CAP theorem). However, what if the machine dies and its hard disk becomes corrupt. We suddenly run into the risk of losing the data which is not acceptable. Imagine losing all your messages because your shard went down and the hard disk got corrupted. That means we definitely need more than one copy of data being written with us.



Q: What problem may arise if we keep multiple copies of data?

3 ()

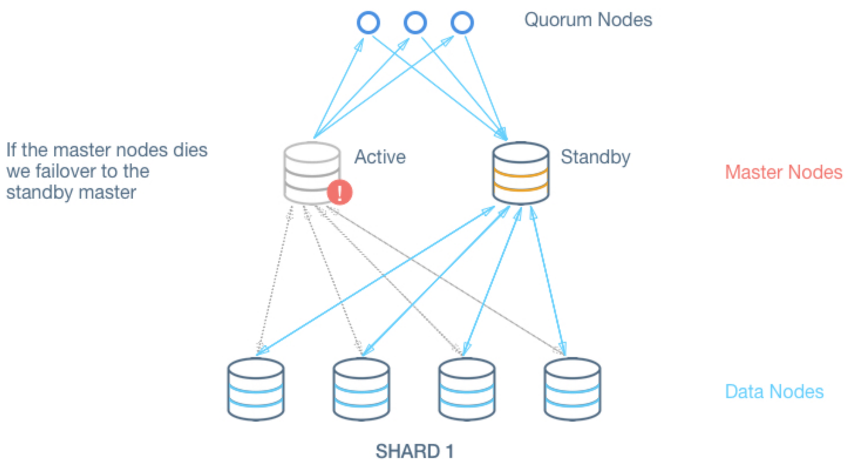
A: Let's say we keep 3 copies of data (The probability of all 3 machines dying and having a corrupted disk is negligible). Now, the issue is how do we maintain all of the copies in absolute sync (Consistency, remember?).
One naive way would be that a write would not succeed unless its written to all 3 copies / machines. That would make our write latency go up significantly apart from making writes very unreliable (My write fails if it fails on any of the machines). Let's see if we can make this a bit better.
If we have to allow writes succeeding even if the write has been written on a majority of machines (2 out of 3, let's say), to maintain consistency, its important that there is a master machine which keeps track of this information. This master machine can track which machines have a particular block in each shard. This means that every read will go through this master machine, figure out the machines with the block and query from the required block. The machines which do not have the block can check with this master machine to see which block are not present on it, and catch up to replicate the block on it.

However, now if this master machine dies, our whole shard is unavailable till this machine comes back up. If this machine has a corrupted hard disk, then the unavailability becomes indefinite (Note that we do not loose data in this case, as total data is the union of data present on 3 nodes). This is not an ideal design, but we will talk about improvements to it later in this question.

Q: What if the master keeping track of where the blocks are stored dies?

5 ()

Anwer: To overcome this problem we keep a standby master which in the failover process becomes the acting master and keeps unavilability to minimum. Now, to keep the standby master upto date we can have a shared network file system. When any namespace modification is performed by the Active master, it durably logs a record of the modification to an edit log file stored in the shared directory. The Standby node constantly watches this directory for edits, and when edits occur, the Standby node applies them to its own namespace. In the event of a failover, the Standby will ensure that it has read all of the edits from the shared storage before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.



A: Going back to our design goals, latency and consistency are our design goals.
A simple way to resolve this is to make sure we only have one machine per shard. Reads and writes would work exactly how they work with a single DB. However, if the machine holding the only copy dies and its hard disk becomes corrupt, we suddenly run into the risk of losing the data which is not acceptable. That means we definitely need more than one copy of data being written with us. Lets say that number is 3. Now, the issue is how do we maintain all of the copies in absolute sync (Consistency, remember?).
One naive way would be that a write would not succeed unless its written to all 3 copies / machines. That would make our write latency go up significantly apart from making writes very unreliable (My write fails if it fails on any of the machines).
If we have to allow writes succeeding when the write has been written on a majority of machines (2 out of 3, lets say), to maintain consistency, its important that there is a master machine which keeps track of this information. This master machine can track which machines have a particular block in each shard. However, now if this master machine dies, our whole shard is unavailable till this machine comes back up. If this machine has a corrupted hard disk, then the unavailability becomes indefinite.

There are couple of ways to keep unavailability to minimum using a standby master keeping track of master node data through a shared file system(Explained in detail in the last hint).

0 ()

You have now mastered this problem!

Discussion

_aman (/profile/_aman) about 1 year ago
-1 fault tolerant
distrinuted
reply

shyam_ (/profile/shyam_) about 1 year ago
-1 what if one of the node in a shard goes down and comes online after sometime. Will the master invalidates the entries belong to the that particular node?
reply

shubham_aggarwal_496 (/profile/shubham_aggarwal_496) 3 months ago
0 consistent hashing
reply

raviteja_yakkaladevi (/profile/raviteja_yakkaladevi) about 1 year ago
3 What are the cons if we use last modified time(solution given in highly available database with W+R>P) for knowing where the block is instead of maintaining a master?
reply

KMeshU (/profile/KMeshU) 10 months ago
0 What if the last modified node itself goes down
reply

raviteja_yakkaladevi (/profile/raviteja_yakkaladevi) 10 months ago
1 W+R>P. This ensures the consistency even in the case that you were saying. You can read it here https://www.interviewbit.com/problems/highly-available-database/
reply

piscado (/profile/piscado) 5 months ago
2 i think this "master" (or i'd say logger) simply acts like a write lock. for the previous high-A architecture, a read can be done in the process of a writing and gives stale data. in this case, on the other hand, we always know if the writing is finished since every read is going through the "master"
reply

sarang (/profile/sarang) about 1 year ago
3 how at paxos or raft?
reply

sarang (/profile/sarang) about 1 year ago
0 abt*
reply

steven_chow (/profile/steven_chow) 8 months ago
0 What are the advantages for this solution comparing with the high availability solution with Casandra when R == W == Replication Factor?
reply

resurrectedVaibhav (/profile/resurrectedVaibhav) 7 months ago
3 Cassandra/Dynamo solution is actually a complex solution and is focussed on business scenarios where writes are very frequent.
That solution is not consistent in case of failures if you notice. I think the only way to make it consistent is to have a higher value of W + R thus decreasing the performance. Also, not to forget that to resolve vector-clock conflicts we need a complex algo in the client.

Though here the load is not truly distributed.

reply

shiyi_chen (/profile/shiyi_chen) 8 months ago
0 distributed transaction
reply

samira_allahverdiyeva (/profile/samira_allahverdiyeva) 7 months ago
2 first, you said QPS is "Let's assume around 100k"m in the Assumptions part you say "Total estimated QPS : Around 10M ". I didn't get this part
reply

sunnyk (/profile/sunnyk) 6 months ago
0 I agree, there is inconsistency in this requirement I think. would be safe to assume one and go ahead with that.
reply

howard_liu (/profile/howard_liu) 2 months ago
0 another reason peer-to-peer system's consistency might break even with W + R > N: https://stackoverflow.com/q/42706811/3024143 In general, it seems to me these systems are designed for availability, not strong consistency.

the proposed every-read-go-through-this-master-machine solution looks unlikely to me, as it could easily overload the master (as opposed to in GFS every client query chunk location once and cache it.), not to mention master's responsibility to sync each shard.

Pinterest's sharding method (https://medium.com/@Pinterest_Engineering/sharding-pinterest-how-we-scaled-our-mysql-fleet-3f341e96ca6f) looks highly consistent to me, as it maintain single master per shard. When master dies the slave is promoted, which is still consistent despite some data between the lag will be unavailable. https://stackoverflow.com/a/25691863/3024143
reply

Write your comment here. Press Enter to submit.