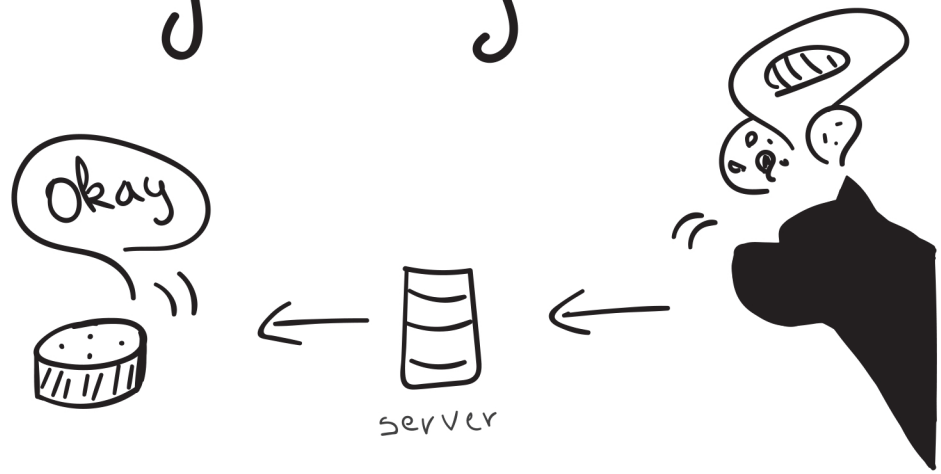
This zine was created as a part of the creative project assignment in CS 128@UCSC taught by Lindsey Kuper!

- to find out more about the class, visit composition.al
- to find out more about me, visit medvetska.com

Gabriela Medvetska, 2019

Wait what, did Alexa just take an order from my dog?



About me

Hi! I am Gabriela and this is my dog Ronnie who learned how to communicate w/ AI!

Jk! The only thing he did is tell Alexa to buy a $300 "premium grande large breed kibble"!

I contacted Amazon and returned it but the mystery persisted...

I investigated futher and found out that my partner (who is in France) put it in our cart but removed it since!

Whew! My dog is not taking over the world!

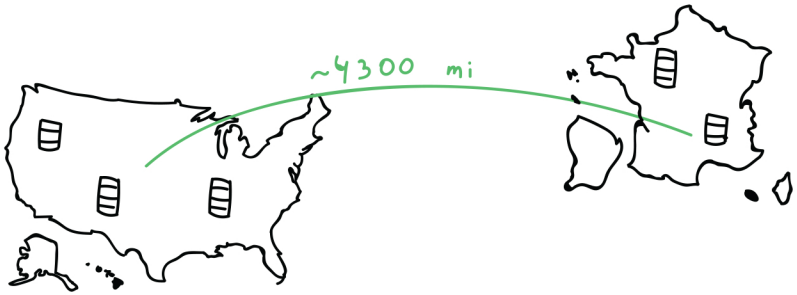
Work cited

1. De Candia, Giuseppe et. al. "Dynamo: Amazon's Highly Available Key-value Store"

Additional resources

- Definitely the original paper itself @ allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- Christopher Batey's talk about Dynamo @ vimeo.com/showcase/4414343

Having said that, I think it's clear now that my order was probably stored on a server in the US, while my partner was busy with the cart abroad in France and her changes propagated to me in the wrong order!



I think it makes sense to store both states of the cart but maybe there is a way to detect such changes and notify the user in advance!

That solved one issue!

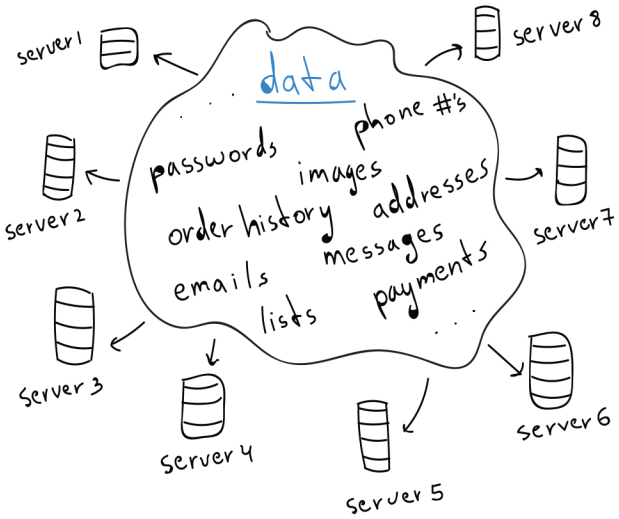
But how can the shopping cart get it wrong in the 1st place?

In this zine I'll try to explain how Dynamo, a key-value store system that powers Amazon's shopping cart and other services, operates and why situations like this happen!

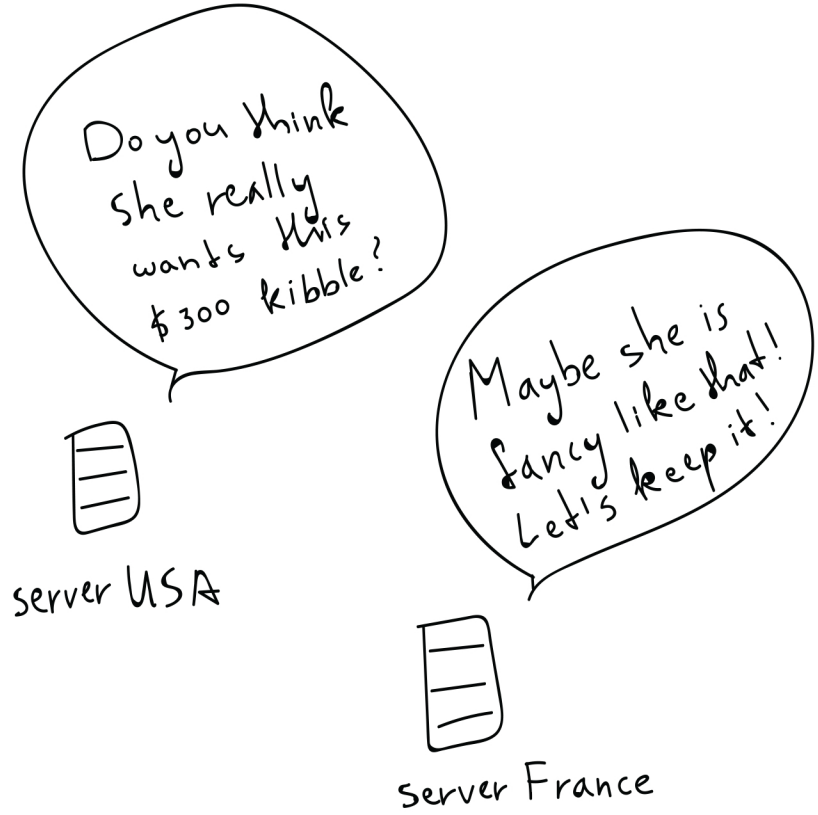
Contents

Consistent Hashing

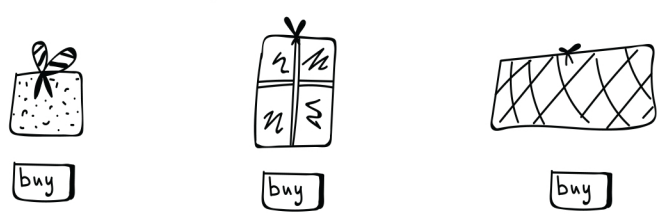
To get a better understanding of how Dynamo works, we must first explore the core technology behind its data store, consistent hashing. To maximize the number of requests that receive a response (availability) of the system, we can partition the data and store it on different servers using a hash function.



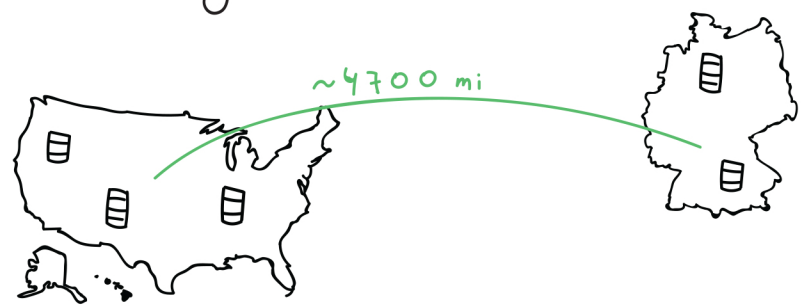
Dynamo leaves it up to the client to resolve issues with the inconsistent data[1]. In regards to shopping cart it's better for both the customer and the company to keep the removed item in the cart in case of uncertainty.



Amazon's shopping cart service trades consistency for availability[1]. In fact, in all of the distributed systems (aka in systems where data is distributed between multiple machines), you have to choose whether to enforce consistency, making sure your data is the same on all machines at all times (which could take a long time to happen), or to enforce availability, prioritizing being able to add and remove items from the cart even during the busiest holiday shopping season.

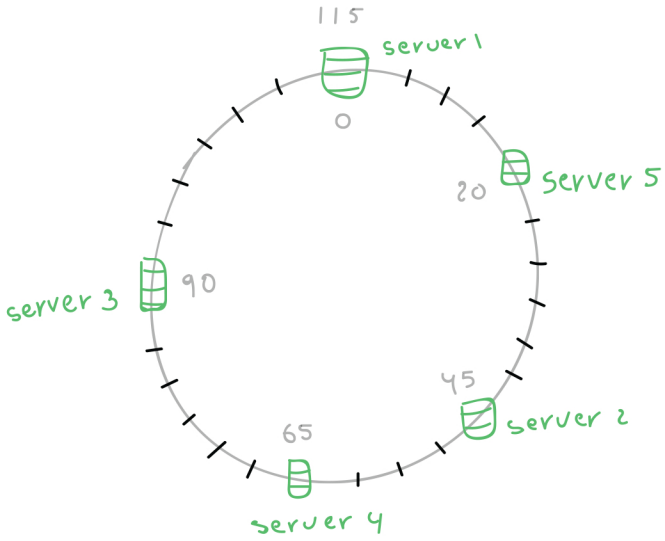


If our hash function is good (it better be!), then it would generate hashes and distribute them in a way that would make it faster for clients to access it. For instance, data pertraining to users in USA could be stored on servers that are located here, while German users' data - on servers in Germany!

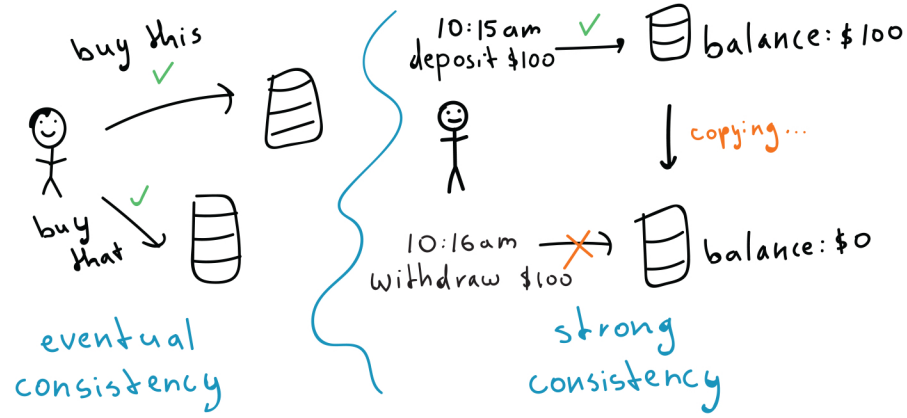


But what if one of the servers fails or a new one is added?

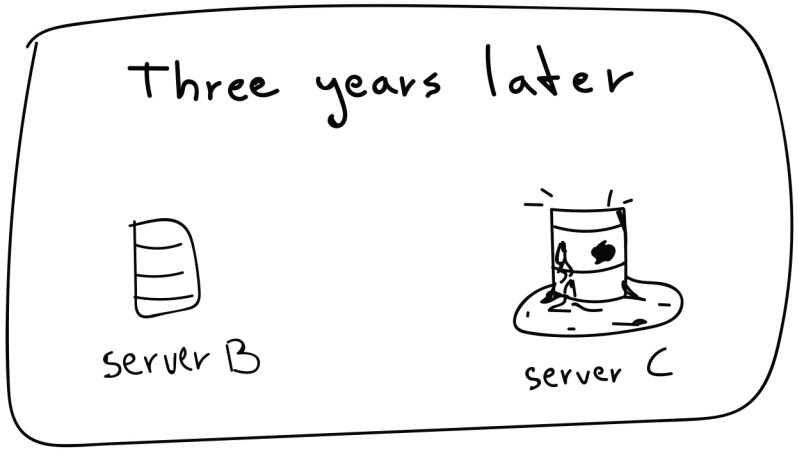
Our hash function would have to re-generate new hash values and and distribute them all over again. With millions of records on a platform like Amazon that could be very expensive in terms of money and time. To ensure that the shopping cart service is highly scalable (servers could be added and deleted all the time), Dynamo uses consistent hashing.



The eventual consistency model states that eventually all servers will have the same data on them, so it permits Dynamo to be an always-on writable system. On the other hand, in the strong consistency model, client has to wait for all nodes to update their values before they could write anything new. Strong consistency is important in systems that power banks where the events are order (time) sensitive.

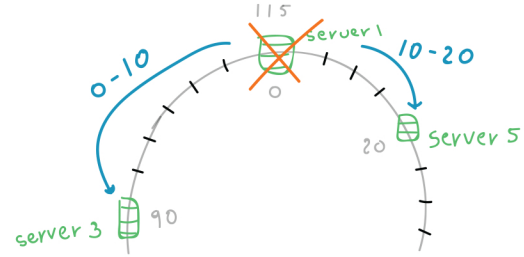


Of course, if B's successor C or D is down, it has to wait before it can pass updated values. A lot of time can be wasted if B keeps waiting...

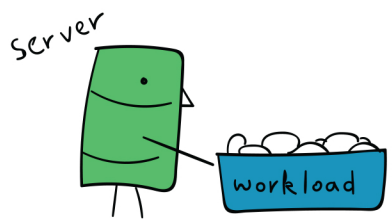


That's why Dynamo follows the eventual consistency model instead of strong consistency.

Consistent hashing is a method of hashing which uses a ring-like structure, where each server is randomly located on the ring and is responsible for a particular range of data keys. For instance, if we have a ring of 115 keys (in reality it's a lot more than that!), server 1 could be responsible for ranges from 0 to 20, server 5 from 20 to 45, server 2 : 45 - 65, server 4 : 65 - 90, and server 3 : 90 - 115. Note the randomness! If server 1 fails, the keys it is responsible for (0 - 20), will be distributed among its neighbours, server 5 and server 3.



If a new server is added then it picks up about the same workload from the other available nodes (in our example, 20-25 keys) and gets its own spot on the ring.



The main feature that makes this method faster than regular hashing is that only neighbouring servers are affected on change. There is no need to re-generate hashes for every single server in the system which increases its availability! This, in turn, makes the system more complex and harder to maintain. Let's see how Dynamo deals with backups!

Dynamo has to make copies of its data (replications) all the time in order to ensure high availability and durability (preservation of data).

If we have data we want to read from server A, but the server is down, we could still read it if other servers have a copy of it. When data is written to a particular server B, B is responsible for sending it to its successors for replication in addition to storing it locally on B.