Lecture 6
Thursday, 17 October 2024 3:04 PM

Key aspects of DynamoDB.
- predictability (whether the query will be executed in a few milliseconds).
- Scalability
- Availability (You don't want failures in your system to take down your database).
- Consistency

These were the reason why people choose DynamoDB & NoSql in comparision to relational databases.

Predictability.

The goal of DynamoDB is for each operation to return within a few milliseconds, regardless to the size of table. The way to do that; is 2 key properties: API (key based queues), resource provisioning (enough resources). DynamoDB is a key-value store.

Get / Put / update / Delete Operations

If you get a "Get" or "Put" in the system of DynamoDB goes across comes in from the application running on some client machine, connects to a DynamoDB point, goes across the network, hits one of the front ends, which is called a request router; that sends the operation to the storage node.

The request router does authentication to make sure that the person who's doing the request is authorized to do that operations.

Issue - Multi-tenant Servers.

The key issue from a provisioning point of view is that the DynamoDB servers are multi-tenant. (i.e. 2 different tables stored on the same server).
(could be different customers)

Solution: Resource Reservation.

Customer A gets certain resources on the server & Customer B get certain. Customer A B is not allowed to use customer A's resource & vice versa.

In Dynamo, they allow customers to tell the no. of reads & writes they want to do to the individual tables, & this is done in the form of read controls & write controls (RC & WC).

RC & WC works in the same way.

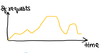
A customer will inform they want 100RC & the pay for 100 RC, & dynmoDB making sure that customers has enough dedicated memory & have dedicated CPU, enough dedicated IO capacity to do their 100 reads per second

Token Bucket algorithm.

There's a set of tokens that control that gets fed at a rate that corresponds to how many RC use the customer has paid for. So if a customer has paid for 100 RC, then 100 token gets put into the buckets per second. Token bucket is a way of guaranteeing that these are resources on the machine & that customers don't interfere with each other.

Non Uniform request distribution over time.

Customers workload where they're low for some part of the data & some part of the day they have spike & they go back to nothing again.



Solution: Over-provisioning

Assuming more RC to avoid the issue of throttling.

Bursting

Change the size of the token buckets; changed from seconds to 5 minutes. If you're underspending reads per seconds your tokens, they just get banked up, you can save them & use it when you have these burst of requests, up to 5 minutes.

Scalability

DynamoDB has hundreds of thousands of servers that are scattered over various regions.

Hash-Ranged Partitioning.

DynamoDB has to partition a table, as the table grows it is partitioned across multiple servers so that operations can be done parallely. Partitioned by customer id; use one way hash function & then partition by hash ranges.

Provisioning.

RC use provide a nice job of isolating customers that are on the same storage node, but now that tables are partitioned across various storage nodes. Solution is to take the token bucket & put it outside of the storage nodes, in the request routers where all the results & all the requests flow. However, there are 10,000 requests routers & request can go to any router. ∴ built a token bucket service.

request comes in → gets a bucket from token service → use the set of token for it's request → once, the token over go back get another → tokens not used, return.

Availability

Every item in DynamoDB, every partition of every table is replicated across 3 storage servers, available in separate availability zone (at 3 zones). Partion Map's in DynamoDB is used to keep a track of where the data is stored. The partition Map is a metadata that's available to every request router. For 'put' request you have to write to 2 out of 3 to be acknowledges, one of those replicas you write has to be leader. Leader can later send it to the 2 secondaries. If leader fails it will send a periodic heartbeat to the other 2. If leader expired then election protocol. In region failure; replicate table across regions, Read & write anywhere, eventual convergence & last-writer-wins conflict resolution.

Use Cases: Disaster Recovery.

Customers often like to read & write in one region & use the second region just as a standby, & later will be updated, The only downside of it is now you're paying cross-region latencies to do this 'get' operation.

Use case: Multi-Access Region.

Global tables allow that because any body can read & write in any other regions in which the data-exists.

Intra Region VS Cross-region Replication.

- It changes the fundamental property of DynamoDB, if replicated across regions then you have issue of whats known as blast radius, that brings down multiple regions at the same time.
- It causes algorithm timeout.

Instead of that; an asynchronous replica is stored across regions.

Intra-region → synchronous replica.

Cross region → asynchronous replica.

Consistency

Since the data is consistent across all the storage nodes, when we do a get operation; Dynamo has 2 options:

- strongly consistent get.

guaranteed to recieve the lastest value that was written, as it goes to the primary server.

- eventually consistent get.

Because it's faster & infact cheaper, as it can go to any of the 3 replicas.

DynamoDB Transaction Architecture.

Among the storage nodes involved in the transaction, the storage node for key A and key B, they do 2 phase protocol, the first phase of the protocol unlike its not your 2 phase commit protocol, your 2 phase commit protol says are you prepared to commit? The reason we need 2 phase is because we have checks in the transaction, making sure all the checks succeed If any checks fail, then we have to abort it.

Transaction ordering.

In transaction protocol is you don't necessarily have to executed transactions in the commit order, as long as the physical or in which you execute them have the same effect as executing them in the time stamp order.

There are 2 types of ordering:

- Transactions arrive at a server
- Transactions are ordered by the their signed timestamps