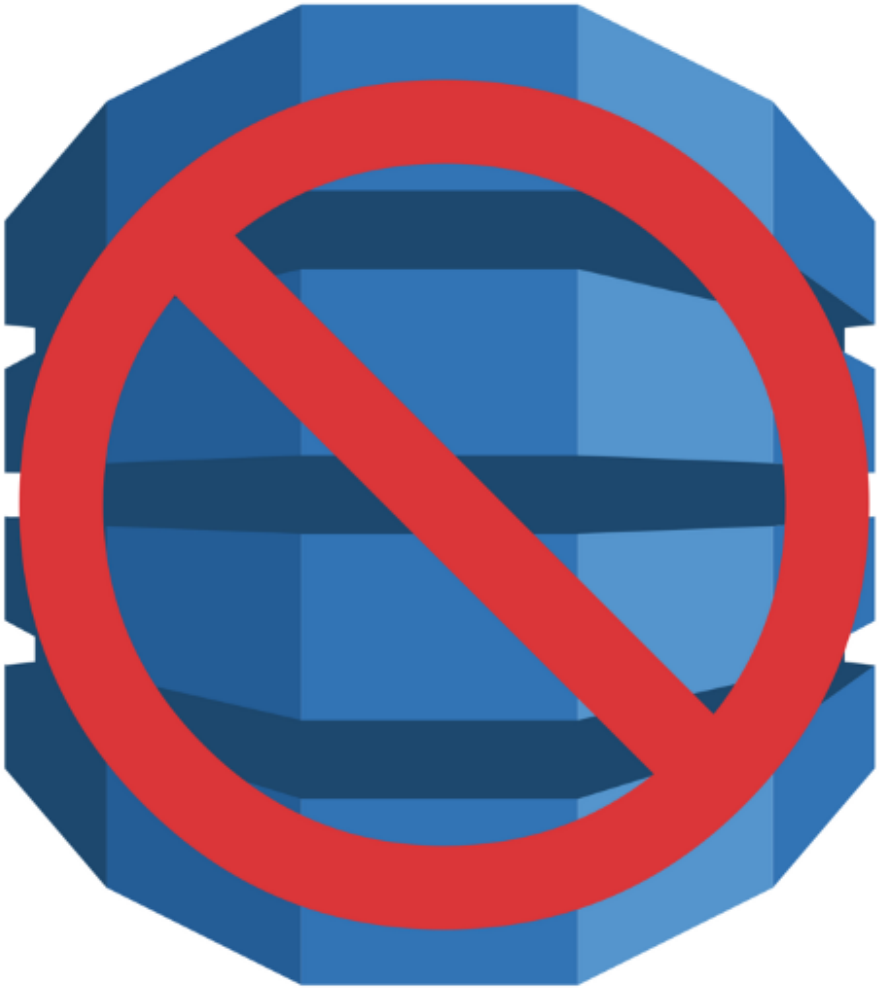
You probably shouldn’t use DynamoDB

Jono MacDougall Follow
Jun 30, 2017 · 5 min read



Edit: This article is firmly out of date now. I have not gone back to try dynamoDB but I have heard they have changed how throughput is calculated and mitigated this issue. So take what is below with a grain of salt!

Avid readers of the Ravelin syslog will remember a story from last year about our use of DynamoDB. It outlined a few gotchas that we experienced and gave advice on different uses cases for the product. The conclusion of the article was that it is possible to build a workable system as long as you are careful and avoid certain traps that we initially fell into. Given a few more months experience (and a lot more data) I have to backtrack on most of that article. I now think there are very few use cases that DynamoDB is suitable for and there are many alternatives that are far better and cheaper.

DynamoDB Doesn't Scale

Well, it does. But it is functionally very challenging to do with large data volumes.

The problem is the distribution of throughput across nodes. DynamoDB works by allocating throughput to nodes. The throughput is set up as follows:

- Each write capacity unit gives 1KB/s of write throughput
- Each read capacity unit gives 4KB/s of read throughput

This seems simple enough but an issue arises in how dynamo decides to distribute the requested capacity. It distributes it evenly across your nodes. So if you provision 100 write capacity units and you have 4 nodes, each node gets 25 write capacity units. The trouble arises because you cannot control how many nodes you have. That is decided by your data volume. For every 10GBs of data you get a new node. So if you have some number of terabytes of data you are going to have hundreds of nodes in your DynamoDB cluster.

The requested throughput is then distributed to these nodes. So if you request a write capacity of 1000 and you have 200 nodes what this really means is that each node only has a write capacity of 5. That is pitifully small and will cause throughput exceptions.

> DynamoDB is an unforgiving beast

Of course the answer to this is the answer to every other NoSQL solution out there. Distribute your keys. Most NoSQL solutions use the given key to decide which node to put that data on. This means that when you want to read or write to that data the system just looks at the key and then knows where to find it. As long as no single key (or no group of keys that point to the same node) is accessed too quickly then you will get the requested throughput. This is true. But DynamoDB is an unforgiving beast and what it considers a hot key is likely not what you consider a hot key.

The hottest of keys

Consider the following situation.

You construct a table which uses a customer ID as partition key. You know your customer ID's are unique and should be uniformly distributed across nodes. Your business has millions of customers and no single customer can do so many actions so quickly that the individual could create a hot key. Under this key you are storing around 2KB of data.

This sounds reasonable.

This will not work at scale in DynamoDb.

When a customer does an action, it is reasonable to think that this kicks off a number of actions in your system. One action might spawn a number of events to be sent into the backend. Or perhaps your frontend is intelligent and batches events together. During a time when a customer is actively engaging in your application, there is a good chance the key for that customer ID is going to be more active. Potentially getting 1–5 requests per second for a given second but certainly not a sustained load of that.

This is not a hot key by any definition. But DynamoDB will fail.

As each of your nodes will only be allocated a small fraction of your total throughput this will cause a throughput exception in DynamoDb. You will have to overprovision by many multiples to avoid the problem and this causes costs to go through the roof.

Where DynamoDB does work

There are many use cases where DynamoDB is likely a good solution. If you know your dataset size is relatively small or your query distribution is high enough that the above scenario doesn't apply to you then DynamoDB could help you solve problems. I have

seen it used as a method for deduping where data is not stored long term so the dataset size does not grow. It is a fast and highly available tool if you can get around the issue I mentioned above. This article is pretty damning of DynamoDB but it really should be read as an indictment of our particular expectations of the database. If you set your expectations differently and work within the restrictions you might find a good use for the product.

Alternatives

There are many NoSQL alternatives and I won't list them here. I will mention another managed solution that is far better than DynamoDB.

Google Bigtable.

Ravelin has recently moved to GCP and a big motivation for that was Bigtable. We will have a blog post discussing the move coming soon but I will briefly discuss our experience with Bigtable here. Before we moved cloud providers we started to use Bigtable for all of our DynamoDB use cases. We immediately saw a benefit. Under our workload, we would get consistent throughput exceptions against DynamoDB and we were set up to retry these off of queues. With Bigtable, not only were these throughputs exceptions gone but the metrics showed we were barely touching the minimum capacity Bigtable provides. To be fair that minimum is fairly high but it was still cheaper than the over stretched throughput we had provisioned for Dynamo.

Better yet, we had to backfill Bigtable with the data we had in DynamoDB. As a load test, we used the same interface as our typical traffic that comes in to do the backfill. We were running at 100x our peak traffic in Bigtable and while we had to scale up, a process that is far simpler in Bigtable and without downscaling restrictions, the cluster performed without fault. In our experience, this backfill would have been impossible in DynamoDB no matter the throughput allocated.

Thanks to Paul Scott and Stephen Whitworth.

AWS Dynamodb Google Bigtable Google Cloud Platform NoSQL