There are very few suitable use cases for DynamoDB (ravelin.com)

232 points by vgt on July 7, 2017 | hide | past | favorite | 140 comments

brandur on July 8, 2017 [–]

I've seen two large usages of DynamoDB at two different companies, and for what it's worth, in both cases we've had similar trouble as the author. In one case we ended up ripping it out and moving to a sharded Postgres scheme, and in the other we've left in place for now because a migration will be such a monumental effort, but it's pretty much universally maligned.

Fundamentally, the problem seems to be that choosing a partitioning key that's appropriate for DynamoDB's operational properties is ... unlikely. In their own docs on choosing a partition key [1] they use "user ID" as an example of one with good uniformity, but in reality if you choose something like that, you're probably about to be in for a world of pain: in many systems big users can be 7+ orders of magnitude bigger than small users, so what initially looked like a respectable partitioning key turns out to be very lopsided.

As mentioned in the article, you can then try to increase throughput, but you won't have enough control over the newly provisioned capacity to really address the problem. You can massively overprovision, but then you're paying for a lot of capacity that's sitting idle, and even then sometimes it's not enough.

Your best bet is probably to choose a partition key that's perfectly uniformly distributed (like a random ID), but at that point you're designing your product around DynamoDB rather than vice versa, and you should probably wonder why you're not looking at alternatives.

---

[1] http://docs.aws.amazon.com/amazondynamodb/latest/developergu...

> hox on July 8, 2017 [–]
>
> Conpletely agreed; using DynamoDB for primary storage suffers from needing to design data models around the underlying technology. This is true for secondary keys as well, and is even constrained by the odd pricing model that DynamoDB follows.
>
> I've been happy using DynamoDB as a large-scale caching layer, but even that only fits very specific use case criteria.
>
> > fragsworth on July 8, 2017 [–]
> >
> > > using DynamoDB for primary storage suffers from needing to design data models around the underlying technology.
> >
> > Every database system, at large scale, suffers from this problem.
> >
> > > grogenaut on July 8, 2017 [–]
> > >
> > > I laugh every time I see this because everyone seems to forget the rough time they had the first time they encountered a relational database and how to map their problem into that space... Sure it's pretty straight forward now but that's the point... You're still doing it... You are still using your domain knowledge of relational to map your problem ilto the underlying tech. You just don't notice it. "Oh companies have people and people have multiple addresses with one marked as primary and addresses have 1-3 address lines and several optional fields based on local and and and and..." That's several years of learning if you had to discover it yourself there, not just on the address domain but also on how to decompose it.

leggomylibro on July 8, 2017 [–]

Hot partitions do always seem to be an issue, for an 'infinitely scalable' NoSQL database that is a bit on the pricey side.

But what's a better option for a distributed, managed database-as-a-service? Rolling your own does mean significant operational burden.

I wonder if you could do something like add a random number to each of your keys before hashing. It would increase your storage size by Nx, but it seems like that would spread your load as well.

vosper on July 8, 2017 [–]

TFA suggests Google's BigTable

xendo on July 8, 2017 [–]

Keys are the main way to access data in Dynamo. They need to be deterministic to provide fast lookup.

nw-fcx on July 8, 2017 [–]

I've been able to improve my partitioning by adding a random suffix from a small set to the key for the writes, and then just trying all suffixes for the reads. Because my work load was relatively write-heavy (used at least ~10x more write capacity than read capacity), and because most of the read capacity was consumed by the size of the entries and not the lookups (so I didn't even have a linear increase in the need for read capacity), I think I came out ahead.

Ultimately, we've decided that dynamodb was not really suited to our use case anyway, but I'd say that's mostly because our total data set was fairly tiny and we our writes were not evenly distributed over time, which is just at odds with the capacity model.

grogenaut on July 8, 2017 [–]

What were you using as keys. I heavily use Dynamo and use uuids for about every key

nw-fcx on July 9, 2017 [–]

Some domain data that was way too coarse, but seemed like the right thing to query by.

foota on July 8, 2017 [–]

If you have significantly more buckets than users though shouldn't the variations in access pattern even out?

solidsnack9000 on July 8, 2017 [–]

It depends on what the extremes look like. Say things are power law distributed -- you have three users that require 1000000x capacity -- then as you shard more you basically guarantee that some shards are much, much more heavily loaded than others.

throwaway91111 on July 8, 2017 [–]

Why on earth don't they hash the key to improve the key distribution?

choward on July 8, 2017 [–]

How would that help? The same user id will still hash to the same thing.

Jach on July 8, 2017 [–]

I think I had the same confusion after reading the GP's last statement. The idea of hashing is that the resulting number will be uniformly distributed across the hash space, as opposed to monotonically increasing bit by bit like an auto-increment or timestamp, but it's not the partition key value itself that needs to be uniformly distributed but your access patterns of the partitions. As the docs note, a device id (and also a user id) could be good or bad to use as a partition key, it depends on the read/write patterns given the key.

xendo on July 8, 2017 [–]

Write-through cache solves many of the issues with uneven access patterns.

imauld on July 8, 2017 [–]

They do.From their getting started docs:

> DynamoDB uses the partition key value as input to an internal hash function; the output from the hash function determines the partition where the item is stored.

idunno246 on July 8, 2017 [–]

they do, thats why the partition key used to be called the hash key

abalone on July 8, 2017 [–]

So, Amazon actually kind of agrees! They talked about this very issue of hot keys and overprovisioning at their most recent conference. The thing is it was buried in a session on Aurora and they didn't mention DynamoDB by name -- they just called it nosql -- but they noted that a customer cut their costs 40% by moving to Aurora. Because it does automatic heat management and just bills you for what IO you use.

This is somewhat at odds with their top-level messaging which still pushes DynamoDB as the most scalable solution. And perhaps it is... there are some scalability limits to Aurora. Writes are bottlenecked by one instance. 64TB max. I think performance drops when you exceed the in memory cache. But those limits are still quite large.

Basically I sense some tension between the DynamoDB and Aurora teams and I wonder where this is all going to shake out in the long run.

Here's the full quote (I transcribed it so may contain errors):

"The one thing that surprised me is that there are some customers who are moving their nosql workload to aurora. There are two reasons for that. One, it's a lot easier to use Aurora because it's mysql compatible compared to nosql because the interfaces and transaction characteristics are so different. What is also interesting is people also saved money because the IO cost is much lower. In no SQL if you have a large table it gets partitioned then the IO gets partitioned across all the table partitions that you have. And if you have one partition that is hot then you have to provision based on the IO requirement of the hot partition. In the case of Aurora we do automatic heat management so we don't have this hot parition issue. Second we don't charge based on provisioned IO. It's only the IO that you use. And that actually saves a lot of money. In this particular case this is a big social company, interaction company, I cannot tell the name, and they reduced their operational costs by 40% by moving from nosql to Aurora" [1]

[1] https://youtu.be/60QumD2QsF0?t=17m01s

windlep on July 8, 2017 [–]

I don't know what the article author is storing, but it's noted that 10GB of data is stored per node. That's quite a bit of data for a single table, and the 10GB is per shard of a table, not per 'database' (DynamoDB only has a notion of tables).

Amazon has deep dive talks on DynamoDB on YouTube[1] that go into lots of these details and how to avoid problems from them. It's not that different from understanding how to structure data for Cassandra, Mongo, etc. All the NoSQL systems require an understanding of how they work both to structure your data and ensure optimal performance.

For example, maybe one's consistency constraints are better met by DynamoDB instead of BigTable's varying consistency on different query types[2] (the author of this article didn't address consistency at all). With DynamoDB, you can get strongly consistent reads and queries all the time [3].

Overall, it seems like a kind of weak reason to make the strong statement about "probably shouldn't be using DynamoDB". Maybe a better title would be "Understanding drawbacks of large datasets in DynamoDB". I do hope the author understands the consistency changes they may experience in BigTable, as it could easily require large changes to the code-base if strong consistency was assumed on all queries.

[1] https://www.youtube.com/watch?v=bCW3lhsJKfw

[2] https://cloud.google.com/datastore/docs/articles/balancing-s...

[3] http://docs.aws.amazon.com/amazondynamodb/latest/developergu...

Edit: Fixed inconsistency in what the 10GB limit was referring to, not per table, but per node (shard) of the table.

mbrukman on July 8, 2017 [–]

Disclaimer: I am the PM of Google Cloud Bigtable.

It appears you're confusing Bigtable with Datastore (to be fair, Datastore is built on Megastore, which is built on Bigtable, so it's an understandable confusion), but let's be clear: Google Cloud Bigtable != Google Cloud Datastore.

The URL you cited about consistency models: https://cloud.google.com/datastore/docs/articles/balancing-s... is entirely about Datastore, but you referred to it as Bigtable.

windlep on July 8, 2017 [–]

Sorry about that, I had difficulty finding anything about the consistency model of BigTable. Upon further reading on SSTables and how BigTable distributes them across nodes, it appears there is full consistency but if a node goes down the data on it is inaccessible?

wora on July 8, 2017 [–]

If a node goes down, another node will replace it quickly. The node (aka tablet server) doesn't own any data. The data is stored on lower level storage layer.

lwansbrough on July 8, 2017 [–]

We store 100-1000 GB of data in each wide table on our untrendy on-prem SQL server boxes. 10GB is peanuts. In fact, I propose (given the limitations suggested by the author) that DynamoDB may have NO practical uses worth exploring. If NoSQL is about scale, and it can't scale, what's it good for?

I can understand having to optimize your key space, but in this case it necessitates extreme premature optimization.

manojlds on July 8, 2017 [–]

From my understanding, node is virtual nodes. This article on Cassandra should give some info - https://docs.datastax.com/en/cassandra/2.1/cassandra/archite...

ryanworl on July 8, 2017 [–]

Each shard is 10GB. The unit of scale is a shard. You can have as many shards as you want.

lwansbrough on July 8, 2017 [–]

It's one shard per node though, right? So you're talking about SERIOUS cost when you want to store _actual_ big data.

ryanworl on July 8, 2017 [–]

You don't pay per node. In fact, the concept of a node is not exposed at all. Each shard is technically on three nodes for high availability. You pay for provisioned capacity and data storage per GB.

pmohan6 on July 8, 2017 [–]

DynamoDB is a multi-tenant service. There is no dedicated node for you. Each shard is replicated across 3 nodes and those nodes contain replicas of other shards/tables.

throwaway91111 on July 8, 2017 [–]

It's amazing how much you can put into a reply without directly addressing any of the concerns. Are you serious that dynamodb can't even hold 10GB? If you intended to defend the database I think you failed. Sqlite3 can store 10 gigs without sweating. What on earth can you use dynamo db for if it doesn't scale with the data? The way you describe it seems much WORSE than the OP.

windlep on July 8, 2017 [–]

I said it was per node (which is a shard), it's not a limit per table. Did you read the original article or my reply? Both of them state this is per node and how DynamoDB decides when to shard to more nodes.

Edit: Sorry, I did mention 10GB originally in relation to a table. That was incorrect of course.

throwaway91111 on July 8, 2017 [–]

I did. However, 10GB still seems extremely small. A commodity postgres, cassandra, or cockroachdb server can serve HUNDREDS of GB per node. Why is the size per node so small for dynamodb?

It seems like poor key space design.

phamilton on July 8, 2017 [–]

While not exactly the same, the Dynamo paper outlines that a single host is composed of many virtual nodes. It is very likely that a physical DynamoDB host will have dozens of nodes. This is done so that the cluster can scale up or down independent of the number of hosts while avoiding a gross imbalance. (12 nodes on 11 hosts means one host has 100% more traffic. 34 nodes on 11 hosts means one host has 33% more traffic.)

pzb on July 8, 2017 [–]

It is important to note that "Dynamo" and "DynamoDB" are two very different things that happen to share many of the same letters. DynamoDB is not Dynamo.

manojlds on July 8, 2017 [–]

This Cassandra article should provide some info on virtual nodes. https://docs.datastax.com/en/cassandra/2.1/cassandra/archite...

windlep on July 8, 2017 [–]

Agreed entirely, I do wonder what kind of internal constraints led Amazon to have these limits. Maybe in the future they'll go away, as so many other seemingly arbitrary limits do.

zwily on July 8, 2017 [–]

10GB is per partition, not per node.

cherioo on July 7, 2017 [–]

"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. ... Potentially getting 1–5 requests per second for a given second but certainly not a sustained load of that. ... This will not work at scale in DynamoDb."

What? Why? Suppose that's 5 million customers, you will only have a 10GB table which fits in a single DynamoDB shard, with no sharding. With the restriction of 1-5 operation per customer per second, this sounds like the ideal use case for DynamoDB.

What am I missing?

fragsworth on July 8, 2017 [–]

Amazon is limiting it to 5 operations per second *per shard*, not *per customer*. That is if you have 200 shards and 1000 write capacity.

It's a problem with how Amazon calculates your capacity units as it scales up Dynamo for you, if you have a particularly large data set with a few very active users.

phamilton on July 8, 2017 [–]

The parent post is suggesting that 5 Million customers does not in any way imply 200 shards. It's much more likely that with 5M customers you have 1 or 2 shards. That's 4KB (one document) per customer.

awfullyjohn on July 8, 2017 [–]

He complains that DynamoDB doesn't work for him, then says you should instead use Google BigTable. But he doesn't offer evidence why you should use BigTable. And just says that it works for him.

I don't buy it. I've used BigTable in the past and found it to be infuriating. Now, because it works for him, I'm supposed to believe that BigTable is right for me?

mbrukman on July 8, 2017 [–]

Hey there, I'm the PM for Google Cloud Bigtable; if you want to talk about your experiences with Bigtable, I'd love to hear what your use case was, what your experience was, and your thoughts on how we can make Bigtable better for you. Feel free to follow me on Twitter (see my profile) and we can chat over DM if you wish.

If we chat and decide that Bigtable is appropriate for your use case, maybe you'd be willing to give it another shot? :-)

Diederich on July 8, 2017 [–]

I believe the article specifically notes that BigTable gave far better performance than DynamoDB.

double-a on July 8, 2017 [–]

What did you find infuriating about BigTable?

luhn on July 8, 2017 [–]

Related is The Million Dollar Engineering Problem [1] from Segment, which showed up on HN a few months ago. They shaved $300K from their annual AWS spend by identifying and dealing with hot shards on DynamoDB.

[1] https://segment.com/blog/the-million-dollar-eng-problem/

karmakaze on July 7, 2017 [–]

TL;DR - Don't use something as coarse as customer_id as a partition key. Alternatively move to GCP/BigTable.

Any DynamoDB tuning advice will say how important it is to have well distributed hash keys. As for the second part, why not use Cloud Spanner? I wish AWS had something like it.

nemothekid on July 8, 2017 [–]

>Any DynamoDB tuning advice will say how important it is to have well distributed hash keys.

Maybe I'm missing something, but as I understand it he did have well distributed hash keys (I'm assuming his customer ids were random UUIDs). The problem he had however was that he had so much data that a single custom could cause throughput exceptions on a very small amount of ops/s.

lotyrin on July 8, 2017 [–]

Customer UUID would only be well-distributed for queries of customers.

If some customers are more active than others in a given unit time, some customers have more data than others, and customers' data is keyed by customer

ID then no, it's not a well distributed key.

qaq on July 8, 2017 [–]

I can see the future:

article that outlines gotchas of Bigtable

article you probably shouldn’t use Bigtable

article the amount of money we could've saved using PG and not rewriting things 3 times.

anything up to 10-15TB there are very few reasons not to use something like PG

x0x0 on July 8, 2017 [–]

And pg supports schema-free data (that they can index!) as columns in your standard sql tables. Check out jsonb if you haven't tried it. It's really really good.

https://www.postgresql.org/docs/9.6/static/datatype-json.htm...

TheAceOfHearts on July 8, 2017 [–]

Many of the comments here are saying that the author's use-case wasn't a good one for DynamoDB. Can anyone share some simple approachable resources that talk about the kinds of use-cases where these tools make sense?

Whenever I read about NoSQL systems, I'm always left a unsure about its use-cases. I've only worked on systems where a traditional RDBMS made the most sense. How do you identify when it's appropriate to reach for one of the many NoSQL tools?

I've had the suspicion that many applications that leverage NoSQL tools are usually used in conjunction with a relational database, and not in isolation. Based on my limited understanding, I can at least wrap my head around a few ways in which this could probably help. Am I off the mark here? One of the points I struggle with is that once you're storing data across multiple data stores, maintaining data integrity becomes much harder.

jjirsa on July 8, 2017 [–]

Can't talk about current employer, but precious employer ran thousands of Cassandra nodes to capture and index billions of events a day into a real time graph. Talking petabytes of data, millions of writes per second, in order to fight hackers.

Doing that with an RDBMS would be ... unpleasant. Doing it with Cassandra isn't trivial, but it's straightforward, cross-cloud capable, cross-DC HA, linear scaling, and tunable consistency.

Data integrity is a nonissue - when you're dealing with the scale nosql was meant to solve, you're probably smart enough to learn how to write multiple places safely.

krallja on July 8, 2017 [–]

> How do you identify when it's appropriate to reach for one of the many NoSQL tools?

Despite the flag on the post, this is a pretty good summary of why you might choose one of several: https://news.ycombinator.com/item?id=14697230

qaq on July 8, 2017 [–]

when your data set grows beyond what can reasonably stored in RDBMS e.g. 20+ TB

phamilton on July 8, 2017 [–]

20TB in an RDBMS has very different characteristics than 100GB in an RDBMS. Once you can no longer store your dataset in memory, you start to see greater variance in latency.

DynamoDB for us has given us very reliable latency for our datasets that are too large to hold in memory on an RDBMS.

qaq on July 8, 2017 [–]

You can get a box with 48TB RAM so 20TB has more to do with time it takes for DB to come up. To provision DynamoDB to the same performance as PG on i3.16xlarge instances you will be paying north of 50K/month vs 5K/month and that's with all limitations that come with DynamoDB.

phamilton on July 8, 2017 [–]

An i3.16xlarge is 488GB, not 48TB. The x1.32xlarge (largest amazon offers) is 2TB.

Additionally, self-managed PG is a huge operational undertaking compared to DynamoDB. RDS is a bit closer, but the largest RDS offers is db.r3.8xlarge, which is 244GB.

The db.r3.8xlarge on RDS Postgres all in with 3000 provisioned IOPS (just provisioning for writes, all reads should be served from memory) and 1.5TB storage ends up around $6k. A 3 year reservation will get that down to $3k.

Comparable DynamoDB (3000 write units, 3000 read units, reserved IOPS) comes out to $1500.

The key here is consistent performance and low operational overhead. As I said, we're pretty happy with it.

anarazel on July 8, 2017 [–]

> The db.r3.8xlarge on RDS Postgres all in with 3000 provisioned IOPS (just provisioning for writes, all reads should be served from memory) and 1.5TB storage ends up around $6k. A 3 year reservation will get that down to $3k. > Comparable DynamoDB (3000 write units, 3000 read units, reserved IOPS) comes out to $1500.

Note those aren't really comparable numbers - postgres will often collapse writes from concurrent sessions / statements.

qaq on July 8, 2017 [–]

Is there any place in the post I claimed i3.16xlarge has 48TB RAM? I said you can get a box with 48TB RAM. Did anywhere in the post I compared to RDS? i3.16xlarge is basically a dedicated box you have 8 NVMe PCI SSDs to be on super conservative side I was using 50,000 write IOPS 200K read IOPS in reality it can do way more.

phamilton on July 8, 2017 [–]

Ok. Let me try again:

> You can get a box with 48TB RAM

Not with AWS. If you don't have AWS as a requirement, go for it.

> i3.16xlarge

An i3.16xlarge still won't hold a 20TB dataset in memory. You are going to see more variance in latency if you have a working dataset too big to fit into memory.

As far as getting superior performance out of an i3.16xlarge, that's fine if you have the expertise and resources to run PG yourself. However you're going to need replication, which will increase that cost. You're going to need failover mechanisms, backup, etc.

We have preferred RDS for Postgres because it gives us something operationally simple. We've found DynamoDB to be even simpler operationally and have more predictable performance. We have frequently

considered running a self managed PG instance and have decided against it for our use case.

qaq on July 8, 2017 [–]

Ok let's try :) Did I ever claim on 48TB on AWS ? You are seriously claiming DynamoDB will be holding full 20TB in RAM if yes I have a bridge to sell you :). The price for i3.16xlarge was quoted for 2 instances the IOPS used for calculation is 1/8th of actual max IOPS that instance can do.

phamilton on July 8, 2017 [–]

> Did I ever claim on 48TB on AWS

When presenting an alternative to DynamoDB, AWS is an implied part of presenting that alternative. While the original article does suggest an alternative outside of AWS, it properly qualifies it as a competitor offering. Regardless, I don't care what was and wasn't claimed. It's not important.

> You are seriously claiming DynamoDB will be holding full 20TB in RAM

No. I'm claiming that DynamoDB will have predictable and consistent latency characteristics at 20TB. The related claim is that an RDBMS will not have consistent latency characteristics if the working dataset does not fit in memory.

qaq on July 8, 2017 [–]

"When presenting an alternative to DynamoDB, AWS is an implied part of presenting that alternative". If that is the way you guys operate well ... We run things in AWS, on prem and in GCP. Our AWS spend is over 3 mil. per month things that require hard latency guarantees are def. not running in AWS. Nothing is AWS has "predictable and consistent latency characteristics" simply due to the nature of how they operate.

chairmanwow on July 8, 2017 [–]

Quick aside:

It seems that a lot of the qualms with various databases stem from a misunderstanding of their use cases. A lot of the features of major SQL databases, namely ACID, are misappropriated to be features of databases in general. This key misunderstanding seems to cause a lot of SWEs insane as they later realize that NoSQL DBs are not always Available and Strongly Consistent.

Responding to Author:

I wasn't totally convinced by the author's argument against DynamoDB. This article [1] offers a good solution to pretty much all of OP's problems. Most significantly, hashing user data using date.

While DynamoDB is certainly different from most other databases, that doesn't mean that there aren't sensical usage methodologies.

Links:

[1] https://medium.com/building-timehop/one-year-of-dynamodb-at-...

sheeshkebab on July 8, 2017 [–]

Dynamodb has a nice js interface and is a good helper store for various small aws specific automation projects.

It's not a good choice for various read heavy enterprise apps (and frankly for write ones neither). It also doesn't scale - it doesn't even have a cross region active/active support (highly surprising for a key value store when cassandra supported it for ages).

Don't use it for anything serious.

xendo on July 8, 2017 [–]

I don't see how the scenario with a single customer ID is credible. Why not just put a cache in front of the db? Now there is even a fully managed solution for that: https://aws.amazon.com/dynamodb/dax/

ezulich on July 10, 2017 [–]

Full disclosure: I am a member of DynamoDB team at AWS, but my opinions here are mine only

On the topic of DynamoDB use cases, here are some of DynamoDB users describing their use cases:

- DataXu: "How DataXu scaled its System to handle billions of events with DynamoDB" -- https://youtu.be/lDiI0JMf_yQ?t=765 (from AWS re:Invent 2016)

- Lyft: "Lyft Easily Scales Up its Ride Location Tracking System with Amazon DynamoDB" -- https://www.youtube.com/watch?v=WlTbaPXj-jc. The story "AWS Helps Fuel Lyft's Expansion" in InformationWeek mentions Lyft's use of DynamoDB as well: http://www.informationweek.com/cloud/infrastructure-as-a-ser...

- Under Armour implemented cross-region replication using DynamoDB: https://youtu.be/NtaTC2Fq7Wo?t=699

- Amazon.com: Amazon Marketplace shares their story of migration to DynamoDB and why they did it: https://www.youtube.com/watch?v=gllNauRR8GM

Finally, DynamoDB was created because Amazon needed a highly reliable and scalable key/value database. As Werner Vogels said it in his blog post announcement of DynamoDB back in 2012 (http://www.allthingsdistributed.com/2012/01/amazon-dynamodb....):

"This non-relational, or NoSQL, database was targeted at use cases that were core to the Amazon ecommerce operation, such as the shopping cart and session service."

natekupp on July 8, 2017 [–]

We use DynamoDB quite a bit at Thumbtack. Our biggest issue is backups - just wrote a short note about our experiences with DynamoDB here: https://medium.com/@natekupp/dynamodb-and-backups-16dba0dbcd...

> Dunedan on July 8, 2017 [–]
>
> Oh yes. Backups for DynamoDB are a pain in the ass, especially as AWS doesn't offer an out-of-the-box solution for that.
>
> That Data Pipeline + EMR solution mentioned in the blog post (here is a better link for it: https://docs.aws.amazon.com/datapipeline/latest/DeveloperGui...) has several drawbacks:
>
> - too many moving parts, especially given the track record of EMR
>
> - might not even be available when your requirement is to keep the data in the same AWS region as the DynamoDB table, as only five regions support Data Pipeline
>
> The best approach I've seen so far is to use DynamoDB Streams and an AWS Lambda function to create incremental backups in an versioned S3-bucket. dynamodb-replicator (https://github.com/mapbox/dynamodb-replicator) implements that together with some scripts to do management tasks like back filling an S3 bucket with data which is already in DynamoDB or joining incremental backups into a single file.

It's still pretty unpolished and definitely needs some love, but I think it's the right approach.

jsemrau on July 7, 2017 [–]

When I started building my first app in 2011 MongoDB was the rage. So I build the back-end using the futuristic 'No-SQL' technology. It turned out to be slow (~1 min Query time), inconsistent, and missing an RDBMS layer. Move the thing to PHP/Mysql problems were gone. I still have not found a use case outside of web (comments/discussion) sites where the high integration with Javascript actually makes sense.

threeseed on July 8, 2017 [–]

Why did you use MongoDB if your domain model wasn't suited ?

I just don't understand people who complain about a technology and say it is useless for all of these use cases when they couldn't even spend a few hours to do a Spike/POC or some basic data domain design. MongoDB has very clear documentation about what you should or shouldn't use it for.

MongoDB is unique in that it is one of the few document stores available today. So if you have use cases such as 360 Customer View or where you need to fetch a lot of nested information with a single id it is blisteringly fast.

If you have a relational data model than use a relational database.

mst on July 8, 2017 [–]

> Why did you use MongoDB if your domain model wasn't suited?

Because MongoDB's marketing sold it as the hot new datastore that made SQL legacy - "newer! faster! web scale!" not "only use this if your data isn't relational".

The reference documentation might be more accurate but the way it was publicised certainly wasn't.

CyberDildonics on July 8, 2017 [–]

What does document store mean and what can that do that other databases can't?

axaxs on July 8, 2017 [–]

As someone being 'forced' to use Mongo for the first time, I'm curious as to when/where you hit that? I'm looking for pros/cons, but the internet is proving rather unhelpful as it seems to be people vehemently for it or against it.

dijit on July 7, 2017 [–]

I hope you don't mind if I piggyback on this to echo this sentiment.

Although I dislike MySQL for its many gotchas (data corruption level stuff too!) I was looking for a _long_ _long_ time for a high consistency NoSQL database.. we basically need document storage of large binary data.

Ironically literally nothing in NoSQL land does write-through to disk, they just write to vfs and hope it works; additionally, those that support clustering opt for eventual consistency.. That just baffles my mind, so much potential for lost or corrupted data.

We ended up doing deterministic sharding on postgresql, it worked incredibly well, even in failure modes you hope never to see.. and no corruptions! :D

jethro_tell on July 8, 2017 [–]

Isn't that the point of no_sql. You trade consistency for speed. Best used to mark temporary state in a web app. If you need atomic consistency in a database you should use a relational database that does atomic writes.

That has always been the case. Why would it be so shocking to you? Atomic writes have already been built. Sounds like you were standing there with a hammer looking for a nail.

dijit on July 8, 2017 [–]

I always thought the term "document store" just meant arbitrary unstructured data that didn't need to be related.

That's how you do things like we did regarding deterministic sharding- it's very easy to scale this way if you don't ever "JOIN" tables etc;

threeseed on July 8, 2017 [–]

Please be specific. NoSQL comprises hundreds of different databases with very different semantics.

Technically PostgreSQL is a NoSQL database and it along with Cassandra for example have modes in which they are fully and atomically consistent.

threeseed on July 8, 2017 [–]

You are talking complete nonsense.

HBase, Cassandra, MongoDB, Riak, Couchbase etc all write through to disk with proper fsync flushes. And I've never heard of any database that has a model where it writes to a virtual file system - whatever that even means.

Please provide some specific examples.

dijit on July 8, 2017 [–]

HBase, Cassandra, MongoDB, Riak, Couchbase And redis were the main candidates.

Hbase: http://mail-archives.apache.org/mod_mbox/hbase-issues/201307...

Cassandra: (fsync to WAL, not full fsync).
https://wiki.apache.org/cassandra/Durability

MongoDB: ... too much wrong here to list, although I hear it's improving in being cluster aware etc.

Redis does support fsync as far as I remember but the write/delete pattern is incredibly sub-optimal, it runs basically out of a WAL by itself and runs very poorly if your dataset does not fit in memory.

linuxhansl on July 8, 2017 [–]

As author of the cited post on HBase lemme clarify:

0. Here are more details on that: http://hadoop-hbase.blogspot.de/2013/07/protected-hbase-agai...

1. By default HBase "flushes" the WAL. Flush here means to make sure that at least 3 machines have change in memory (NOT on disk). A datacenter power outage can lose data.

2. As HDFS closes a block it is not by default forced to disk. So as HBase rewrites old data during compactions by default, old data can be lost during a power outage. Again, by default.

3. HDFS should be configured with sync-on-close, so that old data is forced to disk upon compactions (and sync-behind-writes for performance)

4. HBase now has an option to force and a WAL edit (and all previous edits) to disk (that's what I added in said jira).

5. This is post is 4 years old for chrissake :)... Don't base decisions on 4 year old information.

HBase _is_ a database and it will keep your data safe. Unfortunately it requires some configuration and some knowledge.

scrollaway on July 8, 2017 [–]

Key-value document storage of large binary data... isn't that S3?

dijit on July 10, 2017 [–]

Yeah, shame we were pushing too much for the 5GB/s cap.

Also, their consistency model is hidden and they've lost data before.. so not selling points.

imaginenore on July 8, 2017 [–]

Redis

avitzurel on July 7, 2017 [–]

The title is very... what's the two words I'm looking for?

Dynamo does not scale for this specific use case but I have used it successfully in production (at scale) with ZERO issues.

Dynamo is a key value sharded and zero operations* database that most applications and companies will benefit from IMHO.

is the hot key and evenly sending queries to nodes the only issues you concluded we should not use DynamoDB on?

nikanj on July 8, 2017 [–]

The gist of this seems to be that DynamoDB becomes a problem if you have millions of customers.

Don't worry. You don't. And there will be many good reasons to refactor the architecture before you do.

phamilton on July 8, 2017 [–]

We have close to 25M monthly active users and DynamoDB works pretty well for our use case.

We store user generated content in DynamoDB. Our largest table is over 1 TB. The mapping of users to their generated content is in Postgres. So doing work on behalf of a particular user will generally be distributed across multiple nodes.

We've enjoyed the very predictable performance of DynamoDB as well as the operational simplicity. We've started moving smaller datasets over and are using it more and more.

nnx on July 8, 2017 [–]

How large is user's generated content in average? Have you considered S3 for your use case? Storage in DDB can be quite pricey (when combining storage cost and and read capacity usage cost)

phamilton on July 8, 2017 [–]

We are a messaging application, so every chat message is stored in DynamoDB in an ordered stream. We frequently fetch slices of a stream, so the granularity of storage is a single message.

S3 does not do well (cost wise) with billions of files. Provisioning 1000 Write IOPS on Dynamo costs around $100/month. That's 2.5B writes. On S3 that's going to be on the order of $10k. Similarly 1000 Read IOPS on Dynamo is ~$20/month vs 2.5B reads on s3 costing $1000. Storage is cheaper on S3 by about 10x, but per TB that comes out to $250 vs $25, which is hardly a dominating cost factor.

wanghq on July 8, 2017 [–]

Sounds interesting! But if the mapping is stored in Postgres, your system's availability is capped by that.

phamilton on July 8, 2017 [–]

Yes. We're migrating towards a more available store (one option we're testing out is Aurora) but availability wasn't our primary motivation in

choosing DynamoDB. The main motivation was consistent performance with a dataset too large to fit into memory.

mnutt on July 8, 2017 [–]

I often give this same advice, but assuming this is the case, why reach for DynamoDB at all? Are there small-data use cases where DynamoDB makes more sense than RDS or a hosted redis? At least with those, if you do run into scaling issues you haven't locked yourself into dynamo.

ryandvm on July 8, 2017 [–]

I would imagine the use of AWS Lambda might be a pretty decent incentive to use DynamoDB. I'm guessing that stateless DynamoDB queries are faster to fire off than dealing with Postgres or MySQL connection setup and teardown.

TheAceOfHearts on July 8, 2017 [–]

I'm not familiarized with DynamoDB, but why would connecting to it be any faster than connecting to Postgres or MySQL?

Something1234 on July 8, 2017 [–]

Perhaps DynamoDB is based around HTTP, rather then requiring socket back and forth actions(uname + pass, auth, query, results vs just a single hit where query + auth details are packed together). Never actually looked at dynamodb, so not sure.

apetresc on July 8, 2017 [–]

Yes, that is accurate. DB operations to DynamoDB are just AWS API calls, the same as you would use to spin up an EC2 instance or create an S3 bucket.

solidsnack9000 on July 8, 2017 [–]

Lambda keeps the containers running if there's traffic coming to them, so the setup cost isn't paid per request unless there are few requests.

justinsaccount on July 8, 2017 [–]

I'm not sure how much this applies to dynamodb.. I think I couldn't even figure out how to get started with that.

I used simpledb once as part of project. All I needed it for was to store a small amount of state to be shared between some otherwise stateless ec2 instances. So, kind of the perfect use case.. Except I don't think it is made to scale down that small.

I don't know if I provisioned the capacity wrong or what, but it was so flaky. All I needed it to do was a few reads every few minutes.

I think my issue was I had the provisioned scalability set down to the smallest value I could because I only needed to pull down ~10 rows every 60 seconds or so.. But when the app would start and try to sync up, simpledb would throttle it?

I ended up having to just write a cron job to sync the db to a local json file, because doing the reads from inside my main loop would randomly fail or timeout.

nw-fcx on July 8, 2017 [–]

For my very small-data use case I would be very happy if I could use DynamoDB as a persistent data store because the operational aspect is like ten lines of Terraform

tracker1 on July 8, 2017 [–]

Well, I just spent the past month dealing with issues trying to get around 360M records into Dynamo... At this point, with the number of partitions, I'm unsure if we'll be able to

keep reads to a decent level without bumping the throughput to much higher than what's needed.

dserban on July 8, 2017 [–]

My background is in Cassandra and one company I worked for last year insisted that we use DynamoDB for a project.

Here are a few things that ended up being show stoppers.

1. Both the partition key and the sort key are capped at 1 field. In an attempt to "think Cassandra data model", the ugly workaround was to stringify and concatenate things at the application layer, then parse / split on the other side. This made the code unreadable.

2. DynamoDB-Spark integration is a second-class citizen. (Cassandra-Spark integration is first-class and well-maintained.)

3. The other thing that made code unreadable was the accidental complexity introduced by exception handling / exponential backoff we needed to implement to protect against accidental read capacity underprovisioning.

Although I made repeated pleas to switch to Cassandra, the (non-technical) CEO insisted that we keep using DynamoDB. I'm no longer at that company but I hear they have meanwhile switched to RedShift.

pishpash on July 8, 2017 [–]

Cassandra isn't any better in those regards. For 1, Cassandra just does the concatenation under the hood and 3 happens all the time on Cassandra. The broken thing is using NoSQL as a DB, not Cassandra or DynamoDB.

ckalantzis on July 8, 2017 [–]

Quick! Someone should tell Netflix and Apple, that. They've been using Cassandra and Datastax as their main DB for almost a decade.

They're always hiring. You should apply and show them the error of their ways.

jjirsa on July 8, 2017 [–]

Cassandra allows multiple clustering columns, it's not simple concatenation under the hood.

There are no hidden exponential backoffs in Cassandra in any of the hot query paths, period.

Suggesting that using nosql as a DB is broken is so ludicrous I'm not even going to try to refute it - you simply don't know what you're talking about.

kevan on July 8, 2017 [–]

>This will not work at scale in DynamoDb.

I don't think we're getting the whole story from the author. I'm not the biggest fan of Dynamo either for reasons I won't get into here, but this type of workload is exactly what Dynamo was built for: serving a website with millions of customers.

Disclaimer: I work at Amazon, my views are my own.

thinkloop on July 8, 2017 [–]

So what about the issue of requiring heavy over-provisioning for common usage patterns, like a customer being logged in causing temporary bursts of access on specific keys?

mrep on July 8, 2017 [–]

Aka, the status quo before AWS allowed you to dynamically scale certain parts of your infrastructure. It's not perfect for all parts of your stack, but do you have a better alternative for databases?

dyeje on July 8, 2017 [–]

Our team has been using it alongside Postgres as a scalable metric store and things have been pretty good. We had some growing pains tweaking our storage keys and switching to a daily table to avoid partitioning issues, but it's been quite stable for a while now.

hox on July 8, 2017 [–]

The biggest issue faced with DynamoDB from my perspective has been the problem with any hosted service - that is, the operational stability is in the hands of others entrusted with operating a massive multi-tenant system, and any outage cannot follow your own operational recovery mechanisms unless you plan for failover yourself. And the moment you plan for failover, you need to evaluate why you don't just handle the primary system as well.

> apetresc on July 8, 2017 [–]
>
> There are many things people justifiably complain about DynamoDB, but operational support has rarely been one of them.
>
> > hox on July 9, 2017 [–]
> >
> > Dynamodb experienced a severe outage in us-east in September of 2015, dropping the service to three nines of uptime on just the single event. During the incident there was little-to-no knowledge of when service would be restored, so companies that relied on the service were flying blind.
> >
> > Such an outage can sometimes be mitigated when operations are internal to a company; but with DynamoDB, one has to build internal failover or implement a complex cross-region replication scheme. The complexity and reliability starts to reduce the attractiveness of the hosted service in general.
>
> elvinyung on July 8, 2017 [–]
>
> I mean... of course? But I feel like the idea here is that you *can* "entrust" them with operating the system and not worry about rolling your own failover mechanism, as long as they seem to be upholding their SLAs.
>
> (Of course, cloud vs. bare metal is a long-running debate, but I think I take the position that multitenant clouds are generally cheap enough that it's worth the cost for the extra feature velocity, if you're not truly at scale.)

cyberferret on July 8, 2017 [–]

Hmm... We dabbled with DynamoDB on a couple of very small projects, but found some real oddities in how it stores JSON data, and manages keys. Querying the dataset was also a mammoth, frustrating task.

Ended up switching to RethinkDB and haven't looked back - far better query syntax/language (REQL) and we can organise the JSON content just how we want it.

jjirsa on July 8, 2017 [–]

The best argument against dynamodb is the aws "well architected" guidelines - how are you designing for resiliency with your single region, active-passive database with clunky bolt on replication using kinesis?

Cheaper active-active-active options exist that don't require manual dr failover drills and manual failback when regions inevitably crash.

macinjosh on July 8, 2017 [–]

My company is in the online form builder space and usage of our MySQL RDS instances are nearing the limit of what is offered. I looked into moving certain high-volume/high-traffic data models to DynamoDB and realized we could not because each record is limited to a measly (for our needs) 400kb.

jey on July 8, 2017 [–]

So it's fine as long as I precompute the full transaction to apply to the key, then do it all at once? I expect large overall transaction volumes but am happy to only issue one DB change per user interaction.

pishpash on July 8, 2017 [–]

NoSQL should stay as a quick and dirty solution for storing key-values. When you start to schematize it and use it as a "DB," you are going down the wrong path. That is because NoSQL is a glorified cache, not a DB. It is essentially using the memory on a large number of nodes to buffer bursty throughput, and using background processes to collate the data later onto disk. There is almost no case where an explicit distributed caching or queuing solution backed by a traditional DB isn't strictly better.

> fennecfoxen on July 8, 2017 [–]
>
> > It is essentially using the memory on a large number of nodes to buffer bursty throughput, and using background processes to collate the data later onto disk.
>
> You're not describing NoSQL. You're describing something like MongoDB specifically. You're right about the specific case, wrong in the general case.
>
> > There is almost no case where an explicit distributed caching or queuing solution backed by a traditional DB isn't strictly better.
>
> Bah.
>
> If you're looking at big enough data sets (think "Walmart scale", not "web scale") then you have to design your data store to cope with this volume. First, it has to be replicated to multiple servers to serve more queries and for reliability, and is an eventually consistent system a consequence (basically any time you use your read replica, or do anything on a multimaster configuration). You'll *need* something shardable or multimaster to achieve horizontally scalablity and you'll probably also lose your foreign key enforcement and uniqueness constraints when you do this, so you might as well go the distance and fully denormalise everything for consistent access times. (Have you ever had the Postgres query planner suddenly make something take 10x longer than it usually does because your data patterns changed and decided to use a different query plan than it's ever used before? Yeah, that's a world of fun. Now make that thing a time-critical daily batch job and *really* squirm.)
>
> When you do the things to make this work, you're basically restricting yourself, abandoning database features. As a consequence, you are going to be in for a world of pain, period, *sorry*. One key advantage of good NoSQL software is that it doesn't give you those features to begin with, so it's harder to get it wrong. It also has data structures designed to make this sort of thing work well and be performant in ways that general purpose databases won't be, on account of being designed for general purposes.

didibus on July 8, 2017 [–]

The article forgets a very important detail:

> A single partition can hold approximately 10 GB of data, and can support a maximum of 3,000 read capacity units or 1,000 write capacity units.

DynamoDb will also split your data if you provision more then 3000 reads or 1000 writes. And the caveat is that it will not join back the shards if you later reduce the throughput back down. Instead, each shard will just get even less throughput then you might believe.

> didibus on July 8, 2017 [–]
>
> Another caveat is that when a shard is split because it has reached its max size, the resulting new 2 shards each get half of the throughput the initial shard had, and not a proportion of the total throughput.
>
> So say you have 4000 write capacity and 0 reads (for simplicity). DynamoDb will allocate 4 shards for it, each getting 1000 writes. Now say shard 2 gets too big, and goes above 10GB. DynamoDb will split it in two. Now you have 5 shards, but they won't all get 4000 / 5 = 800 writes. Instead, the original shard 1 3 and 4 will each still have 1000, and shard 2a and 2b will have 500 each. That's because when dynamodb splits a shard, it redistribute the throughput of its parent to the shards.
>
> > nw-fcx on July 8, 2017 [–]
> >
> > I am really surprised by this, I was convinced that the table-wide capacity is re-distributed evenly across shards.

Looks like you're entirely correct:
http://docs.aws.amazon.com/amazondynamodb/latest/developergu...

I wonder how many times I scrolled past the explanation without internalizing it.

banku_brougham on July 8, 2017 [–]

The first part of the article sounds convincing, but i'm pretty sure amazon is using DynamoDB extensively and successfully at massive scales.

coredog64 on July 8, 2017 [–]

Does the internal charge back model for Dynamo work the same as it does for AWS customers?

If I understand the author's point, you _can_ get great performance, but it requires over provisioning. So it's not so much a technical failing as much as it is a poor monetisation of the capability.

I see that elsewhere in AWS: They require huge instance types for large EBS volumes with ES. For our workload, said instances barely break a sweat.

imauld on July 8, 2017 [–]

We DynamoDB to store user alerts and it works fairly well for our use case. However we still run into issues with consumed throughput being well under provisioned but still getting throughput exceptions.

We don't use the auto incremented user ID's either. We create a hash of the user ID and some of the other data contained in the alert and it would appear we still have hot partitions.

luhn on July 8, 2017 [–]

I took a look at BigTable, as recommended by the article, because I was evaluating DynamoDB myself just yesterday. It looks like the minimum price for that is ~$1500 a month. Granted, you're getting what you pay for (3 nodes that support 10,000QPS each), but the pricing is out of reach for smaller projects.

5ersi on July 8, 2017 [–]

You should take a look at Google Cloud Database which is a better option and has per-operation pricing instead of per-node pricing.

luhn on July 8, 2017 [–]

That is very attractive pricing. It looks like you don't even need to preallocate capacity, just read and write as you please.

datashovel on July 8, 2017 [–]

Generally speaking. Given that DynamoDB is a NoSQL database service, I'm not certain that moving larger clients to their own dedicated AWS resources should cause too many negative side effects. Especially ones who are so large they're causing scalability issues.

rushi_agrawal on July 9, 2017 [–]

If DynamoDB guys can come up with a mechanism similar to their CPU credits concept, that'd be a really nice feature to have. Of course it can't be as straightforward as CPU credits.

znep on July 8, 2017 [–]

Does anyone have any real world experience with how the Amazon DynamoDB Accelerator that AWS released as a preview a few months ago can help out?

iampims on July 8, 2017 [–]

rather poor example of "how dynamodb doesn't scale".

DynamoDB is definitely not the silver bullet people hope it is but does work exceptionally well for what it was designed for.

ParadisoShlee on July 8, 2017 [–]

Use case. Use case. Use case.

ezulich on July 10, 2017 [–]

Full disclosure: I am a member of DynamoDB team at AWS, but my opinions here are mine only

Here are some of DynamoDB customers describing their use cases:

- DataXu: "How DataXu scaled its System to handle billions of events with DynamoDB" -- https://youtu.be/lDiI0JMf_yQ?t=765 (from AWS re:Invent 2016)

- Lyft: "Lyft Easily Scales Up its Ride Location Tracking System with Amazon DynamoDB" -- https://www.youtube.com/watch?v=WlTbaPXj-jc. The story "AWS Helps Fuel Lyft's Expansion" in InformationWeek mentions Lyft's use of DynamoDB as well: http://www.informationweek.com/cloud/infrastructure-as-a-ser...

- Under Armour implemented cross-region replication using DynamoDB: https://youtu.be/NtaTC2Fq7Wo?t=699

- Amazon.com: Amazon Marketplace shares their story of migration to DynamoDB: https://www.youtube.com/watch?v=gllNauRR8GM

Finally, DynamoDB was created because Amazon needed a highly reliable and scalable key/value database. As Werner Vogels said it in his blog post announcement of DynamoDB back in 2012 (http://www.allthingsdistributed.com/2012/01/amazon-dynamodb....):

"This non-relational, or NoSQL, database was targeted at use cases that were core to the Amazon ecommerce operation, such as the shopping cart and session service."

ZGF4 on July 8, 2017 [–]

There are a few misguided views in this article and in some of these comments.

1. Every shardable database (Cassandra, Dynamo, BigTable) has to worry about hot spots. Picking a UUID as a partition key is only step one. What happens if one user is a huge majority of your traffic? All of their reads/writes are going to a single partition and of course you are going to suffer from performance issues from that hot spot. It becomes important to further break down your partition into synthetic shards or break up your data by time (only keep a day of data per shard). BigTable does not innately solve this, they may deal better with a large partition but it will inevitably become a problem.

2. Some people are criticizing the choice of NoSQL citing the data size. Note you can have a small data size but have huge write traffic. An unsharded RDBMS will not scale well to this since you cannot distribute the writes across multiple nodes. Don't assume just because someone has a small data set they don't need to use NoSQL to deal with their volume

heisenbit on July 8, 2017 [–]

> Every shardable database (Cassandra, Dynamo, BigTable) has to worry about hot spots.

Yeah, but the issue with DynamoDB seems to be bursts of access triggering "throughput exceptions" caused a very static bandwidth allocation which is going down with the number of shards and not so graceful handling of overload situations.

It is imho. an anti-pattern to split up the bandwidth like they do. It negates the multiplexing gain for no good reason except a rigid control model.

jjirsa on July 8, 2017 [–]

If one user is the majority of your traffic, you don't shard on user ID.

sametmax on July 8, 2017 [–]

It depends a lot of the write. If they can be batched, you can put them in a queue or in redis until it reach a threshold and write the update down in the RDBMS. It won't work for all the use cases, but more often that people think.

jjirsa on July 8, 2017 [–]

If you have a persistent data problem, and introduce redis, then you'll have two persistent data problems.

ZGF4 on July 8, 2017 [–]

Yes microbatching is a great way to get a fixed write rate regardless of traffic, can even do it right at the application layer. The trade-off is that you can theoretically lose one interval of data when your service goes down. This might not matter for analytics workloads with a margin of error but some usecases require confirmed writes

amygdyl on July 8, 2017 [–]

I think the problem is that there are very few suitable length articles that are available for you at no cost, and the debate has been suffering from a lack of understanding in the widest sense of the problem and so little can be found on any website that tries to make a commercial subsistence which is any way better than you scan reading SO randomly the day before you go to present to management for their architectural planning meeting.

/s is I hope self evident

Only seriously for my purposes, the use I get from the typical article on databases which I find is linked to from HN, is a reverse index to the better discussions, long after the discussion is off the front page here.

I'm merely a little confused about the fact that widespread consternation of the quality of geek journalism for programmers is not much more than merely a occasional mention or moan.

Genuinely is a good dose of cynicism in force, which is invisible to me?

I understand that when I see a story about science intended for a general audience, then HN is likely to become home to much greater detail and depth in discussion.

But with a subject line arguing generalised conclusion from a subject matter of database architecture?

Sometimes I think that I'm confused about whether I'm supposed to be confused about the point of the TFA.