freshdesk.com

How we scaled Freshdesk (Part II) - The Sharding and Rebalancing techniques we used - Freshdesk Blogs

Authors

6-7 minutes

This is the second part of the How We Scaled Freshdesk story. In my previous post, I talked about how we tried to vertically scale Freshdesk before we decided to implement sharding with our database. You can find it here.

After using a bunch of methods to scale like R/W split, MySQL partitioning and witnessing them being inadequate for our rapid growth, we came to the conclusion that scaling vertically can only get you so far. The more we thought about it, the more it made sense for us to shard our database.

It was kind of inevitable because sharding was pretty much the only cost-effective way to increase write scalability beyond the instance size. We could also horizontally scale our DB infrastructure in terms of database size, backups, schema changes and more. In our case, we had two main concerns before we took the final call on sharding:

1. No distributed transaction – We wanted all tenant details to be in one shard

2. Rebalancing the shards should be easy – We wanted control over which tenant sits in which shard and to be able to move them around when needed

A little research showed us that there was only one way to go, directory based sharding.

Directory based sharding

Directory based sharding suited our requirements much better than hash key based or range based mainly because it’s simpler to implement. And rebalancing the shards was far easier than with other methods. So, we started caching the directory lookups for fast access and maintaining multiple copies of directory database. We take regular backups of it to avoid a single point of failure.

A typical directory entry looks like this:

tenant_info	shard_details	shard_status
Stark Industries	shard 1	Read & Write

Where, **tenant_info** is the unique key referring to the DB entry, **Shard_details** is the shard in which that tenant exists and **shard_status** tells what kind of activity the tenant is ready for. We have multiple shard statuses like Not Ready, Only Reads, Read & Write etc.

How the directory lookup works

Data can be accessed through multiple entry points like web, background jobs, analytics etc. When a request comes in, an API wrapper accesses the directory to get the appropriate shard and status of the tenant. We tuned the API wrapper to accept the tenant information in multiple forms like tenant URL, tenant ID etc. The shard information returned by the API wrapper contains the shard_details and shard_status of the data. The sharding API even acts as a unique ID generator so that the tenant ID generated is unique across shards.

Why we care about rebalancing

There are instances when a user who initially started out with 1000 tickets per day grows to processing about 10,000 tickets per day. This user could sorely affect the performance of the whole shard. However, we can't solve the problem by splitting up his data into multiple shards because we didn't want the mess of distributed transactions.

So, in these cases, we decided that we'd move the noisy customer to a shard of his own. That way, everybody wins.

How rebalancing works

1. Every shard has its own slave to scale the reads. For example, say Wayne Enterprises and Stark industries are in Shard 1. The directory entry looks like this

Wayne Enterprises	shard1	Read & Write
Stark Industries	shard1	Read & Write

2. Seeing the rate at which Wayne enterprises is growing, moving it to another shard is for the best (averting the danger of Bruce Wayne and Tony Stark being mad at us the same time).

3. So we boot up a new slave to Shard 1, call it Shard 2, attach a read replica to

the new slave and wait for it to sync with the master.

4. We'd then stop the writes for Wayne Enterprises by changing the shard status in the directory.

Wayne Enterprises	shard1	Read only
Stark Industries	shard1	Read & Write

5. Then we'd stop the replication of master data in shard 2 and promote it to master.

6. Now the directory entry would be updated accordingly

Wayne Enterprises	shard2	Read & Write
Stark Industries	shard1	Read & Write

7. This effectively moves Wayne Enterprises to its own shard. Batman's happy and so is Iron Man.

How freshdesk scaled

Now, the only thing left to do is to clean up the duplicate Wayne enterprises data in Shard 1. But that can be done leisurely because it will not affect the writes in either of the shards. And all this can be done through API in Amazon RDS and a few clicks via the Amazon RDS console.

Word of caution

If you are thinking about sharding for your DB, I have some advice for you,

1. Don't do it unless it's absolutely necessary. You will have to rewrite code for your whole app, and maintain it.

2. You could use Functional partitioning (moving an over-sized table to another DB altogether) to completely avoid sharding if writes are not a problem.

3. Choosing the right sharding algorithm is a bit tricky as each has its own benefits and drawbacks. You need to make a thorough study of all your requirements while picking one.

4. You will need to take care of the Unique ID generation across shards.

What's next for Freshdesk

We get 250,000 tickets across Freshdesk every day and 100 M queries during the same time (with a peak of 3-4k QPS). We have a separate shard now for all new sign ups. And each shard can roughly carry 20,000 tenants.

In the future, we'd like to move beyond sharding and explore Multi-pod architecture and also look at Proxy architecture using Mysql Fabric, Scalebase etc.

Further reading

[Tumblr. Massively Sharded MySqQL]

Database Sharding at Netlog

Why don’t you want to shard