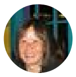
June 5, 2014 DATA · INFRA

HydraBase – The evolution of HBase@Facebook



Zelaine Fong

Rishit Shroff

When we revamped **Messages in 2010** to integrate SMS, chat, email and Facebook Messages into one inbox, we built the product on open-source Apache HBase, a distributed key value data store running on top of HDFS, and extended it to meet our requirements. At the time, HBase was chosen as the underlying durable data store because it provided the high write throughput and low latency random read performance necessary for our Messages platform. In addition, it provided other important features, including horizontal scalability, strong consistency, and high availability via automatic failover. Since then, we've expanded the HBase footprint across Facebook, using it not only for point-read, online transaction processing workloads like Messages, but also for online analytics processing workloads where large data scans are prevalent. Today, in addition to Messages, HBase is used in production by other Facebook services, including our internal monitoring system, the recently launched Nearby Friends feature, search indexing, streaming data analysis, and data scraping for our internal data warehouses.

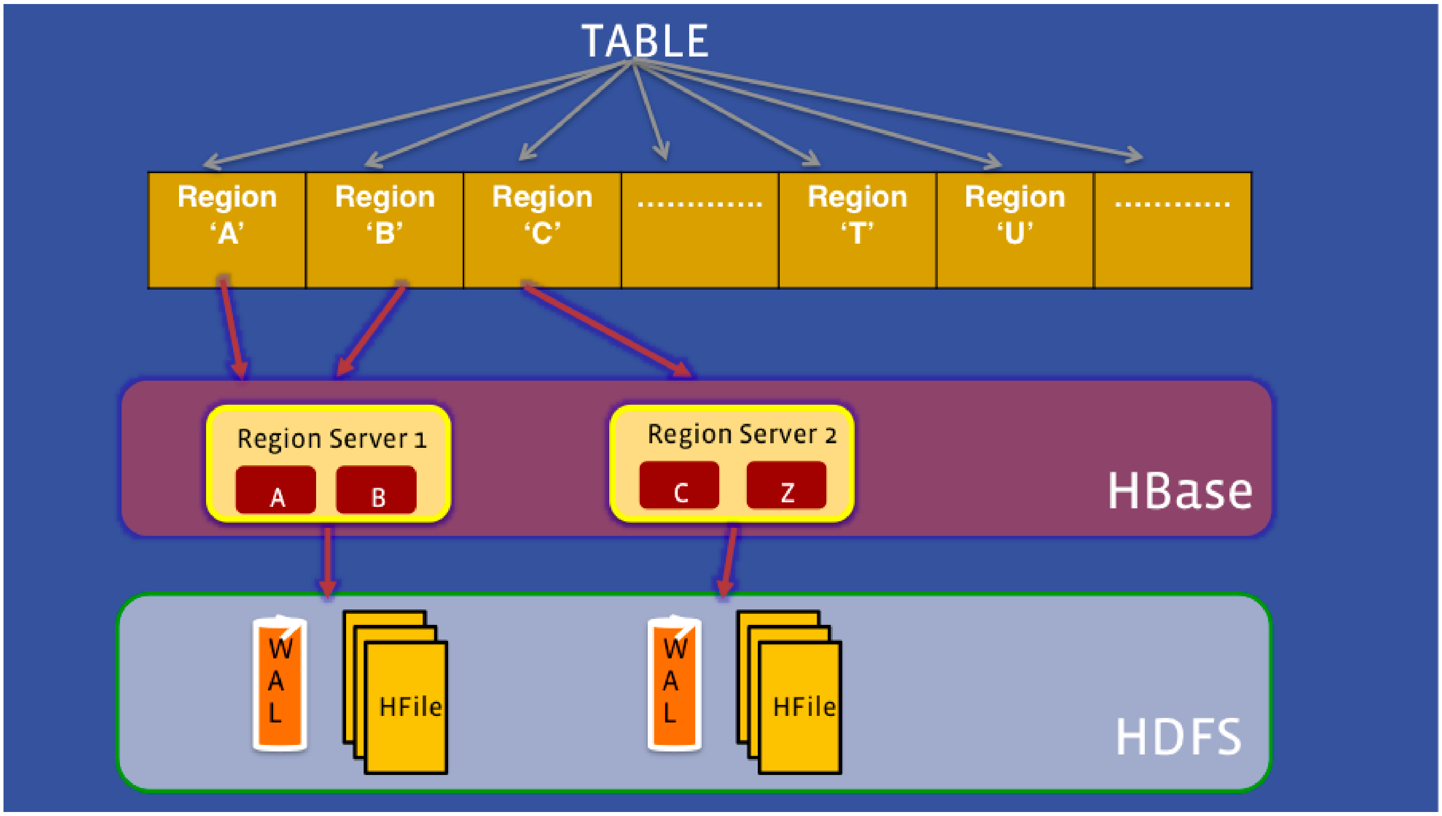
HBase reliability

As is always the case at Facebook, to meet our ever-increasing scale and reach, we are continuing to make improvements in HBase. Reliability, in particular, continues to be an important area for us. Among the challenges that we encounter today with HBase availability are the time to recover individual failing hosts, rack-level failures, and the nuances of running on dense storage. One way of addressing these availability issues is to run HBase in a master-slave setup, asynchronously replicating updates between the two clusters. But this requires failover to be carried out at the cluster level, causes outages on the order of minutes of downtime when failing over from master to slave, and potentially results in data loss due to the asynchronous nature of the replication.

Over the last year, we've been working on HydraBase, which avoids these limitations and provides higher reliability with the same storage footprint as a master-slave replicated setup.

HBase basics

Before going into the details of HydraBase, let's first define some basic HBase concepts. In HBase, data is physically sharded into what are known as regions. A single region server hosts each region, and each region server is responsible for one or more regions. When data is added to HBase, it's first written to a write-ahead log (WAL) known as the HLog. Once written to the HLog, the data is then stored in an in-memory MemStore. Once the data in memory exceeds a certain threshold, it's flushed as an HFile to disk. As the number of HFiles increases with MemStore flushes, HBase will merge several smaller files into a few larger ones, to reduce the overhead of reads. This is known as compaction.



When a region server fails, all the regions hosted by that region server will migrate to another region server, providing automatic failover. Due to the nature of how failover is architected in HBase, this entails splitting and replaying the contents of the WAL associated with each region, which lengthens the failover time.

Enter HydraBase

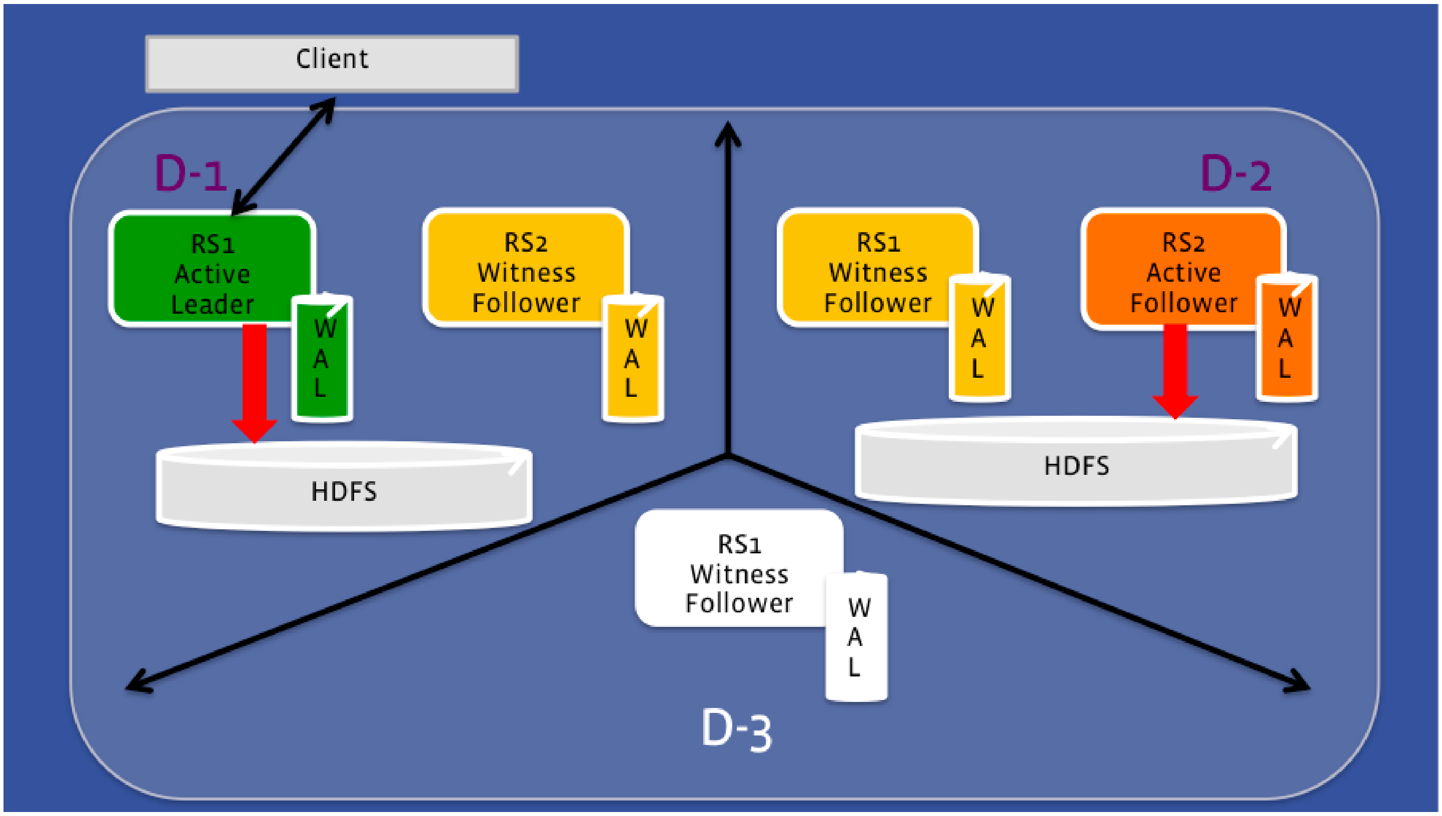
That's where HydraBase fundamentally differs from HBase. Instead of having each region being served by a single region server, in HydraBase, each region is hosted by a set of region servers.

When a region server fails, there are standby region servers ready to service those regions. These standby region servers can be spread across different racks or even data centers, providing availability across different failure domains. The set of region servers serving each region form a quorum. Each quorum has a leader that services read and write requests from the client. HydraBase uses the RAFT consensus protocol to ensure consistency across the quorum. With a quorum of 2F+1, HydraBase can tolerate up to F failures. Each hosting region server synchronously writes to the WAL corresponding the modified region, but only a majority of the region servers need to complete their writes to ensure consistency.

Within each quorum, each member is either in active or witness mode. Active mode members are writing to HDFS and performing data flushes and compactions. Witness mode members only participate in replicating the WALs, but can assume the role of the leader when the leader region server fails.

HydraBase deployment example

The following diagram illustrates an example of what a HydraBase deployment might look like.



In this case, the deployment is across three data centers, with a quorum size of five. With such a setup, leadership can fail over to a witness region server within the same domain. For example, if only the Active Leader fails (see Figure 1), then the Witness Follower in that same data center will assume the role of the leader, and client requests will be re-directed to that region server. If we lose that entire data center (see Figure 2), the Active Follower in the second data center will become the leader. Since the region server in that second data center is also writing data to HDFS, the data is still accessible even though the HDFS cluster running on the first data center is unavailable.

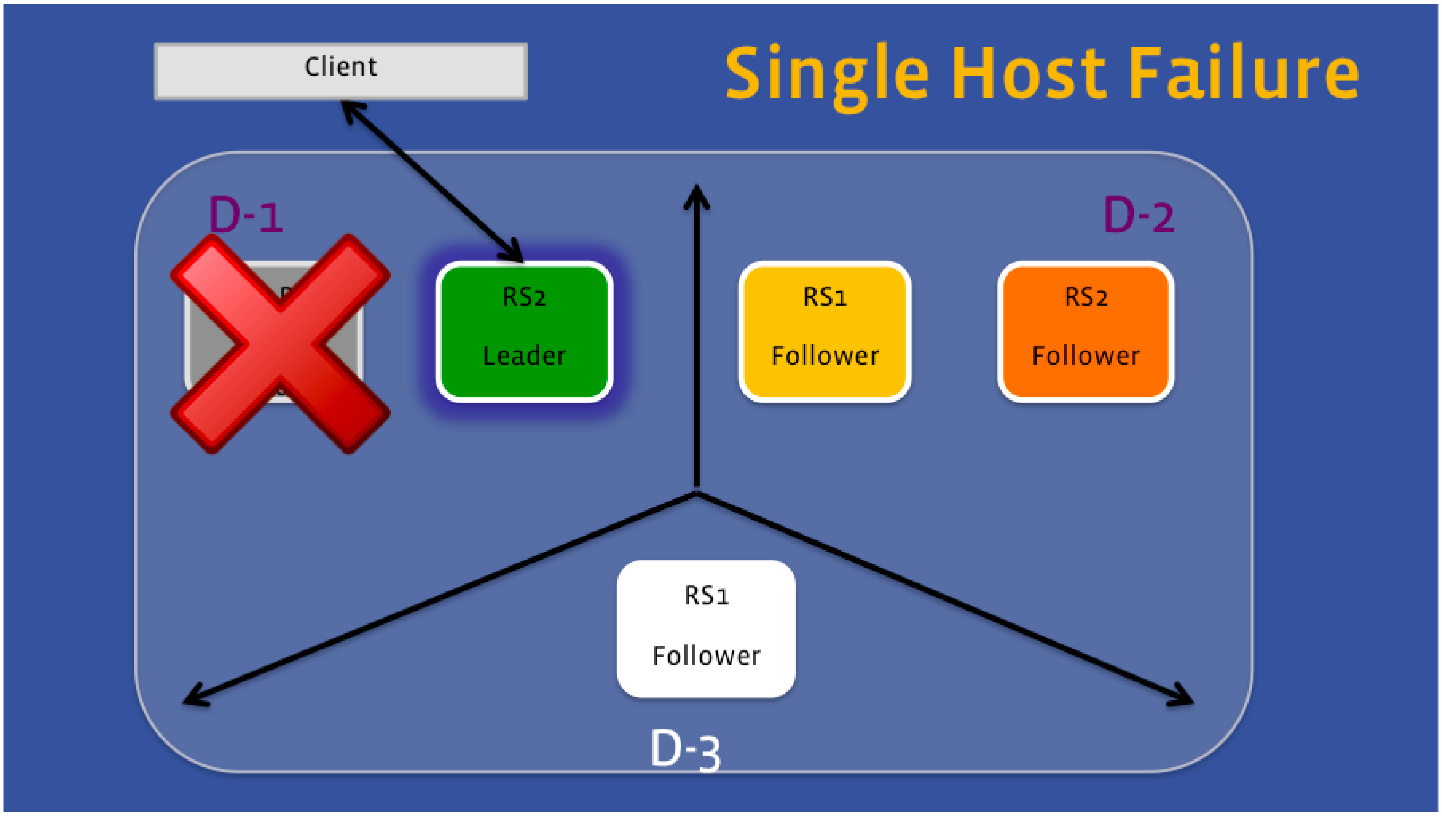


Figure 1

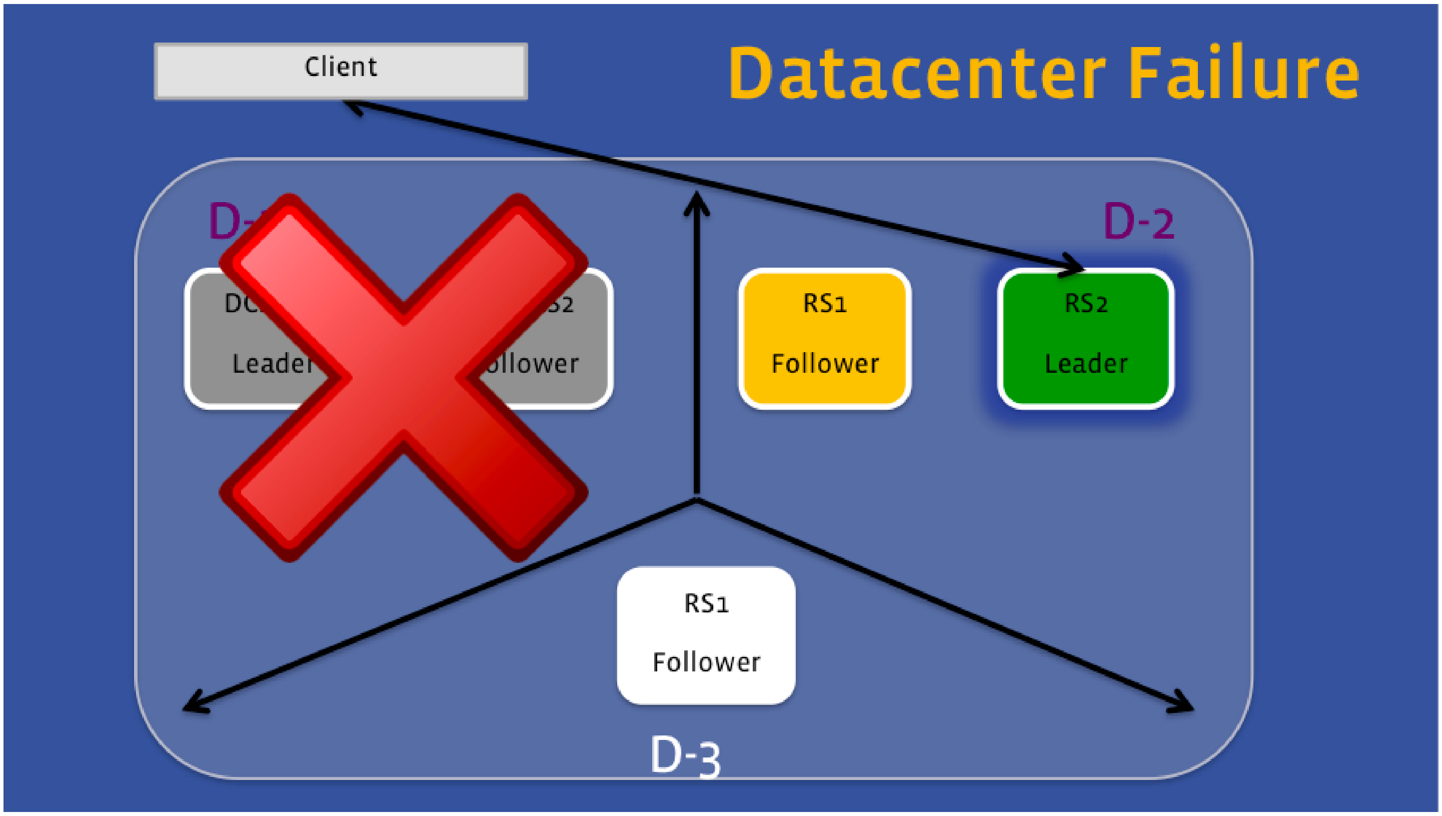


Figure 2

Another way of thinking about HydraBase is we've effectively decoupled the logical and physical replication. In the above example, we have five logical replicas of the data (in each of the hosting region servers), but with the same amount of storage as a master-slave replicated setup. With HydraBase, we can now fail over at the granularity of a single region. Moreover, the failover time is shorter because we no longer need to do log splitting, given that the corresponding write-ahead logs are already on each hosting region server.

Conclusion

With HydraBase, we have the potential to increase the reliability of HBase from 99.99% availability to 99.999% if we deploy HydraBase in a cross-data center configuration as shown above. This would translate to no more than five minutes of downtime in a year.

We also have the flexibility of deploying HydraBase within a single data center. This provides improved failover across hosts as well as racks, compared with HBase.

We're currently testing HydraBase and plan to roll it out in phases across our production clusters. Looking ahead, other enhancements we're considering include allowing reads to be serviced from members of the quorum other than the leader, as well as keeping in-memory copies of the data in witness mode quorum members to further speed up failover.

Thanks to all the engineers who have contributed to the HydraBase project including Amitanand Aiyer, Arjen Roodselaar, Gaurav Menghani, Jiqing Tang, Liyin Tang, Michael Chen, and Yunfan Zhong.

Like Share 940 people like this. Sign Up to see what your friends like.

More to Read

Saving capacity with HDFS RAID

Recommended

TAO: The power of the graph

The Underlying Technology of Messages

Under the Hood: Building and open-sourcing RocksDB

Presto: Interacting with petabytes of data at Facebook

Want to work with us?

Join the team, we're hiring! Here are some of our current open positions:

Partner Engineer, Games and Apps
Web Software Developer, Web Client (WhatsApp)
Solutions Architect (Instagram)

More Engineering Positions

Connect

Like 1,282,089 people like this. Sign Up to see what your friends like.

Follow us on Twitter

Keep Updated

Stay up-to-date via RSS with the latest open source project releases from Facebook, news from our Engineering teams, and upcoming events.

Subscribe