Apache HBase – Read-Replica clusters

Zach York and Ajay Jadhav

Problem

Users of Apache HBase who rely on HBase for mission critical applications need their data to be reliably accessible to a high degree. HBase has made great strides in answering this need by adding features such as region replicas, multiple masters, etc. However, this only allows for high availability of data at the cluster level. If something goes wrong with a cluster and that cluster goes down, the user will not be able to access their data until the cluster returns to a functional state or another cluster has been brought up to service the data.

Recent changes in HBase have enabled users to run their HBase cluster against an external distributed file system. This decouples the storage and compute layers of HBase allowing users to shut down their cluster without losing data. This feature allows HBase to take resiliency to the next level: multiple clusters. Since the data is outside of the HBase system, multiple read clusters can point to the same set of data.

We propose adding functionality to allow HBase users to configure multiple read-only clusters to the same set of data to ensure read availability in the case of one cluster having trouble.

Technical Approach

The current approach is based off of a fairly manual approach. For proposals on how we'd like to improve this, see Future Work. This approach tends to work best with bulk loading data into the system to reduce the number of refreshes.

Multiple Meta tables

HBase stores Region information in the hbase:meta table including start and end key and RegionServer IP address. This causes issues when there are multiple distinct HBase clusters pointed to the same root directory as these clusters will assume that RegionServers are unhealthy when the HMaster is unable to communicate with the RegionServers. Therefore, it will reassign these regions to a new RegionServer causing hbase:meta contention/deadlocking between the two clusters.

To address this, we have created separate meta tables for each read replica cluster. The approach here is to create a new hbase:meta* table underneath the root directory by appending a custom suffix at the end of the table name (see https://issues.apache.org/jira/browse/HBASE-18444). This allows each cluster to maintain its own metadata.

Global Read-Only mode

To ensure that only one HBase cluster can write/modify data or metadata, a global read-only status needs to be set for all HBase read-replica clusters. This will effectively disable all DML (Data

Manipulation Language) and DDL (Data Definition Language) operations on the read-replica cluster to prevent modifying any of the underlying data. All attempts to write will throw an exception.

Add ability to keep metadata in sync

Since the primary (writer) cluster can do operations such as create/drop tables, create/drop namespaces, and split/merge regions, the read-replica clusters need a way to update their hbase:meta table to reflect these metadata changes. This ability will look at the backing data and make sure its hbase:meta is in sync. This is also used to populate the hbase:meta table of a newly created read-replica as it will have an empty hbase:meta.

Add ability to refresh HFiles

Since the primary cluster is the only HBase cluster that accepts writes, the read-replica clusters need to be able to refresh the HFiles that they can see. This is exposing pre-existing HBase functionality for refreshing HFiles from the storage. This work is being done here: https://issues.apache.org/jira/browse/HBASE-18448.

Upgrade considerations

Upgrading with HBase read-replica clusters will be fairly simple as the clusters are for the most part, independent. The main case where upgrading would be an issue would be if something with the storage format changed that requires migrations. In this case, it would not be likely to do a live upgrade.

Please comment if there are any upgrade considerations that are not addressed.

Testing

All functionality will be tested with unit tests. This includes all the major sections laid out in the technical approach section. Integration between multiple clusters will be tested with Integration testing for ensuring read-replica clusters exhibit all the desired behavior. Integration testing will be especially important for the multiple meta tables, add ability to keep metadata in sync, and add ability to refresh HFiles sections.

Documentation

Documentation will be very important as this is currently manual so we need to provide procedural and conceptual information to the user.

To ensure that users understand how to use this new feature, we will add a new section in the HBase book/reference guide to talk about all the new user facing options and interactions.

If needed, a blog post should introduce the feature and explain how to use and configure read-replica clusters. A blog post about this feature on EMR can be found here: https://aws.amazon.com/blogs/big-data/setting-up-read-replica-clusters-with-hbase-on-amazon-s3/

Limitations

- Zookeeper (ZK) notifications: Each cluster in read-replica has its own ZK for maintaining the quorum status, configuration management etc. What this means is that any ACL, quota related updates on primary will not be automatically propagated to replicas. Currently, one will have to re-run these configuration changes on replicas as well.
- Data locality: In case the data resides mostly on primary cluster, this can impact the query performance on replicas. Essentially, every query will result in a network call as the data is on primary cluster till it is cached on replicas. For AWS S3, this is not a concern at all as all the clusters will experience the same latency till the data is cached locally in-memory or disks.

Milestones

Milestone 1: Create JIRAs

Summary: Create known JIRAs and attach pre-existing patches
Schedule: 1 week
Exit Criteria:

- JIRAs created
- Patches attached for initial feedback.

Milestone 2: Gather Community Feedback

Summary: Gather any major design issues to address
Schedule: 1 week
Exit Criteria:

- Gather additional considerations brought up by community
- Gather Integration Testing cases

Milestone 3: Implement Base Functionality with Unit Tests

Summary: Push the basic functionality back into master or a feature branch with Unit Tests
Schedule: 3 weeks
Exit Criteria:

- All initial JIRAs/patches are pushed to master
- Follow-up JIRAs created for additional issues

Milestone 4: Integration Testing

Summary: Add any gaps in Integration testing

Schedule: 2 weeks
Exit Criteria:

- Integration Tests for all external features of read-replica clusters

Milestone 5: Reference Guide

Summary: Add section in the reference guide for read-replica clusters
Schedule: 1-2 weeks
Exit Criteria:

- Section for read-replica clusters complete in reference guide

Milestone 6: Backport to HBase 1.x

Summary: Read-Replica clusters backported to the latest unreleased 1.x HBase.
Schedule: 2 weeks (Originally developed off of 1.3.x, but need to backport any additional features)
Exit Criteria:

- All Read-replica patches backported to the next 1.x release

Future Work

Automate replication work

While this feature is manual in its current form, in order to have wider adoption, there needs to be effort taken to make it work automatically when the feature is enabled. This includes automatically refreshing HFiles and metadata. This effort will likely take a significant amount of design which is why it is not included in the first release.