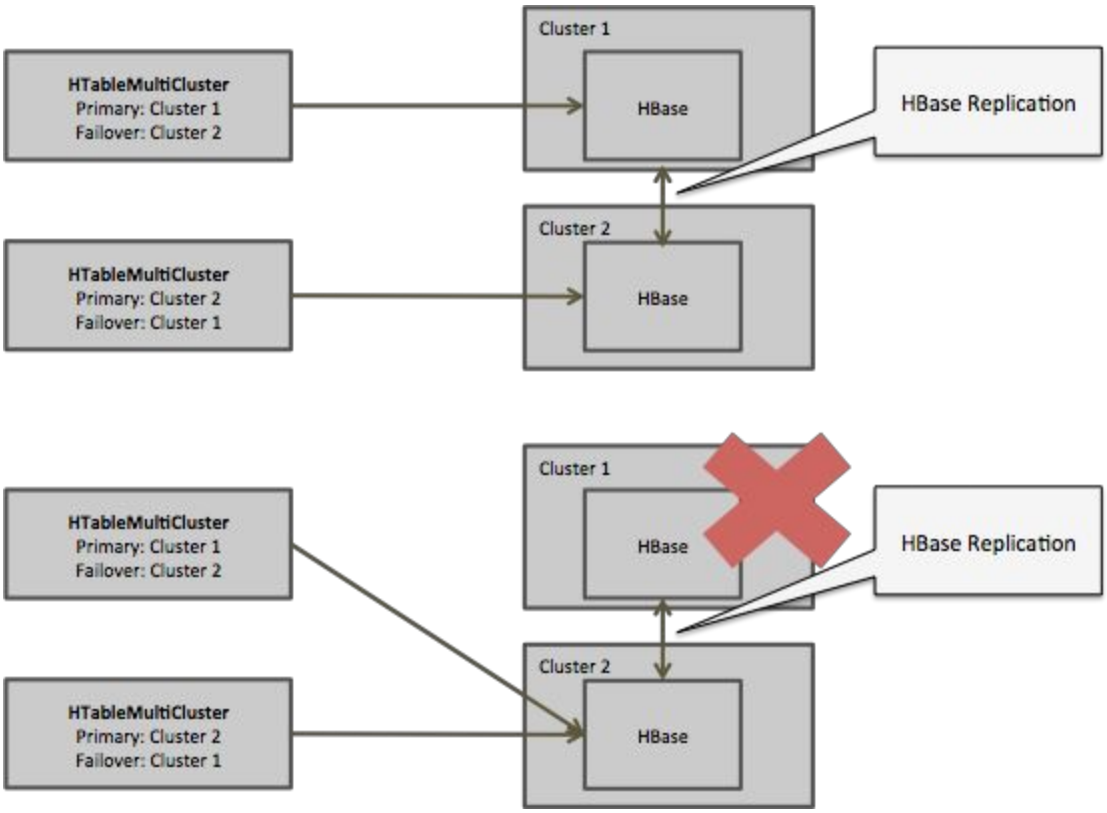
Multi-HBase Cluster Client
Design Document

Use Case

Allow an HBase Client to have higher up time SLAs then one HBase cluster can provide. A single HBase cluster may be affected by low running GCs, Region Server Failures, planned cluster downtime, or unplanned cluster down time.

Goal

The proposed solution is a multi-cluster HBase client that relies on the existing HBase Replication functionality to provide an eventual consistent solution in cases of primary cluster down time.



Additional goals are:

- Be able to switch between single HBase clusters to Multi-HBase Client with limited or no code changes. This means using the HConnectionManager, Connection, and HTable interfaces to hide complexities from the developer (Connection and Table are the new classes for HConnection, and HTableInterface in HBase version 0.99).

- Offer thresholds to allow developers to decide between degrees of strongly consistent and eventually consistent.
- Support N number of linked HBase Clusters

PoC

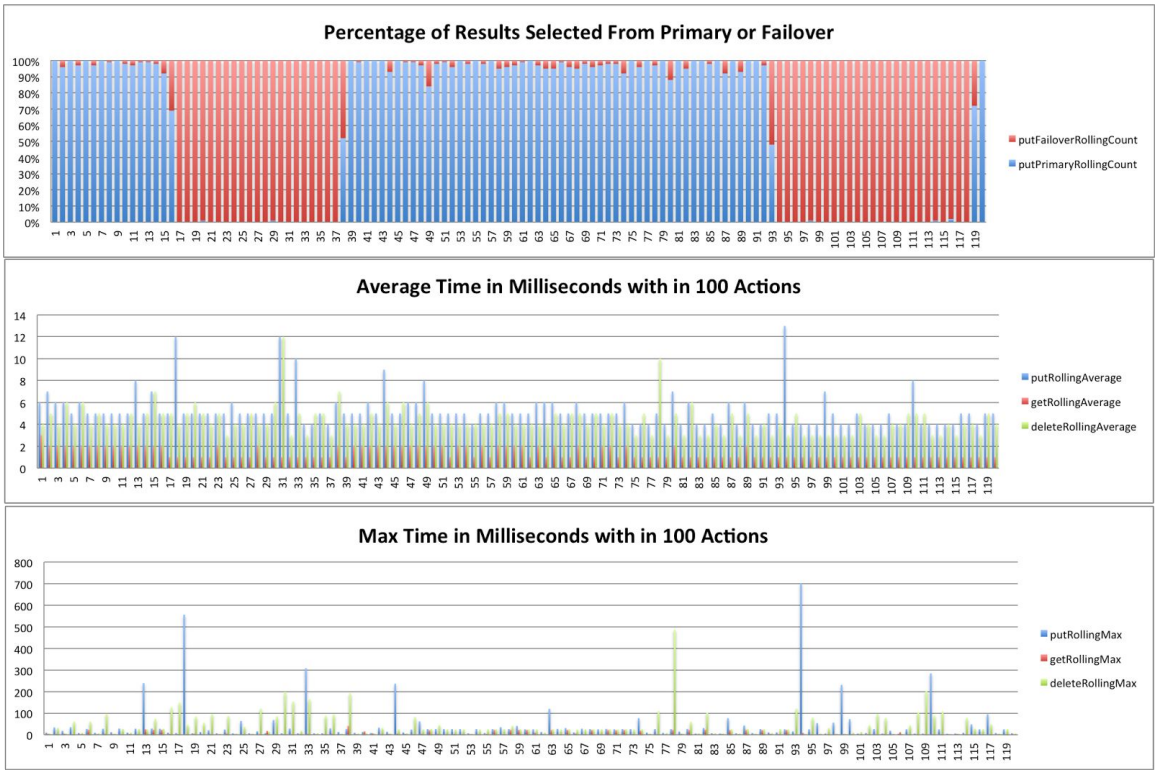
This document is accompanied by an early draft of the implementation that can be found at the following GitHub, under the apache license.

https://github.com/tmalaska/HBase.MCC

There are a number of improvements that need to be made but this document won't go into that level of detail. But here is some high level points.

- Improve Scan failure to recover mid scan
- Improve thread usage
- Remove Multi Cluster HAdmin from initial code drop because it is only best effort
- Allow checkAndPuts and checkAndDeletes if developer deliberately ask for it in the configuration.

The initial results from the PoC implementation are positive. There is a lot of work still to be done, but these times are a lot better than any single cluster can give you.



On the top graph the blue bars are when the operations hit the primary cluster and red lines are operations that executed on the failover cluster. When you see more red it is because the primary cluster was bounced.

The next graph shows the average execution time for gets, puts, and deletes. The final graph shows the slowest 1% for the same time period.

Two interesting points about graphs:

1. These graphs are generated from output from the client so we can give the client user full understand of where the requests are coming from
2. Note that even when both clusters are up some requests are going to the primary cluster. This is because the client was located on the failover cluster and the network latency would make the request to the primary to slow to allow the failover to win with the parameters that were given.

Technical Approach

In the following sections we go over additional configurations and code changes to allow us to reach the goals above. But before going into that, lets define some terms that we will use throughout the document.

Term	Definition
Primary HBase Cluster	The HBase Cluster as the Cluster you first try.
Failover HBase Clusters	These are N number of HBase Cluster that we can go to when the Primary Cluster is down
Strongly Consistent	Means that when you request a value it will be the most up to date as possible
Eventual Consistency	Means when you request a value it may be right or it may be out of date, but at some point it will be right.

Configuration

The key to this solution avoiding coding change to the end developer is through additional configurations parameters. There are only a couple but they are very powerful so we will take time to drill down into each one of them.

failout.clusters

This is a comma separated list of cluster names that will represent the clusters we will fail over too if our primary cluster is not responsive. The design of this parameter is very much like how Flume defines Sinks, Sources, or Channels for a flume agent.

Now you will note that there is no primary cluster definition, that is because we will leave the normal configurations as pointing the client to the primary HBase Cluster.

hbase.failover.cluster.{failover.cluster.name}.{overrided client parameter}

These parameters will allow us the option to override client configurations from the primary HBase Cluster configurations for the one that differ in the failover clusters.

Here is an example that will allow us to point to different zookeeper quorums for the different HBase clusters

```
hbase.failover.clusters=foo,bar
hbase.zookeeper.quorum=1.0.0.1
hbase.failover.cluster.foo.hbase.zookeeper.quorum=1.0.0.2
hbase.failover.cluster.bar.hbase.zookeeper.quorum=1.0.0.3
```

hbase.failover.mode

To identify that we are running in Master-Master or Master-Slave. The two options would be “Master-Master” or “Master-Slave”. This will affect if puts/deletes will be sent to a failover cluster.

hbase.wait.time.before.accepting.failover.result

Define how many milliseconds the client will wait for the primary request to return before we accept a response from a failover cluster.

There are several options for the developer here. If this value is set to 0 then the client will accept the first response that returns no matter if it comes from the primary cluster or if it comes from the failover cluster. The advantage of setting this to 0 is the user will get an answer as fast as possible, but the downside is we may turn down a primary response for a failover response because of a couple millisecond difference.

You may ask, "why is this a big deal?" The answer is the responses from a failover cluster have eventual consistent values so the value may not be up to date, whereas the primary cluster will have strongly consistent values.

An alternative approach would be to set this value to 50 milliseconds, which will give the primary cluster more then enough time to get a response on a common get that even has to hit disk. With a delayed acceptance of a failover response we get a much higher percentage of strongly consistent values. In the optimal experience this will allow the user to get strongly consistent values in most cases until there is a major GC or Region Server failure, and only then will the user get eventual consistent values.

hbase.wait.time.before.request.failover
This parameter can be used in combination with "*hbase.wait.time.before.accepting.failover.result*" but is very different. This parameter will define a delay before the addition request is sent out.

If this value is set to 0 then in all cases a get/scan command will be sent out to the Primary HBase cluster and all Failover Clusters, which will give us our highest possible chance of our lowest response times.

However there are reasons for setting this to a higher value like 30 milliseconds.

1. Reduce the number of outgoing response from the client. Which consume CPU and network resources.
2. Better leverage the block caches of the clusters. One can imagine a design where depending on the rowkey a different primary HBase Cluster is defined. This would allow the block cache to focus on caching different information across the two clusters. If a get always goes to both clusters then the same data is cached in both cluster.

Now note, most of the cases where reads are slowed down because of GCs and Region Server failures will be addressed from the HBASE-10070. This documents design for a multi-cluster client solution will be independent from HBASE-10070 solution and in fact be improved through it. However, it is important to note HBASE-10070 because it will provide faster failover with in the primary cluster.

hbase.wait.time.before.mutating.failover

This parameter defines how long the client waits for a put/delete submission to complete from Primary HBase Cluster before sending it to the failover clusters.

One thing to note is we will be setting the timestamp in the client so even if the put or delete make it to all clusters they will resolve correctly.

So if this parameter is set to 0 then in all cases a put or delete will be sent to all clusters. The downside to this is time and bandwidth and more stress on all the WALs on all the Clusters.

If we sent this parameter to something high like 10 milliseconds we rarely have to send it to anything but the Primary HBase Cluster.

Also not increment command is not included in this list, this is because increment is a little more complex to get correctly with multiple clusters. For now increment commands will only be sent to the primary cluster.

Other Configs

There are a number of additional configs of lesser importance like:

- hbase.wait.time.before.mutating.failover.with.primary.exception
- hbase.multi.cluster.connection.pool.size
- hbase.wait.time.before.trying.primary.after.failure
- hbase.multi.cluster.thread.pool.size
- hbase.multi.cluster.allow.check.and.mutate

HBase Core Changes

Connection / HConnectionMultiCluster

This will be a new class that will extends Connection/*HConnection*. It will also have Connections/*HConnections* for all the HBase clusters configured for primary and failover. Then it returns a Table/*HTableInterface* it will return a *HTableMultiCluster*. The developer should never directly use this class, unless they want to do specific multi-cluster functions like may be list cluster names. But the hope is they will just have in their code Connection/*HConnection* so they can switch between single or multi-cluster mode without code changes.

Table / HTableMultiCluster

This will override *HTable*, which implements Table/*HTableInterface*. It will override functions like put, get, delete, checkPut, … and add logic for failover. Also the goal is the normal developer not use this class directory but use it through Table/*HTableInterface* so that they can switch between single and multi-cluster mode without code changes.

Major point to note is how the timestamp is set in the HTableMultiCluster implementation. If the users don't changed the timestamp for a mutation then the

HTableMultiCluster will set the timestamp to the time on the client box. This is different from the normal HTable in that the timestamp is set on the Region Server. This change is because we can find ourselves in a case where a put may land on the primary cluster and the failover cluster. Having the same timestamp will allow no issues after replication happens.

HConnectionManager

The only change here is to check if the incoming config is one that is request to run in multi-cluster mode and if so construct and return an instance of *HConnectionMultiCluster.*

HTable Function Break Down

Here is a list of the functions in HTable and how they would function in multi cluster mode.

Function	**Behavior in Multi Cluster Mode**
append	Will not work in multi cluster mode
batch	Will work but will not allow Appends or Increments
batchCallback	Will work but will not allow Appends or Increments
batchCoProcesserServer	Not in first release
checkAndDelete	Only if hbase.multi.cluster.allow.check.and.mutate is marked as true
checkAndMutate	Only if hbase.multi.cluster.allow.check.and.mutate is marked as true
checkAndPut	Only if hbase.multi.cluster.allow.check.and.mutate is marked as true
clearRegionCache	Will do best effort
close	Will do best effort
coprocessorService	Not in first release
delete	If hbase.failover.mode is set to Master-Master
exists	Yes
existsAll	Yes
flushCommits	If hbase.failover.mode is set to Master-Master
get	Yes

getAllRegionLocations	No
getMaxKeyValueSize	No
getName	No
getOperationTimeout	No
getRegionCachePreFetch	No
getRegionLocation	No
getRegionInRange	No
getRowOrBefore	Yes
getScanner	Yes
getTableDescriptor	Yes
getTableName	Yes
getWriteBufferSize	Yes
increment	No
incrementColumnValue	No
isAutoFlush	Yes
isTableEnabled	No
mutateRow	If hbase.failover.mode is set to Master-Master. Also no increment or append
processBatch	If hbase.failover.mode is set to Master-Master. Also no increment or append
processBatchCallback	If hbase.failover.mode is set to Master-Master. Also no increment or append
put	If hbase.failover.mode is set to Master-Master.
setAutoFlush	Yes
setAutoFlushTo	Yes
setOperationTimeout	Yes
setWriteBufferSize	Yes