HBASE-11165 - Scaling to 1 Million regions and beyond
Virag Kothari, Francis Liu

## Overview

Being able to support a high number of regions in a cluster (ie millions) enables us to scale a cluster without having to deal with large regions. Which for HBase has a number of problems:

1. Write amplification – A common problem with LSM: in an unevenly distributed write workload, more data will have to be rewritten during compaction even though a very small subset of a region's key space is updated.
2. Load balancing – smaller regions allow for more granular distribution of load across region servers.
    a. Region management – major compaction, splits, etc are much less taxing on a few region servers.
    b. Workload parallelism
    c. Failure recovery- with larger regions data locality takes much longer to recover when a region fails over. Failed over traffic to a single RS will be much higher as well.
3. Ridiculously long region scan times for scans with high selectivity

Scaling a HBase cluster to support regions in the order of millions poses scalability challenges across two dimensions:

1. Cluster region count
    a. Master region management (assignment, creation, etc)
    b. RS region management (open, close, split, etc)
    c. Region metadata storage AKA hbase:meta
2. RS Region density
    a. efficient use of memstore
    b. tradeoff between partial flushes and disk utilization

This may also push the limits of HBase core dependencies:

1. HDFS scalability - # of files, # of blocks, NN ops throughput
2. Zookeeper – r/w throughput, notifications

There will likely be other issues and the intent is to surface them, so that they can be addressed or help drive features.

As a first step, experiments have been done to identify issues related to cluster region count. We present the observations, recommendations along with the supporting experiments (and jiras).

## Observations/Recommendations

**Regionstates lock thrashing**

- Allow for more parallelism when synchronizing on RegionStates (ZkEvent worker blocked 44% of time on this). Less “global” locking.
- Reduce the no of calls on regionStates in zk event worker thread

**Assignment notification improvement**

- Separate zk watcher for watching assignment related events, avoid any delay caused by other listeners. As well as avoid the overhead of other listeners have to got through millions of notifications.
- Skipping OPENING events from waiting queue if OPENED events are present

**Split Meta** ( 3gb with 3 M entries using HexStringSplit as algo)

- Existing table with ~70k regions is 200MB, then 7Million is 20GB, 70Million is 200GB

**Write less to ZK or store state elsewhere (or use forceSync=no)**

- with forceSync=yes disk utilization was hitting 50% peak (85%)

**Parallize ZK deletion**

- Delete unassigned znodes during clean cluster startup takes forever
- 300,000 znodes under region-in-transition takes more than an hour

**Horizontally scale region Assignment**

- High cpu utilization during assignment of 3M regions, need more cpu

**Parallelize meta scan**

- Single scanner takes a few minutes for 3M regions with batch size of 100 and even 10,000
- Also start assigning in parallel along with scanning
- Also whether so many meta scans required or not (3 before clean assignment)

**Parallelize processing of unassigned znodes during master failover mode**

- Region processing is done serially

**Some sort of bulk assignment during balance**

- 20 mins for 34000 regions
- 44 mins for 69,000 regions -

**BulkDisabler should use a bulk RPC call for opening regions (just like BulkAssigner)**

- 30 mins (183 ops/sec)

**Flush table cmd should make RPC calls to region server in parallel**

- Maybe some primitive bulk rpc api?

**Improve logging in ZkEventWorker and other places**

- Bulk disabler prints all the regions (3M entries)

**Require 4gb to list hdfs region dirs under table dir**

- Have an HDFS list api to return entries in batch.
- bucket/tree-ify regions

# Region Count Experiments

**Setup**
330 nodes

**Scenarios**
- 50 regions per RS (16500 regions on cluster)
- 1,000 regions per RS (330,000 regions on cluster)
- 10,000 regions per RS (3,300,000 regions on cluster)

*Actual number of regions may slightly vary as some nodes turned up faulty during the course of the experiments.

**Tests**
- Clean Startup
- Failover startup
- Flush
- Failover (1 rack down, 2 rack down)
- Failover w/ META (1 rack down)
- Balance (1 rack is empty, 2 rack is empty)
- Create table
- Disable table
- Enable table

**Configs**
hbase.bulk.assignment.threadpool.size=30
hbase.assignment.zkevent.workers=20
hbase.hregion.open.and.init.threads.max=50
hbase.master.executor.openregion.threads = 15
hbase.master.executor.closeregion.threads=15
hbase.master.executor.serverops.threads=15
hbase.regionserver.executor.openregion.threads=10
hbase.regionserver.executor.closeregion.threads=10

**50 Regions**

Create table

- Entry in hdfs and updating meta (45 secs)
- Bulk assigning finished in ~3 mins (HMaster in memory states also updated)

Disable

- 3mins

Enable

- 3mins

Kill a rack (34 servers)

- 54 secs from detection

Balance (1 rack/34 servers)

- 72 secs

Kill two racks (with meta)

- 2 mins 16 secs

Balance(2 racks - 69 servers )

- 137 secs

Master failover: (Almost all regions to be assigned)

- 3 mins 44 secs

**1000 Regions**

Create table:

- 12 minutes for create region files on hdfs (460 regions/sec)
- ~3 min to write to META
- Bulk assigning finished in ~3 mins (Bulk assign RPC call done for all RS)
- Regions Online time - ?
- Master state update : ~20 mins
- Assignment - total time 23 mins (239 assignments/sec)

Disable

- 30 mins (Both assignment and master memory zk update)
- RPC calls in parallel but not a bulk RPC call for a single region server

Enable

- 21 mins (Both assignment and master memory zk update)

Recover after kill a rack (no meta - 34 servers (34000 regions)) :

- ~2mins from detection

Balance a rack (34,000 regions)

- 20 mins for Simpleloadbalancer - Not using bulk assigner

Kill two racks (with meta)

- Assignment - 2mins
- Master memory zk update - 4 min

Balance(69 servers (69,000 regions))

- 44 mins for Simpleloadbalancer

Master failover: (Almost all regions to be assigned)

- 1 hr 10 mins
- Takes more time as each dead region server is processed by a servershutdownhandler thread and each thread does round robin assignment of regions belonging to the dead server

Size of meta: (331,000 regions (~2kb each))

**10,000 Regions**

*Incomplete*

Create Table

- Took ~4 hours to create the regions on hdfs
- Had issues with assignment

Clean Startup

- Meta scan took some time since it was done serially (and done twice)
    - Scanning META for 3M entries: 6 minutes - batch size is 100
    - scanning META for 3M : 1 minute: batch size is 10000
- Took ~35 mins for RS to register all regions are online (3.3M)
- Of the 3.3M only ~500 regions are marked online in RegionStates, the rest are still in regions in progress
- Most time was spent due to lock contention on RegionStates
- ZK notification had no backlog
- No outstanding requests on zk server metrics
- High cpu usage 60-70% seems to be peaking at lock contention for RegionStates

Extra changes:

Separate ZookeeperWatcher instance for AssignementManager, share watcher caused some listener to slow down notification processing.

HDFS config
dfs.namenode.fs-limits.max-directory-items=6000000

ZK config
jute.maxbuffer=200MB

hadoop dfs -ls required HEAP of > 3GB

**Follow up test on smaller cluster (1 Master, 3 ZK, 23 RS, 1 Million regions)**
**Configs:**
HDFS config
dfs.namenode.fs-limits.max-directory-items=6000000

ZK config
jute.maxbuffer=200MB

hbase
mslab.chunksize=1024

**Observations**

Startup took ~7 hours
region shutdown handling - ~4mins (1 server), ~9 mins (2 servers)
bring up one server and balance > 1 hr
Even at stable state cpu utilization was jumping between 5% to 50%
20GB heap was not enough for master to startup
zkserver disk utilization was around 50% and peaked at 85%

**With forceSync=no**

Startup took ~1.5 - hours this will likely be faster on a larger cluster (ie in 300 node cluster RS opened all 3.3M regions in 30 mins). While in this case master was not behind by only a few mins towards the end.

**Hacks on 1M test**

- give AM it's own ZKW
    - Slowdown issues due to other listeners in the pipeline
- have regionStates and regionsInTransition in RegionStates class use ConcurrentSkipListMap so that getter access doesn't have to be synchronized
    - avoid lock thrashing
    - even refreshing master admin page caused contention
    - Without this master would die because of memory/gc issues.