New Developments in HBase

Ted Yu
Agenda

• Bulk Loaded HFile Replication
• End to end Offheap Read Path
• Date Tiered Compaction
• Q/A
About myself

• Been working on HBase for 5 years
• HBase committer / PMC
• Staff Engineer at Hortonworks
Replication Overview

• Data synchronization between clusters
• Supports cyclic replication
• Configurable at table / column family level
• Push based implementation through log shipping
• Disaster recovery, geo-distributed serving
Replication state

- Maintains its state in ZooKeeper
- Default: /hbase/replication
- The Peers znode:
  - Default: /hbase/replication/peers

```
/hbase/replication/peers:
  /peer1: zk1.host.com,...:2181:/hbase
    /peer-state: ENABLED
    /tableCFs: table1;table2:cf1
  /peerN: ....
```

Cluster key = ZK quorum:ZK client port:HBase root znode

Replication Status

Only Replicate this table column families
Replication state for region servers

- **The RS znode:**
  - Default: /hbase/replication/rs

  /hbase/replication/rs:
  - /rs1,16201,1234:<
    - /peer1: <
      - /wal1:9086<
      - /walN:0
    - /peerN: ...
  - /rsN,16201,1234: ...

  - Server name = hostname, port, startcode
  - Peer ID, where logs should be replicated
  - WAL log name & read offset
Previous Replication Design

Source Cluster

Batch

Bulk load

ZooKeeper

.../peers/
.../rs/

Region Server

Replication Source/End Point

WAL

1

2

1

Replication Source/End Point

Peer Cluster 1 [tableCfs - 1]

Region Server

Replication Sink

1

Batch

Table

Peer Cluster 2 [tableCfs - ]

Region Server

Replication Sink

2

Batch

1

Table

.../peers/
.../rs/
Bulk Loaded HFile Replication
HBASE-13153

Source Cluster

Region Server

Replication Source/End Point

WAL
1
2
1

Replication Source Manager

Peer Cluster 1 [tableCfs - 1]

Region Server

Replication Sink

1
1

Table

Batch

Peer Cluster 2 [tableCfs - ]

Region Server

Replication Sink

2
1

Table

Batch

ZooKeeper

.../peers/
.../rs/
.../hfile-refs/
Replication state

```
/hbase/replication:
/hbase/replication/peers:
/hbase/replication/peers/1: \[XX.XX.XX.XX:2181\]/hbase
/hbase/replication/peers/1/peer-state: ENABLED
/hbase/replication/rs:
/hbase/replication/rs/host-XX-XX-XX-XX,16000,1444052020616:
/hbase/replication/rs/host-XX-XX-XX-XX,16000,1444052020616/1:
/hbase/replication/rs/host-XX-XX-XX-XX,16000,1444052020616/1/host-XX-XX-XX-XX%2C16000%2C1444052020616.1444062830797:
/hbase/replication/rs/host-XX-XX-XX-XX,16201,1444052024891:
/hbase/replication/rs/host-XX-XX-XX-XX,16201,1444052024891/1:
/hbase/replication/rs/host-XX-XX-XX-XX,16201,1444052024891/1/host-XX-XX-XX-XX%2C16201%2C1444052024891.1444062831922: 9086
/hbase/replication/hfile-refs:
/hbase/replication/hfile-refs/1: 054b0ba74c684bf58da71f2390998920_SeqId_14__, b124a633a07d4f3f9c6af41ac23efed2_SeqId_14__, e02d8006d32d40d5a8216c6b95389f09_SeqId_14__, 853c558f93834270b8519facefd5c2ab_SeqId_14__, 3837019154634df1b7702842f8877a79_SeqId_14__, f376f4e18efa4046863d6e0e2b26c5ba5_SeqId_14__, 1be5276f6a77447e80d60e25af7eab0_SeqId_14__, 0a521bda3bdf4dc5a9cf0679a09cc926_SeqId_14__, d16c4fd80b594306a185bf8031e87aad_SeqId_14__, b454831193ea497782e3ea0971a8bfd3_SeqId_14__, f9bf4c0974c646728f0b0756367b247e_SeqId_14__,
```
Motivation for Offheap Caching

- On heap LRU cache limited by Java heap size
- Solution for GC pause - Offheap bucket cache
- Ideally data can be directly served from off heap
- Block copied to new byte arrays every time the block is read from off heap cache – increases GC
- Cells are backed by byte arrays only (See getXXXArray() APIs)
Offheap Caching – Cont’d

• HBASE-11425 Cell/DBB end-to-end on the read-path
• HFileBlock backed by offheap ByteBuffers (BC buckets)
• ByteBuff: an abstract class
• MultiByteBuff - Data split across N ByteBuffers. One block backed by one MultiByteBuff
• Change read path to deal with BBs rather than byte[]
• Change KVComparator to CellComparator
• Comparators do comparison on ByteBuffers
Offheap Caching – Cont’d

• Cells backed by ByteBuffers
• New ByteBufferedCell (HBASE-13387): server side extension for Cell
• Getters of the position - getXXXPosition()
• Comparators use getXXXByteBuffer() API for offheap ByteBuffer backed cells (L2 block cache)
Offheap Caching – Cont’d

- BucketCache buckets are buffer pools
- Eviction of blocks frees the BB and clears its content
- HBASE-12295 Reference count based block eviction
- Increment count when reader gets block from BucketCache
- Return block once the block usage is over and all cells from that block are written to IPC layer
- RpcCallBack mechanism is introduced
Offheap Caching – Cont’d

- **Performance Evaluation Tool (PE)**
- Table with one CF and one cell per row. 40 GB total data. Each row with 1K value size
- Single node cluster
  - CPU: Intel(R) Xeon(R) CPU with 8 cores.
  - RAM: 150 GB
  - JDK: 1.8
  - HBase configuration
    - HBASE_HEAPSIZE = 9 GB
    - HBASE_OFFHEAPSIZE = 50 GB
    - hbase.bucketcache.size = 40GB

Multi-get with 100 rows
Every thread doing 100000 operations
Avg completion run time of each thread (In secs)

- 45 – 52% latency reduction with different number of threads
Offheap Caching – Cont’d

**PE Random Range Scan 10K range with filter**
Each thread doing range scan for 1000 times
Avg completion run time of each thread (in secs)

![Bar chart showing latency reduction](chart1)

- **45 – 52% latency reduction**

**PE Random Range Scan 10K range**
Each thread doing range scan for 1000 times
Returning 1% of rows back to client
Avg completion run time of each thread (in secs)

![Bar chart showing latency gain](chart2)

- **17 – 21% latency gain**

![Bar chart showing latency gain](chart3)

- **52% latency gain**

© Hortonworks Inc. 2014
YCSB
- Table with one CF and 10 columns per row. Each row with 1K value. 150 GB total data
- Four node cluster. 1 HM and 3 RSs
  - CPU: Intel(R) Xeon(R) CPU with 56 cores.
  - RAM: 400 GB
  - JDK: 1.8
  - HBase configuration
    - HBASE_HEAPSIZE = 32 GB
    - HBASE_OFFHEAPSIZE = 250 GB
    - hbase.bucketcache.size = 220 GB

Random range scan with each thread doing 1000000 operations

Multi get with 100 rows
Every thread doing 5000000 operations

![Scans Graph]
- 11 – 19 % more throughput

![MultiGets Graph]
- 11 – 34 % more throughput
Reduction in GC pause

MultiGets – without HBASE-11425 (25 threads)

Multigets – with HBASE-11425(25 threads)
Date Tiered Compaction

- Write access pattern is mainly sequential writes by time of data arrival
- Read access pattern is mainly time-range scans
- Better granularity beyond major compaction intervals for efficient timespan scans
- Reduced IO cost of compactions
- Efficient data retention
- Better performance
- Consistent within major compaction intervals
Date Tiered Compaction Cont’d

- New time windows appear
- Old ones get merged into exponentially larger windows

Figure 2: base window = 1 hour, windows per tier = 4
Major config parameters

• Base window: smallest time window for first tier
• Windows per tier: scale factor of window sizes from one tier to the next
• Max storefile age: how old it has to be before compaction stops – biggest tier
• Incoming windows threshold: number of files in incoming window before we compact to first tier
Date Tiered Compaction Cont’d

• Files are ordered by sequence Id
• Max timestamp is used to determine order of files and compaction window as secondary order
• Suitable for time series data loaded periodically with minimum time range overlap … and more cases
Date Tiered Compaction Cont’d

• Undesirable scenario: file on the lower tier has long tails
• HBASE 15400, major and minor compactions with splitting by window boundaries will help
• This compaction policy is unsuitable for cases where future timestamp is used in writes
### IO Savings

- **Savings against ExploringCompactionPolicy**

<table>
<thead>
<tr>
<th></th>
<th>Reduction% on HDFS IO</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-day lookback job</td>
<td>79%</td>
<td></td>
</tr>
<tr>
<td>7-day lookback job</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>Minor Compaction</td>
<td>85%</td>
<td>This number is still provisional as data has not aged into some of the largest tiers and we expect the eventual reduction in minor compaction will be less significant</td>
</tr>
</tbody>
</table>
Perf Validation (days after turned on)

- **HDFS Read Bytes**

- **Mapper Run Time Minutes**
Q/A
Thank you.