Bulk Loading Your Big Data into Apache HBase, a Full Walkthrough

Jean-Daniel Cryans
Strata + Hadoop World NYC 2014
About me

• Software Engineer at Cloudera, Storage team.
• Apache HBase committer since 2008, PMC member.
Agenda

1. HBase’s write path
2. Bulk loading concepts
3. ETL example
4. Issues and gotchas
Getting your BIG data in HBase

• Thrift/REST
• Java API
• MapReduce
Getting your BIG data in HBase

- Thrift/REST
  - Low throughput due to indirection.
  - Need a way to have many clients.
Getting your BIG data in HBase

• Java API
  • Indirection problem is solved.
  • Still need a way to have many clients.
Getting your BIG data in HBase

- MapReduce
- No indirection.
- No distribution problem, but...
Getting your BIG data in HBase

- MapReduce
- No indirection.
- No distribution problem, but...
hbase-user@hadoop.apache.org

My region servers are always dying???

Hey list,

I'm using HBase 0.94 and trying to import a few TBs of data. Originally it was slow when sending the data from Python, I estimated it would take over a month in the best case, but now I wrote this MR job that's super fast for a few hours but then everything crashes!

When my region servers die I see a lot of HDFS stack traces and eventually there's a spooky YouAreDeadException.

Can someone help please?

Thx,

J-D
14.3 Java

14.3.1 The Garbage Collector and Apache HBase

14.4 HBase Configurations

14.4.1 Managing Compactions
14.4.2 hbase.regionserver.handler.count
14.4.3 hfile.block.cache.size
14.4.4 Prefetch Option for Blockcache
14.4.5 hbase.regionserver.global.memstore.size
14.4.6 hbase.regionserver.global.memstore.size.lower.limit
14.4.7 hbase.hstore.blockingStoreFiles
14.4.8 hbase.hregion.memstore.block.multiplier
14.4.9 hbase.regionserver.checksum.verify
14.4.10 Tuning callqueue Options

14.5 ZooKeeper

14.6 Schema Design

14.6.1 Number of Column Families
14.6.2 Key and Attribute Lengths
Log-structured merge-trees

C₁ tree

Disk

C₀ tree

Memory

A quick intro to LSM trees

- Data is written in memory to C0.
- C0 flushes upon reaching a certain threshold.
- On-disk components C1–Ck are merged in the background.

Figure 3.1. An LSM-tree of K+1 components

LSM trees in HBase
LSM trees in HBase
LSM trees in HBase

1. Insert
2. Insert
3. Memstore
4. Flush
5. HFile
LSM trees in HBase

Insert → Memstore
Insert → Memstore
Insert → Memstore
Flush → HFile
HFile
HFile
LSM trees in HBase

Insert → Memstore
Insert → Memstore
Insert → Memstore

Flush → HFile
Flush → HFile
LSM trees in HBase

1. Insert → Memstore
2. Insert → Memstore → Flush → HFile
3. Insert → Memstore → Flush → HFile
4. Insert → Memstore → HFile
5. Insert → Memstore → HFile
LSM trees in HBase

1. Insert → Memstore
2. Insert → Memstore → Flush → HFile
3. Insert → Memstore → Flush → HFile
4. Insert → Memstore → Flush → HFile → HFile

HFile
LSM trees in HBase

- Insert
  - Memstore
  - Flush
  - HFile

- Insert
  - Memstore
  - Flush
  - HFile
  - HFile

- Insert
  - Memstore
  - Flush
  - HFile
  - HFile

- Insert
  - Memstore
  - Flush
  - HFile
  - HFile
  - HFile

Compaction!
LSM trees in HBase
LSM trees in HBase

Inserts

Inserts

Memstore
LSM trees in HBase
LSM trees in HBase
LSM trees in HBase

Inserts → Memstore → Flush! → HFile
Inserts → Memstore → Flush! → HFile
Inserts → Memstore → Flush! → HFile
LSM trees in HBase
LSM trees in HBase

Inserts → Memstore → HFile
Inserts → Memstore → HFile → HFile
Inserts → Memstore → HFile → HFile → HFile
Inserts → Memstore → HFile → HFile → HFile → HFile
LSM trees in HBase
LSM trees in HBase

- How many times will data be rewritten?
- What kind of tuning could make this better?
- What about splitting those regions?
LSM trees in HBase

Or is there a way to just get the final result directly in HBase?
Agenda

1. HBase’s write path
2. Bulk loading concepts
3. ETL example
4. Issues and gotchas
Bulk loading overview

• Goal: generate data files in HBase’s own format, respecting the region boundaries, and give them to the region servers.
• Use cases:
Bulk loading overview

• Goal: generate data files in HBase’s own format, respecting the region boundaries, and give them to the region servers.

• Use cases:

  - Initial Data Import Example:
Bulk loading overview

• Goal: generate data files in HBase’s own format, respecting the region boundaries, and give them to the region servers.

• Use cases:

  Initial Data Import
  Example:

  MySQL

  Regular Imports
  Example:
Bulk loading data flow

Extract

Data source

Transform

MapReduce

HDFS

Load

Apache HBase
Bulk loading data flow

1. Extract
   - Data source

2. Transform
   - MapReduce
   - HDFS

3. Load
   - Apache HBase
Bulk loading data flow

Extract

Data source

Transform

MapReduce

HDFS

Load

Apache HBase
Transforming data into HFiles

```java
HTable table = new HTable(conf, tableName);
job.setReducerClass(PutSortReducer.class);
Path outputDir = new Path(hfileOutPath);
FileOutputFormat.setOutputPath(job, outputDir);
job.setMapOutputKeyClass(ImmutableBytesWritable.class);
job.setMapOutputValueClass(Put.class);
HFileOutputFormat.configureIncrementalLoad(job, table);
```
Transforming data into HFiles

HFileOutputFormat.configureIncrementalLoad(job, table);
Transforming data into HFiles

HFileOutputFormat.configureIncrementalLoad(job, table);

Mapper 1
Mapper 2
Mapper 3
Mapper 4
Mapper n
Transforming data into HFiles

HFileOutputFormat.configureIncrementalLoad(job, table);

Mapper 1

key [aaa,bbb]

key [bbb,ccc]

Mapper 2

Mapper 3

Mapper 4

Mapper n

TotalOrder Partitioner
Transforming data into HFiles

HFileOutputFormat.configureIncrementalLoad(job, table);

Each reducer outputs one file per region.
Bulk loading data flow

Choose your source of data, extract data, transform it using MapReduce, and then load it into HBase.
Loading HFiles

$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles <files_location> <table_name>

/ job_output/
  hfile1
  hfile2
  hfileN
Loading HFiles

$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>

/ job_output/
hfile1
hfile2
hfileN

read the HFiles’ metadata
Loading HFiles

$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>

read the HFiles' metadata

/ job_output/
hfile1
hfile2
hfileN

LoadIncrementalHFiles

split files or group them if needed
Loading HFiles

```bash
$ hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles
<files_location> <table_name>
```

- Read the HFiles' metadata
- Move hfile x to region y
- Split files or group them if needed
Agenda

1. HBase’s write path
2. Bulk loading concepts
3. ETL example
4. Issues and gotchas
MySQL Import

- Extract
  - CSV dump into file.
- Transform
  - Map columns, create HFiles.
- Load
  - Use LoadIncrementalHFiles.
SELECT * INTO OUTFILE 'dump.csv'
   FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '''
   LINES TERMINATED BY '\n'
FROM table

hdfs dfs -put dump.csv
Transform

Map dump.csv

Reduce to output/

hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.seperator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2 table-name dump.csv
Transform

Map dump.csv

```
hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.separator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2 table-name dump.csv
```

Reduce to output/
Transform

Map dump.csv

```
hadoop jar /usr/lib/hbase/hbase-0.98.6-cdh5.2.0-security.jar importtsv
-Dimporttsv.separator=,
-Dimporttsv.bulk.output=output
-Dimporttsv.columns=HBASE_ROW_KEY,f:col1,f:col2
```

Reduce to output/

dump.csv

```
table-name dump.csv
```
Load

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles output table-name
```

List the files under output/

Tell each RS to move them.
Agenda

1. HBase’s write path
2. Bulk loading concepts
3. ETL example
4. Issues and gotchas
Planning the bulk load; gotchas

• Initial import
  • Tables must still be created, pre-split.

    create 'table-name', {NAME => 'f'}, {SPLITS => ['a', 'b', 'c', 'd']}

• Plan for the files to fit in the regions else it will split.

    alter 'table-name', {MAX_FILESIZE => 10737418240}
Planning the bulk load; gotchas

• Regular import
  • Loading data on HDFS still not free, IO-wise.
    • Especially the Transform phase.

• Data won’t be in the block cache once Loaded.

• Block locality isn’t guaranteed.
Gotchas: Security

- Problem:
  - The user “hbase” must move files it doesn’t have access to.
Gotchas: Security

• Problem:
  • The user “hbase” must move files it doesn’t have access to.

• SecureBulkLoadEndpoint
  • Must be installed as part of enabling security.
  • A secret staging directory with 777 perms is used.
  • LoadIncrementalHFiles moves files there and then the RS moves it into its regions’ directories.
Gotchas: HBASE-8521 (fixed 0.94.13+)
Gotchas: HBASE-8521 (fixed 0.94.13+)

Sequence IDs

Bulk loaded

Flushed

3 HFiles fit the compaction selection criteria...
Gotchas: HBASE-8521 (fixed 0.94.13+)

Sequence IDs

Bulk loaded | Flushed
0 0 0 | 1 4 6 9

3 HFiles fit the compaction selection criteria...
0 0 0 | 1 4 6 9

Major compaction
0 0 0 | 1 4 6 9

It means we also select all the files that come after!
Gotchas: HBASE-8521 (fixed 0.94.13+)

Solution: assign sequence IDs to the bulk loaded files. Optional in 0.94, on by default in 0.96+.
Gotchas: HBASE-8283 (fixed 0.94.9+)

Sequence IDs

Bulk loaded

Flushed

0 0 0 1 4 6 9
Gotchas: HBASE-8283 (fixed 0.94.9+)

New selection algorithm considers multiple alternatives and doesn’t work in only one direction
Gotchas: HBASE-8283 (fixed 0.94.9+)

New selection algorithm considers multiple alternatives and doesn’t work in only one direction.

The selection that lowers the amount of seeks while compacting the least amount of data is chosen.
HBASE-10958 AKA Blindspot

Current MemStore

31402, 31403.................

Flushed

HFile
seqID
10203
HFile
seqID
31401
The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's HFile seqID.

- HFile seqID 10203
- HFile seqID 31401

Flushed

Current MemStore

31402, 31403..............

HFile seqID 31634
The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore's HFile. When loading edits from a failed RS's log, replay only the edits coming after the HFile's highest sequence ID.
The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore’s HFile seqID. When loading edits from a failed RS’s log, replay only the edits coming after the HFile’s highest sequence ID.
The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore’s

When loading edits from a failed RS’s log, replay only the edits coming after the HFile’s highest sequence ID.

Thankfully, we can recognize when a file is a bulk loaded one...
The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore’s `HFile seqID 10203` and `seqID 31401`.

Flushed

The bulk loaded file is assigned a sequence ID that ends up somewhere in the MemStore’s `HFile seqID 31634`.

When loading edits from a failed RS’s log, replay only the edits coming after the HFile’s highest sequence ID.

Thankfully, we can recognize when a file is a bulk loaded one.
HBASE-10958 AKA Blindspot

- HFile seqID 10203
- HFile seqID 31401
- Current MemStore: 31402, 31403
HBASE-10958 AKA Blindspot

Current MemStore

Flushed

HFile seqID 10203

HFile seqID 31401

HFile seqID 31402, 31403

HFile seqID 31634

HFile seqID 31764

HFile seqID 32562
HBASE-10958 AKA Blindspot

Current MemStore

HFile seqID 10203
HFile seqID 31401

Flushed

31402, 31403.................
HBASE-10958 AKA Blindspot

The bulk loaded status is lost through compaction, the resulting HFile looks like any other!
The bulk loaded status is lost through compaction, the resulting HFile looks like any other!

The MemStore’s data, starting from the beginning and up to 32562, will be lost even if it was logged.
The bulk loaded status is lost through compaction, the resulting HFile looks like any other!
The MemStore’s data, starting from the beginning and up to 32562, will be lost even if it was logged.

The proposed solution is to force flush when bulk loading with sequence IDs, since the way it currently works goes against log replay’s assumptions.
In conclusion

• Loading via the “normal” APIs can be slow and/or disrupt the cluster.
In conclusion

• Loading via the “normal” APIs can be slow and/or disrupt the cluster.
• Bulk loading can create files that HBase can directly use.
In conclusion

• Loading via the “normal” APIs can be slow and/or disrupt the cluster.
• Bulk loading can create files that HBase can directly use.
• Useful for your original data import or incremental ones.
In conclusion

• Loading via the “normal” APIs can be slow and/or disrupt the cluster.
• Bulk loading can create files that HBase can directly use.
• Useful for your original data import or incremental ones.
• Recommended to use HBase versions released during the past year.