Wide column stores

Lenart Treven

January 9, 2019

Since a lot of what is in those two mandatory readings [1] and [2] was either covered in classes or is not (in my opinion) suitable for exam question I will just point out those ideas which were new in there or were not pointed out enough (in my opinion) during lectures.

Chapter 1. Introduction, [1]

• Problems with Relational Database Systems such as sharding, mulitple joins are expensive, limited number of columns

• Consistency models are discussed - how long does it take that user can observe changes which were made in database

• Column families have to be created in the begining, there should not be too many of them, columns can be added on the fly, usually a great number of them

• The name of the column family must be composed of printable characters, a notable difference from all other names or values.

• All rows are always sorted lexicographically by their row key - the design of row key is imoprtant for fast queries

• The user can specify how many versions of a value should be kept

• Initially there is only one region for a table, and as you start adding data to it, the system is monitoring it to ensure that you do not exceed a configured maximum size. If you exceed the limit, the region is split into two at the middle key creating two roughly equal halves. Splitting is also very fast—close to instantaneous
• There is also the option to run client-supplied code in the address space of the server. The server-side framework to support this is called coprocessors. The code has access to the server local data and can be used to implement lightweight batch jobs, or use expressions to analyze or summarize data based on a variety of operators.

• Lookups can be performed with a single disk seek

• Because store files are immutable, you cannot simply delete values by removing the key/value pair from them. Instead, a delete marker is written to indicate the fact that the given key has been deleted

• There are three major components to HBase: the client library, one master server, and many region servers. The region servers can be added or removed while the system is up and running to accommodate changing workloads. The master is responsible for assigning regions to region servers and uses Apache ZooKeeper, a reliable, highly available, persistent and distributed coordination service, to facilitate that task.

Chapter 3. Client API: The Basics, [1]

• HBase is wide column store

• Updates are atomic on a per-row basis

• Java API: Put, Get, Update Method

• We can batch different operations across multiple rows. Be aware that you should not mix a Delete and Put operation for the same row in one batch call. The operations will be applied in a different order that guarantees the best performance, but also causes unpredictable results. In some cases, you may see fluctuating results due to race conditions.

• Scan Method. The start row is always inclusive, while the end row is exclusive.

Chapter 8. Architecture, [1]

• Indexing: Log-Structured Merge-Trees, B+ Trees
• The client issues Put request. The first step is to write the data to the write-ahead log. Once the data is written to the WAL, it is placed in the MemStore. At the same time, it is checked to see if the MemStore is full and, if so, a flush to disk is requested. The data is written to a new HFile located in HDFS. Also the last written sequence number is saved so that the system knows what was persisted so far.

• HFile: default size is 64 KB, in practice, the majority of blocks will be slightly larger.

• Write ahead log: if writing the record to the WAL fails, the whole operation must be considered as a failure.

• Replication in HBase: Column family-level

Chapter 9. Advanced Usage, [1]

• Secondary indexes: Although HBase has no native support for secondary indexes, there are use cases that need them. The requirements are usually that you can look up a cell with not just the primary coordinates—the row key, column family name, and qualifier—but also an alternative coordinate. In addition, you can scan a range of rows from the main table, but ordered by the secondary index. There are some solutions to address this problem one is using coprocessors.

• Versioning: timeline - we have total order. Important that the clock on your servers is synchronized. You can also specify your own timestamp values—and therefore create your own versioning scheme—while over-riding the server-side timestamp generation based on the synchronized server time.

• Bloom filters

Chapter 20. HBase, [2]

• Use when you require real-time read/write random access to very large datasets

• HBase tables are like those in an RDBMS, only cells are versioned, rows are sorted, and columns can be added on the fly by the client as long as the column family they belong to preexists
• It is advised that all column family members have the same general access pattern and size characteristics

• Problem: Running out of file descriptors - Because we keep files open, on a loaded cluster it doesn’t take long before we run into system- and Hadoop-imposed limits. For instance, say we have a cluster that has three nodes, each running an instance of a datanode and a regionserver, and we’re running an upload into a table that is currently at 100 regions and 10 column families. Allow that each column family has on average two flush files. Doing the math, we can have $100 \times 10 \times 2$, or 2,000, files open at any one time. Add to this total other miscellaneous descriptors consumed by outstanding scanners and Java libraries. Each open file consumes at least one descriptor over on the remote datanode. The default limit on the number of file descriptors per process is 1,024. When we exceed the filesystem ulimit, we’ll see the complaint about “Too many open files” in logs, but often we’ll first see indeterminate behavior in HBase. The fix requires increasing the file descriptor ulimit count; 10,240 is a common setting.

• Another problem mentioned is running out of datanode threads - Similarly, the Hadoop datanode has an upper bound on the number of threads it can run at any one time. Hadoop 1 had a low default of 256 for this setting, which would cause HBase to behave erratically. Hadoop 2 increased the default to 4,096, so you are much less likely to see a problem for recent versions of HBase (which only run on Hadoop 2 and later).

References


Wide column stores

January 30, 2019

Some content was already covered in the lectures, I included ideas which were new. Java examples are meant to supplement what we have done in the exercise class.

Chapter 1. Introduction, [1]

- There can be stored multiple versions of each cell. User can specify how many versions should be stored.
- Rows are sorted lexicographically by row key. You may have to pad keys to get the sorting order you really want (without padding 10 is lexicographically smaller than 2). Row keys are unique and can be arbitrary bytes (not needed to consist of printable characters like in case of column families).
- Number of column families is limited to the low tens, number of columns is unlimited.
- NULL values are free, because you simply omit them when storing data. If you have a fixed schema, like in SQL, this is not the case.
- Region server can store multiple regions, but each region is stored on a single region server.
- Splitting of regions is very fast.
- Every HFile has a block index, so lookups can be performed with a single disk seek.
- Memstores are already sorted by keys. Therefore, they can be flushed to HFiles without additional sorting.
- When you delete a key-value, it is not removed, but a delete marker is written instead. Those key-values are then ignored when retrieving key-values.
- Compactions:
  - Minor compaction: reduce the number of HFiles by rewriting smaller files into fewer but larger ones.
  - Major compaction: rewrite all HFiles within a column family for a region into a single HFile.
- Row key lookup is of logarithmic order or even constant order if Bloom filters are used.

Chapter 3. Client API: The Basics, [1]

- When you insert a value you can specify timestamp yourself. If you do not do that, timestamp is generated by RegionServer when value is inserted.
- Put example in Java:

```java
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, "table");
Put put = new Put(Bytes.toBytes("row"));
put.add(Bytes.toBytes("columnFamily"),
        Bytes.toBytes("qualifier"),
        Bytes.toBytes("value"));
table.put(put);
```
Example of multiple puts in Java (if one of them fails, other still get executed):

```java
List<Put> puts = new ArrayList<Put>();
Put put1 = new Put(Bytes.toBytes("row"));
put1.add(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"), Bytes.toBytes("value"));
puts.add(put1);
Put put2 = new Put(Bytes.toBytes("otherRow"));
put2.add(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"), Bytes.toBytes("value"));
puts.add(put2);
table.put(puts);
```

Get example in Java:

```java
Get get = new Get(Bytes.toBytes("row"));
get.addColumn(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"));
Result result = table.get(get);
byte[] value = result.getValue(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"));
String stringValue = Bytes.toString(value);
```

Multiple gets are similar to multiple puts:

```java
List<Get> gets = new ArrayList<Get>();
...
Result[] results = table.get(gets);
```

Delete example in Java:

```java
Delete delete = new Delete(Bytes.toBytes("row"));
// All following methods can have specified timestamp parameter
// which is always last parameter
// I did not include it in methods

// In following examples:
// 1. means timestamp not specified
// 2. means timestamp specified

// 1. Delete last version of column
// 2. Delete specific version of columns
delete.deleteColumn(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"));

// 1. Delete all version of column
// 2. Delete versions of column older or same as timestamp
delete.deleteColumns(Bytes.toBytes("columnFamily"), Bytes.toBytes("qualifier"));

// 1. Delete whole family, all versions
// 2. Delete versions of whole family older or same as timestamp
delete.deleteFamily(Bytes.toBytes("columnFamily"));
table.delete(delete);
```
• Multiple deletes are similar to multiple puts.

```java
List<Delete> deletes = new ArrayList<Delete>();
...
table.delete(deletes);
```

• Aforementioned operations can be combined using batch (it is not advisable to use delete and put operations for same row in same batch operation):

```java
List<Row> batch = new ArrayList<Row>();
...
batch.add(put);
...
batch.add(get);
Object[] results = new Object[batch.size()];
table.batch(batch, results);
```

• Row locks are used by default (see slides). However, user can also use explicit lock like:

```java
RowLock lock = table.lockRow(Bytes.toBytes("row"));
...
table.unlockRow(lock);
```

• Scan example in Java (more features covered in exercise):

```java
// Note that start row is included and end row is not
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("columnFamily"),
Bytes.toBytes("qualifier"))
 .addColumn(Bytes.toBytes("columnFamily2"),
Bytes.toBytes("qualifier2"))
 .setStartRow(Bytes.toBytes("row-1"))
 .setStopRow(Bytes.toBytes("row-100"));
```

Chapter 8. Architecture, [1]

• Blocks in HFile can be compressed (for instance LZO compression algorithm).

• There is one logfile (Write-Ahead Log) per region server. It is used to replay actions when server fails or cluster starts. Since there is only one WAL per region server log needs to be splitted according to regions before it can be replayed. This process is called log splitting.

• Get operation is internally implemented the same as scan operation on a single row.

• Region lookups:

1. Client contacts ZooKeeper to obtain location of -ROOT- table region, which is stored at one of the region servers (rs1).
2. Client queries rs1 to obtain location of matching .META. table region, which is stored at one of the region servers (rs2).
3. Client queries rs2 to obtain location of table region than contains requested data, which is stored at one of the region servers (rs3).
4. Client queries rs3 to obtain relevant data.

Note that client caches locations of regions.

• It is possible to replicate HBase cluster for higher availability and recovery in case of disaster.
Chapter 9. Advanced Usage, [1]

- When there is a lot of data incoming at the same time, it is written to the same region, which can be problematic, because it has to be processed by a single server. This problem can be solved by prefixing row keys with different values, which ensures that data is stored across all region servers.
- Each value can be made more complex if Avro or Protocol Buffers are used.
- Secondary indices are not natively supported, however there are solutions that enable their use.
- Transactions are not usually used with HBase, however there are solutions that enable their use.
- Usually row Bloom filters are used, but row+column Bloom filters can also be used.

Chapter 20. HBase, [2]

- HBase is suitable for storing large and sparse tables on commodity hardware.
- Column family prefix must consist of printable characters, column family qualifier can be arbitrary.
- Regionservers manage region splits and inform HMaster about new regions.
- How to remove table from HBase with command line:
  ```
  disable '<table name>'
  drop '<table name>'
  ```

- How to create table with Java:
  ```java
  Configuration config = HBaseConfiguration.create();
  HBaseAdmin admin = new HBaseAdmin(config);
  TableName name = TableName.valueOf("tableName");
  HTableDescriptor td = new HTableDescriptor(name);
  HColumnDescriptor cd = new HColumnDescriptor("columnFamilyName");
  htd.addFamily(cd);
  admin.createTable(td);
  ```

- It is possible to do reverse scans (scanning in decreasing order). In Java it can be done like:
  ```java
  Scan scan = new Scan();
  scan.setReversed(true);
  ```

References
