Apache Accumulo User Manual
Version 1.3

July 3, 2012

Contents

Chapter 1

Introduction

Apache Accumulo is a highly scalable structured store based on Google's BigTable. Accumulo is written in Java and operates over the Hadoop Distributed File System (HDFS), which is part of the popular Apache Hadoop project. Accumulo supports efficient storage and retrieval of structured data, including queries for ranges, and provides support for using Accumulo tables as input and output for MapReduce jobs.

Accumulo features automatic load-balancing and partitioning, data compression and fine-grained security labels.

Chapter 2

Accumulo Design

2.1 Data Model

Accumulo provides a richer data model than simple key-value stores, but is not a fully relational database. Data is represented as key-value pairs, where the key and value are comprised of the following elements:

<table>
<tr><td colspan="4">Key</td><td rowspan="2">Value</td></tr>
<tr><td rowspan="2">Row ID</td><td colspan="3">Column</td><td rowspan="2">Timestamp</td></tr>
<tr><td>Family</td><td>Qualifier</td><td>Visibility</td></tr>
</table>

All elements of the Key and the Value are represented as byte arrays except for Timestamp, which is a Long. Accumulo sorts keys by element and lexicographically in ascending order. Timestamps are sorted in descending order so that later versions of the same Key appear first in a sequential scan. Tables consist of a set of sorted key-value pairs.

2.2 Architecture

Accumulo is a distributed data storage and retrieval system and as such consists of several architectural components, some of which run on many individual servers. Much of the work Accumulo does involves maintaining certain properties of the data, such as organization, availability, and integrity, across many commodity-class machines.

2.3 Components

An instance of Accumulo includes many TabletServers, write-ahead Logger servers, one Garbage Collector process, one Master server and many Clients.

2.3.1 Tablet Server

The TabletServer manages some subset of all the tablets (partitions of tables). This includes receiving writes from clients, persisting writes to a writeâĂŘahead log, sorting new keyâĂŘvalue pairs in memory, periodically flushing sorted keyâĂŘvalue pairs to new files in HDFS, and responding to reads from clients, forming a mergeâĂŘsorted view of all keys and values from all the files it has created and the sorted inâĂŘmemory store.

TabletServers also perform recovery of a tablet that was previously on a server that failed, reapplying any writes found in the write-ahead log to the tablet.

2.3.2 Loggers

The Loggers accept updates to Tablet servers and write them to local on-disk storage. Each tablet server will write their updates to multiple loggers to preserve data in case of hardware failure.

2.3.3 Garbage Collector

Accumulo processes will share files stored in HDFS. Periodically, the Garbage Collector will identify files that are no longer needed by any process, and delete them.

2.3.4 Master

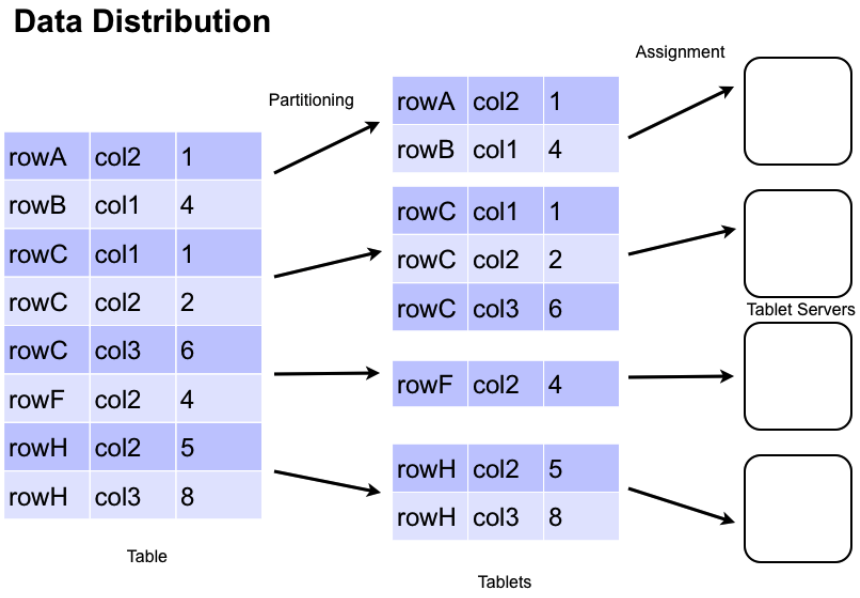
The Accumulo Master is responsible for detecting and responding to TabletServer failure. It tries to balance the load across TabletServer by assigning tablets carefully and instructing TabletServers to migrate tablets when necessary. The Master ensures all tablets are assigned to one TabletServer each, and handles table creation, alteration, and deletion requests from clients. The Master also coordinates startup, graceful shutdown and recovery of changes in write-ahead logs when Tablet servers fail.

2.3.5 Client

Accumulo includes a client library that is linked to every application. The client library contains logic for finding servers managing a particular tablet, and communicating with TabletServers to write and retrieve key-value pairs.

2.4 Data Management

Accumulo stores data in tables, which are partitioned into tablets. Tablets are partitioned on row boundaries so that all of the columns and values for a particular row are found together within the same tablet. The Master assigns Tablets to one TabletServer at a time. This enables row-level transactions to take place without using distributed locking or some other complicated synchronization mechanism. As clients insert and query data, and as machines are added and removed from the cluster, the Master migrates tablets to ensure they remain available and that the ingest and query load is balanced across the cluster.



2.5 Tablet Service

When a write arrives at a TabletServer it is written to a WriteâĂŘAhead Log and then inserted into a sorted data structure in memory called a MemTable. When the MemTable reaches a certain size the TabletServer writes out the sorted key-value pairs to a file in HDFS called Indexed Sequential Access Method (ISAM) file. This process is called a minor compaction. A new MemTable is then created and the fact of the compaction is recorded in the WriteâĂŘAhead Log.

When a request to read data arrives at a TabletServer, the TabletServer does a binary search across the MemTable as well as the in-memory indexes associated with each ISAM file to find the relevant values. If clients are performing a scan, several keyâĂŘvalue pairs are returned to the client in order from the MemTable and the set of ISAM files by performing a mergeâĂŘsort as they are read.

2.6 Compactions

In order to manage the number of files per tablet, periodically the TabletServer performs Major Compactions of files within a tablet, in which some set of ISAM files are combined into one file. The previous files will eventually be removed by the Garbage Collector. This also provides an opportunity to permanently remove deleted keyâĂŘvalue pairs by omitting keyâĂŘvalue pairs suppressed by a delete entry when the new file is created.

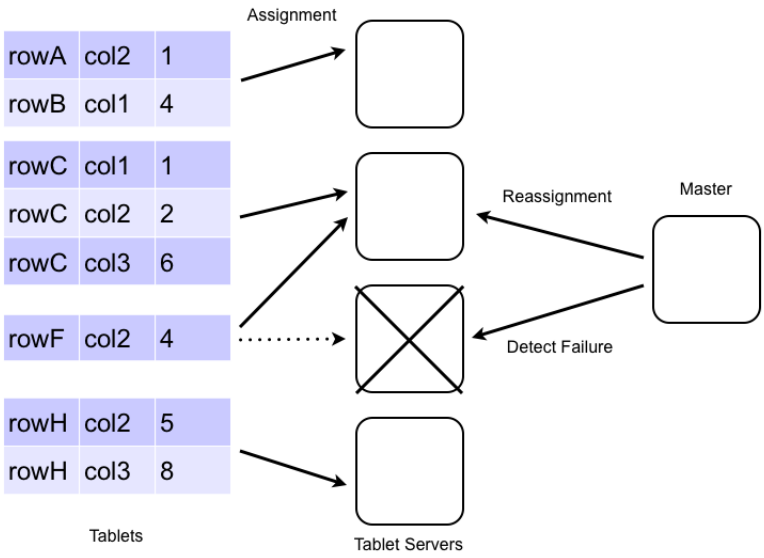
2.7 Fault-Tolerance

If a TabletServer fails, the Master detects it and automatically reassigns the tablets assigned from the failed server to other servers. Any key-value pairs that were in memory at the time the TabletServer are automatically reapplied from the Write-Ahead Log to prevent any loss of data.

The Master will coordinate the copying of write-ahead logs to HDFS so the logs are available to all tablet servers. To make recovery efficient, the updates within a log are grouped by tablet. The sorting process can be performed by Hadoops MapReduce or the Logger server. TabletServers can quickly apply the mutations from the sorted logs that are destined for the tablets they have now been assigned.

TabletServer failures are noted on the Master's monitor page, accessible via http://master-address:50095/monitor.

Automatic Failure Handling



Chapter 3

Accumulo Shell

Accumulo provides a simple shell that can be used to examine the contents and configuration settings of tables, apply individual mutations, and change configuration settings.

The shell can be started by the following command:

```
$ACCUMULO_HOME/bin/accumulo shell -u [username]
```

The shell will prompt for the corresponding password to the username specified and then display the following prompt:

```
Shell - Apache Accumulo Interactive Shell
-
- version 1.3
- instance name: myinstance
- instance id: 00000000-0000-0000-0000-000000000000
-
- type 'help' for a list of available commands
-
```

3.1 Basic Administration

The Accumulo shell can be used to create and delete tables, as well as to configure table and instance specific options.

```
root@myinstance> tables
!METADATA

root@myinstance> createtable mytable
```

```
root@myinstance mytable>

root@myinstance mytable> tables
!METADATA
mytable

root@myinstance mytable> createtable testtable

root@myinstance testtable>

root@myinstance junk> deletetable testtable

root@myinstance>
```

The Shell can also be used to insert updates and scan tables. This is useful for inspecting tables.

```
root@myinstance mytable> scan

root@myinstance mytable> insert row1 colf colq value1
insert successful

root@myinstance mytable> scan
row1 colf:colq [] value1
```

3.2 Table Maintenance

The **compact** command instructs Accumulo to schedule a compaction of the table during which files are consolidated and deleted entries are removed.

```
root@myinstance mytable> compact -t mytable
07 16:13:53,201 [shell.Shell] INFO : Compaction of table mytable
scheduled for 20100707161353EDT
```

The **flush** command instructs Accumulo to write all entries currently in memory for a given table to disk.

```
root@myinstance mytable> flush -t mytable
07 16:14:19,351 [shell.Shell] INFO : Flush of table mytable
initiated...
```

3.3 User Administration

The Shell can be used to add, remove, and grant privileges to users.

```
root@myinstance mytable> createuser bob
Enter new password for 'bob': *********
Please confirm new password for 'bob': *********

root@myinstance mytable> authenticate bob
Enter current password for 'bob': *********
Valid

root@myinstance mytable> grant System.CREATE_TABLE -s -u bob

root@myinstance mytable> user bob
Enter current password for 'bob': *********

bob@myinstance mytable> userpermissions
System permissions: System.CREATE_TABLE
Table permissions (!METADATA): Table.READ
Table permissions (mytable): NONE

bob@myinstance mytable> createtable bobstable
bob@myinstance bobstable>

bob@myinstance bobstable> user root
Enter current password for 'root': *********

root@myinstance bobstable> revoke System.CREATE_TABLE -s -u bob
```

Chapter 4

Writing Accumulo Clients

All clients must first identify the Accumulo instance to which they will be communicating. Code to do this is as follows:

```
String instanceName = "myinstance";
String zooServers = "zooserver-one,zooserver-two"
Instance inst = new ZooKeeperInstance(instanceName, zooServers);

Connector conn = new Connector(inst, "user","passwd".getBytes());
```

4.1 Writing Data

Data are written to Accumulo by creating Mutation objects that represent all the changes to the columns of a single row. The changes are made atomically in the TabletServer. Clients then add Mutations to a BatchWriter which submits them to the appropriate TabletServers.

Mutations can be created thus:

```
Text rowID = new Text("row1");
Text colFam = new Text("myColFam");
Text colQual = new Text("myColQual");
ColumnVisibility colVis = new ColumnVisibility("public");
long timestamp = System.currentTimeMillis();

Value value = new Value("myValue".getBytes());

Mutation mutation = new Mutation(rowID);
mutation.put(colFam, colQual, colVis, timestamp, value);
```

4.1.1 BatchWriter

The BatchWriter is highly optimized to send Mutations to multiple TabletServers and automatically batches Mutations destined for the same TabletServer to amortize network overhead. Care must be taken to avoid changing the contents of any Object passed to the BatchWriter since it keeps objects in memory while batching.

Mutations are added to a BatchWriter thus:

```
long memBuf = 1000000L; // bytes to store before sending a batch
long timeout = 1000L; // milliseconds to wait before sending
int numThreads = 10;

BatchWriter writer =
    conn.createBatchWriter("table", memBuf, timeout, numThreads)

writer.add(mutation);

writer.close();
```

An example of using the batch writer can be found at
accumulo/docs/examples/README.batch

4.2 Reading Data

Accumulo is optimized to quickly retrieve the value associated with a given key, and to efficiently return ranges of consecutive keys and their associated values.

4.2.1 Scanner

To retrieve data, Clients use a Scanner, which provides acts like an Iterator over keys and values. Scanners can be configured to start and stop at particular keys, and to return a subset of the columns available.

```
// specify which visibilities we are allowed to see
Authorizations auths = new Authorizations("public");

Scanner scan =
    conn.createScanner("table", auths);

scan.setRange(new Range("harry","john"));
scan.fetchFamily("attributes");
```

```
for(Entry<Key,Value> entry : scan) {
    String row = e.getKey().getRow();
    Value value = e.getValue();
}
```

4.2.2 BatchScanner

For some types of access, it is more efficient to retrieve several ranges simultaneously. This arises when accessing a set of rows that are not consecutive whose IDs have been retrieved from a secondary index, for example.

The BatchScanner is configured similarly to the Scanner; it can be configured to retrieve a subset of the columns available, but rather than passing a single Range, BatchScanners accept a set of Ranges. It is important to note that the keys returned by a BatchScanner are not in sorted order since the keys streamed are from multiple TabletServers in parallel.

```
ArrayList<Range> ranges = new ArrayList<Range>();
// populate list of ranges ...

BatchScanner bscan =
    conn.createBatchScanner("table", auths, 10);

bscan.setRanges(ranges);
bscan.fetchFamily("attributes");

for(Entry<Key,Value> entry : scan)
    System.out.println(e.getValue());
```

An example of the BatchScanner can be found at
accumulo/docs/examples/README.batch

Chapter 5

Table Configuration

Accumulo tables have a few options that can be configured to alter the default behavior of Accumulo as well as improve performance based on the data stored. These include locality groups, constraints, and iterators.

5.1 Locality Groups

Accumulo supports storing of sets of column families separately on disk to allow clients to scan over columns that are frequently used together efficient and to avoid scanning over column families that are not requested. After a locality group is set Scanner and BatchScanner operations will automatically take advantage of them whenever the fetchColumnFamilies() method is used.

By default tables place all column families into the same “default" locality group. Additional locality groups can be configured anytime via the shell or programmatically as follows:

5.1.1 Managing Locality Groups via the Shell

```
usage: setgroups <group>=<col fam>{,<col fam>}{ <group>=<col fam>{,<col
fam>}} [-?] -t <table>

user@myinstance mytable> setgroups -t mytable group_one=colf1,colf2

user@myinstance mytable> getgroups -t mytable
group_one=colf1,colf2
```

5.1.2 Managing Locality Groups via the Client API

```
Connector conn;

HashMap<String,Set<Text>> localityGroups =
    new HashMap<String, Set<Text>>();

HashSet<Text> metadataColumns = new HashSet<Text>();
metadataColumns.add(new Text("domain"));
metadataColumns.add(new Text("link"));

HashSet<Text> contentColumns = new HashSet<Text>();
contentColumns.add(new Text("body"));
contentColumns.add(new Text("images"));

localityGroups.put("metadata", metadataColumns);
localityGroups.put("content", contentColumns);

conn.tableOperations().setLocalityGroups("mytable", localityGroups);

// existing locality groups can be obtained as follows
Map<String, Set<Text>> groups =
    conn.tableOperations().getLocalityGroups("mytable");
```

The assignment of Column Families to Locality Groups can be changed anytime. The physical movement of column families into their new locality groups takes place via the periodic Major Compaction process that takes place continuously in the background. Major Compaction can also be scheduled to take place immediately through the shell:

```
user@myinstance mytable> compact -t mytable
```

5.2 Constraints

Accumulo supports constraints applied on mutations at insert time. This can be used to disallow certain inserts according to a user defined policy. Any mutation that fails to meet the requirements of the constraint is rejected and sent back to the client.

Constraints can be enabled by setting a table property as follows:

```
user@myinstance mytable> config -t mytable -s table.constraint.1=com.test.ExampleConstraint
user@myinstance mytable> config -t mytable -s table.constraint.2=com.test.AnotherConstraint
user@myinstance mytable> config -t mytable -f constraint
---------+--------------------------------+----------------------------
```

```
SCOPE    | NAME                             | VALUE
---------+----------------------------------+-----------------------------
table    | table.constraint.1.............. | com.test.ExampleConstraint
table    | table.constraint.2.............. | com.test.AnotherConstraint
---------+----------------------------------+-----------------------------
```

Currently there are no general-purpose constraints provided with the Accumulo distribution. New constraints can be created by writing a Java class that implements the org.apache.accumulo.core.constraints.Co interface.

To deploy a new constraint, create a jar file containing the class implementing the new constraint and place it in the lib directory of the Accumulo installation. New constraint jars can be added to Accumulo and enabled without restarting but any change to an existing constraint class requires Accumulo to be restarted.

An example of constraints can be found in `accumulo/docs/examples/README.constraints` with corresponding code under `accumulo/src/examples/main/java/accumulo/examples/constraints` .

5.3 Bloom Filters

As mutations are applied to an Accumulo table, several files are created per tablet. If bloom filters are enabled, Accumulo will create and load a small data structure into memory to determine whether a file contains a given key before opening the file. This can speed up lookups considerably.

To enable bloom filters, enter the following command in the Shell:

```
user@myinstance> config -t mytable -s table.bloom.enabled=true
```

An extensive example of using Bloom Filters can be found at `accumulo/docs/examples/README.bloom` .

5.4 Iterators

Iterators provide a modular mechanism for adding functionality to be executed by TabletServers when scanning or compacting data. This allows users to efficiently summarize, filter, and aggregate data. In fact, the built-in features of cell-level security and age-off are implemented using Iterators.

5.4.1 Setting Iterators via the Shell

```
usage: setiter [-?] -agg | -class <name> | -filter | -nolabel |
-regex | -vers [-majc] [-minc] [-n <itername>] -p <pri> [-scan]
[-t <table>]

user@myinstance mytable> setiter -t mytable -scan -p 10 -n myiter
```

5.4.2 Setting Iterators Programmatically

```
scanner.setScanIterators(
    15, // priority
    "com.company.MyIterator", // class name
    "myiter"); // name this iterator
```

Some iterators take additional parameters from client code, as in the following example:

```
bscan.setIteratorOption(
    "myiter", // iterator reference
    "myoptionname",
    "myoptionvalue");
```

Tables support separate Iterator settings to be applied at scan time, upon minor compaction and upon major compaction. For most uses, tables will have identical iterator settings for all three to avoid inconsistent results.

5.4.3 Versioning Iterators and Timestamps

Accumulo provides the capability to manage versioned data through the use of timestamps within the Key. If a timestamp is not specified in the key created by the client then the system will set the timestamp to the current time. Two keys with identical rowIDs and columns but different timestamps are considered two versions of the same key. If two inserts are made into accumulo with the same rowID, column, and timestamp, then the behavior is non-deterministic.

Timestamps are sorted in descending order, so the most recent data comes first. Accumulo can be configured to return the top k versions, or versions later than a given date. The default is to return the one most recent version.

The version policy can be changed by changing the VersioningIterator options for a table as follows:

```
user@myinstance mytable> config -t mytable -s
table.iterator.scan.vers.opt.maxVersions=3
```

```
user@myinstance mytable> config -t mytable -s
table.iterator.minc.vers.opt.maxVersions=3

user@myinstance mytable> config -t mytable -s
table.iterator.majc.vers.opt.maxVersions=3
```

Logical Time

Accumulo 1.2 introduces the concept of logical time. This ensures that timestamps set by accumulo always move forward. This helps avoid problems caused by TabletServers that have different time settings. The per tablet counter gives unique one up time stamps on a per mutation basis. When using time in milliseconds, if two things arrive within the same millisecond then both receive the same timestamp.

A table can be configured to use logical timestamps at creation time as follows:

```
user@myinstance> createtable -tl logical
```

Deletes

Deletes are special keys in accumulo that get sorted along will all the other data. When a delete key is inserted, accumulo will not show anything that has a timestamp less than or equal to the delete key. During major compaction, any keys older than a delete key are omitted from the new file created, and the omitted keys are removed from disk as part of the regular garbage collection process.

5.4.4 Filtering Iterators

When scanning over a set of key-value pairs it is possible to apply an arbitrary filtering policy through the use of a FilteringIterator. These types of iterators return only key-value pairs that satisfy the filter logic. Accumulo has two built-in filtering iterators that can be configured on any table: AgeOff and RegEx. More can be added by writing a Java class that implements the org.apache.accumulo.core.iterators.filter.Filter interface.

To configure the AgeOff filter to remove data older than a certain date or a fixed amount of time from the present. The following example sets a table to delete everything inserted over 30 seconds ago:

```
user@myinstance> createtable filtertest
user@myinstance filtertest> setiter -t filtertest -scan -minc -majc -p
```

```
10 -n myfilter -filter

FilteringIterator uses Filters to accept or reject key/value pairs
----------> entering options: <filterPriorityNumber>
<ageoff|regex|filterClass>

----------> set org.apache.accumulo.core.iterators.FilteringIterator option
(<name> <value>, hit enter to skip): 0 ageoff

----------> set org.apache.accumulo.core.iterators.FilteringIterator option
(<name> <value>, hit enter to skip):
AgeOffFilter removes entries with timestamps more than <ttl>
milliseconds old

----------> set org.apache.accumulo.core.iterators.filter.AgeOffFilter parameter
currentTime, if set, use the given value as the absolute time in
milliseconds as the current time of day:

----------> set org.apache.accumulo.core.iterators.filter.AgeOffFilter parameter
ttl, time to live (milliseconds): 30000

user@myinstance filtertest>
user@myinstance filtertest> scan
user@myinstance filtertest> insert foo a b c
insert successful
user@myinstance filtertest> scan
foo a:b [] c

... wait 30 seconds ...

user@myinstance filtertest> scan
user@myinstance filtertest>
```

To see the iterator settings for a table, use:

```
user@example filtertest> config -t filtertest -f iterator
---------+---------------------------------------------+------------------
SCOPE    | NAME                                        | VALUE
---------+---------------------------------------------+------------------
table    | table.iterator.majc.myfilter ........... |
10,org.apache.accumulo.core.iterators.FilteringIterator
table    | table.iterator.majc.myfilter.opt.0 ..... |
org.apache.accumulo.core.iterators.filter.AgeOffFilter
table    | table.iterator.majc.myfilter.opt.0.ttl . | 30000
table    | table.iterator.minc.myfilter ........... |
```

```
10,org.apache.accumulo.core.iterators.FilteringIterator
table    | table.iterator.minc.myfilter.opt.0 ..... |
org.apache.accumulo.core.iterators.filter.AgeOffFilter
table    | table.iterator.minc.myfilter.opt.0.ttl . | 30000
table    | table.iterator.scan.myfilter ........... |
10,org.apache.accumulo.core.iterators.FilteringIterator
table    | table.iterator.scan.myfilter.opt.0 ..... |
org.apache.accumulo.core.iterators.filter.AgeOffFilter
table    | table.iterator.scan.myfilter.opt.0.ttl . | 30000
---------+------------------------------------------+------------------
```

5.5 Aggregating Iterators

Accumulo allows aggregating iterators to be configured on tables and column families. When an aggregating iterator is set, the iterator is applied across the values associated with any keys that share rowID, column family, and column qualifier. This is similar to the reduce step in MapReduce, which applied some function to all the values associated with a particular key.

For example, if an aggregating iterator were configured on a table and the following mutations were inserted:

```
Row     Family Qualifier Timestamp  Value
rowID1  colfA  colqA     20100101   1
rowID1  colfA  colqA     20100102   1
```

The table would reflect only one aggregate value:

```
rowID1  colfA  colqA     -          2
```

Aggregating iterators can be enabled for a table as follows:

```
user@myinstance> createtable perDayCounts -a
day=org.apache.accumulo.core.iterators.aggregation.StringSummation

user@myinstance perDayCounts> insert row1 day 20080101 1
user@myinstance perDayCounts> insert row1 day 20080101 1
user@myinstance perDayCounts> insert row1 day 20080103 1
user@myinstance perDayCounts> insert row2 day 20080101 1
user@myinstance perDayCounts> insert row2 day 20080101 1

user@myinstance perDayCounts> scan
row1 day:20080101 [] 2
row1 day:20080103 [] 1
row2 day:20080101 [] 2
```

Accumulo includes the following aggregators:

- **LongSummation**: expects values of type long and adds them.
- **StringSummation**: expects numbers represented as strings and adds them.
- **StringMax**: expects numbers as strings and retains the maximum number inserted.
- **StringMin**: expects numbers as strings and retains the minimum number inserted.

Additional Aggregators can be added by creating a Java class that implements **org.apache.accumulo.core.iterators.aggregation.Aggregator** and adding a jar containing that class to Accumulo's lib directory.

An example of an aggregator can be found under accumulo/src/examples/main/java/org/apache/accumulo/examples/aggregation/SortedSetAggregator.java

5.6 Block Cache

In order to increase throughput of commonly accessed entries, Accumulo employs a block cache. This block cache buffers data in memory so that it doesn't have to be read off of disk. The RFile format that Accumulo prefers is a mix of index blocks and data blocks, where the index blocks are used to find the appropriate data blocks. Typical queries to Accumulo result in a binary search over several index blocks followed by a linear scan of one or more data blocks.

The block cache can be configured on a per-table basis, and all tablets hosted on a tablet server share a single resource pool. To configure the size of the tablet server's block cache, set the following properties:

`tserver.cache.data.size: Specifies the size of the cache for file data blocks.`
`tserver.cache.index.size: Specifies the size of the cache for file indices.`

To enable the block cache for your table, set the following properties:

`table.cache.block.enable: Determines whether file (data) block cache is enabled.`
`table.cache.index.enable: Determines whether index cache is enabled.`

The block cache can have a significant effect on alleviating hot spots, as well as reducing query latency. It is enabled by default for the !METADATA table.

Chapter 6

Table Design

6.1 Basic Table

Since Accumulo tables are sorted by row ID, each table can be thought of as being indexed by the row ID. Lookups performed row ID can be executed quickly, by doing a binary search, first across the tablets, and then within a tablet. Clients should choose a row ID carefully in order to support their desired application. A simple rule is to select a unique identifier as the row ID for each entity to be stored and assign all the other attributes to be tracked to be columns under this row ID. For example, if we have the following data in a comma-separated file:

```
userid,age,address,account-balance
```

We might choose to store this data using the userid as the rowID and the rest of the data in column families:

```
Mutation m = new Mutation(new Text(userid));
m.put(new Text("age"), age);
m.put(new Text("address"), address);
m.put(new Text("balance"), account_balance);

writer.add(m);
```

We could then retrieve any of the columns for a specific userid by specifying the userid as the range of a scanner and fetching specific columns:

```
Range r = new Range(userid, userid); // single row
Scanner s = conn.createScanner("userdata", auths);
s.setRange(r);
s.fetchColumnFamily(new Text("age"));
```

```
for(Entry<Key,Value> entry : s)
    System.out.println(entry.getValue().toString());
```

6.2 RowID Design

Often it is necessary to transform the rowID in order to have rows ordered in a way that is optimal for anticipated access patterns. A good example of this is reversing the order of components of internet domain names in order to group rows of the same parent domain together:

```
com.google.code
com.google.labs
com.google.mail
com.yahoo.mail
com.yahoo.research
```

Some data may result in the creation of very large rows - rows with many columns. In this case the table designer may wish to split up these rows for better load balancing while keeping them sorted together for scanning purposes. This can be done by appending a random substring at the end of the row:

```
com.google.code_00
com.google.code_01
com.google.code_02
com.google.labs_00
com.google.mail_00
com.google.mail_01
```

It could also be done by adding a string representation of some period of time such as date to the week or month:

```
com.google.code_201003
com.google.code_201004
com.google.code_201005
com.google.labs_201003
com.google.mail_201003
com.google.mail_201004
```

Appending dates provides the additional capability of restricting a scan to a given date range.

6.3 Indexing

In order to support lookups via more than one attribute of an entity, additional indexes can be built. However, because Accumulo tables can support any number of columns without specifying them beforehand, a single additional index will often suffice for supporting lookups of records in the main table. Here, the index has, as the rowID, the Value or Term from the main table, the column families are the same, and the column qualifier of the index table contains the rowID from the main table.

Key					Value
Row ID	Column			Timestamp	
	Family	Qualifier	Visibility		
Term	Field Name	MainRowID			

Note: We store rowIDs in the column qualifier rather than the Value so that we can have more than one rowID associated with a particular term within the index. If we stored this in the Value we would only see one of the rows in which the value appears since Accumulo is configured by default to return the one most recent value associated with a key.

Lookups can then be done by scanning the Index Table first for occurrences of the desired values in the columns specified, which returns a list of row ID from the main table. These can then be used to retrieve each matching record, in their entirety, or a subset of their columns, from the Main Table.

To support efficient lookups of multiple rowIDs from the same table, the Accumulo client library provides a BatchScanner. Users specify a set of Ranges to the BatchScanner, which performs the lookups in multiple threads to multiple servers and returns an Iterator over all the rows retrieved. The rows returned are NOT in sorted order, as is the case with the basic Scanner interface.

```
// first we scan the index for IDs of rows matching our query

Text term = new Text("mySearchTerm");

HashSet<Text> matchingRows = new HashSet<Text>();

Scanner indexScanner = createScanner("index", auths);
indexScanner.setRange(new Range(term, term));

// we retrieve the matching rowIDs and create a set of ranges
for(Entry<Key,Value> entry : indexScanner)
    matchingRows.add(new Text(entry.getValue()));
```

```
// now we pass the set of rowIDs to the batch scanner to retrieve them
BatchScanner bscan = conn.createBatchScanner("table", auths, 10);

bscan.setRanges(matchingRows);
bscan.fetchFamily("attributes");

for(Entry<Key,Value> entry : scan)
    System.out.println(e.getValue());
```

One advantage of the dynamic schema capabilities of Accumulo is that different fields may be indexed into the same physical table. However, it may be necessary to create different index tables if the terms must be formatted differently in order to maintain proper sort order. For example, real numbers must be formatted differently than their usual notation in order to be sorted correctly. In these cases, usually one index per unique data type will suffice.

6.4 Entity-Attribute and Graph Tables

Accumulo is ideal for storing entities and their attributes, especially of the attributes are sparse. It is often useful to join several datasets together on common entities within the same table. This can allow for the representation of graphs, including nodes, their attributes, and connections to other nodes.

Rather than storing individual events, Entity-Attribute or Graph tables store aggregate information about the entities involved in the events and the relationships between entities. This is often preferrable when single events aren't very useful and when a continuously updated summarization is desired.

The physical shema for an entity-attribute or graph table is as follows:

Key					Value
Row ID	Column			Timestamp	
	Family	Qualifier	Visibility		
EntityID	Attribute Name	Attribute Value			Weight
EntityID	Edge Type	Related EntityID			Weight

For example, to keep track of employees, managers and products the following entity-attribute table could be used. Note that the weights are not always necessary and are set to 0 when not used.

RowID	ColumnFamily	ColumnQualifier	Value
$E001$	*name*	*bob*	0
$E001$	*department*	*sales*	0
$E001$	*hire_date*	20030102	0
$E001$	*units_sold*	$P001$	780
$E002$	*name*	*george*	0
$E002$	*department*	*sales*	0
$E002$	*manager_of*	$E001$	0
$E002$	*manager_of*	$E003$	0
$E003$	*name*	*harry*	0
$E003$	*department*	*accounts_recv*	0
$E003$	*hire_date*	20000405	0
$E003$	*units_sold*	$P002$	566
$E003$	*units_sold*	$P001$	232
$P001$	*product_name*	*nike_airs*	0
$P001$	*product_type*	*shoe*	0
$P001$	*in_stock*	*germany*	900
$P001$	*in_stock*	*brazil*	200
$P002$	*product_name*	*basic_jacket*	0
$P002$	*product_type*	*clothing*	0
$P002$	*in_stock*	*usa*	3454
$P002$	*in_stock*	*germany*	700

To allow efficient updating of edge weights, an aggregating iterator can be configured to add the value of all mutations applied with the same key. These types of tables can easily be created from raw events by simply extracting the entities, attributes, and relationships from individual events and inserting the keys into Accumulo each with a count of 1. The aggregating iterator will take care of maintaining the edge weights.

6.5 Document-Partitioned Indexing

Using a simple index as described above works well when looking for records that match one of a set of given criteria. When looking for records that match more than one criterion simultane-

ously, such as when looking for documents that contain all of the words 'the' and 'white' and 'house', there are several issues.

First is that the set of all records matching any one of the search terms must be sent to the client, which incurs a lot of network traffic. The second problem is that the client is responsible for performing set intersection on the sets of records returned to eliminate all but the records matching all search terms. The memory of the client may easily be overwhelmed during this operation.

For these reasons Accumulo includes support for a scheme known as sharded indexing, in which these set operations can be performed at the TabletServers and decisions about which records to include in the result set can be made without incurring network traffic.

This is accomplished via partitioning records into bins that each reside on at most one TabletServer, and then creating an index of terms per record within each bin as follows:

Key					Value
Row ID	Column			Timestamp	
	Family	Qualifier	Visibility		
BinID	Term	DocID			Weight

Documents or records are mapped into bins by a user-defined ingest application. By storing the BinID as the RowID we ensure that all the information for a particular bin is contained in a single tablet and hosted on a single TabletServer since Accumulo never splits rows across tablets. Storing the Terms as column families serves to enable fast lookups of all the documents within this bin that contain the given term.

Finally, we perform set intersection operations on the TabletServer via a special iterator called the Intersecting Iterator. Since documents are partitioned into many bins, a search of all documents must search every bin. We can use the BatchScanner to scan all bins in parallel. The Intersecting Iterator should be enabled on a BatchScanner within user query code as follows:

```
Text[] terms = {new Text("the"), new Text("white"), new Text("house")};

BatchScanner bs = conn.createBatchScanner(table, auths, 20);
bs.setScanIterators(20, IntersectingIterator.class.getName(), "ii");

// tells scanner to look for terms in the column family and sends terms
bs.setScanIteratorOption("ii",
    IntersectingIterator.columnFamiliesOptionName,
    IntersectingIterator.encodeColumns(terms));

bs.setRanges(Collections.singleton(new Range()));
```

```
for(Entry<Key,Value> entry : bs) {
    System.out.println(" " + entry.getKey().getColumnQualifier());
}
```

This code effectively has the BatchScanner scan all tablets of a table, looking for documents that match all the given terms. Because all tablets are being scanned for every query, each query is more expensive than other Accumulo scans, which typically involve a small number of TabletServers. This reduces the number of concurrent queries supported and is subject to what is known as the 'straggler' problem in which every query runs as slow as the slowest server participating.

Of course, fast servers will return their results to the client which can display them to the user immediately while they wait for the rest of the results to arrive. If the results are unordered this is quite effective as the first results to arrive are as good as any others to the user.

Chapter 7

High-Speed Ingest

Accumulo is often used as part of a larger data processing and storage system. To maximize the performance of a parallel system involving Accumulo, the ingestion and query components should be designed to provide enough parallelism and concurrency to avoid creating bottlenecks for users and other systems writing to and reading from Accumulo. There are several ways to achieve high ingest performance.

7.1 Pre-Splitting New Tables

New tables consist of a single tablet by default. As mutations are applied, the table grows and splits into multiple tablets which are balanced by the Master across TabletServers. This implies that the aggregate ingest rate will be limited to fewer servers than are available within the cluster until the table has reached the point where there are tablets on every TabletServer.

Pre-splitting a table ensures that there are as many tablets as desired available before ingest begins to take advantage of all the parallelism possible with the cluster hardware. Tables can be split anytime by using the shell:

```
user@myinstance mytable> addsplits -sf /local_splitfile -t mytable
```

For the purposes of providing parallelism to ingest it is not necessary to create more tablets than there are physical machines within the cluster as the aggregate ingest rate is a function of the number of physical machines. Note that the aggregate ingest rate is still subject to the number of machines running ingest clients, and the distribution of rowIDs across the table. The aggregation ingest rate will be suboptimal if there are many inserts into a small number of rowIDs.

7.2 Multiple Ingester Clients

Accumulo is capable of scaling to very high rates of ingest, which is dependent upon not just the number of TabletServers in operation but also the number of ingest clients. This is because a single client, while capable of batching mutations and sending them to all TabletServers, is ultimately limited by the amount of data that can be processed on a single machine. The aggregate ingest rate will scale linearly with the number of clients up to the point at which either the aggregate I/O of TabletServers or total network bandwidth capacity is reached.

In operational settings where high rates of ingest are paramount, clusters are often configured to dedicate some number of machines solely to running Ingester Clients. The exact ratio of clients to TabletServers necessary for optimum ingestion rates will vary according to the distribution of resources per machine and by data type.

7.3 Bulk Ingest

Accumulo supports the ability to import files produced by an external process such as MapReduce into an existing table. In some cases it may be faster to load data this way rather than via ingesting through clients using BatchWriters. This allows a large number of machines to format data the way Accumulo expects. The new files can then simply be introduced to Accumulo via a shell command.

To configure MapReduce to format data in preparation for bulk loading, the job should be set to use a range partitioner instead of the default hash partitioner. The range partitioner uses the split points of the Accumulo table that will receive the data. The split points can be obtained from the shell and used by the MapReduce RangePartitioner. Note that this is only useful if the existing table is already split into multiple tablets.

```
user@myinstance mytable> getsplits
aa
ab
ac
...
zx
zy
zz
```

Run the MapReduce job, using the AccumuloFileOutputFormat to create the files to be introduced to Accumulo. Once this is complete, the files can be added to Accumulo via the shell:

```
user@myinstance mytable> importdirectory /files_dir /failures
```

Note that the paths referenced are directories within the same HDFS instance over which Accumulo is running. Accumulo places any files that failed to be added to the second directory specified.

A complete example of using Bulk Ingest can be found at
accumulo/docs/examples/README.bulkIngest

7.4 MapReduce Ingest

It is possible to efficiently write many mutations to Accumulo in parallel via a MapReduce job. In this scenario the MapReduce is written to process data that lives in HDFS and write mutations to Accumulo using the AccumuloOutputFormat. See the MapReduce section under Analytics for details.

An example of using MapReduce can be found under
accumulo/docs/examples/README.mapred

Chapter 8

Analytics

Accumulo supports more advanced data processing than simply keeping keys sorted and performing efficient lookups. Analytics can be developed by using MapReduce and Iterators in conjunction with Accumulo tables.

8.1 MapReduce

Accumulo tables can be used as the source and destination of MapReduce jobs. To use an Accumulo table with a MapReduce job (specifically with the new Hadoop API as of version 0.20), configure the job parameters to use the AccumuloInputFormat and AccumuloOutputFormat. Accumulo specific parameters can be set via these two format classes to do the following:

- Authenticate and provide user credentials for the input
- Restrict the scan to a range of rows
- Restrict the input to a subset of available columns

8.1.1 Mapper and Reducer classes

To read from an Accumulo table create a Mapper with the following class parameterization and be sure to configure the AccumuloInputFormat.

```
class MyMapper extends Mapper<Key,Value,WritableComparable,Writable> {
    public void map(Key k, Value v, Context c) {
        // transform key and value data here
```

```
    }
}
```

To write to an Accumulo table, create a Reducer with the following class parameterization and be sure to configure the AccumuloOutputFormat. The key emitted from the Reducer identifies the table to which the mutation is sent. This allows a single Reducer to write to more than one table if desired. A default table can be configured using the AccumuloOutputFormat, in which case the output table name does not have to be passed to the Context object within the Reducer.

```
class MyReducer extends Reducer<WritableComparable, Writable, Text, Mutation> {

    public void reduce(WritableComparable key, Iterator<Text> values, Context c) {

        Mutation m;

        // create the mutation based on input key and value

        c.write(new Text("output-table"), m);
    }
}
```

The Text object passed as the output should contain the name of the table to which this mutation should be applied. The Text can be null in which case the mutation will be applied to the default table name specified in the AccumuloOutputFormat options.

8.1.2 AccumuloInputFormat options

```
Job job = new Job(getConf());
AccumuloInputFormat.setInputInfo(job,
        "user",
        "passwd".getBytes(),
        "table",
        new Authorizations());

AccumuloInputFormat.setZooKeeperInstance(job, "myinstance",
        "zooserver-one,zooserver-two");
```

Optional settings:

To restrict Accumulo to a set of row ranges:

```
ArrayList<Range> ranges = new ArrayList<Range>();
// populate array list of row ranges ...
AccumuloInputFormat.setRanges(job, ranges);
```

To restrict accumulo to a list of columns:

```
ArrayList<Pair<Text,Text>> columns = new ArrayList<Pair<Text,Text>>();
// populate list of columns
AccumuloInputFormat.fetchColumns(job, columns);
```

To use a regular expression to match row IDs:

```
AccumuloInputFormat.setRegex(job, RegexType.ROW, "^.*");
```

8.1.3 AccumuloOutputFormat options

```
boolean createTables = true;
String defaultTable = "mytable";

AccumuloOutputFormat.setOutputInfo(job,
        "user",
        "passwd".getBytes(),
        createTables,
        defaultTable);

AccumuloOutputFormat.setZooKeeperInstance(job, "myinstance",
        "zooserver-one,zooserver-two");
```

Optional Settings:

```
AccumuloOutputFormat.setMaxLatency(job, 300); // milliseconds
AccumuloOutputFormat.setMaxMutationBufferSize(job, 5000000); // bytes
```

An example of using MapReduce with Accumulo can be found at
accumulo/docs/examples/README.mapred

8.2 Aggregating Iterators

Many applications can benefit from the ability to aggregate values across common keys. This can be done via aggregating iterators and is similar to the Reduce step in MapReduce. This provides the ability to define online, incrementally updated analytics without the overhead or latency associated with batch-oriented MapReduce jobs.

All that is needed to aggregate values of a table is to identify the fields over which values will be grouped, insert mutations with those fields as the key, and configure the table with an aggregating iterator that supports the summarization operation desired.

The only restriction on an aggregating iterator is that the aggregator developer should not assume that all values for a given key have been seen, since new mutations can be inserted at anytime. This precludes using the total number of values in the aggregation such as when calculating an average, for example.

8.2.1 Feature Vectors

An interesting use of aggregating iterators within an Accumulo table is to store feature vectors for use in machine learning algorithms. For example, many algorithms such as k-means clustering, support vector machines, anomaly detection, etc. use the concept of a feature vector and the calculation of distance metrics to learn a particular model. The columns in an Accumulo table can be used to efficiently store sparse features and their weights to be incrementally updated via the use of an aggregating iterator.

8.3 Statistical Modeling

Statistical models that need to be updated by many machines in parallel could be similarly stored within an Accumulo table. For example, a MapReduce job that is iteratively updating a global statistical model could have each map or reduce worker reference the parts of the model to be read and updated through an embedded Accumulo client.

Using Accumulo this way enables efficient and fast lookups and updates of small pieces of information in a random access pattern, which is complementary to MapReduce's sequential access model.

Chapter 9

Security

Accumulo extends the BigTable data model to implement a security mechanism known as cell-level security. Every key-value pair has its own security label, stored under the column visibility element of the key, which is used to determine whether a given user meets the security requirements to read the value. This enables data of various security levels to be stored within the same row, and users of varying degrees of access to query the same table, while preserving data confidentiality.

9.1 Security Label Expressions

When mutations are applied, users can specify a security label for each value. This is done as the Mutation is created by passing a ColumnVisibility object to the put() method:

```
Text rowID = new Text("row1");
Text colFam = new Text("myColFam");
Text colQual = new Text("myColQual");
ColumnVisibility colVis = new ColumnVisibility("public");
long timestamp = System.currentTimeMillis();

Value value = new Value("myValue");

Mutation mutation = new Mutation(rowID);
mutation.put(colFam, colQual, colVis, timestamp, value);
```

9.2 Security Label Expression Syntax

Security labels consist of a set of user-defined tokens that are required to read the value the label is associated with. The set of tokens required can be specified using syntax that supports logical AND and OR combinations of tokens, as well as nesting groups of tokens together.

For example, suppose within our organization we want to label our data values with security labels defined in terms of user roles. We might have tokens such as:

```
admin
audit
system
```

These can be specified alone or combined using logical operators:

```
// Users must have admin privileges:
admin

// Users must have admin and audit privileges
admin&audit

// Users with either admin or audit privileges
admin|audit

// Users must have audit and one or both of admin or system
(admin|system)&audit
```

When both | and & operators are used, parentheses must be used to specify precedence of the operators.

9.3 Authorization

When clients attempt to read data from Accumulo, any security labels present are examined against the set of authorizations passed by the client code when the Scanner or BatchScanner are created. If the authorizations are determined to be insufficient to satisfy the security label, the value is suppressed from the set of results sent back to the client.

Authorizations are specified as a comma-separated list of tokens the user possesses:

```
// user possess both admin and system level access
Authorization auths = new Authorization("admin","system");

Scanner s = connector.createScanner("table", auths);
```

9.4 Secure Authorizations Handling

Because the client can pass any authorization tokens to Accumulo, applications must be designed to obtain users' authorization tokens from a trusted 3rd party rather than having the users specify their authorizations directly.

Often production systems will integrate with Public-Key Infrastructure (PKI) and designate client code within the query layer to negotiate with PKI servers in order to authenticate users and retrieve their authorization tokens (credentials). This requires users to specify only the information necessary to authenticate themselves to the system. Once user identity is established, their credentials can be accessed by the client code and passed to Accumulo outside of the reach of the user.

9.5 Query Services Layer

Since the primary method of interaction with Accumulo is through the Java API, production environments often call for the implementation of a Query layer. This can be done using web services in containers such as Apache Tomcat, but is not a requirement. The Query Services Layer provides a mechanism for providing a platform on which user facing applications can be built. This allows the application designers to isolate potentially complex query logic, and enables a convenient point at which to perform essential security functions.

Several production environments choose to implement authentication at this layer, where users identifiers are used to retrieve their access credentials which are then cached within the query layer and presented to Accumulo through the Authorizations mechanism.

Typically, the query services layer sits between Accumulo and user workstations.

Chapter 10

Administration

10.1 Hardware

Because we are running essentially two or three systems simultaneously layered across the cluster: HDFS, Accumulo and MapReduce, it is typical for hardware to consist of 4 to 8 cores, and 8 to 32 GB RAM. This is so each running process can have at least one core and 2 - 4 GB each.

One core running HDFS can typically keep 2 to 4 disks busy, so each machine may typically have as little as 2 x 300GB disks and as much as 4 x 1TB or 2TB disks.

It is possible to do with less than this, such as with 1u servers with 2 cores and 4GB each, but in this case it is recommended to only run up to two processes per machine - i.e. DataNode and TabletServer or DataNode and MapReduce worker but not all three. The constraint here is having enough available heap space for all the processes on a machine.

10.2 Network

Accumulo communicates via remote procedure calls over TCP/IP for both passing data and control messages. In addition, Accumulo uses HDFS clients to communicate with HDFS. To achieve good ingest and query performance, sufficient network bandwidth must be available between any two machines.

10.3 Installation

Choose a directory for the Accumulo installation. This directory will be referenced by the environment variable `$ACCUMULO_HOME`. Run the following:

```
$ tar xzf $ACCUMULO_HOME/accumulo.tar.gz
```

Repeat this step at each machine within the cluster. Usually all machines have the same `$ACCUMULO_HOME`.

10.4 Dependencies

Accumulo requires HDFS and ZooKeeper to be configured and running before starting. Passwordless SSH should be configured between at least the Accumulo master and TabletServer machines. It is also a good idea to run Network Time Protocol (NTP) within the cluster to ensure nodes' clocks don't get too out of sync, which can cause problems with automatically timestamped data. Accumulo will remove from the set of TabletServers those machines whose times differ too much from the master's.

10.5 Configuration

Accumulo is configured by editing several Shell and XML files found in `$ACCUMULO_HOME/conf`. The structure closely resembles Hadoop's configuration files.

10.5.1 Edit conf/accumulo-env.sh

Accumulo needs to know where to find the software it depends on. Edit accumuloenv. sh and specify the following:

1. Enter the location of the installation directory of Accumulo for `$ACCUMULO_HOME`
2. Enter your system's Java home for `$JAVA_HOME`
3. Enter the location of Hadoop for `$HADOOP_HOME`
4. Choose a location for Accumulo logs and enter it for `$ACCUMULO_LOG_DIR`
5. Enter the location of ZooKeeper for `$ZOOKEEPER_HOME`

By default Accumulo TabletServers are set to use 1GB of memory. You may change this by altering the value of `$ACCUMULO_TSERVER_OPTS`. Note the syntax is that of the Java JVM command line options. This value should be less than the physical memory of the machines running TabletServers.

There are similar options for the master's memory usage and the garbage collector process. Reduce these if they exceed the physical RAM of your hardware and increase them, within the bounds of the physical RAM, if a process fails because of insufficient memory.

Note that you will be specifying the Java heap space in accumulo-env.sh. You should make sure that the total heap space used for the Accumulo tserver and the Hadoop DataNode and TaskTracker is less than the available memory on each slave node in the cluster. On large clusters, it is recommended that the Accumulo master, Hadoop NameNode, secondary NameNode, and Hadoop JobTracker all be run on separate machines to allow them to use more heap space. If you are running these on the same machine on a small cluster, likewise make sure their heap space settings fit within the available memory.

10.5.2 Cluster Specification

On the machine that will serve as the Accumulo master:

1. Write the IP address or domain name of the Accumulo Master to the `$ACCUMULO_HOME/conf/masters` file.
2. Write the IP addresses or domain name of the machines that will be TabletServers in `$ACCUMULO_HOME/conf/slaves`, one per line.

Note that if using domain names rather than IP addresses, DNS must be configured properly for all machines participating in the cluster. DNS can be a confusing source of errors.

10.5.3 Accumulo Settings

Specify appropriate values for the following settings in `$ACCUMULO_HOME/conf/accumulo-site.xml` :

```
<property>
    <name>zookeeper</name>
    <value>zooserver-one:2181,zooserver-two:2181</value>
    <description>list of zookeeper servers</description>
</property>
<property>
    <name>walog</name>
```

```
    <value>/var/accumulo/walogs</value>
    <description>local directory for write ahead logs</description>
</property>
```

This enables Accumulo to find ZooKeeper. Accumulo uses ZooKeeper to coordinate settings between processes and helps finalize TabletServer failure.

Accumulo records all changes to tables to a write-ahead log before committing them to the table. The 'walog' setting specifies the local directory on each machine to which write-ahead logs are written. This directory should exist on all machines acting as TabletServers.

Some settings can be modified via the Accumulo shell and take effect immediately. However, any settings that should be persisted across system restarts must be recorded in the accumulo-site.xml file.

10.5.4 Deploy Configuration

Copy the masters, slaves, accumulo-env.sh, and if necessary, accumulo-site.xml from the `$ACCUMULO_HOME/conf/` directory on the master to all the machines specified in the slaves file.

10.6 Initialization

Accumulo must be initialized to create the structures it uses internally to locate data across the cluster. HDFS is required to be configured and running before Accumulo can be initialized.

Once HDFS is started, initialization can be performed by executing `$ACCUMULO_HOME/bin/accumulo init` . This script will prompt for a name for this instance of Accumulo. The instance name is used to identify a set of tables and instance-specific settings. The script will then write some information into HDFS so Accumulo can start properly.

The initialization script will prompt you to set a root password. Once Accumulo is initialized it can be started.

10.7 Running

10.7.1 Starting Accumulo

Make sure Hadoop is configured on all of the machines in the cluster, including access to a shared HDFS instance. Make sure HDFS and ZooKeeper are running. Make sure ZooKeeper

is configured and running on at least one machine in the cluster. Start Accumulo using the `bin/start-all.sh` script.

To verify that Accumulo is running, check the Status page as described under *Monitoring*. In addition, the Shell can provide some information about the status of tables via reading the !METADATA table.

10.7.2 Stopping Accumulo

To shutdown cleanly, run `bin/stop-all.sh` and the master will orchestrate the shutdown of all the tablet servers. Shutdown waits for all minor compactions to finish, so it may take some time for particular configurations.

10.8 Monitoring

The Accumulo Master provides an interface for monitoring the status and health of Accumulo components. This interface can be accessed by pointing a web browser to
`http://accumulomaster:50095/status`

10.9 Logging

Accumulo processes each write to a set of log files. By default these are found under
`$ACCUMULO/logs/`.

10.10 Recovery

In the event of TabletServer failure or error on shutting Accumulo down, some mutations may not have been minor compacted to HDFS properly. In this case, Accumulo will automatically reapply such mutations from the write-ahead log either when the tablets from the failed server are reassigned by the Master, in the case of a single TabletServer failure or the next time Accumulo starts, in the event of failure during shutdown.

Recovery is performed by asking the loggers to copy their write-ahead logs into HDFS. As the logs are copied, they are also sorted, so that tablets can easily find their missing updates. The copy/sort status of each file is displayed on Accumulo monitor status page. Once the recovery

is complete any tablets involved should return to an “online" state. Until then those tablets will be unavailable to clients.

The Accumulo client library is configured to retry failed mutations and in many cases clients will be able to continue processing after the recovery process without throwing an exception.

Note that because Accumulo uses timestamps to order mutations, any mutations that are applied as part of the recovery process should appear to have been applied when they originally arrived at the TabletServer that failed. This makes the ordering of mutations consistent in the presence of failure.

Appendix A

Shell Commands

?

```
usage: ? [ <command> <command> ] [-?] [-np]
description: provides information about the available commands
  -?,--help  display this help
  -np,--no-pagination  disables pagination of output
```

about

```
usage: about [-?] [-v]
description: displays information about this program
  -?,--help  display this help
  -v,--verbose displays details session information
```

addsplits

```
usage: addsplits [<split> <split> ] [-?] [-b64] [-sf <filename>] -t <tableName>
description: add split points to an existing table
  -?,--help  display this help
  -b64,--base64encoded decode encoded split points
  -sf,--splits-file <filename>file with newline separated list of rows to add
       to table
  -t,--table <tableName>  name of a table to add split points to
```

authenticate

```
    usage: authenticate <username> [-?]
    description: verifies a user's credentials
      -?,--help  display this help
```

bye

```
    usage: bye [-?]
    description: exits the shell
      -?,--help  display this help
```

classpath

```
    usage: classpath [-?]
    description: lists the current files on the classpath
      -?,--help  display this help
```

clear

```
    usage: clear [-?]
    description: clears the screen
      -?,--help  display this help
```

cls

```
    usage: cls [-?]
    description: clears the screen
      -?,--help  display this help
```

compact

```
    usage: compact [-?] [--override] -p <pattern> | -t <tableName>
    description: sets all tablets for a table to major compact as soon as possible
           (based on current time)
      -?,--help  display this help
      --override  override a future scheduled compaction
      -p,--pattern <pattern>  regex pattern of table names to flush
      -t,--table <tableName>  name of a table to flush
```

config

```
usage: config [-?] [-d <property> | -f <string> | -s <property=value>] [-np]
       [-t <table>]
description: prints system properties and table specific properties
  -?,--help  display this help
  -d,--delete <property>  delete a per-table property
  -f,--filter <string>show only properties that contain this string
  -np,--no-pagination  disables pagination of output
  -s,--set <property=value>  set a per-table property
  -t,--table <table>  display/set/delete properties for specified table
```

createtable

```
usage: createtable <tableName> [-?] [-a
       <<columnfamily>[:<columnqualifier>]=<aggregation_class>>] [-b64]
       [-cc <table>] [-cs <table> | -sf <filename>] [-ndi]  [-tl | -tm]
description: creates a new table, with optional aggregators and optionally
       pre-split
  -?,--help  display this help
  -a,--aggregator <<columnfamily>[:<columnqualifier>]=<aggregation_class>>
       comma separated column=aggregator
  -b64,--base64encoded decode encoded split points
  -cc,--copy-config <table>  table to copy configuration from
  -cs,--copy-splits <table>  table to copy current splits from
  -ndi,--no-default-iterators  prevents creation of the normal default iterator
       set
  -sf,--splits-file <filename>file with newline separated list of rows to
       create a pre-split table
  -tl,--time-logical  use logical time
  -tm,--time-millis  use time in milliseconds
```

createuser

```
usage: createuser <username> [-?] [-s <comma-separated-authorizations>]
description: creates a new user
  -?,--help  display this help
  -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
```

debug

```
usage: debug [ on | off ] [-?]
description: turns debug logging on or off
  -?,--help  display this help
```

delete

```
usage: delete <row> <colfamily> <colqualifier> [-?] [-l <expression>] [-t
       <timestamp>]
description: deletes a record from a table
  -?,--help  display this help
  -l,--authorization-label <expression>  formatted authorization label expression
  -t,--timestamp <timestamp>  timestamp to use for insert
```

deleteiter

```
usage: deleteiter [-?] [-majc] [-minc] -n <itername> [-scan] [-t <table>]
description: deletes a table-specific iterator
  -?,--help  display this help
  -majc,--major-compaction  applied at major compaction
  -minc,--minor-compaction  applied at minor compaction
  -n,--name <itername>iterator to delete
  -scan,--scan-time  applied at scan time
  -t,--table <table>  tableName
```

deletemany

```
usage: deletemany [-?] [-b <start-row>] [-c
       <<columnfamily>[:<columnqualifier>]>] [-e <end-row>] [-f] [-np]
       [-s <comma-separated-authorizations>] [-st]
description: scans a table and deletes the resulting records
  -?,--help  display this help
  -b,--begin-row <start-row>  begin row (inclusive)
  -c,--columns <<columnfamily>[:<columnqualifier>]>  comma-separated columns
  -e,--end-row <end-row>  end row (inclusive)
  -f,--force  forces deletion without prompting
  -np,--no-pagination  disables pagination of output
  -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
       (all user auths are used if this argument is not specified)
  -st,--show-timestamps  enables displaying timestamps
```

deletescaniter

```
usage: deletescaniter [-?] [-a] [-n <itername>] [-t <table>]
description: deletes a table-specific scan iterator so it is no longer used
       during this shell session
  -?,--help  display this help
  -a,--all  delete all for tableName
  -n,--name <itername>iterator to delete
  -t,--table <table>  tableName
```

deletetable

```
usage: deletetable <tableName> [-?]
description: deletes a table
  -?,--help  display this help
```

deleteuser

```
usage: deleteuser <username> [-?]
description: deletes a user
  -?,--help  display this help
```

droptable

```
usage: droptable <tableName> [-?]
description: deletes a table
  -?,--help  display this help
```

dropuser

```
usage: dropuser <username> [-?]
description: deletes a user
  -?,--help  display this help
```

egrep

```
usage: egrep <regex> <regex> [-?] [-b <start-row>] [-c
       <<columnfamily>[:<columnqualifier>]>] [-e <end-row>] [-np] [-s
       <comma-separated-authorizations>] [-st] [-t <arg>]
description: egreps a table in parallel on the server side (uses java regex)
```

```
  -?,--help  display this help
  -b,--begin-row <start-row>  begin row (inclusive)
  -c,--columns <<columnfamily>[:<columnqualifier>]>  comma-separated columns
  -e,--end-row <end-row>  end row (inclusive)
  -np,--no-pagination  disables pagination of output
  -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
       (all user auths are used if this argument is not specified)
  -st,--show-timestamps  enables displaying timestamps
  -t,--num-threads <arg>  num threads
```

execfile

```
usage: execfile [-?] [-v]
description: specifies a file containing accumulo commands to execute
  -?,--help  display this help
  -v,--verbose displays command prompt as commands are executed
```

exit

```
usage: exit [-?]
description: exits the shell
  -?,--help  display this help
```

flush

```
usage: flush [-?] -p <pattern> | -t <tableName>
description: makes a best effort to flush tables from memory to disk
  -?,--help  display this help
  -p,--pattern <pattern>  regex pattern of table names to flush
  -t,--table <tableName>  name of a table to flush
```

formatter

```
usage: formatter [-?] -f <className> | -l | -r
description: specifies a formatter to use for displaying database entries
  -?,--help  display this help
  -f,--formatter <className>  fully qualified name of formatter class to use
  -l,--list  display the current formatter
  -r,--reset  reset to default formatter
```

getauths

```
usage: getauths [-?] [-u <user>]
description: displays the maximum scan authorizations for a user
  -?,--help  display this help
  -u,--user <user>  user to operate on
```

getgroups

```
usage: getgroups [-?] -t <table>
description: gets the locality groups for a given table
  -?,--help  display this help
  -t,--table <table>  get locality groups for specified table
```

getsplits

```
usage: getsplits [-?] [-b64] [-m <num>] [-o <file>] [-v]
description: retrieves the current split points for tablets in the current table
  -?,--help  display this help
  -b64,--base64encoded encode the split points
  -m,--max <num>  specifies the maximum number of splits to create
  -o,--output <file>  specifies a local file to write the splits to
  -v,--verbose print out the tablet information with start/end rows
```

grant

```
usage: grant <permission> [-?] -p <pattern> | -s | -t <table>  -u <username>
description: grants system or table permissions for a user
  -?,--help  display this help
  -p,--pattern <pattern>  regex pattern of tables to grant permissions on
  -s,--system  grant a system permission
  -t,--table <table>  grant a table permission on this table
  -u,--user <username>user to operate on
```

grep

```
usage: grep <term> <term> [-?] [-b <start-row>] [-c
       <<columnfamily>[:<columnqualifier>]>] [-e <end-row>] [-np] [-s
       <comma-separated-authorizations>] [-st] [-t <arg>]
description: searches a table for a substring, in parallel, on the server side
```

```
  -?,--help  display this help
  -b,--begin-row <start-row>  begin row (inclusive)
  -c,--columns <<columnfamily>[:<columnqualifier>]>  comma-separated columns
  -e,--end-row <end-row>  end row (inclusive)
  -np,--no-pagination  disables pagination of output
  -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
       (all user auths are used if this argument is not specified)
  -st,--show-timestamps  enables displaying timestamps
  -t,--num-threads <arg>  num threads
```

help

```
usage: help [ <command> <command> ] [-?] [-np]
description: provides information about the available commands
  -?,--help  display this help
  -np,--no-pagination  disables pagination of output
```

importdirectory

```
usage: importdirectory <directory> <failureDirectory> [-?] [-a <num>] [-f <num>]
       [-g] [-v]
description: bulk imports an entire directory of data files to the current table
  -?,--help  display this help
  -a,--numAssignThreads <num>  number of assign threads for import (default: 20)
  -f,--numFileThreads <num>  number of threads to process files (default: 8)
  -g,--disableGC  prevents imported files from being deleted by the garbage
       collector
  -v,--verbose displays statistics from the import
```

info

```
usage: info [-?] [-v]
description: displays information about this program
  -?,--help  display this help
  -v,--verbose displays details session information
```

insert

```
usage: insert <row> <colfamily> <colqualifier> <value> [-?] [-l <expression>] [-t
       <timestamp>]
```

```
    description: inserts a record
      -?,--help  display this help
      -l,--authorization-label <expression>  formatted authorization label expression
      -t,--timestamp <timestamp>  timestamp to use for insert
```

listscans

```
    usage: listscans [-?] [-np] [-ts <tablet server>]
    description: list what scans are currently running in accumulo. See the
           org.apache.accumulo.core.client.admin.ActiveScan javadoc for more information
           about columns.
      -?,--help  display this help
      -np,--no-pagination  disables pagination of output
      -ts,--tabletServer <tablet server>  list scans for a specific tablet server
```

masterstate

```
    usage: masterstate <NORMAL|SAFE_MODE|CLEAN_STOP> [-?]
    description: set the master state: NORMAL, SAFE_MODE or CLEAN_STOP
      -?,--help  display this help
```

offline

```
    usage: offline [-?] -p <pattern> | -t <tableName>
    description: starts the process of taking table offline
      -?,--help  display this help
      -p,--pattern <pattern>  regex pattern of table names to flush
      -t,--table <tableName>  name of a table to flush
```

online

```
    usage: online [-?] -p <pattern> | -t <tableName>
    description: starts the process of putting a table online
      -?,--help  display this help
      -p,--pattern <pattern>  regex pattern of table names to flush
      -t,--table <tableName>  name of a table to flush
```

passwd

```
    usage: passwd [-?] [-u <user>]
```

```
    description: changes a user's password
      -?,--help  display this help
      -u,--user <user>  user to operate on
```

quit

```
    usage: quit [-?]
    description: exits the shell
      -?,--help  display this help
```

renametable

```
    usage: renametable <current table name> <new table name> [-?]
    description: rename a table
      -?,--help  display this help
```

revoke

```
    usage: revoke <permission> [-?] -s | -t <table>  -u <username>
    description: revokes system or table permissions from a user
      -?,--help  display this help
      -s,--system  revoke a system permission
      -t,--table <table>  revoke a table permission on this table
      -u,--user <username>user to operate on
```

scan

```
    usage: scan [-?] [-b <start-row>] [-c <<columnfamily>[:<columnqualifier>]>] [-e
           <end-row>] [-np] [-s <comma-separated-authorizations>] [-st]
    description: scans the table, and displays the resulting records
      -?,--help  display this help
      -b,--begin-row <start-row>  begin row (inclusive)
      -c,--columns <<columnfamily>[:<columnqualifier>]>  comma-separated columns
      -e,--end-row <end-row>  end row (inclusive)
      -np,--no-pagination  disables pagination of output
      -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
           (all user auths are used if this argument is not specified)
      -st,--show-timestamps  enables displaying timestamps
```

select

```
    usage: select <row> <columnfamily> <columnqualifier> [-?] [-np] [-s
           <comma-separated-authorizations>] [-st]
    description: scans for and displays a single record
      -?,--help  display this help
      -np,--no-pagination  disables pagination of output
      -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
      -st,--show-timestamps  enables displaying timestamps
```

selectrow

```
    usage: selectrow <row> [-?] [-np] [-s <comma-separated-authorizations>] [-st]
    description: scans a single row and displays all resulting records
      -?,--help  display this help
      -np,--no-pagination  disables pagination of output
      -s,--scan-authorizations <comma-separated-authorizations>  scan authorizations
      -st,--show-timestamps  enables displaying timestamps
```

setauths

```
    usage: setauths [-?] -c | -s <comma-separated-authorizations>  [-u <user>]
    description: sets the maximum scan authorizations for a user
      -?,--help  display this help
      -c,--clear-authorizations  clears the scan authorizations
      -s,--scan-authorizations <comma-separated-authorizations>  set the scan
           authorizations
      -u,--user <user>  user to operate on
```

setgroups

```
    usage: setgroups <group>=<col fam>,<col fam> <group>=<col fam>,<col fam>
           [-?] -t <table>
    description: sets the locality groups for a given table (for binary or commas,
           use Java API)
      -?,--help  display this help
      -t,--table <table>  get locality groups for specified table
```

setiter

```
    usage: setiter [-?] -agg | -class <name> | -filter | -nolabel | -regex | -vers
```

```
       [-majc] [-minc] [-n <itername>]  -p <pri>  [-scan] [-t <table>]
  description: sets a table-specific iterator
    -?,--help  display this help
    -agg,--aggregator  an aggregating type
    -class,--class-name <name>  a java class type
    -filter,--filter  a filtering type
    -majc,--major-compaction  applied at major compaction
    -minc,--minor-compaction  applied at minor compaction
    -n,--name <itername>iterator to set
    -nolabel,--no-label  a no-labeling type
    -p,--priority <pri>  the order in which the iterator is applied
    -regex,--regular-expression  a regex matching type
    -scan,--scan-time  applied at scan time
    -t,--table <table>  tableName
    -vers,--version  a versioning type
```

setscaniter

```
  usage: setscaniter [-?] -agg | -class <name> | -filter | -nolabel | -regex |
       -vers  [-n <itername>]  -p <pri>[-t <table>]
  description: sets a table-specific scan iterator for this shell session
    -?,--help  display this help
    -agg,--aggregator  an aggregating type
    -class,--class-name <name>  a java class type
    -filter,--filter  a filtering type
    -n,--name <itername>iterator to set
    -nolabel,--no-label  a no-labeling type
    -p,--priority <pri>  the order in which the iterator is applied
    -regex,--regular-expression  a regex matching type
    -t,--table <table>  tableName
    -vers,--version  a versioning type
```

systempermissions

```
  usage: systempermissions [-?]
  description: displays a list of valid system permissions
    -?,--help  display this help
```

table

```
    usage: table <tableName> [-?]
    description: switches to the specified table
      -?,--help  display this help
```

tablepermissions

```
    usage: tablepermissions [-?]
    description: displays a list of valid table permissions
      -?,--help  display this help
```

tables

```
    usage: tables [-?] [-l]
    description: displays a list of all existing tables
      -?,--help  display this help
      -l,--list-ids  display internal table ids along with the table name
```

trace

```
    usage: trace [ on | off ] [-?]
    description: turns trace logging on or off
      -?,--help  display this help
```

user

```
    usage: user <username> [-?]
    description: switches to the specified user
      -?,--help  display this help
```

userpermissions

```
    usage: userpermissions [-?] [-u <user>]
    description: displays a user's system and table permissions
      -?,--help  display this help
      -u,--user <user>  user to operate on
```

users

```
    usage: users [-?]
    description: displays a list of existing users
```

```
  -?,--help  display this help
```

whoami

```
usage: whoami [-?]
description: reports the current user name
  -?,--help  display this help
```