HBase

Week 3

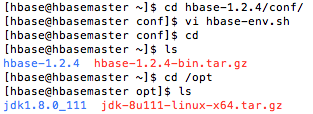
Sean O’Malley

* HBase
  + Is a distributed column oriented database built on top of HDFS
  + Hadoop application used when you require real time read/write random access to a very large dataset
* Column family oriented store
  + Applications store data in labeled tables.
  + Table cells, the intersection of row and column coordinates are versioned
  + By default, their version is a timestamp auto-assigned by HBase at the time of the cell insertion.
  + A cell’s content is an uninterpreted array of bytes
  + Table row keys are also byte arrays, so theoretically anything can serve as a row key, from strings to binary representations of long or even serialized data structures.
  + Table rows are sorted by row key, aka the table’s primary key. The sort is byte ordered. All table access are via the primary key
  + Row columns are grouped into column families. All column family members have a common prefix. The column family prefix must be composed of printable characters.
  + The column family and column qualifier are always separated by a colon.
    - EG: info:format
  + A table’s column families must be specified up front as part of the table schema definition, but new column qualifiers can be added on demand.
  + Physically, all column family members are stored together on the filesystem
  + Synopsis:
    - 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 pre-exists.
* Regions
  + Tables are automatically partitioned horizontally by HBase into regions.
  + Each region comprises a subset of a table’s rows. A region is denoted by the table it belongs to, its first row, and its last row.
  + Initially a table is comprised of a single region, but as the region grows it eventually crosses a configurable size threshold, at which point it splits at a row boundary into two new regions of approximately equal size.
  + Until the first split happens, all loading will be against the single server hosting the original region.
  + As the table grows, the number of its regions grows
  + Regions are the units that get distributed over an HBase cluster.
    - In this way, a table that is too big for any one server can be carried by a cluster of servers, with each node hosting a subset of the table’s total regions.
    - This is also the means by which the loading on a table gets distributed.
* Locking
  + Row updates are atomic, no matter how many row columns constitute the row-level transaction. This keeps the model locking simple.
* Implementation
  + HBase is made up of an HBase master node orchestrating a cluster of one or more regionserver workers.
  + The HBase master is responsible for bootstrapping a virgin install, for assigning regions to registered regionservers, and for recovering regionserver failures.
  + The master node is lightly loaded.
  + The regionservers carry zero or more regions and filed client read / write requests and also manager region splits, informing the HBase master about the new daughter regions so it can manage the offlining of parent regions and assignment of the replacement daughters.
* Zookeeper interaction
  + HBase depends on Zookeeper and by default managers a Zookeeper instance as the authority on cluster state, although it can be configured to use an existing Zookeeper cluster instead
  + Zookeeper ensemble hosts the location of the hbase:meta catalog table and the address of the current cluster master.
  + The assignments of regions is mediated via Zookeeper in case participating servers crash mid-assignment
  + Hosting the assignment transaction state in Zookeeper makes it so recovery can pick up on the assignment where the crashed server left off
  + At a minimum, when bootstrapping a client connection to an HBase cluster, the client must be passed the location of the Zookeeper ensemble. Thereafter, the client navigates the ZooKeeper hierarchy to learn cluster attributes such as server locations
* Configuration
  + Regionserver worker nodes are listed in HBase **conf/regionservers** file, as you would list datanodes and node managers in the Hadoop **etc/hadoop/slaves** file.
  + Start and stop scripts like those in Hadoop and use the same SSH-based mechanism for running remote commands.
  + A cluster’s site-specific configuration is done in the HBase **conf/hbase-site.xml** and **conf/hbase-env.sh** files, which have the same equivalents in the Hadoop parent project.
* Data Storage FYI
  + HBase persists data via the Hadoop filesystem API.
  + Most people using HBase run it on HDFS storage, though by default, and unless told otherwise, HBase writes to the local filesystem.
* HBase in Operation
  + Hbase keeps a special catalog table named **hbase:meta**, within which it maintains the current list, state, and locations of all user space regions afloat in the cluster.
  + Entries in **hbase:meta** are keyed by region name, where a region name is made up of the name of the table the region belongs to, the region’s start row, its time of creation and finally and MD5 hash of all of these.
    - Commas delimit the table name, start row, and timestamp. The MD5 hash is surrounded by a leading and trailing period
    - **TestTable,xyz,1234343589082.1be980s8ve9vdd908807exy.**
  + Fresh clients connect to Zookeeper cluster first to learn the location of hbase:meta.
  + The client then does a lookup against the appropriate **hbase:meta** region to figure out the hosting user-space region and its location. Thereafter, the client interacts directly with the hosting regionserver.
  + Writes arriving at a regionserver are first appended to a commit log and then added to an in-memory memstore. When a memstore fills, its contents are flushed to the filesystem
  + The commit log is hosted on HDFS, so it remains available through a regionserver crash. When the master notices the regionserver is no longer reachable, it splits the dead regionserver’s commit log by region. On reassignment and before they reopen for business, regions that were on the dead regionserver will pick up their just-split files of not-yet-persisted edits and replay them to bring themselves up to date with the state they had before failure.
  + A separate process running the regionserver monitors flush file sizes, splitting the region when they grow in excess of the configured maximum.

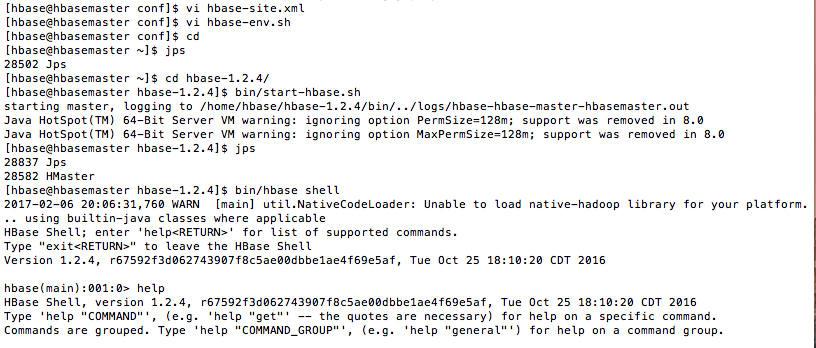
**(Installation instructions below)**

**Installation**

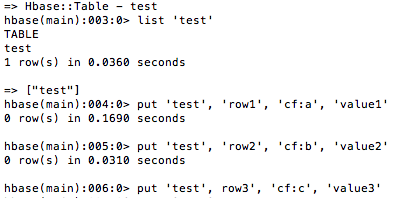
* + I created a CentOS virtual machine, calling it hbasemaster. I downloaded the necessary yum, nano, .bashrc, jdk and jps updates under the root user then created a passwordless user named hbase.
  + I located the hbase and jdk file locations in my local machine and reflected those changes in my hbase-env.sh and hbase-site.xml.



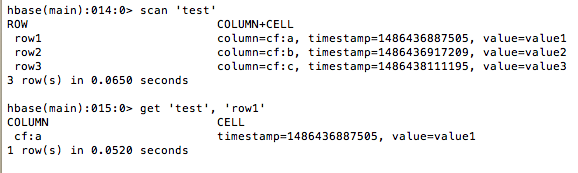
* + Confirming that I had built my hbase xml and sh correctly, I ran the start-hbase.sh.
  + I then ran jps to confirm that HMaster was running successfully
  + After doing so I ran the hbase shell and asked for the help, giving me the operating commands within the hbase command line interface.



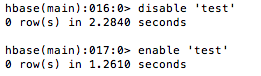
* + I then went into the Hbase system and created 3 rows:



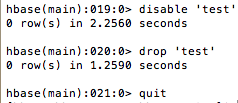
* + Next, I scanned my table called test and then performed some EDA only on the first row



* + I then disabled and enabled the ‘test’ table



* + Followed by re-disabling / dropping the table ‘test’ and quitting hbase



* + I then quit HBase and confirmed that HMaster had shut down with the jps command