Crash course

A beginner's guide

INTRODUCTION

Hbase is a distributed column-oriented database built on top of HDFS used when we require a real-time read/write random-access to to very large data set. Hadoop is built from the ground-up to scale linearly just by adding nodes. It is not relational and does not support SQL.

WEBTABLE

Canonical use of Hadoop is webtable - a table of crawled web pages and their attributes (such as language and MIME type) keyed by the web page URL. The webtable is large, with row counts that run into the billions.
Batch analytic and parsing MapReduce jobs are continuously run against the webtable deriving statistics and adding new columns for verified MIME type and parsed text content for later indexing by search engine.

DATA MODEL

Applications store data into labeled tables which are made of rows and columns. Table cells are versioned - their version is a timestamp auto-assigned by HBase at the time of cell insertion. Cell's content is an uninterpreted array of bytes table row keys are also byte arrays. Table rows are sorted by row key, the table's primary key (the sort is byte ordered).Row columns are grouped into column families and all column family members have a common prefix. Column families must be specified up front as part of the table schema definition, but new family column members can be added on demand. Physically all column family members are stored up front as part of the table schema definition.

REGIONS

Tables are automatically partitioned horizontally by HBase into regions. Initially, a table comprises a single region , but as the size grows, it splits at row boundary into two new regions of approximately equal size. Regions are the units that get distributed over an HBase cluster.

LOCKING

Row updates are atomic, no matter how many row columns constitute the row-level transaction.

IMPLEMENTATION

HBase master node is orchestrating a cluster of one or more regionserver slaves. It is responsible for bootstrapping a virgin install, for assigning regions to registered regionservers and for recovering regionserver failures. Regionserver slave nodes are listed in the HBase *conf/regionservers* file. Cluster site-specific configuration is made in the HBase *conf/hbase-site.xml* and *conf/hbase-env.sh* files.

HBASE IN OPERATION

Current list, state and location are maintained in special catalog table named –ROOT and .META. ROOT table holds the list of .META table regions. The .META table holds the list of table regions.

INSTALLATION

1. download a stable release from an Apache Download Mirror and unpack it on your local file system

```
% tar xzf hbase-x.y.z.gz
```

2. set the Java installation that HBase uses by editing HBase's *conf/hbase-env.sh* and specifying JAVA_HOME variable to get the list of HBase options type

TESTDRIVE

To administer HBase instance, launch the HBase shell by typing

```
% hbase shell
```

This will bring up a JRuby IRB interpreter that has had some HBase-specific commands added to it. You can type help and RETURN to see the list of shell commands grouped into categories.

To create table , name and schema is required

```
hbase(main):007:0> create 'test', 'data'
```

Creates table named test with a single column family name data using defaults for table and column family attributes.

To check if the new table was created successfully, run the list command

```
hbase(main):019:0> list
```

To insert data into three different rows and columns in the data column family, and then list the table content, do the following

```
hbase(main) :021:0> put 'test', 'row1', 'data:1' , 'value1'
                    0 row(s) in 0.0454 seconds
                    hbase(main):022:0> put 'test', 'row2', 'data:2', 'value2'
                    0 row(s) in 0.0035 seconds
                    hbase(main):023:0 > put 'test', 'row3' , 'data:3', 'value3'
                    0 row(s) in 0.0090 seconds
                    hbase(main):024:0> scan 'test'
```

```
ROW                    COLUMN+CELL
row1                   column=data:1, timestamp=1240148026198, value=value1
row2                   column=data:2, timestamp=1240148040025, value=value2
row3                   column=data:3, timestamp=1240148047497, value=value3
3 row(s) in 0.0825 seconds
```

You can shut down your HBase by running

```
% stop-hbase.sh
```

MAPREDUCE

HBase classes and utilities in the *org.apache.hadoop.hbase.mapreduce package* facilitate using HBase as a source and/or sink in MapReduce jobs. *TableInputFormat* class makes splits on region boundaries so maps are handed a single region to work on, *TableOutputFormat* will write the result of reduce into HBase.

Example: MapReduce application to count the number of rows in an HBase table

```
public class RowCounter {
                /** Name of this 'program' */
                static final String NAME = "rowcounter";

                static class RowCounterMapper
                extends TableMapper<ImmutableByteWritable, Result> {
                        /** Counter enumeration to count the actual rows. */
                        public static enum Counters {ROWS}

                        @Override
                        public void map(ImmutableBytesWritable row, Result values, Context context)
                        throws IOException {
                                for (KeyValue value: values.list()) {
                                        if(value.getValue().length >0) {
                                                context.getCounter(Counters.ROWS).increment(1);
                                                break;
                                        }
                                }

                        }

                }

                public static Job createSubmittableJob(Configuration conf, String[] args)
                throws IOException {
                        String tableName = args[0];
                        Job job = new Job(conf, NAME +"_"  + tableName);
                        job.setJarByClass(RowCount.class);
                        //Columns are space delimited

                        StringBuilder sb = new StringBuilder();
                        final int columnoffset = 1;
                        for(int i = columnoffset; i < args.length; i++) {
                                if(i > columnoffset) {
                                        sb.append(" ");
                                }
                                sb.append(args[i]);
                        }
```

```
                Scan scan = new Scan();
                scan.setFilter(new FirstKeyOnlyFilter());
                if(sb.length() > 0) {
                        for(String columnName: sb.toString().split(" ")) {
                                String[] fields = columnName.split(":");
                                if(fields.length == 1) {
                                        scan.addFamily(Bytes.toBytes(fields[0]));
                                } else {
                                        scan.addColumn(Bytes.toBytes(fields[0]),
                Bytes.toBytes(fields[1]));
                                }
                        }
                }
                // Second argument is the table name
                job.setOutputFormatClass(NullOutputFormat.class);
                TableMapReduceUtil.initTableMapperJob(tableName, scan, RowCounterMapper.class ,
                ImmutableBytesWritable.class,
                job);
                job.setNumberReduceTasks(0);
                return job;
        }

        public static void main(String[] args) throws Exception {
                Configuration conf = HBaseConfiguration.create();
                String[] otherArgs = new GenricOptionsParser(conf, args).getRemainingArgs();
                if(otherArgs.length < 1) {
                        System.err.println("ERROR: Wrong number of parameters: " + args.length);
                        System.err.println("Usage: RowCounter <tablename> [<column1> <column2> ...]");
                        System.exit(-1);
                }

                Job job = createSubmittableJob(conf, otherArgs);
                System.exit(job.waitForCompletetion(true) ? 0:1);


        }
```

AVRO, REST, THIRFT

Those tools are used when the interacting application is written in a language other than Java.

REST

To put up a stargate instance (stargate is the name for the HBase REST service) start it, using the following command:

```
% hbase-daemon.sh start rest
```

This will start a server instance, by default on port 8080, background it, and catch any emission by the server in logfilesunder the HBase logs directory. Clients can ask for the response to be formatted as JSON , Gooogle's protobugs, or as XML , depending on how the HTTP Accept header is set.

To stop the REST server, type:

```
% habse-daemon.sh stop rest
```

THRIFT

You can start Thirft by running the following

```
% hbase-daemon.sh start thrift
```

This will start the server instance, by default on port 9090, background it and catch any emissions by the servers logfiles under the HBase logs directory.

To stop Thrift server, type

```
% hbase-daemon.sh stop thirft
```

AVRO

The Avro server is started and stopped in the same manner as you'd start and stop the Thrift or REST services. The Avro server by default uses port 9090.

HBASE IN ACTION – EXAMPLE

Simple web interface that allows a user to navigate the different stations and page through their historical temperature observations in time order, which implies a massive dataset. HDFS and MapReduce are powerful tools for processing batch operations over large datasets, but do not provide ways to read or write individual records efficiently.

SCHEMAS

Stations - table to hold station data

- *stationid* – rowkey
- *info* - column family tha acts as key/val dictionary for station information
- the dictionary keys: *info:name*, *info:location* , *info:description*

Observations - table to hold temperature observations

- row key - a composite key of *stationid* + reverse order timestamp
- column family data will contain one column *airtemp* with the observed temperature as the column value

Defining tables in shell

```
hbase(main):036:0 > create 'stations', {Name => 'info', VERSIONS => 1}
                              0 row(s) in 0.1304 seconds
 hbase(main) :037:0 > create 'observations' , {NAME => 'data', VERSIONS => 1}
                               0  row(s) in 0.1332 seconds
```

LOADING DATA

Let's assume that there are billions of individual observations to be loaded - this kind of import is normally an extremely complex and long-running database operation, but MapReduce and HBase's distribution model allow us to make full use of the cluster.

MapReduce application to import temperature data from HDFS into an HBase table.

```
public class HBaseTemperatureMapper<K, V> extends Configured implements Tool {

        // Inner-class for map
        static class HBaseTemperatureMapper<K, V> extends MapReduceBase implements
                        Mapper<LongWritable, Text, K, V> {

                private NcdcRecordParser parser = new NcdcRecordParser();
                private HTable table;

                public void map(LongWritable key, Text value,
                                OutputCollector<K, V> output, Reporter reporter)
                                throws IOException {
                        parser.parse(value.toString());
                        if (parser.isValidTemperature()) {
                                byte[] rowKey = RowKeyConverter.makeObservationRowKey(parser
                                                .getStationId(), parser.getObservationDate().getTime());
                                Put p = new Put(rowKey);
                                p.add(HBaseTemperatureCli.DATA_COLUMNFAMILY,
                                                HBaseTemperatureCli.AIRTEMP_QUALIFIER,
                                                Bytes.toBytes(parser.getAirTemperature()));
                                table.put(p);
                        }
                }

                public void configure(JobConf jc) {
                                super.configure(jc);
                                //Create the HBase table client once up-front and keep it around
                                //rather than create on each map invocation

                                try {
                                        this.table = new HTable(new HBaseConfiguration(jc), "observations");
                                }
                                catch(IOException e) {
                                        throw new RuntimeException("Failed HTable construction", e);
                                }
                          }             @Override
                public void close() throws IOException {
                        super.close();
                        table.close();
                }
        }

        public int run(String[] args) throws IOException {
                if (args.length != 1) {
                        System.err.println("Usage: HBaseTemperatureImporter <input>");
                        return -1;
                }
                JobConf c = new JobConf(getConf(), getClass());
                FileInputeFormat.addInputPath(jc, new Path(args[0]));
                jc.setMapperClass(HBaseTemperatureMapper.class);
                jc.setNumberReduceTasks(0);
                jc.setOutputFormat(NullOutputFormat.class);
                JobClient.runJob(jc);
                return 0;
        }

        public static void main(String[] args) throws Exception {
                int exitCode = ToolRunner.run(new HbaseConfiguration(),
                                new HBaseTemperatureImporter(), args);
                System.exit(exitCode);
        }

}
```

We can run the program with the following:

```
% hbase HBaseTemperatureImporter input/ncdc/all
```

WEB QUERIES

To implement the web application in HBase Java API can be used directly. Using the *info* family as a *key/value* dictionary (column names as keys, column values as values), the code would look like this

```
public Map<String , String> getStationInfo(HTable table, String stationId)
                  throws IOException
                  {
                        Get get = new Get(Bytes.toBytes(stationId));
                        get.addColumn(INFO_COLUMNFAMILY);
                        Result res = table.get(get);
                        if(res == null) {
                                return null;
                        }

                        Map<String, String> resultMap = New HashMap<String , String>();
                        resultMap.put("name", getValue(res, INFO_COLUMNFAMILY, NAME_QUALIFIER));
                        resultMap.put("location", getValue(res, INFO_COLUMNFAMILY, LOCATION_QUALIFIERS));
                        resultMap.put("description", getValue(res, INFO_COLUMNFAMILY, DESCRIPTION_QUALIFIER));
                        return resultMap;
                  }

                  private static String getValue(Result res,byte[] cf , byte[] qualifier) {
                                byte[] value = res.getValue(cf, qualifier);
                                return value == null?"" : Bytes.toString(value);

                  }
```

Methods for retrieving a range of rows of weather station observation from an HBase table.

```
public NavigableMap<Long, Integer> getStationObservations(HTable table, String stationId,
                        long maxStamp , int maxCount) throws IOException {

                                byte[] startRow = RowKeyConverter.makeObservationRowKey(stationId, maxStamp);
                                NavigableMap<long, Integer> resultMap = new TreeMap<Long, Integer>();
                                Scan scan = new Scan(startRow);

                                scan.addColumn(DATA_COLUMNFAMILY, AIRTEMP_QUALIFIER);
                                ResultScanner scanner = table.getScanner(scan);
                                Result res  = null;

                                int count = 0;

                                try {
                                        while((res = scanner.next()) != null && count++ < maxCount) {
                                                byte[] row = res.getRow();
                                                byte[] value = res.getValue(DATA_COLUMNFAMILY,
AIRTEMP_QUALIFIER);

                                                Long stamp = Long.MAX_VALUE -
                                                        Bytes.toLong(row, row.length - Bytes.SIZEOF_LONG,
Bytes.SIZEOF_LONG);

                                                Integer temp = Bytes.toInt(value);
                                                resultMap.put(stamp, temp);
                                        }
                                finally {
                                        scanner.close();
                                }
                                        scanner.close();
                                }
                                        return resultMap;
                        }
```

```
/**
* Return the last ten observation.
 */
public NavigableMap<Long, Integer> getStationObservations(HTable table ,
String stationId) throws IOException {
        return getStationObservations(table, stationId, Long.MAX_VALUE, 10);
}
```

IMPORTING DATASET

The easiest way to import dataset from relational database into Hbase, is to export database from table to CSV file. After this is accomplished you should move CSV file into HDFS files system by using command:

```
% hadoop fs -put ./z_22.csv /shooting
```

This command will copy shooting.csv file into shooting directory on HDFS system.

Next step is to access Hbase shell and create table.

```
% hbase shell
```

Accesses hbase shell.

```
hbase(main):001:0> create 'pothole', 'archivesource', 'text', 'to_user_id',
'from_user','id','from_user_id', 'iso_language_code', 'source', 'profile_image_url',
'geo_type','geo_coordinates','created_at','time','date'
```

Creating table in Hbase: geo_coordinates *and* date *are column families with members.*

Exit Hbase.

```
hbase(main):001:0>exit
```

Import data in CSV file using ImportTsv tool.

```
%hbase  org.apache.hadoop.hbase.mapreduce.ImportTsv   ' -Dimporttsv.separator=,'
-Dimporttsv.columns=archivesource,text,to_user_id,from_user,HBASE_ROW_KEY,
from_user_id,iso_language_code,source,profile_image_url,geo_type,geo_coordinates:0,geo_coordina
tes:1,created_at,time,date:month,date:day,date:year shooting
/user/dlrlhive/shooting/shooting.csv
```

CONCLUSIONS

The purpose of this tutorial is to give a user a brief and basic introduction to HBase and its main features. For more in-depth information and instructions on how to explore HBase, please visit

https://hbase.apache.org/