HBase

HBasics

Distributed column oriented database built on top of HDFS used when real-time read/write random-access is required at large scale. Doesn't support SQL but scales linearly (adding a new node).

Concepts

Cells are timestamped by HBase at insertion. Row key are bytes arrays so anything can serve as row key (strings or binary representations).

Row columns are grouped into *column families* with a common prefix. Tables are partitioned horizontally into *regions* of configured size when it splits into equal size. Until this split happens, all loading will be against a single server.

Row updates are atomic.

Implementation

A *master* node manages many *regionservers* slaves and uses a Zookeeper server by itself.

Regionservers slaves nodes are in *conf/regionservers* much like the *conf/slaves* of Hadoop.

Persists data throught Hadoop API so it can write to KFS, Amazon's S3 and HDFS

HBase in operation

- **-ROOT-**: Holds the list of .META. table regions. Is the first thing clients learn when connected to ZooKeeper
- **.META.**: Holds the list of all user-space regions

Test Drive

```
$ start-hbase.sh
$ hbase shell
```

Will launch a loca HBase on **/tmp/hbase-${USERID}**

To **create a table** use:

```
hbase> create 'test', 'data'
```

To **list** tables use:

```
hbase> list
```

To **insert data** use:

```
hbase> put 'test', 'row1', 'data:1', 'value1'
```

To **remove the table** use:

```
hbase> disable 'test'
hbase> drop 'test'
```

Clients

Java

First gets a org.apache.hadoop.conf.Configuration instance with HBase configuration read from *hbase-site.xml* and and *hbase-default.xml* to creates **HBaseAdmin** for administering and adding and dropping tables **HTable** for access tables.

Avro, REST and Thr.ift

Are slower as a java server is parsing request to HBase.

- **REST**: **Stargate** is initialized with `hbase-daemon.sh start rest` on 8080
- **Thrift**: Is started with `hbase-daemon.sh start thrift` on 9090

- **Avro**: Is started with `hbase-daemon.sh start avro` on 9090 too.

Core features

- **No real indexes**: as row and columns are stored sequentially. Insert performance is based on table size.
- **Automatic partitioning**: As the tables grow they'll split automatically and be distributed across available nodes.
- **Scale linearly and automatically with new nodes**: Add a node, point it to the Zookeeper and regions will rebalance automatically.
- **Commodity hardware**: Nodes of 1000-5000$ nodes rather than 50000$ ones.
- **Fault tolerance**: No worries about individual node downtime
- **Batch processing**: Allow fully parallel distributed jobs

Common problems

Versions

HBase version compatibility with Hadoop is matched up to 0.90 so HBase 0.20.5 would run on an Hadoop 0.20.2 but HBase 0.19.5 would not. 0.90 will work on Hadoop 0.20.x and 0.21.x

HDFS

HBase uses HDFS in a different way that Hadoop.

- **Running out of file descriptors** is when, for example, you have 2000 files opened (100 regions x 10 columns x 2 flush files + other files) and the default limit of file descriptors per process ir 1024
- **Running out of datanode threads**: Hadoop datanode can't run more than 256 threads unless you set a higher limit in `dfs.datanode.max.xcievers`
- **Sync**: You must run HBase on an HDFS that has a working sync. Otherwise, you will lose data. This means running HBase on Hadoop 0.20.205.0 or later

UI and metrics

60010 is the default port and displays a list of basic attributes. Enabling Hadoop metrics and tying them to

Ganglia will give views of what's happening on the cluster

Schema design

HBase has versioned cells, sorted rows and capacity to add columns on the fly. These factors should be considered when designing schemas but specially how the data will be accessed. Also with column-oriented stores can host very populated tables at no incurred cost.

Joins

No native join in HBase but with wide tables there is no need

Row keys

Take time designing the row key

Counters

With **incrementColumnValue()** a counter can the incremented many thousands of times a second