ALSO FROM

DZone

Expert research and learning communities for developers, tech professionals and **smart people everywhere.**

COMMUNITIES

- Share links, write articles, and engage in dialogue with other tech experts
- Topics include Java, Mobile Development, Web Development, SQL, Big Data, Internet of Things, and many more
- Ability to promote your own blog and products through contribution

RESEARCH GUIDES

- Free, unbiased industry insight into key technology topics, trends and vendors
- In-depth articles written by industry experts
- Key findings from our survey of over 1000+ developers and experts
- Profiles and key information on solution providers
- Development checklists and infographic

REFCARDZ

- Access a library of over 200 reference cards covering the latest tech topics, programming languages, and platforms
- Written by industry experts
- Updated monthly

Join DZone today for free and gain access to all of these exclusive benefits and more

JOIN NOW

DZone Refcardz

CONTENTS INCLUDE:

- Configuration
- Start/Stop
- HBase Shell
- Java API
- Web UI: Master & Slaves
- and More!

Apache HBase

The NoSQL Database for Hadoop and Big Data

By Alex Baranau and Otis Gospodnetić

ABOUT HBASE

HBase is the Hadoop database. Think of it as a distributed, scalable Big Data store.

Use HBase when you need random, real-time read/write access to your Big Data. The goal of the HBase project is to host very large tables — billions of rows multiplied by millions of columns — on clusters built with commodity hardware. HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable. Just as Bigtable leverages the distributed data storage provided by the Google File System, HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

CONFIGURATION

OS & Other Pre-requisites

HBase uses the local hostname to self-report its IP address. Both forward- and reverse-DNS resolving should work.

HBase uses many files simultaneously. The default maximum number of allowed open-file descriptors (1024 on most *nix systems) is often insufficient. Increase this setting for any Hbase user.

The nproc setting for a user running HBase also often needs to be increased — when under a load, a low nproc setting can result in the OutOfMemoryError.

Because HBase depends on Hadoop, it bundles an instance of the Hadoop jar under its /lib directory. The bundled jar is ONLY for use in standalone mode. In the distributed mode, it is critical that the version of Hadoop on your cluster matches what is under HBase. If the versions do not match, replace the Hadoop jar in the HBase /lib directory with the Hadoop jar from your cluster.

To increase the maximum number of files HDFS DataNode can serve at one time in hadoop/conf/hdfs-site.xml, just do this:

```
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>4096</value>
</property>
```

hbase-env.sh

You can set HBase environment variables in this file.

Env Variable	Description
HBASE_HEAPSIZE	Shows the maximum amount of heap to use, in MB. Default is 1000. It is essential to give HBase as much memory as you can (avoid swapping!) to achieve good performance.
HBASE_OPTS	Shows extra Java run-time options. You can also add the following to watch for GC: export HBASE_OPTS="$HBASE_OPTS -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps $HBASE_GC_OPTS"

hbase-site.xml

Specific customizations go into this file in the following file format:

```
<configuration>
  <property>
    <name>property_name</name>
    <value>property_value</value>
  </property>
  …
</configuration>
```

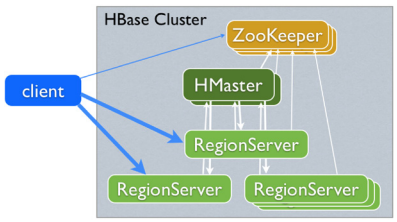
For the list of configurable properties, refer to http://hbase.apache.org/book.html#hbase_default_configurations (or view the raw /conf/hbase-default.xml source file).

These are the most important properties:

Property	Value	Description
hbase.cluster.distributed	true	Set value to true when running in distributed mode.
hbase.zookeeper.quorum	my.zk.server1,my.zk.server2,	HBase depends on a running ZooKeeper cluster. Configure using external ZK. (If not configured, internal instance of ZK is started.)
hbase.rootdir	hdfs://my.hdfs.server/hbase	The directory shared by region servers and where HBase persists. The URL should be 'fully qualified' to include the filesystem scheme.

START/STOP

Running Modes



This is a typical cluster setup.

Mode	Description
Standalone	HBase does not use HDFS it uses the local filesystem instead and it runs all HBase daemons and a local ZooKeeper in the same JVM.
Pseudo-distributed	A pseudo-distributed mode is simply a distributed mode running on a single host. Use this configuration for testing and prototyping on HBase.
Fully-distributed	Daemons are spread across all nodes in the cluster. Use this configuration for production or for evaluating HBase performance.

Hot Tip To manage all nodes from master, set up passwordless ssh.

Start/Stop Commands

from master	on every node	CDH (on every node)
bin/start-hbase.sh	bin/hbase master start bin/hbase regionserver start	service hadoop-hbase-master start service hadoop-hbase-regionserver start
bin/stop-hbase.sh	bin/hbase master stop bin/hbase regionserver stop	service hadoop-hbase-master stop service hadoop-hbase-regionserver stop

HBASE SHELL

To run the HBase shell:

```
$ ./bin/hbase shell
```

For examples of scripting HBase, look for files with the .rb extension in the HBase bin directory. To run one of these scripts, do the following:

```
$ ./bin/hbase org.jruby.Main PATH_TO_SCRIPT
```

Shell Command Example	Description
help	Show shell help
create 'mytable', {NAME => 'colfam1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}, {NAME => 'colfam2'}	Create a table
list	List all tables
disable 'mytable' drop 'mytable'	Drop a table
truncate 'mytable'	Truncate(clear) a table: disable->drop->create
alter 'mytable', {NAME => 'new_colfam'}, {NAME => 'colfam_to_delete', METHOD => 'delete'}	Alter a table / add column family (Note: table must be disabled to make ColumnFamily modifications)
scan 'mytable'	Scan all table records
scan 'mytable', {LIMIT=>10, STARTROW=>'start_row', STOPROW=>'stop_row'}	Scan 10 records in a table starting with 'start_row' and ending with 'end_row' (use double quotes to escape hex-based characters)
get 'mytable', 'row_key'	Get a record
put 'mytable', 'row_key', colfam:qual', 'value'	Add/update a record
delete 'mytable', 'row_key', 'colfam:qual'	Delete a record
major_compact 'mytable'	Run minor/major compaction
split 'mytable'	Split region(s)
status 'detailed'	Get HBase cluster-status details and statistics

JAVA API

Schema Creation

HBase schemata can be created or updated using the HBaseAdmin in the Java API, as shown below:

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);      // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);    // modifying existing ColumnFamily

admin.enableTable(table);
```

HTable

HTable manages connections to the HBase table.

```
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, "mytable");
table.setAutoFlush(false);
table.setWriteBufferSize(2 * 1024 * 1024); // 2 Mb
// … do useful stuff
table.close()
```

Put

"Put" adds new records into the HBase table.

```
HTable table = ...     // instantiate HTable
Put put = new Put(Bytes.toBytes("key1"));
put.add(Bytes.toBytes("colfam"), Bytes.toBytes("qual"),
        Bytes.toBytes("my_value"));
put.add(...);
...
table.put(put);
```

Get

"Get" fetches a single record given its key.

```
HTable table = ...     // instantiate HTable
Get get = new Get(rowKey);
get.addColumn(Bytes.toBytes("colfam"),
              Bytes.toBytes("qual")); // to fetch specific column
get.addFamily("colfam2");  // to fetch the all from column
Result result = aggTable.get(get);
```

Delete

"Delete" deletes a single record given its key.

```
HTable table = ...     // instantiate HTable
Delete toDelete = new Delete(rowKey);
table.delete(delete);
```

Scan

"Scan" searches through multiple rows iteratively for specified attributes.

```
HTable table = ...      // instantiate HTable
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("cf"),Bytes.toBytes("attr"));
scan.setStartRow( Bytes.toBytes("row"));  // start key is inclusive
scan.setStopRow( Bytes.toBytes("row" +  new byte[] {0}));  // stop key is
exclusive
ResultScanner scanner = table.getScanner(scan)
try {
  for(Result result : scanner) {
    // process Result instance
  }
} finally {
  scanner.close();
}
```

WEB UI: MASTER & SLAVES

HMaster and RegionServer web interfaces are handy for checking high-level cluster states as well as individual tables/regions and slaves details. See the Operational Management section for how to access more statistics and metrics exposed by HBase.

Port	Web-Interface	Contains
60010	HMaster web-interface default port	cluster status and statistics table's statistics, regions, information and location different tools to invoke compactions and splits on regions and whole tables various links to RegionServer's web-interface
60030	RegionServer web-interface default port	RegionServer statistics, assigned regions info, and access to logs

DATA MODEL & SCHEMA DESIGN

Data Model

Table
Applications store data into an HBase table. Tables are made of rows and columns. Table cells — the intersection of row and column coordinates — are versioned.

Cell Value
A {row, column, version} tuple precisely specifies a cell in HBase.

Versions
It is possible to have an unbounded number of cells where the row and column are the same but the cell address differs only in its version dimension. A version is specified as a long integer. The HBase version dimension is stored in decreasing order so when reading from a store file, the most recent values are found first.

Row Key
Table row keys are also byte arrays. Therefore almost anything can serve as a row key, from strings to binary representations of longs or even serialized data structures.

Rows are lexicographically sorted with the lowest order appearing first in a table. The empty byte array is used to denote both the start and end of a table's namespace. All table accesses are via the table row key — its primary key.

Columns & Column Families
Columns in HBase are grouped into column families. All column members of a column family have the same prefix. For example, the courses:history and courses:math columns are both members of the courses column family. Physically, all column family members are stored together in the filesystem. Because tuning and storage specifications are done at the column family level, it is recommended that all column family members have the same general access pattern and size characteristics.

Schema Creation & Updating

Tables are declared up front at schema-definition time using the HBase shell or Java API (see earlier sections). Column families are defined at table-creation time. It is possible to alter a table and add new column families, but the table must be disabled at altering time.

When changes are made to either tables or column families (e.g., region size, block size), these changes take effect the next time there is a major compaction and the StoreFiles get re-written.

Row-Key Design

Try to keep row keys short because they are stored with each cell in an HBase table, thus noticeably reducing row-key size results of data needed for storing HBase data. This advice also applies to column family names. Common problems of choosing between sequential row keys and randomly distributed row keys:

Some mixed-design approaches allow fast range scans while distributing data among all clusters when writing sequential (by nature) data. One of the ready-to-use solutions is here: https://github.com/sematext/HBaseWD.

Design Solution	Pros	Cons
Using sequential row keys (e.g. time-series data with row key built based on timestamp)	Makes it possible to perform fast range scans with help of setting start/stop keys on Scanner	Creates single regionserver, hot-spotting problems upon writing data (as row keys go in sequence, all records end up written into a single region at a time)
Using randomly distributed row keys (e.g. UUIDs)	Aims for fastest writing performance by distributing new records over random regions	Does not conduct fastrange scans against written data

And here is the link to access the HBase Reference Guide: http://hbase.apache.org/book.html#rowkey.designRow-key design is essential to gaining maximum performance when using HBase to store your application's data.

Column Families

Currently, HBase does not do well with anything above two or three column families per table. With that said, keep the number of column families in your schema low. Try to make do with one column family in your schemata if you can. Only introduce a second and third column family in the case where data access is usually column-scoped; i.e. you usually query no more than a single column family at one time.

You can also set TTL (in seconds) for a column family. HBase will automatically delete rows once reaching the expiration time.

Versions

The maximum number of row versions that can be stored is configured per column family (the default is 3). This is an important parameter because HBase does not overwrite row values, but rather stores different values per row by time (and qualifier). Setting the number of maximum versions to an exceedingly high level (e.g., hundreds or more) is not a good idea because that will greatly increase StoreFile size.

The minimum number of row versions to keep can also be configured per column family (the default is 0, meaning the feature is disabled). This parameter is used together with TTL and maximum row versions parameters to allow configurations such as "keep the last T minutes worth of data of at least M versions, and at most N versions." This parameter should only be set when TTL is enabled for a column family and must be less than the number of row versions.

Data Types

HBase supports a "bytes-in/bytes-out" interface via Put and Result, so anything that can be converted to an array of bytes can be stored as a value. Input can be strings, numbers, complex objects, or even images, as long as they can be rendered as bytes.

One supported data type that deserves special mention is the "counters" type. This type enables atomic increments of numbers.

MAPREDUCE

Reading from HBase

Job Configuration
The following is an example of using HBase as a MapReduce source in a read-only manner:

```
Configuration config = HBaseConfiguration.create();

config.set(                                  // speculative
  “mapred.map.tasks.speculative.execution”, // execution will
  “false”);                                  // decrease performance
                                             // or damage the data

Job job = new Job(config, “ExampleRead”);
job.setJarByClass(MyReadJob.class);   // class that contains mapper

Scan scan = new Scan();
scan.setCaching(500);        // 1 is the default in Scan,
                             // which will be bad for MapReduce jobs

scan.setCacheBlocks(false);  // don’t set to true for MR jobs

// set other scan attrs

...

TableMapReduceUtil.initTableMapperJob(
  tableName,        // input HBase table name
  scan,             // Scan instance to control CF and attribute selection
  MyMapper.class,   // mapper
  null,             // mapper output key
  null,             // mapper output value
  job);

job.setOutputFormatClass(NullOutputFormat.class); // because we
                         // aren’t emitting anything from mapper

boolean b = job.waitForCompletion(true);

if (!b) {
  throw new IOException(“error with job!”);
}
```

The mapper instance would extend TableMapper, too, like this:

```
public static class MyMapper extends TableMapper<Text, Text> {
  public void map(ImmutableBytesWritable row, Result value, Context context)
throws InterruptedException, IOException {

    // process data for the row from the Result instance.

   }
}
```

Map Tasks Number

When TableInputFormat is used (set by default with TableMapReduceUtil. initTableMapperJob(...)) to read an HBase table for input to a MapReduce job, its splitter will make a map task for each region of the table. Thus, if 100 regions are in the table, there will be 100 map tasks for the job, regardless of how many column families are selected in the Scan.
To implement a different behavior (custom splitters), see the method getSplits in TableInputFormatBase (either override in custom-splitter class or use as example).

Writing to HBase

Job Configuration

The following is an example of using HBase both as a source and as a sink with MapReduce:

```
Configuration config = ...; // configuring reading

Job job = ...;              // from HBase table

Scan scan = ...;            // is the same as in

TableMapReduceUtil          // read-only example

.initTableMapperJob(...); // above

TableMapReduceUtil.initTableReducerJob(
    targetTable,            // output table
    MyTableReducer.class,   // reducer class
    job);

job.setNumReduceTasks(1);   // at least one, adjust as required

boolean b = job.waitForCompletion(true);
```

And the reducer instance would extend TableReducer, as shown here:

```
public static class MyTableReducer extends TableReducer<Text, IntWritable,
ImmutableBytesWritable>  {
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
    ...
    Put put = ...;    // data to be written
    context.write(null, put);
    ...
  }
}
```

PERFORMANCE TUNING

Details about performance tuning could take up this whole Refcard, and even more, so this section contains the most common pointers (in addition to what's mentioned in other sections). Please refer to the HBase documentation (e.g. Apache HBase Reference) for more details.

OS

- Give HBase as much RAM as you can (hbase-env.sh).
- Use a 64-bit platform.
- Don't allow swapping.

Java

- Watch for Garbage Collector behavior (avoid long stop-the-world pauses).
- Use proven-to-work-well java versions.

HBase Configuration

Setting	Comments
hbase.regionserver.handler. count property in hbase-site.xml	Number of threads that are kept open to answer incoming requests to user tables. The rule of thumb is to keep this number low (10 by default) when the payload per request approaches the MB (big puts, scans using a large cache), and high when the payload is small (gets, small puts, ICVs, deletes).
hbase.hregion.max.filesize in hbase-site.xml	Consider going to larger regions to cut down on the total number of regions on your cluster. Generally, less Regions to manage makes for a smoother running cluster. A lower number of regions is preferred, i.e. a number in the range from 20 to the low-hundreds per RegionServer. The default 256Mb is usually too low.
compression	Consider enabling ColumnFamily compression (on CF creation). Several options are near-frictionless, and in most cases they boost performance by reducing the size of StoreFiles, thus reducing I/O as well.
splitting	One may desire to turn off automatic splitting, e.g., for better debugging purposes. To do so, increase:hbase. hregion.max.filesize (e.g. 100GB).
major compactions	A common administrative technique is to manage major compactions manually rather than letting HBase do it. By default, HConstants.MAJOR_ COMPACTION_PERIOD is one day long. Major compactions may kick in when you least desire it - especially on a busy system. To turn off automatic major compactions, set the hbase.hregion. majorcompaction property (hbase-site. xml) value to 0.

Schema Design

- Consider using Bloom Filters.
- Consider increasing column family blocksize (default is 64KB) when cell values are large to reduce StoreFile indexes.
- Consider defining in-memory column families.
- Use column family compression.

Writing to HBase

- For batch loading use the bulk-load tool, if possible. For more about this technique, go here: http://hbase.apache.org/book/arch.bulk.load.html
- Use pre-splitting of regions when bulk loading into empty HBase table (http://hbase.apache.org/book/perf.writing.html).
- Consider disabling WAL or use deferred WAL flushes.

- Make sure that setAutoFlush is set to false on your HTable instance when performing a lot of Puts.
- Watch for and avoid the RegionServer hot-spotting problem. The sign is that one RS is sweating while others are resting, i.e. an uneven load distribution while writing (hot-spotting is also mentioned in the Row-Key Design section).

Reading from HBase

- Use bigger-than-default (1) scan caching when reading a lot of records, e.g., when using the HBase table as a source for a MapReduce job. But beware that overly aggressive caching may cause timeouts—e.g., the UnknownScannerException—if processing of records is heavy/slow
- Narrow down Scan selection by defining column families and columns you need to fetch.
- Close ResultScanners in code.
- Set CacheBlocks to "false" in Scan, a source for the MapReduce job.

OPERATIONAL MANAGEMENT

Health Check

hbck
Use hbck to check the consistency of your cluster:

```
$ ./bin/hbase hbck
```

Metrics
Use metrics exposed by Hadoop and HBase to monitor the state of your cluster. As most processes are Java processes, general JVM monitoring is useful.

You can use JConsole or any other JMX client to access metrics exposed by HBase. Some metrics are also exposed via HMaster and RegionServer web interfaces.

SPM for HBase (http://sematext.com/spm/hbase-performance-monitoring/index.html) monitors all key HBase metrics, renders time-series performance graphs, includes alerts, etc.

The most important metrics to monitor are:

Group	Metric
HBase	Requests count
HBase	Compactions queue
Java	GC metrics
OS	IO Wait
OS	User CPU

Backup Options

Type	Method
Full Shutdown	Use HDFS distcp tool. Stop HBase distcp HBase dir in HDFS Start HBase pointing to copied dir.
Live	Set up replication between two running clusters.
Live	Use CopyTable utility to copy data from one table to another on the same cluster, or to copy data to another table on another cluster.
Live	Use Export tool (run as MapReduce job) to export HBase table contents to HDFS. Then use Import tool to load data into another table from the dump.

See this link—http://hbase.apache.org/book/ops.backup.html—for more about HBase backup methods.

TROUBLESHOOTING & DEBUGGING

Use the info in the Operational Management section for preemptive cluster health checks.

When bad things happen, refer to logs (usually start with HMaster logs) and/or collected metrics. This is where access to historical stats is very useful (see SPM for HBase - http://sematext.com/spm/hbase-performance-monitoring/index.html)

Logs

Process	Log Location
NameNode	$HADOOP_HOME/logs/hadoop-<user>-namenode-<hostname>.log
DataNode	$HADOOP_HOME/logs/hadoop-<user>-datanode-<hostname>.log
JobTracker	$HADOOP_HOME/logs/hadoop-<user>-jobtracker-<hostname>.log
TaskTracker	$HADOOP_HOME/logs/hadoop-<user>-jobtracker-<hostname>.log
HMaster	$HBASE_HOME/logs/hbase-<user>-master-<hostname>.log
RegionServer	$HBASE_HOME/logs/hbase-<user>-regionserver-<hostname>.log
ZooKeeper	/var/log/zookeeper/zookeeper.log

To output GC logs (important for debugging RegionServer failures), uncomment (or add) the following line in the hbase-env.sh file:

```
export HBASE_OPTS="$HBASE_OPTS -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:$HBASE_HOME/logs/gc-hbase.log"
```

Resources

http://search-hadoop.com/
search-hadoop.com indexes all mailing lists, wikis, JIRA issues, sources, code, javadocs, etc. Search here first when you have an issue. More than likely someone has already had your problem.

Mailing Lists
Ask a question on the HBase mailing lists. The 'dev' mailing list is aimed at the community of developers actually building HBase, and it is also for features currently under development. The 'user' list is generally used for questions about released versions of HBase. Before going to the mailing list, first make sure your question has not already been answered by searching the mailing list archives.

IRC
#hbase on irc.freenode.net

JIRA
JIRA is also really helpful when looking for Hadoop/HBase-specific issues at https://issues.apache.org/jira/browse/HBASE.

NON-JAVA APIS

Languages talking to JVM
The HBase wiki (see links in the end) contains examples of how to access HBase using the following languages:

- Jython
- Groovy
- Scala

Languages with a custom protocol
HBase can also be accessed using the following protocols:

- REST
- Thrift

USEFUL LINKS & OTHER SOURCES

HBase project: http://hbase.apache.org

HBase Wiki: http://wiki.apache.org/hadoop/Hbase

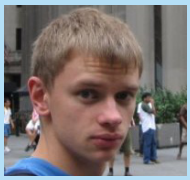
Apache HBase Reference Guide: http://hbase.apache.org/book.html

Search for Hadoop Ecosystem projects (wiki, mailing lists, source code and more): http://search-hadoop.com

Monitoring for HBase: http://sematext.com/spm/hbase-performance-monitoring

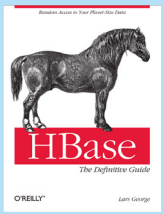
ABOUT THE AUTHORS

Otis Gospodnetić (@otisg) is a coauthor of Lucene in Action (1st and 2nd edition). He has been involved with Lucene since 2000 and Solr since 2006. He is also a member of Nutch, and Mahout development teams, as well as Lucene Project Management Committee. Otis is an Apache Software Foundation member and the founder of Sematext, a products and services company focused on Search & Big Data Analytics using Solr, ElasticSearch, Lucene, Hadoop, HBase, Flume, Mahout, and other open-source technologies to serve customers world-wide.



Alex Baranau (@abaranau) is a Software Engineer at Sematext. For the last few years Alex has been working on complex data analytics-focused projects that utilize Hadoop, HBase, and Flume. During that time Alex contributed to HBase and Flume, and has created several open-sourced projects: HBaseWD and HBaseHUT. He writes articles covering Hadoop, HBase, and related technologies at Sematext Blog.

RECOMMENDED BOOK



HBase: The Definitive Guide

If you're looking for a scalable storage solution to accommodate a virtually endless amount of data, this book shows you how Apache HBase can fulfill your needs. As the open source implementation of Google's BigTable architecture, HBase scales to billions of rows and millions of columns, while ensuring that write and read performance remain constant. Many IT executives are asking pointed questions about HBase. This book provides meaningful answers, whether you're evaluating this non-relational database or planning to put it into practice right away. http://shop.oreilly.com/

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-53-0
ISBN-10: 1-936502-53-4
50795
9 781936 502530
$7.95

Version 1.0