

# Hadoop HBase Performance Evaluation

Kareem Dana  
CPS 212 Project

## Introduction

This project had several goals:

- 1) Install and configure Hadoop and HBase on a working cluster of nodes.
- 2) Study the Hadoop/HBase API and write several HBase test programs to demonstrate functionality.
- 3) Write several HBase programs to test the performance of HBase under a variety of conditions not tested by their own Performance Evaluation test.

The next sections describe the results of the test programs. Installing Hadoop was fairly straight forward. I ran into several problems trying to run the built in performance evaluation test, though. The first problem was a Java class not found exception that was caused by a misconfiguration on my part. However, the performance evaluation test would always fail complaining about a corrupt DFS. This appears to be a known bug either in HBase or the underlying DFS implementation. I have not studied this bug further so it remains unsolved. It may have crept up in a few of my own test programs in a few instances, but for the most part my own tests ran well.

My cluster consisted of six identical Linux nodes. One node acted as the master DFS node and master HBase Server. The other five nodes acted as DFS data nodes. I ran each test with one region server and then again with five region servers to see how well they scale. HBase performed adequately for the most part under my various test conditions. In some instances it scaled poorly and overall performance is still several orders of magnitude worse than BigTable. HBase works well in most situations, especially if it is not pushed to its limits, but it is still a work in progress.

# 1. Column Family Test

Experiment	# of column families (1 Region Server)				Experiment	# of column families (5 Region Servers)			
	10	100	500	1000		10	100	500	1000
Setup Time (sec)	12.3	19.2	45.9	Timeout	Setup Time (sec)	12.2	18.7	46.4	Timeout
Writes/Sec	164	323	419	Timeout	Writes/Sec	29	153	376	Timeout
Reads/Sec	99	139	122	Timeout	Reads/Sec	119	111	120	Timeout

Tables 1 & 2: Number of 1000-byte values written to and read from a single row with increasing number of column families.

## Test Description

The goal of this experiment was to test the question “How does increasing the number of column families on a row affect performance?”. The test worked by creating a table with the specified number of column families. It then wrote a randomized 1000-byte value to each column family. Finally, 5000 reads were performed on random column families to measure read performance as column family size increased. It should be noted that the BigTable paper mentions that the design of BigTable (and therefore HBase) assume that the number of column families will be “in the hundreds at most”. So it is clear that HBase was not designed to handle a very large number of column families.

## Test Analysis

The results in Table 1 show that the major bottleneck is the setup time. That is the time it takes to create the table with all the column families. In fact, trying to create a table with 1000 column families caused the HBase RPC to timeout before it could complete. Once the table was created though, read and write performance were not significantly affected by the number of column families. HBase does not scale well to a large number of column families, but since it is a one-time cost it can be amortized among all the operations.

HBase has an API to allow column families to be added on the fly after the table was created. However, to do this the table must be taken off line first. I attempted this test using that method to create the column families. The setup time remained constant but the writes per second took a huge performance hit. Creating just 5 column families on the fly slowed the writes down so it took 25 seconds per write. This is due to the huge cost of taking a table off line and then adding the column family with each write.

Table 2 shows that setup and read time were unaffected by more region servers. This is to be expected since we are reading just one row and cannot see an improvement with extra servers. The write speed did not scale well. I am unsure why it decreased though.

## 2. Column Test

	# of columns (1 Region Server)		
Experiment	10,000	100,000	1,000,000
Writes/Sec	547	403	Crash
Reads/Sec	138	32	Crash

	# of columns (5 Region Servers)		
Experiment	10,000	100,000	1,000,000
Writes/Sec	470	435	Crash
Reads/Sec	96	18	Crash

Tables 3 & 4: Number of 1000-byte values written to and read from a single row with increasing number of columns.

### Test Description

The BigTable paper claimed that BigTable can handle an unbounded number of columns. This test was designed to test that claim within HBase. The test worked by creating a table with a single column family and then writing out one 1000-byte value to a different column within that column family for the specified number of columns. It then randomly read back 5000 column values to test read performance on rows with a lot of columns.

### Test Analysis

Table 3 shows that HBase does not scale well as the number of columns increases to very large numbers. Write performance suffers somewhat but read performance suffers a lot. This is probably because as the number of columns increases, the reads have a higher chance of having to fetch that row from disk instead of in memory. Writing 1,000,000 columns exposed an HBase bug that caused the test program to crash. The exception was:

```
Crash: Exception e: org.apache.hadoop.hbase.WrongRegionException:  
org.apache.hadoop.hbase.WrongRegionException: Requested row out of range for HRegion  
test,row_1,658683475067880027, startKey='row_1', endKey='row_1', row='row_1'
```

This is odd because row\_1 is not out of range. I was unable to debug this problem any further.

Since this test just writes a single row, additional region servers cannot help speed it up. Table 4 confirms this. In fact the additional region servers just add some overhead to the whole process.

### 3. Sort Test

	# of rows (1 Region Server)		
Experiment	10,000	100,000	1,000,000
Lexicographic	485	432	334
Reverse	451	477	354
Random	462	421	334

	# of rows (5 Region Servers)		
Experiment	10,000	100,000	1,000,000
Lexicographic	488	440	346
Reverse	522	387	343
Random	468	441	370

Tables 5 & 6: Number of 1000-byte values written per second with row keys in lexicographic, reverse lexicographic, and random order.

#### Test Description

HBase stores rows sorted lexicographically by row key. The motivation of this test is to determine if performance changes if row keys are written in reverse lexicographic order or randomly. The test works by writing the specified number of rows dynamically generating a row key with each write. For the lexicographic test the row keys are generated as “aaaaa”, “aaaab”, “aaaac”, ... This is a baseline test. The reverse lexicographic test generates row keys “zzzzz”, “zzzzzy”, “zzzzzx”, .... The random test randomly generates a row key from the same set.

#### Test Analysis

The results show that there is no significant performance loss when rows are inputted in reverse lexicographic order or random order. In fact, the reverse test performed slightly better with 100,000 rows.

While running these tests, I was able to corrupt the HBase table a few times. Trying to drop the test table caused the following error:

```
Hbase> drop Table test;  
Dropping test... Please wait.  
info:regioninfo not found
```

The table would not drop and the entire DFS had to be reformatted. This problem only occurred intermittently and eventually I was able to complete all the tests. It does expose, though, a possible corruption bug within HBase.

The writing speed does not increase even if more region servers are added. This tells me that there is a bottleneck of ~500 writes/sec caused by something else within the cluster. Most probably this bottleneck is I/O time or network bandwidth back to the instance of the test program. It is worth investigating more in the future.

## 4. Interspersed Read/Write Test

	# of writes (1 Region Server)		
Experiment	10,000	100,000	1,000,000
Reads/Sec	95	42	24

	# of writes (5 Region Servers)		
Experiment	10,000	100,000	1,000,000
Reads/Sec	95	34	35

Tables 7 & 8: Number of 1000-byte values read per second on increasingly large tables.

### Test Description

The goal of this test is to determine the performance hit when reading from very large tables. The test works by writing a specified number of rows (10,000 or 100,000 or 1,000,000) and then reading back 5000 of them randomly and averaging the elapsed time.

### Test Analysis

Reads slow down as the number of rows written increases. This means that HBase will not scale that well on very large amounts of data per table. There are several reasons this could be. First of all, if the table is very large odds are high that an arbitrary read will have to hit the disk instead of memory. Secondly scanning data structures will become more costly as their size increases.

Again Table 8 shows no significant change with additional region servers. I now believe that these tests really just stress one region server. Differently designed tests would be needed to really test performance gain from additional region servers. New tests could include running these current tests multiple times simultaneously or reading and writing multiple tables simultaneously.