Parameters

## Experiment 1:

Runtime params that were common for all the steps in this experiment:

onetime -load -db hbase.HbaseClient -P /path/populatedDB

params that changed with each step

1. –p threadcount=1
2. –p threadcount=10
3. –p threadcount=100
4. –p threadcount=1 –p
5. –p threadcount=1 -p insertimage=true -p imagesize=12
6. –p threadcount=10 -p insertimage=true -p imagesize=12
7. –p threadcount=100 -p insertimage=true -p imagesize=12
8. –p threadcount=1 -p insertimage=true -p imagesize=500
9. –p threadcount=10 -p insertimage=true -p imagesize=500
10. –p threadcount=100 -p insertimage=true -p imagesize=500

## Experiment 2:

Runtime params that were common for all the steps in this experiment:

onetime -t -s -db hbase.HbaseClient -P /path/viewProfileAction –p maxexecutiontime=180 –p initapproach=deterministic –p usercount=10000 –p resourcecountperuser=10 –p friendcountperuser=4 –p confperc=1 –p numloadthreads=10 –p useroffset=0

params that changed with each step

1. –p exportfile=/path/exp2-0-1 –p threadcount=1
2. –p exportfile=/path/exp2-0-10 –p threadcount=10
3. –p exportfile=/path/exp2-0-100 –p threadcount=100
4. –p exportfile=/path/exp2-12-1 –p threadcount=1
5. –p exportfile=/path/exp2-12-10 –p threadcount=10
6. –p exportfile=/path/exp2-12-100 –p threadcount=100
7. –p exportfile=/path/exp2-500-1 –p threadcount=1
8. –p exportfile=/path/exp2-500-10 –p threadcount=10
9. –p exportfile=/path/exp2-500-100 –p threadcount=100

## Experiment 3:

For each of the following steps, the friendcountperuser was changed in the populateDB file.

Runtime params that were common for running the listFriendsworkload:

onetime -t -s -db hbase.HbaseClient -P /path/ ListFriendsAction –p threadcount=10 –p maxexecutiontime=180 –p initapproach=deterministic –p usercount=10000 –p friendcountperuser=4 –p confperc=1 –p numloadthreads=10 –p useroffset=0 –p resourcecountperuser=10

params that changed with each step

1. –p exportfile=/path/exp3-10-1 –p threadcount=1 –p friendcountperuser=10
2. –p exportfile=/path/exp3-10-10 –p threadcount=10 –p friendcountperuser=10
3. –p exportfile=/path/exp3-10-100 –p threadcount=10 –p friendcountperuser=10
4. –p exportfile=/path/exp3-20-1 –p threadcount=1 –p friendcountperuser=20
5. –p exportfile=/path/exp3-20-10 –p threadcount=10 –p friendcountperuser=20
6. –p exportfile=/path/exp3-20-100 –p threadcount=100 –p friendcountperuser=20
7. –p exportfile=/path/exp3-50-1 –p threadcount=1 –p friendcountperuser=50
8. –p exportfile=/path/exp3-50-10 –p threadcount=10 –p friendcountperuser=50
9. –p exportfile=/path/exp3-50-100 –p threadcount=100 –p friendcountperuser=50
10. –p exportfile=/path/exp3-12-10-1 –p threadcount=1 –p friendcountperuser=10
11. –p exportfile=/path/exp3-12-10-10 –p threadcount=10 –p friendcountperuser=10
12. –p exportfile=/path/exp3-12-10-100 –p threadcount=10 –p friendcountperuser=10
13. –p exportfile=/path/exp3-12-20-1 –p threadcount=1 –p friendcountperuser=20
14. –p exportfile=/path/exp3-12-20-10 –p threadcount=10 –p friendcountperuser=20
15. –p exportfile=/path/exp3-12-20-100 –p threadcount=100 –p friendcountperuser=20
16. –p exportfile=/path/exp3-12-50-1 –p threadcount=1 –p friendcountperuser=50
17. –p exportfile=/path/exp3-12-50-10 –p threadcount=10 –p friendcountperuser=50
18. –p exportfile=/path/exp3-12-50-100 –p threadcount=100 –p friendcountperuser=50

## Experiment 4

Runtime params that were common for all the steps in this experiment:

onetime -t -s -db hbase.HbaseClient–p threadcount=10 –p maxexecutiontime=180 –p initapproach=deterministic –p usercount=10000 –p resourcecountperuser=10 –p friendcountperuser=4 –p confperc=1 –p numloadthreads=10 –p useroffset=0

params that changed with each step

1. -P /path/SymmetricVeryLowUpdateActions –p threadcount=1 –p exportfile=exp4-1-1
2. -P /path/SymmetricVeryLowUpdateActions –p threadcount=10 –p exportfile=exp4-1-10
3. -P /path/SymmetricVeryLowUpdateActions –p threadcount=100 –p exportfile=exp4-1-100
4. -P /path/SymmetricLowUpdateActions –p threadcount=1 –p exportfile=exp4-2-1
5. -P /path/SymmetricLowUpdateActions –p threadcount=10 –p exportfile=exp4-2-10
6. -P /path/SymmetricLowUpdateActions –p threadcount=100 –p exportfile=exp4-2-100
7. -P /path/SymmetricLowUpdateActions –p threadcount=1 –p exportfile=exp4-3-1
8. -P /path/SymmetricHighUpdateActions –p threadcount=10 –p exportfile=exp4-3-10
9. -P /path/SymmetricHighUpdateActions –p threadcount=100 –p exportfile=exp4-3-100

Analysis:

## Experiment 1:

In experiment 1, the goal was to understand the effect of the number of client threads (threadcount) and image size on the load time. It was observed that as the image size is increased the average load time increases. Keeping the image size constant, the threadcount was varied to three values, 1, 10 and 100. Irrespective of the image size, the observed trend was the same when the thread count was varied i.e. the highest load time was experience while using 1 thread, and the lowest load time was experienced using 10 threads. While using the 100 threads, the load time was a little higher than that of 10 threads.

The increase in load time with the increase in image size can be attributed to more disc IO. As image sizes increase, more data needs to be written and hence more IOs need to be made.

The increase in load time when thread count is exceedingly high i.e. 100 is due to contention on locks. On the other hand, the load time is high when there is only one thread because all cores are not being fully utilized.

## Experiment 2

The objective of this experiment was to study how image size and number of client threads affect throughput for the ViewProfileAction. It was observed that increasing the image size resulted in poor performance. Adding 12kb images resulted in reduction in throughput by half. Increasing the size to 500kb reduced it even further by almost 95%.

Keeping the image size constant and varying the threadcount, the throughput was low when only one thread was used and higher when more thread were used. The difference between throughput using 10 threads and that of 100 threads was not significant, with the later being lesser than the former.

The lower throughput as a result of image size can be attributed to more IOs as size of the record increases and fewer records can fit on a disc page. The low throughput when only one thread is used can be again attributed to low utilization of cores.

## Experiment 3

This objective of this experiment was to understand the impact of number of threads (threadcount), image size and the number of friends per user on the throughput of the ListFriendsAction. It was observed that with the introduction of images the throughput was adversely affected. Keeping the image size constant, and varying the number of friends from 10 to 20 and then 50, the average throughput decreased. As for threadcount, the throughput was lower when only one thread was used but was almost the same for when 10 or 100 threads were used.

The low throughput when only one thread is used can be again attributed to low utilization of cores. The lower throughput as a result of image size can be attributed to more IOs as size of the record increases and fewer records can fit on a disc page.

Overall, the throughput of this action was very low. The reason for this low performance is that in order to achieve this action, a linear scan of the table is required.

## Experiment 4

The goal of this experiment was to determine the performance of the systems when writes are introduced. The overall trend observed here was that throughput was not severely affected when the percentage of writes increased from very low to high update actions. The trend when the thread count was varied was similar. The lowest throughput was observed for 1 thread. The throughput for 100 threads was higher and much closer to that of 10 threads, which observed the highest throughput.

No stale data was observed. The reason for this is that Hbase provides row level locking.

War stories

During the load phase while using 500kb images, the computer ran out of disk space. In order circumvent that, a disk clean up was conducted.

During the benchmark phase, an exception was thrown indicating that heap space has run out. In order to circumvent that, additional parameters to the java VM was added.

Also during all the phases, the CPU utilization went up to 95%. All other applications were closed in order to reduce the load.

Data Store Benchmarking Survey

How many man-hours did you spend optimizing your schema?

0

How many man-hours did you spend optimizing your actions?

0

How many hours did you spend understanding BG’s workloads?

1

How many man-hours did you spend understanding the homework deliverables?

1

How many hours did you spend understanding and setting up BG’s experiment parameters?

1

How many hours did you spend testing and debugging your code?

1

How many hours did you spend understanding the output of BG’s benchmarking phase?

1

How many hours did you spend analyzing your results?

2

What were the resources (e.g. bgbenchmark.org, in-class tutorial, BG paper, BG slides, google  forum, TA’s help) you used for doing this part of the homework and which one was more useful?

Bgbenchmark.org and forum.