Get&Scan against different BlockCache Deploys

Constant sized rows

For HBASE-11323 Bucket Cache all the time!

Short report on test runs comparing LruBlockCache to onheap/offheap BucketCache when dataset fits in cache, ~half fits in cache, and when only a small subset fits cache. Rows are all of the same size!

1 Preamble

Preloaded a table with 1B 1k rows (200odd regions). We then run a *randomRead* test and a *scanRange1000* row test from PerformanceEvaluation (see its usage for description). For each test type (*randomRead* and *scanRange1000*), we do a run where accesses are all in-cache, mostly in-cache, and then mostly cache misses. In each test run, we first do a short warm up run and then the actual test run. We do one test run only (just looking for rough numbers here).

We run ten concurrent clients on one machine against single regionserver with all regions up on it. Under HBase is an HDFS running on five nodes.

See below for more detail on setup and test script.

2 Takeaway

LruBucketCache has lower latency and uses less CPU in all cases (all-in-cache ~20% better, mostly-in-cache ~7% better, mostly-out-of-cache about the same). Bucket cache does almost half the GC and has slightly better 95th and 99th percentiles when cache misses. Offheap performs slightly better than onheap bucket cache.

Need to redo with different sized rows.

3 Comparison

PE emits histograms per client on latencies. Those for randomRead are extracted below.

3.1 All-in-cache randomRead

LruBlockCache performs best. Better average and 99thpercentile. Test completed almost 20% faster.

3.1.1 LruBlockCache

```
…
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest Min      210.0
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest Avg      382.2252723934864
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest StdDev   423.16364579200695
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest 50th     350.0
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest 95th     520.0
14:19:08,016  hbase.PerformanceEvaluation: RandomReadTest 99th     764.0

...
2014-06-11 14:19:08,120 INFO  [main] hbase.PerformanceEvaluation: [RandomReadTest]
Min: 77356ms    Max: 81706ms    Avg: 80098ms
```

3.1.2 BlockCache offheap randomRead

```
...
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest Min      257.0
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest Avg      449.00544071716376
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest StdDev   2258.4520566094207
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest 50th     399.0
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest 95th     570.0
15:34:04,417  hbase.PerformanceEvaluation: RandomReadTest 99th     936.0
...
2014-06-11 15:34:04,522 INFO  [main] hbase.PerformanceEvaluation: [RandomReadTest]
Min: 90285ms    Max: 95693ms    Avg: 93526ms
```

3.1.3 BlockCache onheap randomRead

```
…
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest Min      263.0
```

```
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest Avg      483.9708270748397
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest StdDev   935.1543411219835
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest 50th     410.0
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest 95th     579.0
16:54:25,175  hbase.PerformanceEvaluation: RandomReadTest 99th     958.8399999999965
...
2014-06-11 16:54:25,286 INFO  [main] hbase.PerformanceEvaluation: [RandomReadTest]
Min: 99260ms    Max: 103043ms   Avg: 101496ms
```

3.2 Part-cache

LruBlockCache performs best, ~7%. Better average by about the same. Better 99thpercentile too.

3.2.1 LruBlockCache

```
...
14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest Min      221.0
14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest Avg      578.898754119873
14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest StdDev   1339.4837212737773
014-06-11 14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest 50th     416.0
14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest 95th     1128.0
14:29:18,970 hbase.PerformanceEvaluation: RandomReadTest 99th     1478.0
...
14:29:19,088 hbase.PerformanceEvaluation: [RandomReadTest]      Min: 299570ms   Max:
307129ms   Avg: 303347ms
```

3.2.2 BlockCache offheap

```
...
15:46:14,478 hbase.PerformanceEvaluation: RandomReadTest Min      243.0
15:46:14,479 hbase.PerformanceEvaluation: RandomReadTest Avg      618.1643657684326
15:46:14,479 hbase.PerformanceEvaluation: RandomReadTest StdDev   797.147055017307
15:46:14,479 hbase.PerformanceEvaluation: RandomReadTest 50th     467.0
15:46:14,479 hbase.PerformanceEvaluation: RandomReadTest 95th     1194.0
15:46:14,479 hbase.PerformanceEvaluation: RandomReadTest 99th     1615.0
...
15:46:14,591 hbase.PerformanceEvaluation: [RandomReadTest]      Min: 313086ms   Max:
327715ms   Avg: 322387ms
```

3.2.3 BlockCache onheap

```
...
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest Min      266.0
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest Avg      655.0100002288818
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest StdDev   1210.434867713139
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest 50th     475.0
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest 95th     1202.0
17:05:16,173 hbase.PerformanceEvaluation: RandomReadTest 99th     1617.0
...
17:05:16,277 INFO  [main] hbase.PerformanceEvaluation: [RandomReadTest]      Min:
325534ms   Max: 347074ms   Avg: 341593ms
```

3.3 Missing-cache

Don’t have numbers here. Just the pictures below.

4 Pictures

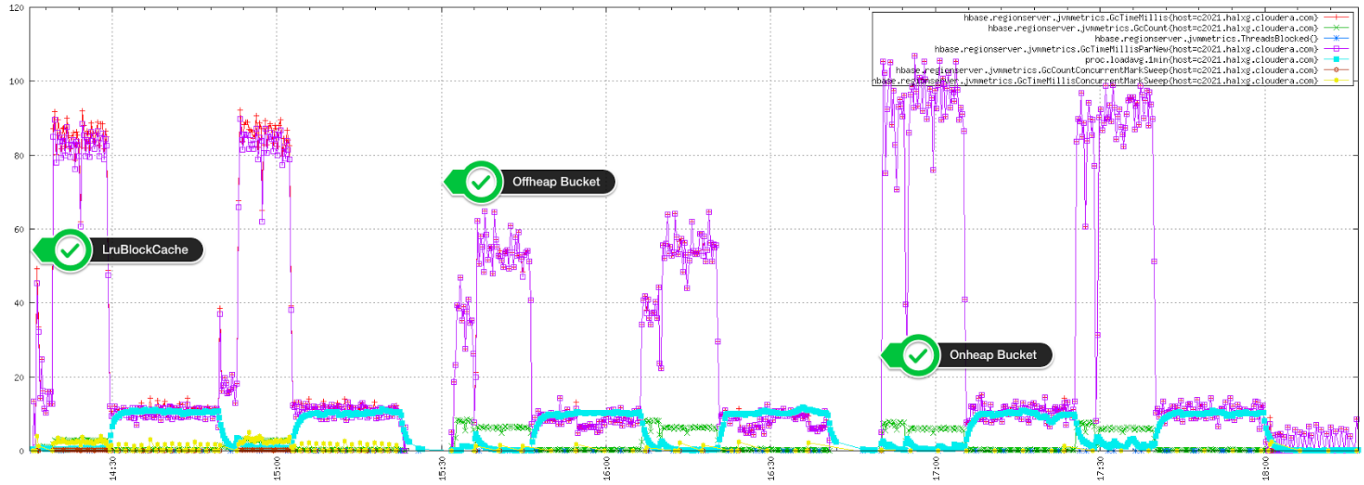
Use the below to figure what is running when in the attached diagrams. The diagrams span our three deploy types, i.e.: Pure LruBlockCache, BucketCache offheap, and then BucketCache onheap. When size is 2.8, we are hitting cache all the time. When 5.6. less so and then when 100 hardly at all. All diagrams are for the same time period so the humps all line up.

```
lrublockcache:Wed Jun 11 14:16:11 PDT 2014 run randomRead warmup time=300 size=2.8
lrublockcache:Wed Jun 11 14:17:43 PDT 2014 run randomRead time=900 size=2.8
lrublockcache:Wed Jun 11 14:19:08 PDT 2014 run randomRead warmup time=300 size=5.6
lrublockcache:Wed Jun 11 14:24:08 PDT 2014 run randomRead time=900 size=5.6
lrublockcache:Wed Jun 11 14:29:19 PDT 2014 run randomRead warmup time=300 size=100
lrublockcache:Wed Jun 11 14:34:19 PDT 2014 run randomRead time=900 size=100
lrublockcache:Wed Jun 11 14:49:20 PDT 2014 run scanRange1000 warmup time=300 size=2.8
lrublockcache:Wed Jun 11 14:50:53 PDT 2014 run scanRange1000 time=900 size=2.8
lrublockcache:Wed Jun 11 14:52:18 PDT 2014 run scanRange1000 warmup time=300 size=5.6
lrublockcache:Wed Jun 11 14:57:18 PDT 2014 run scanRange1000 time=900 size=5.6
lrublockcache:Wed Jun 11 15:02:33 PDT 2014 run scanRange1000 warmup time=300 size=100
lrublockcache:Wed Jun 11 15:07:33 PDT 2014 run scanRange1000 time=900 size=100

bucketcache.offheap:Wed Jun 11 15:26:06 PDT 2014 run randomRead warmup time=300 size=2.8
bucketcache.offheap:Wed Jun 11 15:32:25 PDT 2014 run randomRead warmup time=300 size=2.8
bucketcache.offheap:Wed Jun 11 15:34:04 PDT 2014 run randomRead time=900 size=2.8
bucketcache.offheap:Wed Jun 11 15:35:43 PDT 2014 run randomRead warmup time=300 size=5.6
bucketcache.offheap:Wed Jun 11 15:40:43 PDT 2014 run randomRead time=900 size=5.6
bucketcache.offheap:Wed Jun 11 15:46:14 PDT 2014 run randomRead warmup time=300 size=100
bucketcache.offheap:Wed Jun 11 15:51:15 PDT 2014 run randomRead time=900 size=100
bucketcache.offheap:Wed Jun 11 16:06:15 PDT 2014 run scanRange1000 warmup time=300
size=2.8
bucketcache.offheap:Wed Jun 11 16:08:03 PDT 2014 run scanRange1000 time=900 size=2.8
bucketcache.offheap:Wed Jun 11 16:09:43 PDT 2014 run scanRange1000 warmup time=300
size=5.6
bucketcache.offheap:Wed Jun 11 16:14:43 PDT 2014 run scanRange1000 time=900 size=5.6
bucketcache.offheap:Wed Jun 11 16:20:18 PDT 2014 run scanRange1000 warmup time=300
size=100
bucketcache.offheap:Wed Jun 11 16:25:18 PDT 2014 run scanRange1000 time=900 size=100

bucketcache.onheap:Wed Jun 11 16:48:26 PDT 2014 run randomRead warmup time=300 size=2.8
bucketcache.onheap:Wed Jun 11 16:50:52 PDT 2014 run randomRead warmup time=300 size=2.8
bucketcache.onheap:Wed Jun 11 16:52:39 PDT 2014 run randomRead time=900 size=2.8
bucketcache.onheap:Wed Jun 11 16:54:25 PDT 2014 run randomRead warmup time=300 size=5.6
bucketcache.onheap:Wed Jun 11 16:59:25 PDT 2014 run randomRead time=900 size=5.6
bucketcache.onheap:Wed Jun 11 17:05:16 PDT 2014 run randomRead warmup time=300 size=100
bucketcache.onheap:Wed Jun 11 17:10:16 PDT 2014 run randomRead time=900 size=100
bucketcache.onheap:Wed Jun 11 17:25:17 PDT 2014 run scanRange1000 warmup time=300
size=2.8
bucketcache.onheap:Wed Jun 11 17:27:11 PDT 2014 run scanRange1000 time=900 size=2.8
bucketcache.onheap:Wed Jun 11 17:28:58 PDT 2014 run scanRange1000 warmup time=300
size=5.6
bucketcache.onheap:Wed Jun 11 17:33:59 PDT 2014 run scanRange1000 time=900 size=5.6
bucketcache.onheap:Wed Jun 11 17:39:47 PDT 2014 run scanRange1000 warmup time=300
size=100
bucketcache.onheap:Wed Jun 11 17:44:47 PDT 2014 run scanRange1000 time=900 size=100
```

4.1 GC

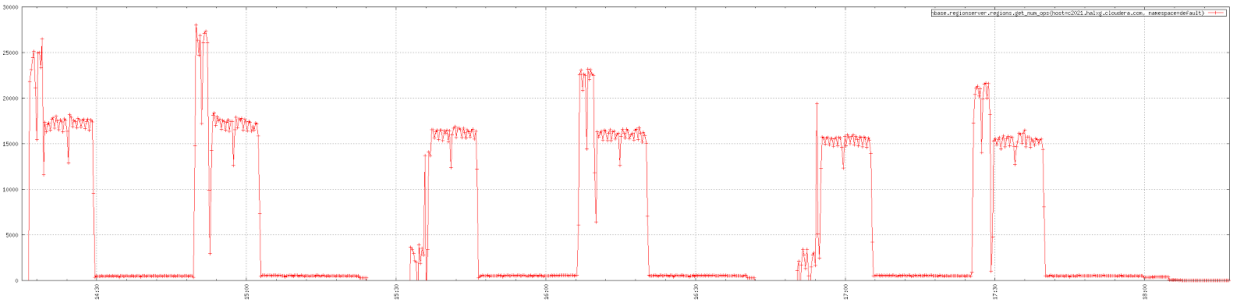
Zoom in to read the legend. The green markers show where each test run starts.

CPU load is about same across the three test. The BucketCache does less GC in both cases (offheap seems better than onheap).



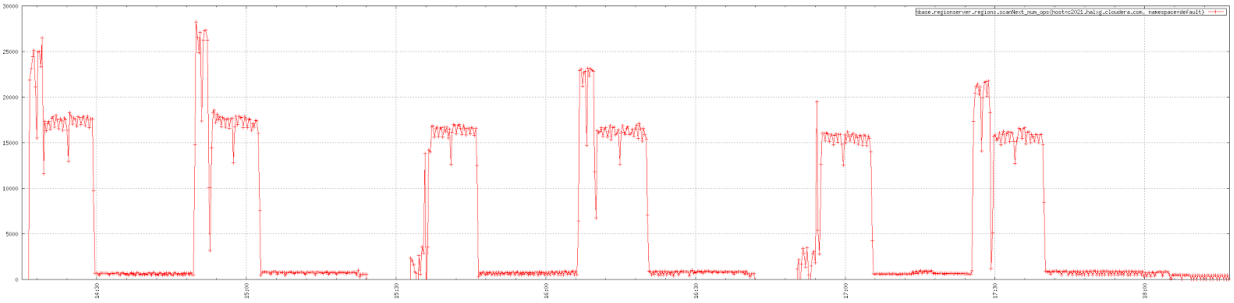
4.2 Number of Gets

LruBlockCache does more Gets especially when hitting the cache. Offheap bucket cache may do more gets than onheap, slightly.



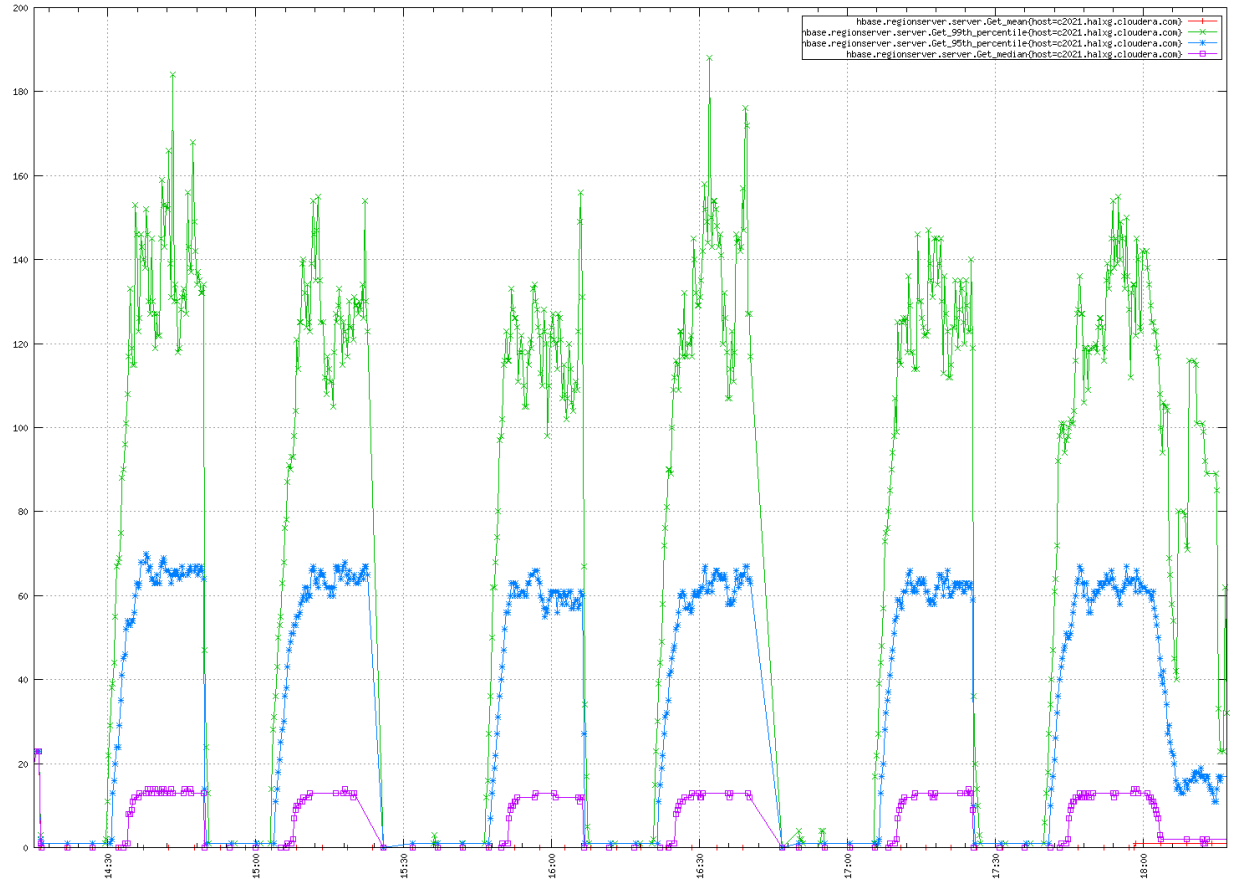
4.3 Number of scan nexts

Same profile as for gets



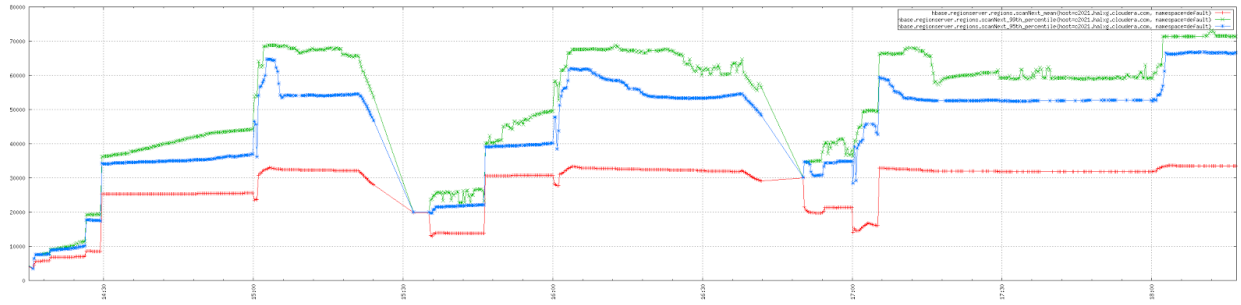
4.4 Get histogram

Bucket cache offheap seems slightly better than onheap which is slightly better than LruBlockCache.



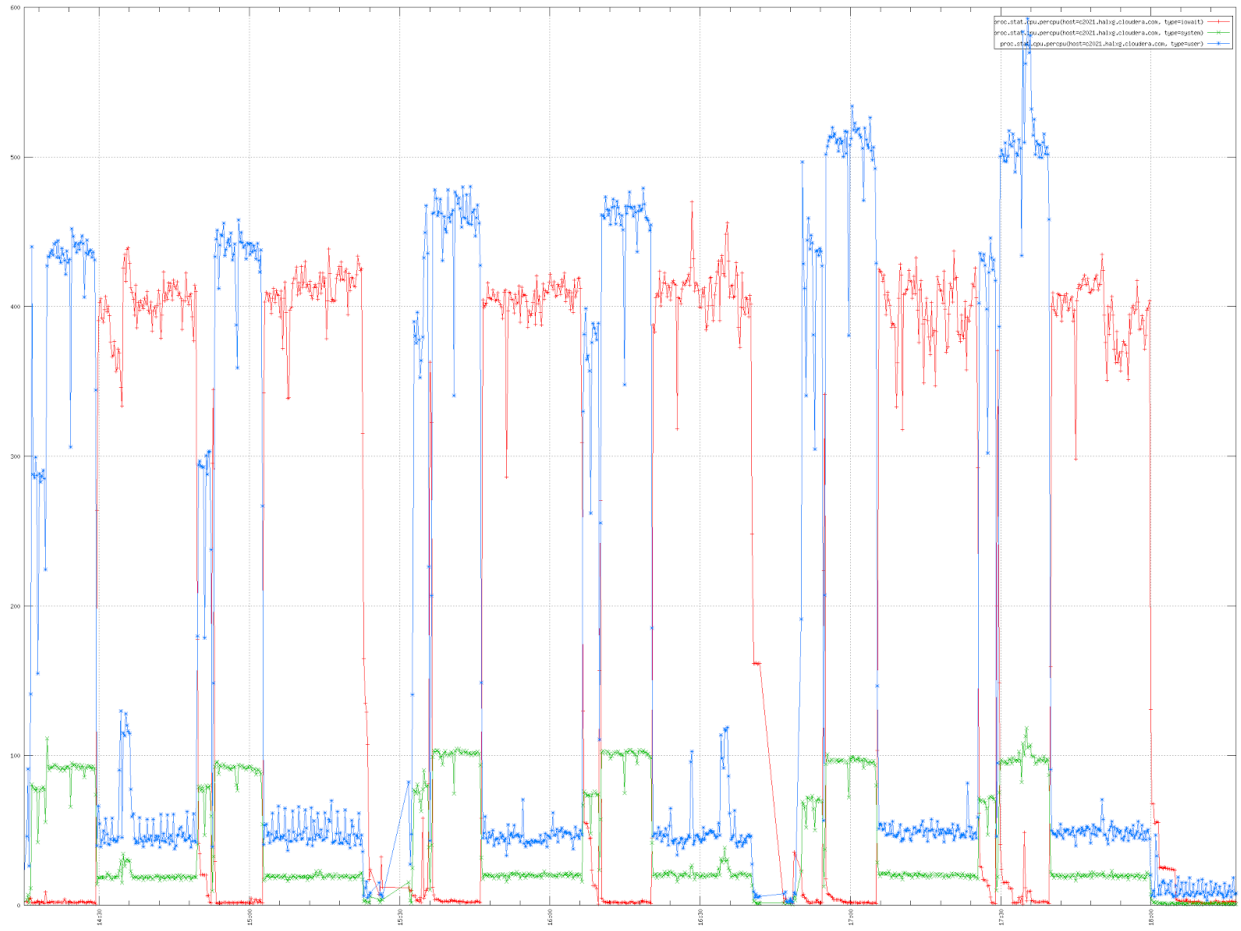
4.5 Scan histogram

LruBlockCache is better.



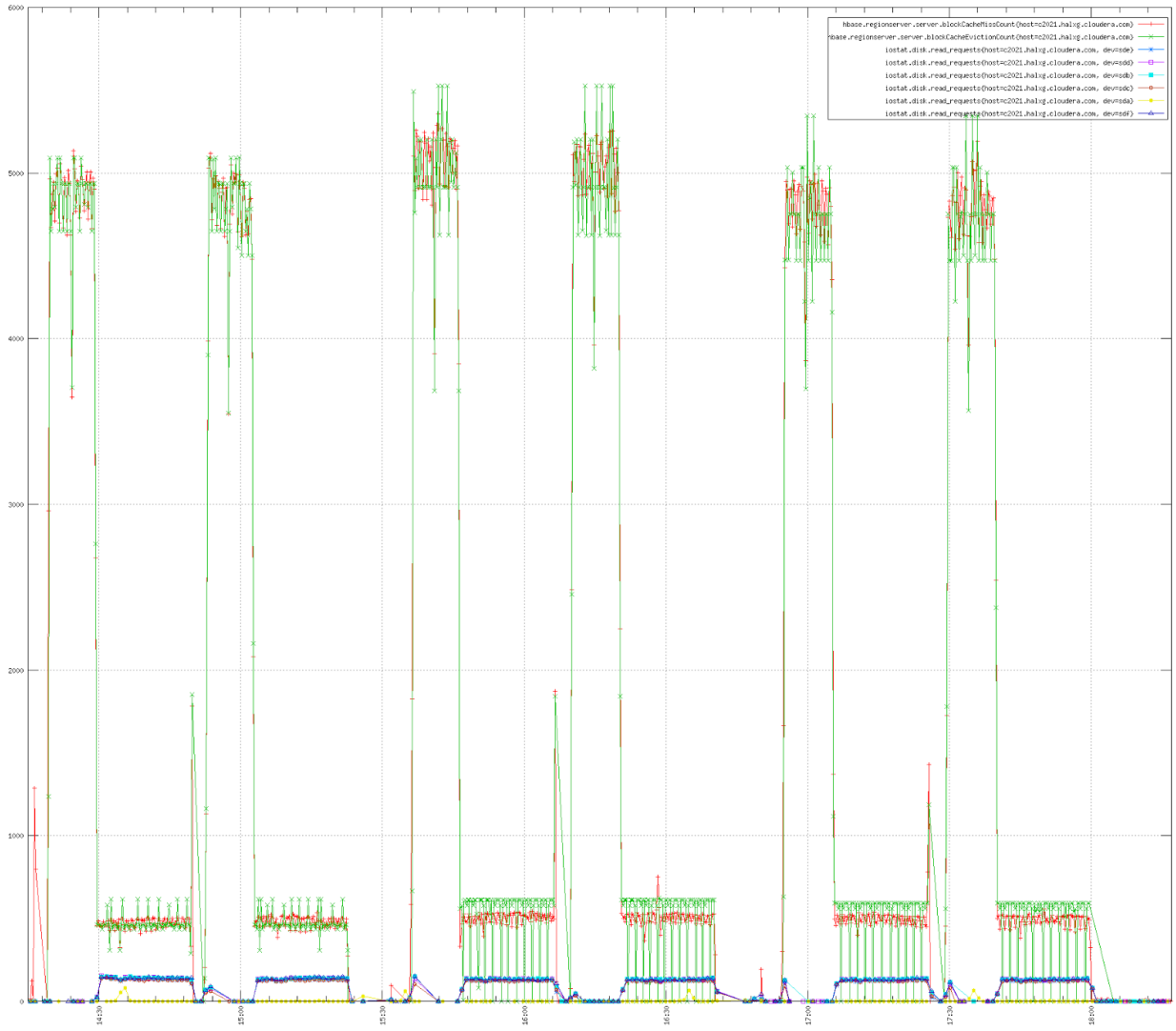
4.6 CPU

LruBlockCache uses less CPU (user) when in-cache. When out of cache, bucket cache offheap might be a bit better. iowait same all around.



4.7 I/O

Same all around.



5 Test Script

Script runs a warmup and then actual test. Does PerformanceEvaluation randomRead tests and then scanRange1000. When size is 2.8G, we never go to disk. When 5.6G we go to disk some (see accompanying diagrams for i/o), and then when size is 100G, we go to disk a bunch.

```
[stack@c2020 ~]$ more bin/bc_test.sh
#!/bin/sh
HOME=/home/stack
testtype=$1
date=`date -u +"%Y-%m-%dT%H:%M:%SZ"
echo testtype=$testtype $date` >> nohup.out
HBASE_HOME=$HOME/hbase-0.99.0-SNAPSHOT
warmuptime=300
runtime=900
for test in randomRead scanRange1000; do
  for i in 2.8 5.6 100; do
    echo "`date` run $test warmup time=$warmuptime size=$i" >> nohup.out
```

```
    timeout $warmuptime nohup ${HBASE_HOME}/bin/hbase --config
  /home/stack/conf_hbase org.apache.hadoop.hbase.PerformanceEvaluation --nomapred
  --size=$i randomRead 10
    echo "`date` run $test time=$runtime size=$i" >> nohup.out
    timeout $runtime nohup ${HBASE_HOME}/bin/hbase --config /home/stack/conf_hbase
  org.apache.hadoop.hbase.PerformanceEvaluation --nomapred --size=$i randomRead 10
    done
  done
  mv $HOME/nohup.out $HOME/nohup.$testtype.$date
```

6 Setup Details

Cluster of five machines all running HDFS. One RegionServer with 200+ regions up on it. 16 processors. 48G. 6 disks. Run clients on same machine as master.

6.1 hbase-env.sh

```
export HBASE_HEAPSIZE=8000
...
export HBASE_OPTS="$HBASE_OPTS -XX:MaxDirectMemorySize=5g"
...
```

6.2 hbase-site.xml

```
  <property>
    <name>hbase.regionserver.global.memstore.upperLimit</name>
    <value>0.3</value>
    <description>Maximum size of all memstores in a region server before new
      updates are blocked and flushes are forced. Defaults to 40% of heap.
      Updates are blocked and flushes are forced until size of all memstores
      in a region server hits
hbase.regionserver.global.memstore.size.lower.limit.</description>
  </property>
  <property>
    <name>hbase.regionserver.global.memstore.size</name>
    <value>0.3</value>
    <description>Maximum size of all memstores in a region server before new
      updates are blocked and flushes are forced. Defaults to 40% of heap.
      Updates are blocked and flushes are forced until size of all memstores
      in a region server hits
hbase.regionserver.global.memstore.size.lower.limit.</description>
  </property>
  <property>
    <name>dfs.client.slow.io.warning.threshold.ms</name>
    <value>100</value>
  </property>

  <!--LRU Cache-->
  <property>
    <name>hfile.block.cache.size</name>
    <value>0.5</value>
  </property>

  <!--Slab Cache-->
  <!--
  <property>
```

```
  <name>hbase.offheapcache.percentage</name>
  <value>0.8</value>
</property>
-->

<!--Bucket cache-->
<!--
-->
<property>
  <name>hbase.bucketcache.ioengine</name>
  <value>offheap</value>
</property>
<property>
  <name>hbase.bucketcache.size</name>
  <value>5120</value>
</property>
<property>
  <name>hbase.bucketcache.percentage.in.combinedcache</name>
  <value>0.8</value>
</property>

<property>
  <name>hbase.bucketcache.ioengine</name>
  <value>heap</value>
</property>
```