[HBASE-13408] HBase In-Memory Memstore Compaction: Scans Evaluation Results

August 20, 2015

Eshcar Hillel (eshcar@yahoo-inc.com)

Anastasia Braginsky (anastas@yahoo-inc.com)

Experimental Settings

In this experiment we evaluate long scans. We set target throughput to 500ops with 99% reads and 1% scans. The size of the scans is determined by a zipfian distribution with maximal size set to 20,000 records. Even though their portion is small (only 1%), the inherently long scan operations span more time during the experiments then update operations do.

As in the read-write workload we set up a small cluster of 3 hdfs nodes, with a single region server (and a master). The heap at each node is allocated 1GB, while the global memstore size is limited to 300MB and the block cache size to 100MB (see the hbase-site.xml below). To avoid memory fragmentation we use mslab chunks of size 2MB.

We use a data set of 128K records, record size is 1KB.
We do not load the table with any initial data, instead we use YCSB to run workloads of 5M operations, and to measure their performance over time. We present the latency result for hotspot distribution.

hbase-site.xml

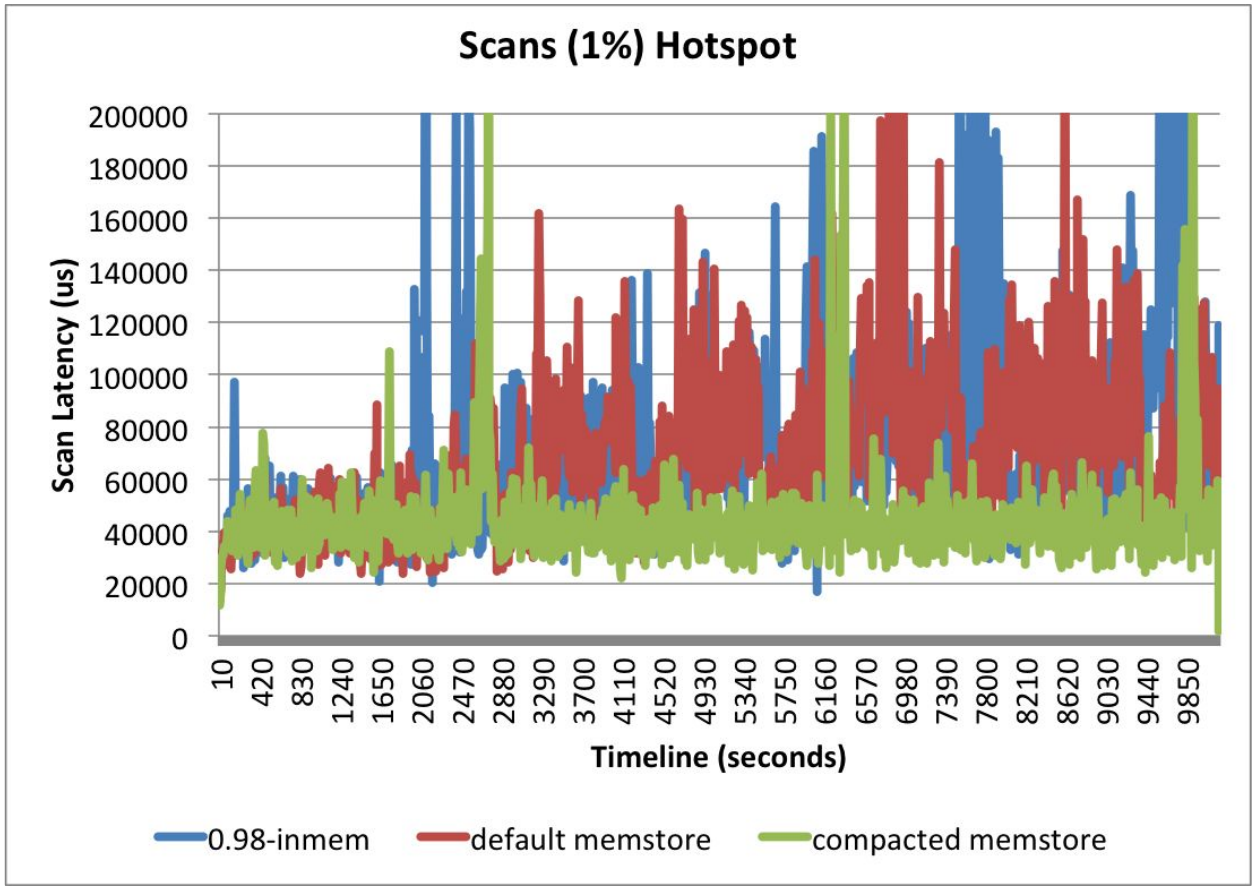
```
<...cluster configuration settings...>
<property>
<name>hbase.hregion.memstore.block.multiplier</name>
<value>4</value>
```

```
</property>
<property>
<name>hbase.regionserver.global.memstore.lowerLimit</name>
<value>0.3f</value>
</property>
<property>
<name>hbase.regionserver.global.memstore.upperLimit</name>
<value>0.3f</value>
</property>
<property>
<name>hfile.block.cache.size</name>
<value>0.1f</value>
</property>
<property>
<name>hbase.hregion.memstore.mslab.enabled</name>
<value>true</value>
</property>
<property>
<name>hbase.hregion.memstore.mslab.chunksize</name>
<value>2097152</value>
</property>
</configuration>
```

Experimental Results

Since the size of the scans is not fixed the latency results are much noisier than the read/write latency results, however that trends are similar, showing 30%-65% speed-up in retrieval performance.

As in the read operations, all implementations start the same -- at 40ms per scan while the content is stored entirely in memory. Since the write load is the same in the scan-write experiment as in the read-write experiment, flush operations fill up the block-cache at roughly the same time, where the performance of the default memstore starts to degrade, reaching peaks of 120ms. The figure shows that scans in the default memstore implementation also benefit from compactions which help reduce the latency to 50ms, but never regain the same performance as the compacted memstore scans.



The second figure shows only the latency results for default memstore with the HBASE-13408 patch and for in-memory memstore without the patch, so the low points are visible and not hidden by the compacted memstore results. It can be seen here more clearly that once not all data is served from memory the latency is above 50ms (most of the time).