HBASE-16417: Policy for in-memory flush&compaction - benchmark results

We run benchmarks on a single machine cluster to compare performance of no compaction (default memstore), index-compaction, and data-compaction under various workloads, with the goal of finding the optimal policy for in-memory flush and compaction.

Benchmark Settings

Hardware

SSD machine, 48GB ram, 12 cores, 2.9 TB disk

HBase configuration (fixed)

16GB heap
50 regions (presplit)
50 columns
Value size 200B

Additional global parameters:

```
<property>
      <name>hbase.regionserver.global.memstore.size</name>
      <value>0.42</value>
</property>
<property>
      <name>hfile.block.cache.size</name>
      <value>0.38</value>
</property>
<property>
      <name>hbase.hstore.flusher.count</name>
      <value>10</value>
</property>
<property>
      <name>hbase.hstore.blockingStoreFiles</name>
      <value>25</value>
</property>
```

In most cases we run with MSLAB enabled

```
<property>
      <name>hbase.hregion.memstore.mslab.enabled</name>
      <value>true</value>
</property>
<property>
      <name>hbase.hregion.memstore.chunkpool.maxsize</name>
      <value>1</value>
</property>
<property>
      <name>hbase.hregion.memstore.chunkpool.initialsize</name>
      <value>0.5</value>
</property>
```

Saturation point: #threads=10

We would like to test the system at a point where it is loaded to the maximum it can stand before pushing back (namely, block updates). This is called the *saturation point*. It is the point of maximum throughput with minimal latency.
To identify the saturation point we
(1) run PE write-only workload with different numbers of threads (5,10,12,15, 20, 30, 50) and,
(2) plot throughput-latency graphs of these executions for each of the latency percentiles reported by PE.

Figure 1 shows that for the 50th and 75th percentiles there is no difference in latency w.r.t. The number of threads. The difference can be seen first in the 95th percentile. IT shows that the saturation point is achieved at 10 threads where the throughput is maximal (207 MB/s) and latency is minimal (80.7 ms).
The same trend is depicted in figure 2 for higher percentiles.

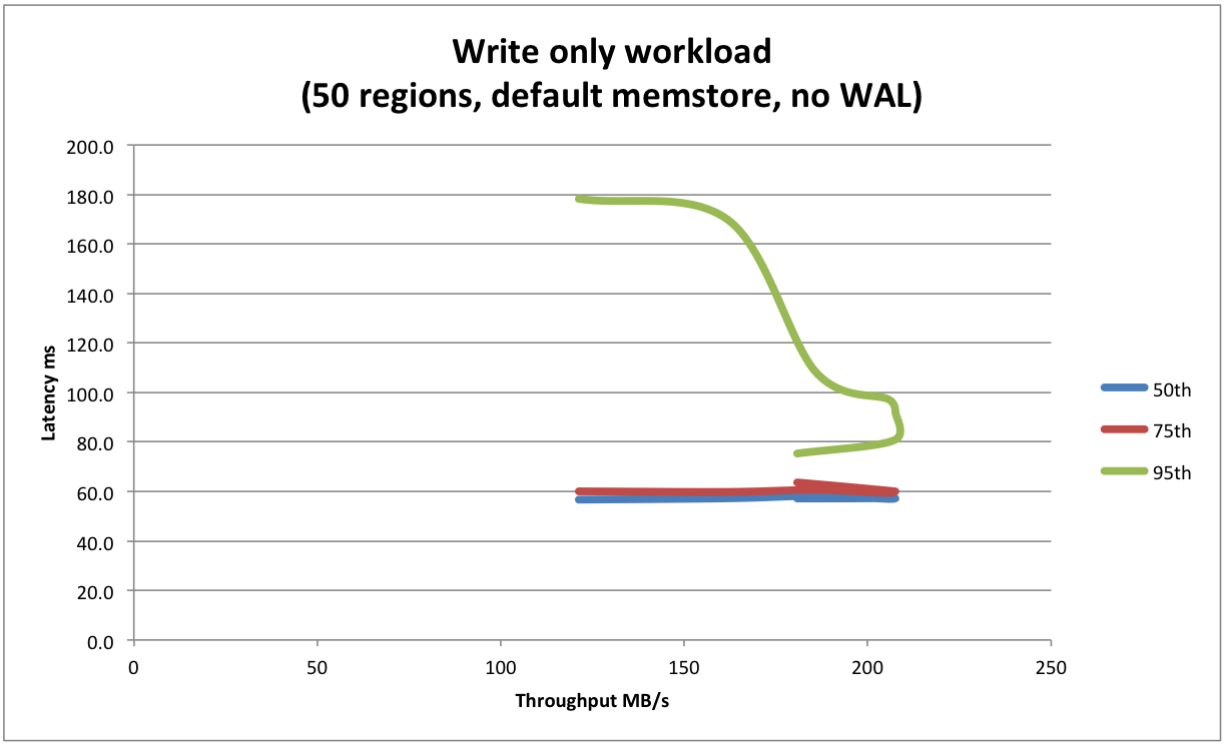

Figure 1. Saturation point at 10 threads (95th percentile)

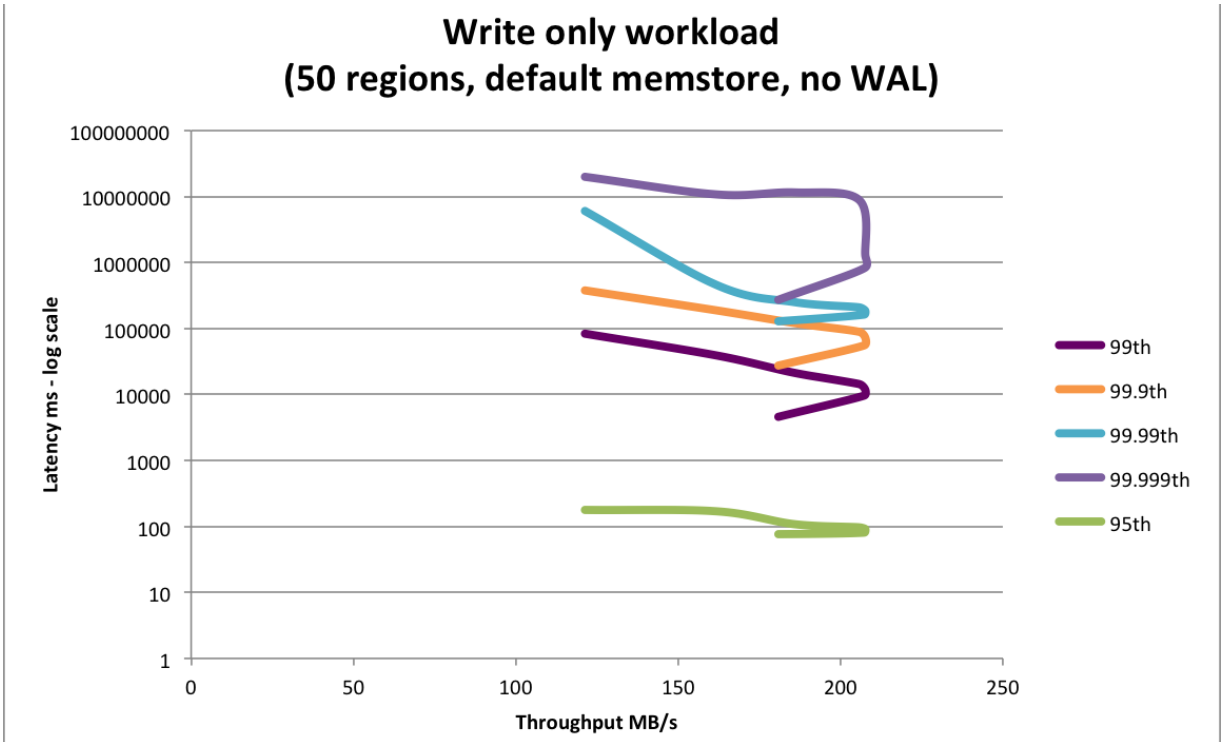

Figure 2. Saturation point at 10 threads (95/99/99.9/99.99/99.999th percentiles)

We ran the same tests with WAL and the saturation point was the same with 10 threads.

We next compare the performance of different options (no-, index-, data-compaction) under different workloads.
While write-only workload with uniform distribution can be tested with PE, other workloads (mixed or with zipfian distribution) calls that we use a different tool, namely, YCSB.

Benchmark results

Write-only workload, uniform distribution

We write 50GB data using PE with the above settings.
We run index compaction with varying number of segments in the pipeline before merging the index: greater than 1 (ic1), greater than 2 (ic2), greater than 3 (ic3). For data compaction we **do not use MSLABs** to avoid the inherent space and computation overhead of copying data during compaction.
Figure 3 shows that up until the 95th percentile all options are comparable. At the 99th percentile data compaction starts to lag behind -- indeed in a uniform workload there is not much point in doing data compaction. The overhead might stem from running SQM to determine which versions to retain. One way to close this gap is to not run data compaction when there is no gain in it. A good policy should be able to identify this with no extra cost.
At the 99.999th percentile index compaction also exhibits significant overhead. This might be due to memory reclamation of temporary indices.

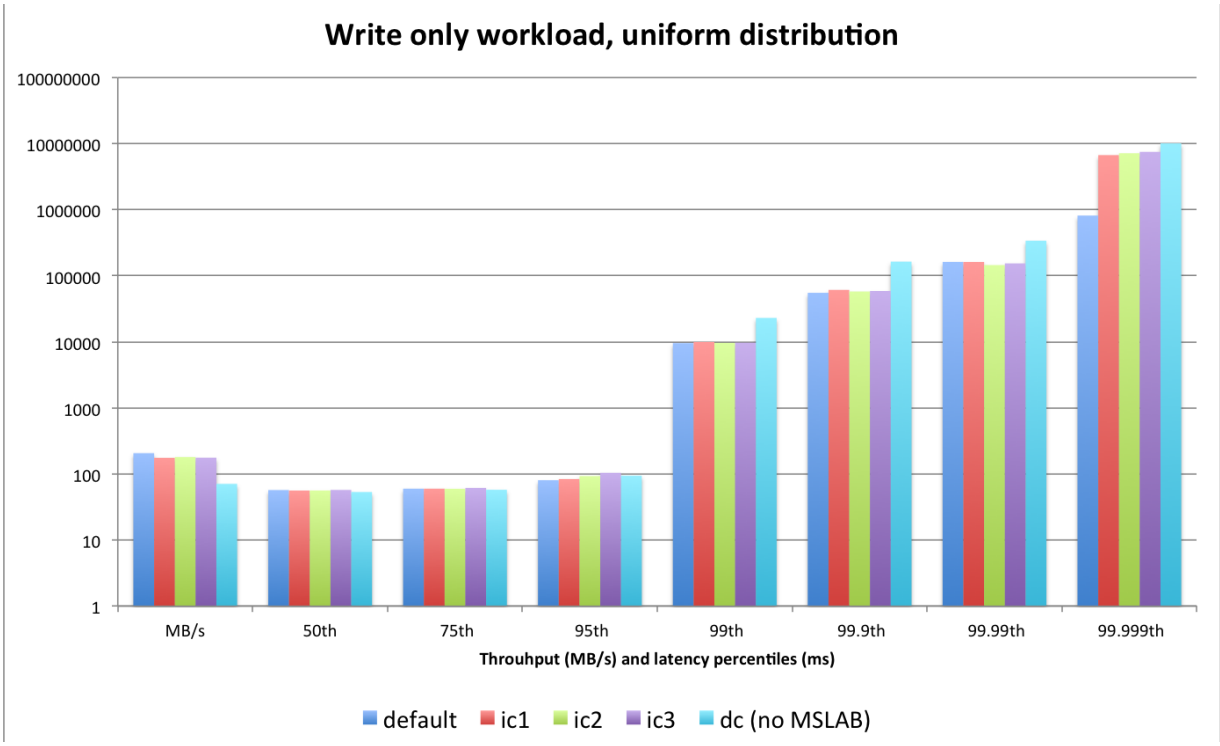

Figure 3. Throughput and latency comparison