HBASE-16417: Benchmark Results for In-memory Flush and Compaction

We run benchmarks on a cluster of 3 machines to compare performance of different compaction policies: NONE - no compaction, BASIC - index-compaction, and EAGER - data-compaction under write-only and mixed (read-write) workloads.

We focus on small data (100B values), uniform and zipfian distribution of keys, Async and sync wal modes.
We investigate

(1) write amplification: both total volume of MB written to files and amplification w.r.t. data written by the client (write-only workload)
(2) Total throughput (write-only workload)
(3) Avg and 99th percentile write latency (write only workload)
(4) Avg, 50th, 75th, 90th, 99th read latency (mixed workload)

Summary of results:
Write only
We run write-only benchmarks with zipfian and uniform distribution of keys.

- Uniform distribution: basic outperforms no compaction by 40%; eager improves throughput of no compaction by 10%, however the 99th latency percentile in eager is slower by 18% than no compaction. This is likely due to unneeded in-memory compactions occurring at the background.
  Basic and eager improve write amplification by 25% and 15% respectively.
- Zipfian distribution: when running in async wal mode eager improves write amplification by 30%. Basic and eager outperform no compaction by 20% and 15%, resp. This can be attributed in part to running less GC and in part to doing less IO. When running sync wal mode eager improves write amplification by 10%. Other than that all policies are comparable. Basic and eager slightly improving over no compaction. In sync wal mode the throughput is much lower. Async wal mode represent a scenario where the system is loaded by many clients with much higher load.

Mixed workload
Eager improves over no compaction by 6-10%. Basic improves the 50th percentile by 7% but the performance of 95th and 99th percentile degrade the performance by 15-30%. This is as a result of reading from multiple segments in the compaction pipeline. Applying merge on pipeline segments more often should eliminate this overhead (initial experiment we did show that it does).

Benchmark Settings

Hardware

4 SSD machines, each with 48GB ram, 12 cores, 2.9 TB disk
1 master, 2 RS, 1 YCSB client

Cluster settings

HDFS is deployed on 3 machines (3-way replication);
HBase: 1 master, 2 region servers with the following setting:
16GB heap, from which 40% allocated to memstore and 40% to block cache (default values)
GC: HBASE_OPTS="-XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:InitiatingHeapOccupancyPercent=60 -XX:G1HeapWastePercent=20 -XX:G1MixedGCCountTarget=8"

Additional global parameters:

```
<property>
      <name>hbase.hstore.flusher.count</name>
      <value>10</value>
</property>
<property>
      <name>hbase.hstore.blockingStoreFiles</name>
      <value>25</value>
</property>
```

Since previous experiments show consistently weaker performance with on-heap mslabs we focus on running experiments with **no chunks pool** and **no mslabs**.

Benchmark results

We run in synchronous and asynch WAL modes.
Having 2 RSs we pre-split the table into 100 regions (50 region per RS); data set of 200M keys.
Batching writes at the client side; buffer size is 10KB

Write-only workload, zipfian distribution, small values

We run 12 threads to run 1G write operations. Keys are chosen from a zipfian distribution over data set of 200M items. Value size is 100B so overall we write 100GB.

Table 1 shows statistics for write amplification in synchronous and asynchronous WAL modes. Write amplification is defined as the size of data written to disk divided by the size of data written by the client.
Write amplification =
3*(total flush size+total compaction size+total wal size) / 100GB
Eager improves the amplification by 10-30%.

	Sync WAL		Async WAL	
	Amplification	Normalized	Amplification	Normalized
None	12.336	1.00	11.523	1.00
Basic	12.297	0.99	11.498	0.99
Eager	11.154	0.90	8.226	0.71

Table 1. Write amplification in zipfian distribution

Table 2 compares the number of flush/compaction/wal files in sync and async wal modes. Eager policy reduces the number of flush and compactions by 35-50% in sync mode and by 45-60% in async mode. It also reduced the number of wal files in async mode.

	Sync WAL			Async Wal		
	#flush	#compact	#wal	#flush	#compact	#wal
None	1189(1.00)	412(1.00)	1673(1.00)	1132(1.00)	366(1.00)	839(1.00)
Basic	1181(0.99)	405(0.98)	1673(1.00)	1137(1.00)	357(0.97)	879(1.04)
Eager	763(0.64)	212(0.51)	1672(1.00)	640(0.56)	147(0.40)	476(0.56)

Table 2. Write counters in zipfian distribution

Figure 2 shows the size of data written by files divided to flush/compaction/wal in sync and async wal modes.

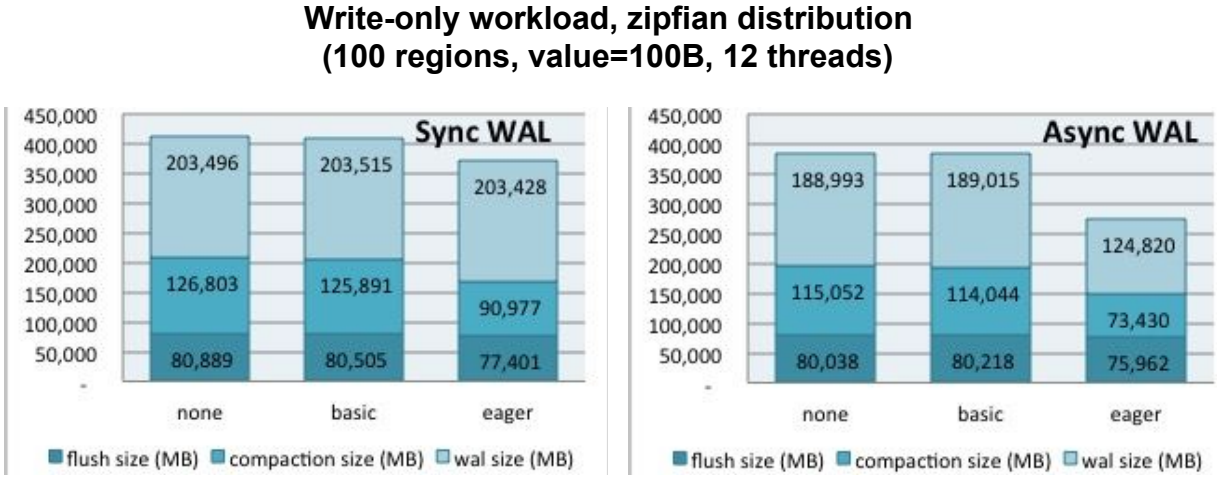

Figure 2. Total data written to files divide by flush/compaction/wal

Figures 3 and 4 show the performance in sync and async wal modes, respectively: overall throughput (op/s), average and 99th latency, and GC counts.
In sync wal mode all policies are comparable. Basic and eager slightly improving over no compaction. This is not statistically significant (2-4%).

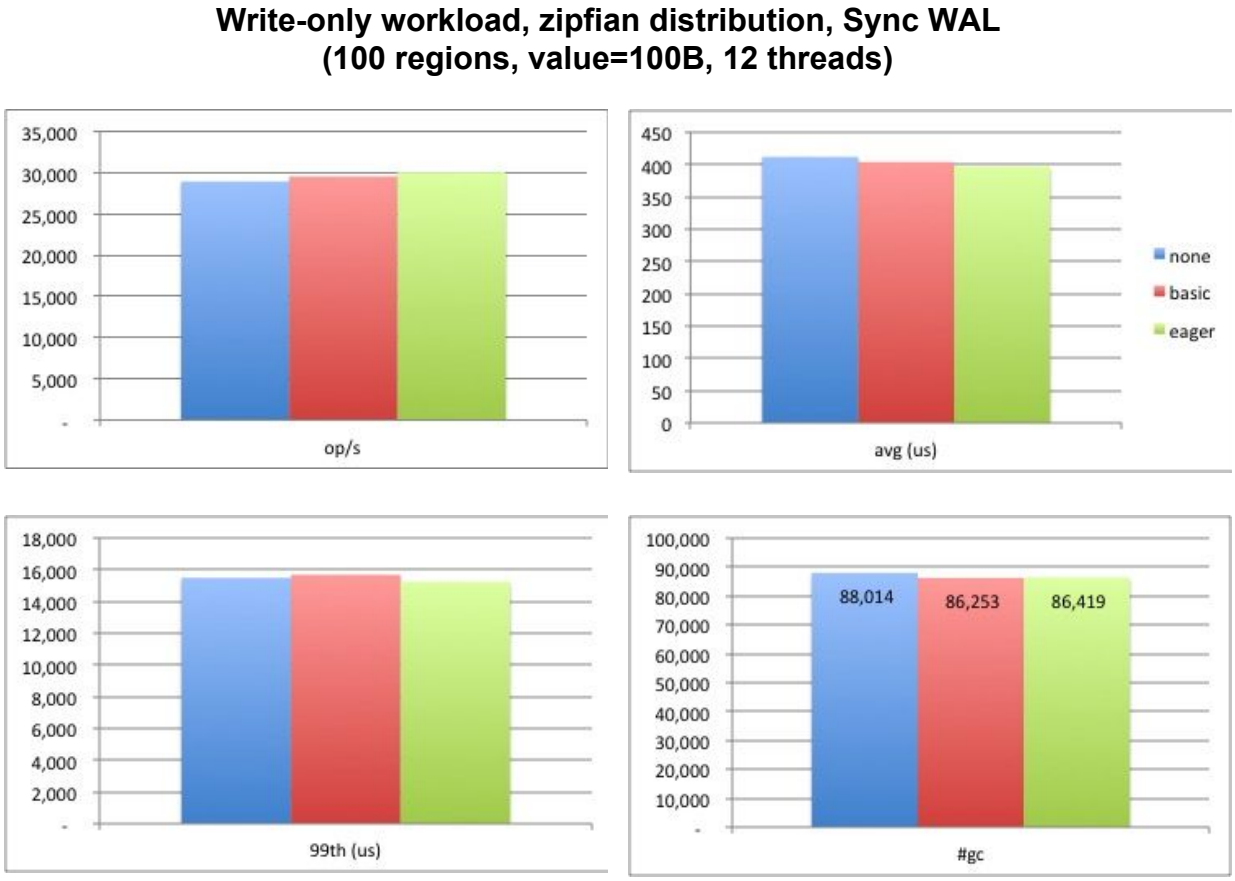

Figure 3. Total throughput, write latency, and gc count comparison (Sync WAL)

In Async WAL basic and eager outperform no compaction by 20% and 15%, resp. This can be attributed in part to running less GC and in part to doing less IO.

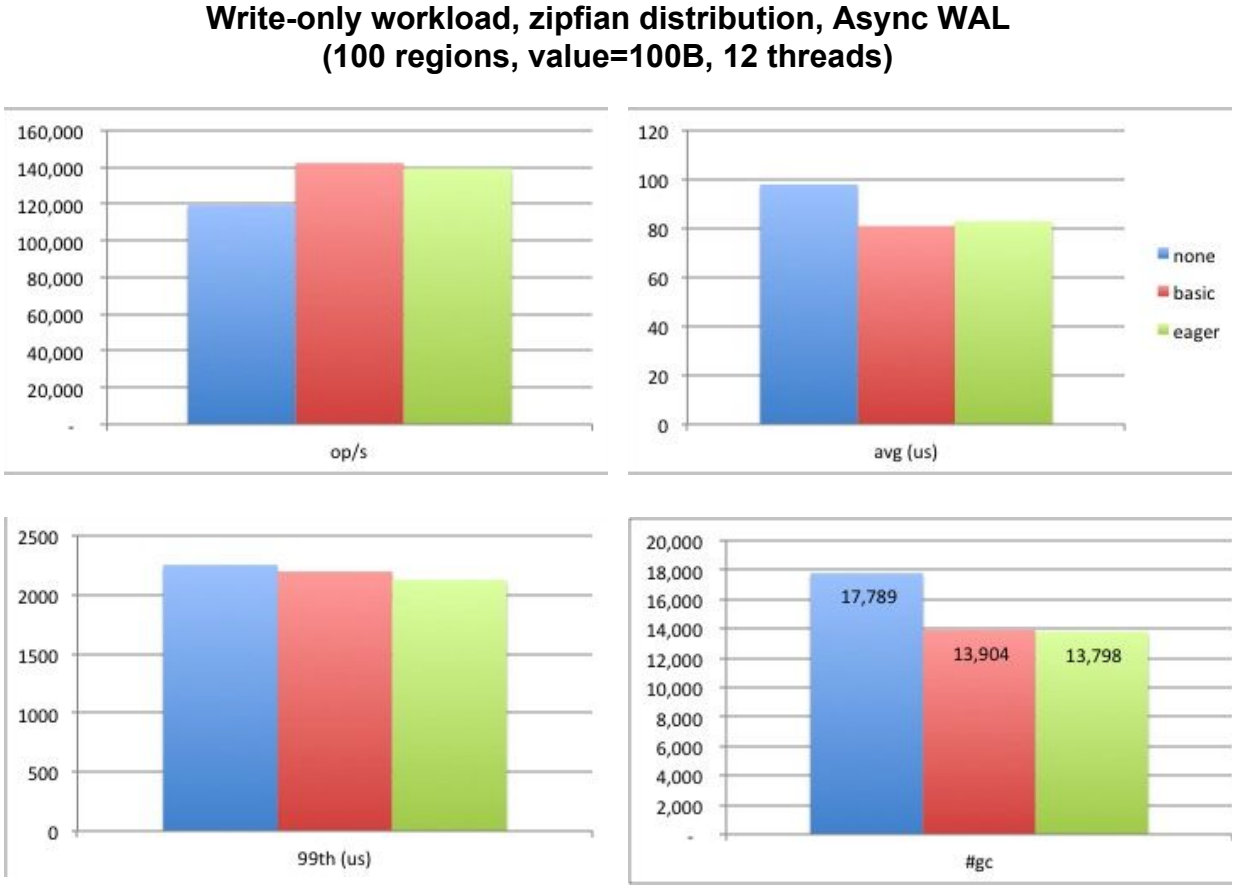

Figure 4. Total throughput, write latency and gc count comparison (Async WAL)

Write-only workload, uniform distribution, small values

We run 12 threads to run 1G write operations. Keys are chosen from a uniform distribution over data set of 200M items. Value size is 100B so overall we write 100GB. We run only in asynchronous WAL mode

Table 3 shows statistics for write amplification.

	#flush	#compact	#wal	Amplification
None	1118	252	378	11.783(1.00)
Basic	1107	169	304	8.906(0.75)
Eager	1229	280	300	10.130(0.85)

Table 3. Write amplification in uniform distribution

Figure 5 shows the size of data written by files divided to flush/compaction/wal.

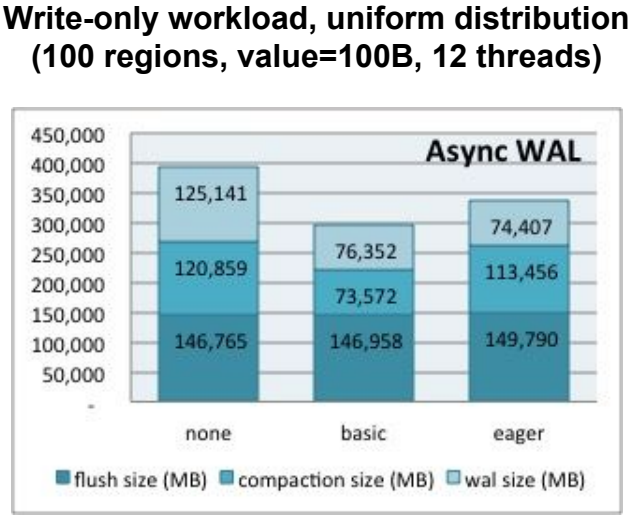

Figure 5. Total data written to files divide by flush/compaction/wal

Figure 6 show performance result for uniform distribution: overall throughput (op/s), average and 99th latency, and GC counts.
Basic and Eager outperform no compaction by 40% and 10%, resp. The 99th percentile in eager is slower by 18% than no compaction.

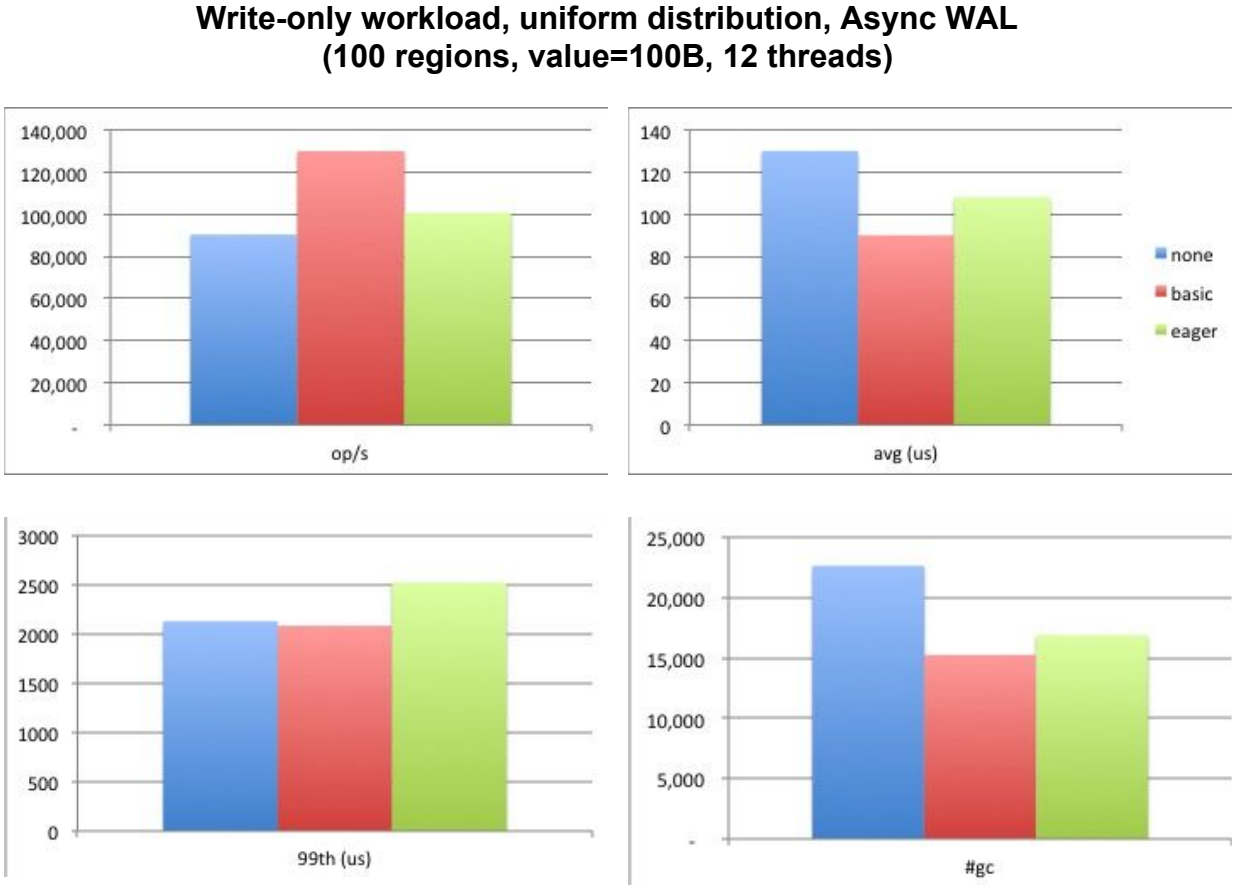

Figure 6. Total throughput, write latency and gc count comparison (Async WAL)

Mixed workload (50%-50%), zipfian distribution, small values

We pre-split the table into 100 regions (50 region per RS); data set of 200M keys.
Initially we pre-load data by writing 200Mx100B cells chosen uniformly at random (total 20GB). Then we measure the performance of 200M operations, 50% writes and 50% reads, with keys chosen from a zipfian distribution. We present results only in async wal mode as wal mode does not affect read latency, but it allows us to test the system with greater load. We see similar trends with sync wal mode.
Eager improves over no compaction by 6-10%. Basic improves the 50th percentile by 7% but the performance of 95th and 99th percentile degrade the performance by 15-30%. This is as a result of reading from multiple segments in the pipeline. Applying merge on pipeline segments more often should eliminate this overhead (initial experiment that we did show that it does).

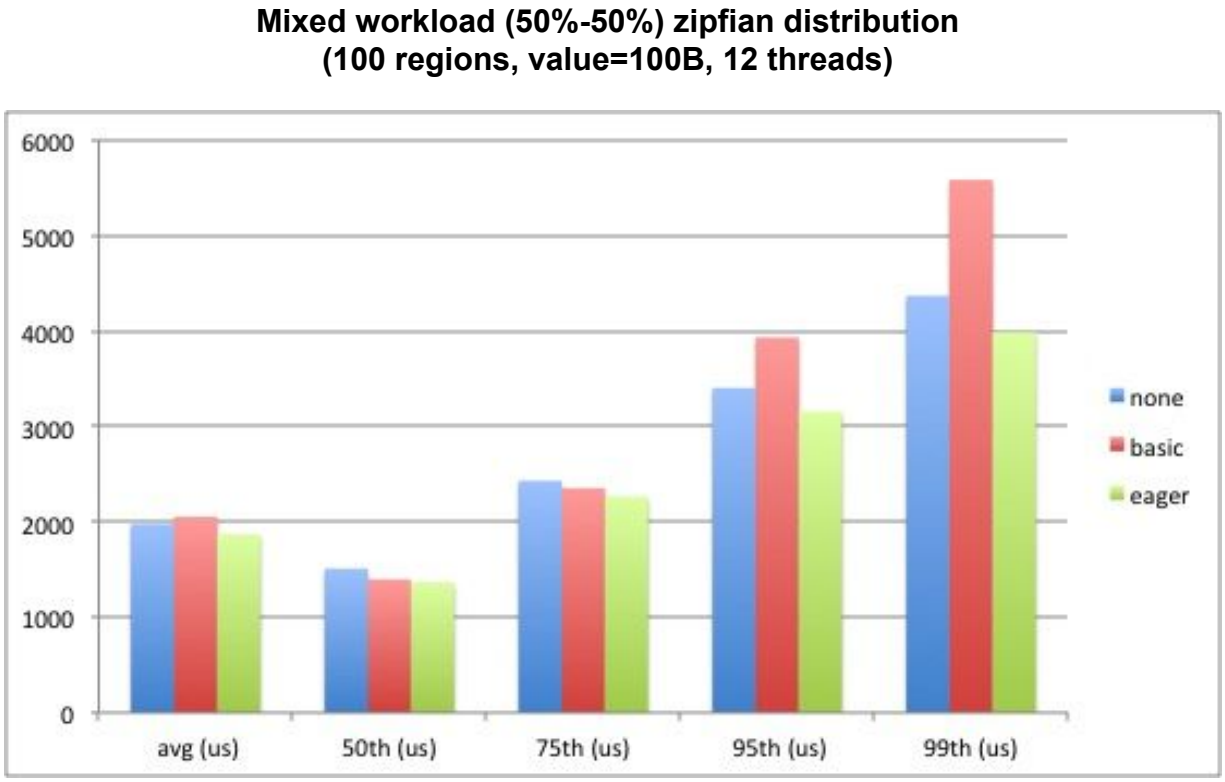

Figure 7. Read latency comparison (Async WAL)