Stripe compaction perf evaluation

sershe, 2012-03-26, HBASE-7667

Table of Contents

Count-based stripes

Iteration 1

Goal

Get the basic picture of perf, look at various tweakable settings to narrow down to what warrants further testing.

Setup and method

5-node EC2 cluster with "m1.large" nodes, 3RSes and no other activity, is used. Recent trunk HBase version with stripe compactions is deployed. For each test, table

is created with one region per server, one CF, with the necessary configuration set via HTD. Splits are prevented via constant policy with very large max_filesize.
Using LoadTestTool, table is then pre-loaded with 120000 records per server with base 8Kb record size, without verification.
After that, 500 iterations of LoadTestTool are executed. First, LTT is used to insert 2000 (8Kb base size) records per server; then, to verify them (in a separate run). Insert and read times are measured, in ms.
After each test, table is dropped and recreated.

Known weaknesses of the setup/method

- Inherent EC2 variability.
    - Some scenarios may be tested repeatedly with shorter tests if this is the concern.
- Workload is not typical (write heavy).
- Workload only reads recent data.
    - Both of these were partially addressed in another test with slightly different load (instead of writing N keys and reading them, write 0.4*N keys, then read them and also 3 random old segments of 0.4*N keys each); however this test merely showed a more stable chart due to less write load, with not much in terms of new insights, and was terminated. Something similar can be done again upon requests.

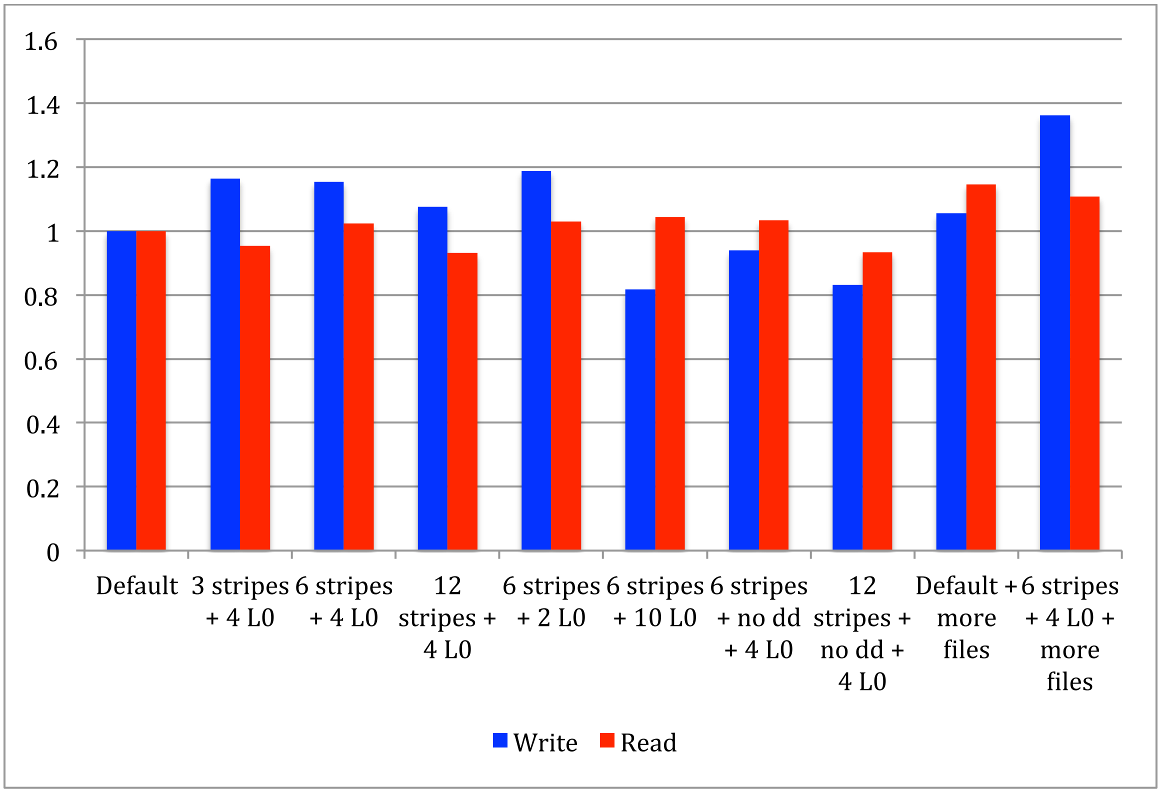
Compaction schemes tested

- Default.
- Stripe with default settings (6 stripes and 4 L0 file limit).
- Stripe with 3 stripes.
- Stripe with 12 stripes.
- Stripe with 2 L0 file limit.
- Stripe with 10 L0 file limit.
- Stripe with assumption of sequential puts (can drop deletes without involving L0).
- Stripe with 12 stripes and assumption of sequential puts (can drop deletes without involving L0).
- Default with 64Mb flush size (more files).
- Stripe with 64Mb flush size (more files).

Results – total time

The relative time taken by writes and reads for each of the above scenarios is shown below (less is better).

- Most (and all practical) stripe scenarios have worse write perf, probably due to many L0 compactions resulting in write amplification.
- Reads in stripe scheme are generally faster.
- Having more L0 files over time, or drop-delete assumption, makes writes faster (than default) but reads somewhat slower (which is expected).
- Having 12 stripes seems to improve write perf regardless of drop-delete assumption.
- Having 12 stripes seems to improve reads regardless of drop-delete assumption, but so does having 3 stripes in one test – that needs to be verified.



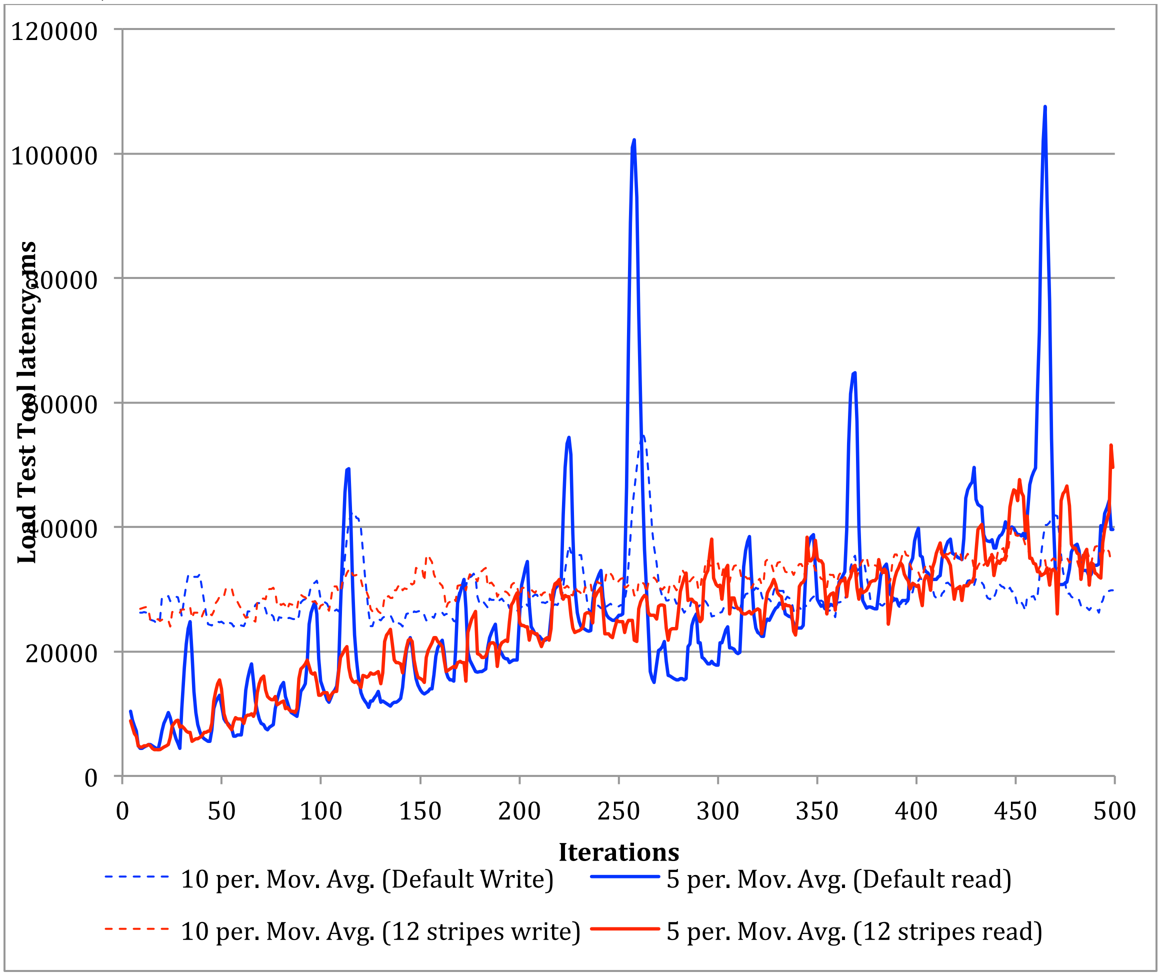
Results – variability

Charts below are LoadTestTool latency over time, separately for reads and writes. Moving average is displayed (5-point for reads, 10 for writes) because otherwise chart is a mess.

The charts clearly show the reduced variability of the stripe scheme. Relative totals for the charts are in previous sections.

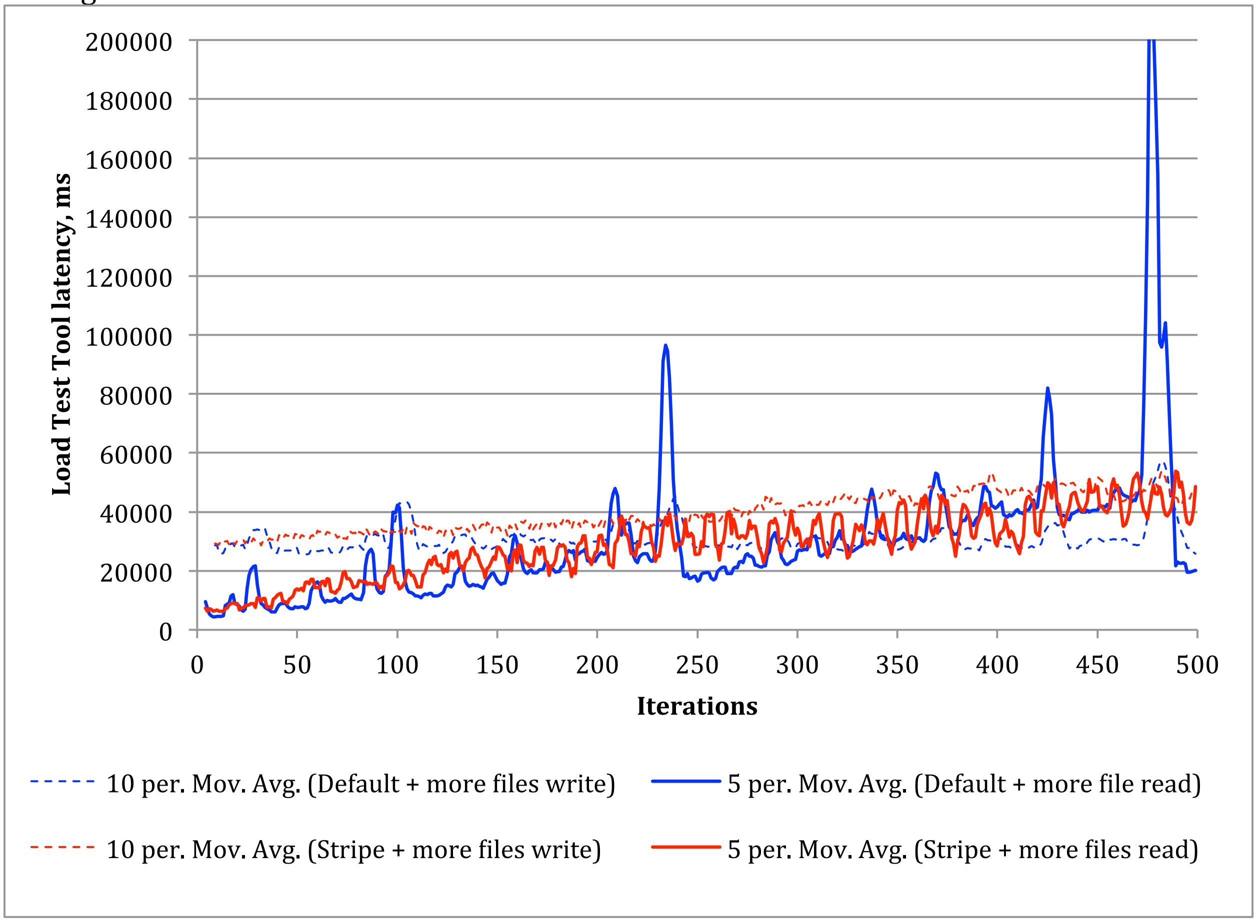
Providing charts for all scenarios is too tedious. Thus, 2 pairs are provided:

Default vs. stripe-with-12-stripes, because it's the best performing stripe scenario without tradeoffs of those tested. Stripe writes are slower except during spikes on default; default reads are slower and much more unstable.



Default vs. stripe, w/64Mb flush size both (i.e. more files), because it has different data for the default scheme (although very similar in pattern given that test is predictable). Writes for stripes are almost always slower. Reads are faster on

average and much more stable.



Conclusion

Count-based stripe approach is viable to reduce compaction impact for large regions. Large number of stripes (larger than the assumption of 5-6) seems to be better. It may make sense to compare few more numbers of stripes and L0 files to verify read and write tradeoffs. Users that do not use out of order puts can benefit from "don't include L0 to drop deletes" setting.

Iteration 2

Goal

Compare performance on specific scenarios on less IO-constrained EC2 instances.

Setup and method

Same as Iteration 1, however "c1.xlarge" instances are used.

Known weaknesses of the setup/method

Same as Iteration 1, EC2 variability should be somewhat reduced. e Some scenarios may be tested repeatedly with shorter tests if this is the concern.

Compaction schemes tested

- Default (twice).
- Stripe with 10 stripes (twice).
- Stripe with 3 stripes (twice).
- Stripe with 25 stripes (twice).

Results – total time

Tests are still running.

Results – variability

Tests are still running.

Conclusion

Tests are still running.

Size-based stripes

Iteration 1

Goal

Make sure stripe compactions work as intended. Because the perf benefit is supposed to be qualitative and not quantitative, ascertaining its existence should be enough.

Setup and method

5-node EC2 cluster with "m1.large" nodes, 3RSes and no other activity, is used. Recent trunk HBase version with stripe compactions is deployed. For each test, table is created with one region per server, one CF, with the necessary configuration set via HTD. Splits are prevented via constant policy with very large max_filesize. Regions are pre-split predictably to have a particular fixed prefix for the test below. 300 iterations of MultiThreadedWriter and –Reader (the classes that LoadTestTool uses) with custom data generator are then executed, each inserting 2000 records per server and verifying. The store compactions are tracked as the store size grows. Data generator in question generates sequential numbers partitioned into requisite number of regions so that the region size will be approximately equal, and ever increasing.
After each test, table is dropped and recreated.

Known weaknesses of the setup/method

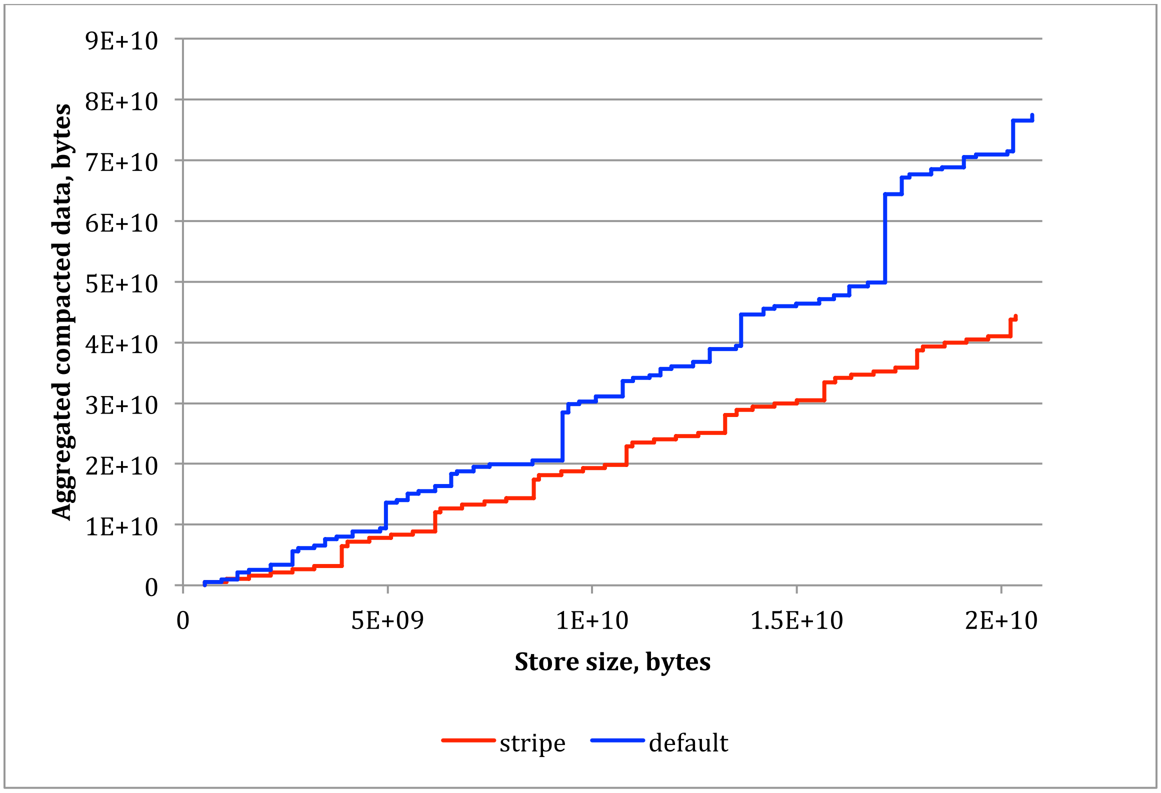
- Inherent EC2 variability.
    - Some scenarios may be tested repeatedly with shorter tests if this is the concern.

Compaction schemes tested

- Default.
- Stripe with size-based stripes.

Results

Analyzing store compactions shows that in this scenario, despite rewriting, stripe compaction scheme reduces the amount of data compacted as the store grows over time.



Conclusion

In ideal sequential data case, size-based stripes achieves much lower compaction volume despite some unnecessary data rewriting due to L0, at the same time making sure that, due to key range based separation, out-of-order keys will still be handled properly and will not require the unnecessary compaction of unaffected data.