Btree vs LSM

agorrod edited this page on Sep 11 2014 · 3 revisions

Introduction

The benchmarks on this page aim to demonstrate the strengths and weaknesses of Log Structured Merge trees and Btree table types. WiredTiger supports both LSM and Btree based tables.

Benchmark Overview

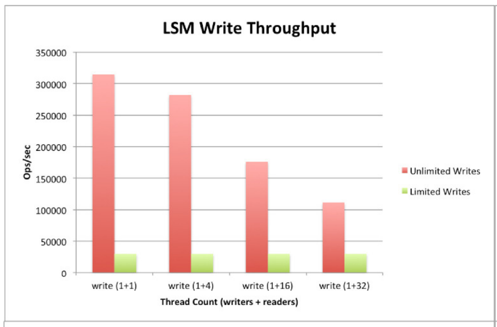
The benchmark creates a database by sequentially inserting 100 million key/value pairs with 100 byte value size. Generating around 10GB of data.

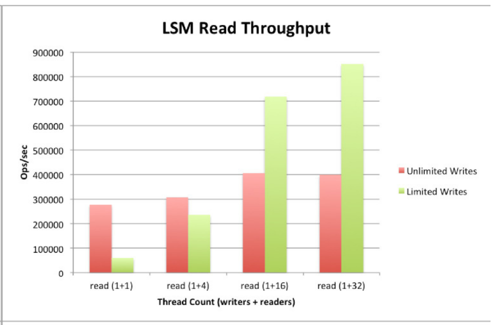
The benchmark runs a single thread doing inserts, and a set of threads that will concurrently run point queries. The following graphs run each workload phase for 10 minutes.

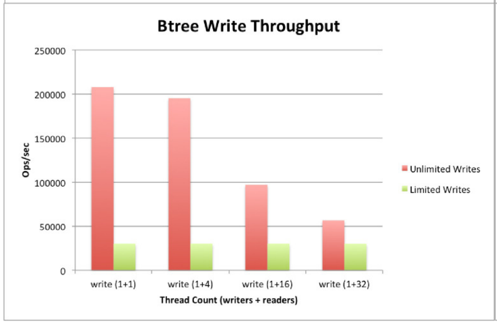
The variables between runs are:

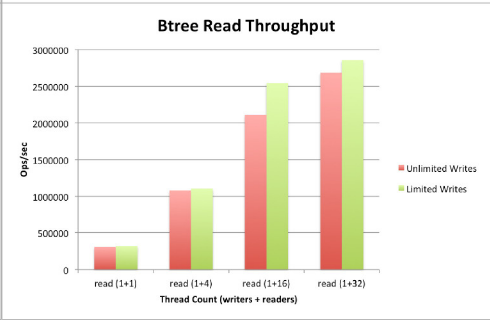
- Whether or not the number of updates per second limited
- The number of threads doing queries

Results









Analysis

Here is how I interpret the above graphs:

Write Throughput

Comparing the *LSM Write Throughput* to the *Btree Write Throughput*. Notice that both are able to sustain the 30,000 inserts a second requested by the rate limited workload. The LSM setup manages to sustain between 1.5x and 2x the insert throughput of btree. This is expected - LSM is optimized for write heavy workloads.

It is also interesting that there is only a single writer thread, but as the number of threads doing reads increases, the writes throughput decreases. The effect is quite dramatic with 32 read threads - which is to be expected since the machine running the benchmark has 24 CPU cores - having 32 application threads leads to context switching. The write throughput reduction with 16 read threads bears further investigation.

Read Throughput

Comparing the *LSM Read Throughput* to the *Btree Read Throughput*.

The key observation between the two graphs is that the read throughput using a btree is between 1.5x and 3x that achieved using an LSM tree. The difference becomes more pronounced as more read threads are added.

The first set of results for LSM is a side effect of how the benchmark is run. The benchmark runs a populate phase followed immediately by a sequence of read phases. With an LSM tree, the populate phase is going to leave the tree in a state where it has lots of background maintenance that can be done. In WiredTiger, when there is not much activity, we automatically increase the amount of maintenance work being done. So the set of read (1+1) results will be skewed while the LSM tree is "settling down". This settling work helps improve the read performance in later configurations. Btrees don't require the background maintenance, so this isn't a factor in their performance.

Take aways

- If you don't require extreme write throughput btree is likely to be a better choice. Read throughput is better and high volumes of writes can be maintained.
- If you have a workload that requires a high write throughput LSM is the best choice.
- WiredTiger makes it trivial to choose either Btree or LSM data structures - we recommend trying each for your particular workload.
- These workloads are trivial. Each database contains a single table with a very simple schema. The insert operations are unconditional updates within a constrained key space. Each read operation is a single search. YMMV.
- If you have a more complex schema it may best to use a combination of btree and LSM trees in your database.

Details

Hardware

Item	Specification
CPU	4 x Intel Xeon CPU X5650 @ 2.67GHz (24 cores total)
Memory	144 GB RAM
Disk	400GB Intel S3700 SSD

Software

WiredTiger release 2.3.1, configured with `../configure --with-builtins=snappy,zlib`

LevelDB benchmark built with: `CXXFLAGS="-I$WT_HOME/include" LDFLAGS="-L$WT_HOME/lib" make db_bench_wiredtiger`

Runner script

```
export TEST_TMPDIR=/ssd1/scratch/wiredtiger/ldb
NUM=100000000
# some engines appear to ignore the cache_size and just grab as much RAM as they want
CACHE=34359738368
# some engines honor the cache_size, give them more.
CACHE2=137438953472
WRATE=30000
STATS=1048576
DUR=600
TIME="/usr/bin/time -v"

export LD_LIBRARY_PATH=$WT_HOME/lib:$LD_LIBRARY_PATH
rm -rf $TEST_TMPDIR/*
echo "Running wiredtiger twice, once as LSM, once as Btree"
rm -rf $TEST_TMPDIR/*
LD_PRELOAD=/usr/lib64/libjemalloc.so LD_LIBRARY_PATH=$WT_HOME/lib:$LD_LIBRARY_PATH $TIME ./
#for THREADS in 1 2 4 8 16 32 64; do
for THREADS in 1 4 16 32; do
echo THREADS=$THREADS
LD_PRELOAD=/usr/lib64/libjemalloc.so LD_LIBRARY_PATH=$WT_HOME/lib:$LD_LIBRARY_PATH $TIME ./
du $TEST_TMPDIR
done
echo "Running wiredtiger Btree"
rm -rf $TEST_TMPDIR/*
LD_PRELOAD=/usr/lib64/libjemalloc.so LD_LIBRARY_PATH=$WT_HOME/lib:$LD_LIBRARY_PATH $TIME ./
#for THREADS in 1 2 4 8 16 32 64; do
for THREADS in 1 4 16 32; do
echo THREADS=$THREADS
LD_PRELOAD=/usr/lib64/libjemalloc.so LD_LIBRARY_PATH=$WT_HOME/lib:$LD_LIBRARY_PATH $TIME ./
du $TEST_TMPDIR
done
exit
```

Rate limited version has `WRATE=30000`, unlimited version has `WRATE=0`