LevelDB VS Wiredtiger Benchmark

Introduction

The goal of this benchmark is to compare WiredTiger and LevelDB for the original LevelDB benchmark.

Configurations

The LevelDB code, including the WT benchmark found in doc/bench/db_bench_wiredtiger.cc, can be found here.

Hardware

- Amazon Web Services (AWS) High I/O instance:
- 60.5 GiB of memory
- 35 EC2 Compute Units (16 virtual cores*)
- 2 SSD-based volumes each with 1024 GB of instance storage
- 64-bit platform
- I/O Performance: Very High (10 Gigabit Ethernet)
- API name: `hi1.4xlarge`

Operating System

- Amazon Linux AMI x86_64 EBS (3.2.30-49.59.amzn1.x86_64)

File System

- XFS

Benchmark Characteristics

- Keys are 16 bytes each.
- Values are 100 bytes each (50 bytes after compression).

- Number of entries is 1000000.
- One operation, fill100K, uses values of 100Kb and 1000 entries.
- LevelDB standard random implementation (which does not guarantee uniqueness of keys in random tests). A discussion of this can be found here. Our experiments found that about 37% of the possible keys are never written or read during the random operations.

WiredTiger

We run all the tests for WiredTiger and LevelDB with a cache size of 128Mb. This size is not quite large enough to contain the one million 16 byte keys, 100 bytes values plus any overhead.

- Compression: Snappy compression
- LSM and Btree configuration (internal to db_bench_wiredtiger.cc): `"internal_page_max=16kb,leaf_page_max=16kb,lsm_chunk_size=20MB,block_compressor=snappy,memory_page_max=120795955"`
- Database configuration: `"key_format=S,value_format=S,prefix_compression=false,checksum=off"`
- Cursor configuration: `"bulk=true"` (only for the sequential write cursor)
- Arguments to db_bench_wiredtiger: `"--cache_size=134217728 [--use_lsm=0]"`
- WiredTiger release 1.4.2

To run the benchmark, compile the WiredTiger library for a production system (we use -O3). The benchmark defaults to running with LSM trees. Use the --use_lsm=0 argument to run with Btree. Compile the benchmark. You may need to send in the path to the WT library. Run with:

- [env LD_LIBRARY_PATH=] ./db_bench_wiredtiger --cache_size=134217728 [--use_lsm=0]

LevelDB

- Release 1.5
- Compression: Snappy compression
- Arguments to db_bench: `"--cache_size=134217728"`

Results

The raw data and scripts used to process it are here.

The entire benchmark was run five times for each database product. The benchmark runs some operations more than once. The results script uses the last instance of an operation in a run's output file as the final value. We eliminate the minimum and maximum value for each operation and then average the remaining three values.

Sequential Insert Performance

WiredTiger's Btree configuration performs 86% better than LevelDB for sequential insert, due to the effect of the bulk insert configuration option for the cursor. WiredTiger's LSM configuration performs 12% better than LevelDB for sequential inserts.

Random Insert Performance

WiredTiger's LSM configuration performs about 10% better than LevelDB for random inserts.

Random Overwrite Performance

WiredTiger's LSM configuration performs 76% better than LevelDB for random overwrites. WiredTiger's Btree configuration performs 46% better than LevelDB.

Random Large Value Performance

WiredTiger's Btree configuration achieves over 7 times the performance of LevelDB for randomly writing large values. WiredTiger's LSM configuration performs over 3 times better.

Sequential Read Forward and Reverse

WiredTiger's Btree implementation performs 71% better than LevelDB for sequential reads. WiredTiger's LSM configuration performs 47% better than LevelDB.

Both Btree and LSM configurations get more than double the operations/second of LevelDB for reading reverse. LevelDB suffers a significant performance hit of 54% in its reverse direction compared to its forward direction.

Random Read

WiredTiger's Btree configuration performs 122% better than LevelDB for random reads. WiredTiger's LSM configuration performs almost 60% better than LevelDB.