

RocksDB

A embedded key-value store for fast storage

Sunday, November 24, 2013

The History of RocksDB

The Past

It was mid 2011, I had been developing [HDFS/HBase](#) for five years and I was in love with the [Hadoop](#) ecosystem. Here is a system that can store and query hundreds of petabytes of data without blinking an eye. Hadoop has always focussed on scalability first and Hadoop's scalability was increasing by leaps and bounds. It was apparent that we will be able to store an exabyte of data in a single Hadoop cluster in a few years. I wanted to take on a new challenge...to see if we can extend HDFS's success story from Data Analytics to Query Serving workloads as well.

A Query Serving workload mostly consists of point lookups, random short range scans and point updates. The primary requirement of this workload is *low-latency* queries. On the other hand, a Big Data Analytics query typically involves large sequential scans and joins of two or more data sets with a very low volume of updates if any. Thus, the following year I spent comparing HBase/HDFS and [MySQL](#) for a Query Serving workload. The advantage of using HBase is that we can have multiple copies of data within a single data center. I wanted to determine what is needed to migrate a very large Query Serving workload from a cluster of MySQL servers to an HBase/HDFS cluster. This multi-petabyte dataset was stored on spinning disks. After multiple rounds of enhancements to HBase and were able to make it such that HBase latencies were only twice as slow as a MySQL server and HBase was using only three times more IOPs to serve the same workload on the same hardware. We were making steady progress towards our goal... but then something changed!

[Flash storage](#) became a reality. The data set migrated from spinning disks to flash. Now, the million dollar question that came up is whether HBase is capable of efficiently using flash hardware. I benchmarked HDFS and HBase with data on flash storage and posted the results in an [earlier post](#). The short story is that the HDFS/HBase of 2012 had a few software bottlenecks because of which it was not able to use flash storage efficiently. It became clear that if data is stored in flash storage, then we need a new storage engine to be able to serve a random workload on that data efficiently. I started to look out for techniques to build the next generation key-value store, especially designed to serve data from fast storage.

Why do we need an Embedded Database?

Flash is fundamentally different from spinning storage in performance. For the purpose of this discussion, let's assume that a read or write to spinning disk takes about 10 milliseconds while a read or write to flash storage takes about 100 microseconds. Network network-latency between two machines remains around 50 microseconds. These numbers are not cast in stone and your hardware could be very different from this one, but these numbers demonstrate the relative differences between two scenarios. What does this have anything to do with application-systems architecture? A client wants to store and access data from a database. There are two alternatives, it can store data on locally attached disks or it can store data over the network on a remote server that have disks attached to it. If we consider latency, then the locally attached disks can serve a read request in about 10 milliseconds. And in the client-server architecture, accessing the same data over a network results in a latency of 10.05 milliseconds, the overhead imposed by the network being only a miniscule 0.5%. Given this fact, it is easy to understand why a majority of currently-deployed systems use the client-server model of accessing data. (For the purpose of the discussion, I am ignoring network bandwidth limitations).

Now, let's consider the same scenario but with disks replaced by flash drives. A data access in the case of locally attached flash storage is 100 microseconds whereas accessing the same data via the network is 150 micros. Network data access is 50% higher overhead than

Blog Archive

- [2017](#) (2)
- ▼ [2013](#) (2)
 - ▼ [November](#) (1)
 - [The History of RocksDB](#)
 - [October](#) (1)

About Me



Dhruva Borthakur

Connect to me at
<http://www.facebook.com/dhruva>

[View my complete profile](#)

local data access and 50% is a pretty big number. This means that databases that run embedded within an application could have much lower latency than applications that access data over a network. Thus, the necessity of an [Embedded Database](#).

The above hypothesis does not state that client-server models will become extinct. The client-server-data-access-model has inherent advantages in the areas of data management and will continue to remain prominent in application deployment scenarios.

Aren't there any existing embedded databases?

Of course there are many existing embedded databases: [BerkeleyDB](#), [KyotoDB](#), [SQLite3](#), [leveldb](#), etc. [Open-source benchmarks](#) seem to state that [leveldb](#) is the fastest of the lot. But not all of them are suitable for storing data on Flash storage. Flash has limited write-endurance; updates to a block of data on Flash typically introduces write-amplification within the Flash driver. Given that we want to run our database on flash, we focussed on measuring write-amplification for evaluating databases technologies.

HBase and [Cassandra](#) are Log Structured Merge ([LSM](#)) style databases, but it will take a lot of engineering work to make HBase and Cassandra be an embeddable library. Both of them are servers with an entire ecosystem of built-in management, configuration and deployments. I was looking for a simple c/c++ library: [leveldb](#) was the apparent first choice for our benchmarking.

Why was [leveldb](#) insufficient for our purpose?

I started to benchmark [leveldb](#) and found that it was unsuitable for our server workloads. [leveldb](#) was a cool piece of work but was not designed for server workloads. The open-source [benchmark results](#) looks awesome at first sight, but I quickly realized that those results were for a database whose size was smaller than the size of RAM on the test machine, i.e. the entire database has to fit in the OS page cache. When I performed the same benchmarks on a database that was at least 5 times larger than main memory, the performance results was dismal.

[leveldb](#)'s single-threaded compaction process was insufficient to drive server workloads. Frequent write-stalls caused 99-percentile latency to be tremendously large. Mmap-ing a file into the OS cache introduced performance bottlenecks for reads. [leveldb](#) was unable to consume all the IOs offered by the underlying flash storage.

On the other hand, I was seeing server storage hardware evolve fast in different dimensions, For example, an experimental system where a storage volume is striped across 10 flash cards can provide upwards of a million IOs per second. A NVRAM based storage can support a few million data accesses per second. I would like to use a database that can drive these types of fast storage hardware. A natural evolution of flash storage could lead us to storage that has a very limited erase cycles and I envisioned that a database that can allow elegant tradeoffs of read amplification, write amplification and space amplification would be a dire necessity for these type of storage. [leveldb](#) was not designed to achieve these goals. The best path was to fork the [leveldb](#) code and change its architecture to suit these needs. So, [RocksDB](#) was born!

The vision for [RocksDB](#)

1. An embedded key-value store with point lookups and range scans
2. Optimized for fast storage, e.g. Flash and RAM
3. Server Side database with full production support
4. Scale linearly with number of CPU cores and with storage IOPs

[RocksDB](#) is not a distributed database. It does not have fault-tolerance or replication built into it. It does not know anything about data-sharding. It is upto the application that is using [RocksDB](#) to implement replication, fault-tolerance and sharding if needed.

Architecture of [RocksDB](#)

[RocksDB](#) is a C++ library that can be used to persistently store keys and values. Keys and values are arbitrary byte streams. Keys are stored in sorted runs. New writes occur to new places in the storage and a background compaction process eliminates duplicates and processes delete markers. There is support for atomically writing a set of keys into the database. Backward and forward iteration over the keys are supported.

[RockDB](#) is built using a "pluggable" architecture. This makes it easy to replace parts of it without impacting the overall architecture of the system. This architecture makes me confident that [RocksDB](#) will be easily tunable for different workloads and on different hardware.

For example, one can plug in various compression modules (snappy, zlib, bzip, etc) without changing any RocksDB code. Similarly, an application can plug in their own compaction filter to process keys during compaction; an example application can use it to implement a *expiry-time* for keys in the database. RocksDB has pluggable [memtables](#) so that an application can design a custom data structure to cache their writes, one example is prefix-hash-memtable where a portion of the key is hashed and the remainder of the key is arranged in the form of a btree. The implementation of a [sst](#) file is pluggable too and an application can design their own format for sst files. RocksDB supports a MergeType record that allows applications to build higher level constructs (Lists, Counters, etc) by avoiding a read-modify-write.

RocksDB currently supports two styles of compaction: the level style compaction and the universal style compaction. These two styles offers flexible performance tradeoffs. Compactions are inherently multi-threaded so that large databases can sustain high update rates. I will write a separate post on the pros and cons of these two styles of compaction.

RocksDB exposes interfaces for incremental online backup which is needed for any kind of production usage. It supports setting bloom filters on a sub-part of the key, which is one possible technique to reduce iops needed for range-scans.

RocksDB's apis are [stackable](#), which means that you can wrap lower level apis with higher level easy-to-use wrappers. This is the topic of a future post.

Potential Use-cases of RocksDB

RocksDB can be used by applications that need low latency database accesses. A user-facing application that stores the viewing history and state of users of a website can potentially store this content on RocksDB. A spam detection application that needs fast access to big data sets can use RocksDB. A graph-search query that needs to scan a data set in realtime can use RocksDB. RocksDB can be used to cache data from Hadoop, thereby allowing applications to query Hadoop data in realtime. A message-queue that supports a high number of inserts and deletes can use RocksDB.

The Road Ahead

There is work-in-progress to make RocksDB be able to serve data at memory speeds when the entire database fits in RAM. RocksDB would evolve to be compatible with highly multi-core machines. With the advent of super-low-endurance-flash storage, we expect RocksDB to store data with minimal write amplification. Maybe sometime in the future, someone will port RocksDB to the Android and iOS Platform. Another nice feature would be to support [Column Families](#) to support better clustering of related data.

I am hoping that software programmers and database developers would use, enhance and customize RocksDB for their use-cases. The code base is in <http://github.com/facebook/rocksdb>. You can join the Facebook Group <https://www.facebook.com/groups/rocksdb.dev> to participate in the engineering design discussions about RocksDB.

Posted by [Dhruba Borthakur](#) at [8:13 PM](#)

Labels: [database](#), [flash storage](#), [hbase](#), [lsm](#), [rocksdb](#)

Location: [Sunnyvale, CA, USA](#)

13 comments:

[Vinoth Chandar](#) November 26, 2013 at 2:31 PM

Just curious.. Did you benchmark this against BDB Java edition (which is log structured and embeddable)? We use it at massive scale for Voldemort @ linkedin.. Or that won't fly at all since the server code is (of course) in c/c++?

[Reply](#)



[Dhruba Borthakur](#) November 26, 2013 at 4:39 PM

There was a time when I used to keep the java-jni api updated, but then fell behind and finally deleted it from the rocksdb code base. We had one application that was using

rocksdb-jni but it switched to using rocksdb-via-thrift, so I could get rid of the rocksdb-jni code completely. If you look at older versions of the rocksdb repo, you will be able to find it.

I have not benchmarked against BDB Java. If you happen to benchmark and compare it, can you pl share the results so that I can improve rocksdb if it falls short on that workload.

[Reply](#)

Vinoth Chandar November 27, 2013 at 8:36 AM

sure ... I have a JNA binding going <https://github.com/vinothchandar/rocksdb-jna> and already plugged it into my benchmark tool as well.. Will do some tests and let you know.. <https://github.com/vinothchandar/kv-perf>

Anyways, nice job guys! keep you posted..

[Reply](#)



Dhruba Borthakur July 19, 2014 at 2:46 PM

Vinoth: as you already know, rocksdb now supports a full-fledged java api via <https://github.com/facebook/rocksdb/tree/master/java>, thanks to Ankit and YH. Have you been able to benchmark voldermort with a rocksdb storage engine?

[Reply](#)

rackmountpro March 10, 2015 at 1:33 AM

This comment has been removed by a blog administrator.

[Reply](#)

Anonymous June 6, 2017 at 1:54 AM

What is the levelDB version, is rocksDB still merging update from levelDB?

[Reply](#)



Tejuteju May 3, 2018 at 9:58 PM

Nice post Thankful to sharing [Big Data Hadoop Online course India](#)

[Reply](#)



Tejuteju May 3, 2018 at 10:06 PM

Nice post Thankful to sharing [Big Data Hadoop Online course India](#)

[Reply](#)



Dhruba Borthakur May 4, 2018 at 12:24 AM

RocksDB has taken a life of its own. Most code now is original RocksDB code and is not inherited code from LevelDB. The question of merging in new code changes from LevelDB does not arise anymore.

[Reply](#)



Unknown October 15, 2018 at 4:07 AM

It is really a helpful blog to find some different source to add my knowledge. I came into aware of new professional blog and I am impressed with suggestions of author.

[เขาพื้นที่เก็บของ](#)

[Reply](#)



Helish Lawera April 8, 2019 at 9:02 PM

I am thankful to this blog for assisting me. I added some specified clues which are really important for me to use them in my writing skill. Really helpful stuff made by this blog.

[הערות](#)

[Reply](#)

Anonymous [April 22, 2019 at 11:32 PM](#)

Hi

does in RocksDb each level contains many files, and is all files inside one level are non overlapping sorted

[Reply](#)

Anonymous [June 3, 2021 at 11:10 AM](#)

I have installed RocksDB using vcpkg. I am the 1st time user and never ran this before. Can anyone point me to right steps how to run this RocksDB upon installation? Thank You

[Reply](#)



Enter Comment

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

Simple theme. Powered by [Blogger](#).