Under the Hood: Building and open-sourcing RocksDB

By Dhruba Borthakur on Thursday, November 21, 2013 at 10:01am

Every time one of the 1.2 billion people who use Facebook visits the site, they see a completely unique, dynamically generated home page. There are several different applications powering this experience--and others across the site--that require global, real-time data fetching.

Storing and accessing hundreds of petabytes of data is a huge challenge, and we're constantly improving and overhauling our tools to make this as fast and efficient as possible. Today, we are open-sourcing RocksDB, an embeddable, persistent key-value store for fast storage that we built and use here at Facebook.

Why build an embedded database?
Applications traditionally access their data via remote procedure calls over a network connection, but that can be slow--especially when we need to power user-facing products in real time. With the advent of flash storage, we are starting to see newer applications that can access data quickly by managing their own dataset on flash instead of accessing data over a network. These new applications are using what we call an embedded database.

There are several reasons for choosing an embedded database. When database requests are frequently served from memory or from very fast flash storage, network latency can slow the query response time. Accessing the network within a data center can take about 50 microseconds, as can fast-flash access latency. This means that accessing data over a network could potentially be twice as slow as an application accessing data locally.

Secondly, we are starting to see servers with an increasing number of cores and with storage-IOPS reaching millions of requests per second. Lock contention and a high number of context switches in traditional database software prevents it from being able to saturate the storage-IOPS. We're finding we need new database software that is flexible enough to be customized for many of these emerging hardware trends.

The vision for RocksDB
RocksDB builds on LevelDB, Google's open source key value database library, to satisfy several goals:

1. Scales to run on servers with many CPU cores.
2. Uses fast storage efficiently.
3. Is flexible to allow for innovation.
4. Supports IO-bound, in-memory, and write-once workloads.

1. RocksDB scales to run on servers with many CPU cores
Commodity servers have many CPU cores, and it can be difficult to write a database server for which throughput increases with the CPU core count. Getting close to linear scaling is even harder. However, RocksDB has advantages that help it deal effectively with multi-core servers. One advantage is that the semantics it provides are simpler than a traditional DBMS. For example, it supports MVCC, but only for read-only transactions. Another advantage is that the database is logically partitioned into a read-only path and a read-write path. Both of these reduce the usage of locks, and reducing lock contention is a prerequisite for supporting high-concurrency workloads.

2. RocksDB makes efficient use of storage -- more IOPS, improved compression, less write wear
Modern storage devices support up to 100,000 random operations per second on a single flash storage card. If you stripe 10 of these cards, you get about 1 million random operations

Facebook Engineering

Notes by Facebook Engineering

All Notes

Get Notes via RSS

Embed Post

per second. Today, RocksDB is able to run fast enough to avoid being the bottleneck when operating on this type of storage.

Compared to an update-in-place B-tree, it is possible to get better compression and less write amplification with a write-optimized database like RocksDB. With better compression, you can use less storage; and with less write-amplification, flash devices will last longer and you may be able to use lower-endurance flash devices.

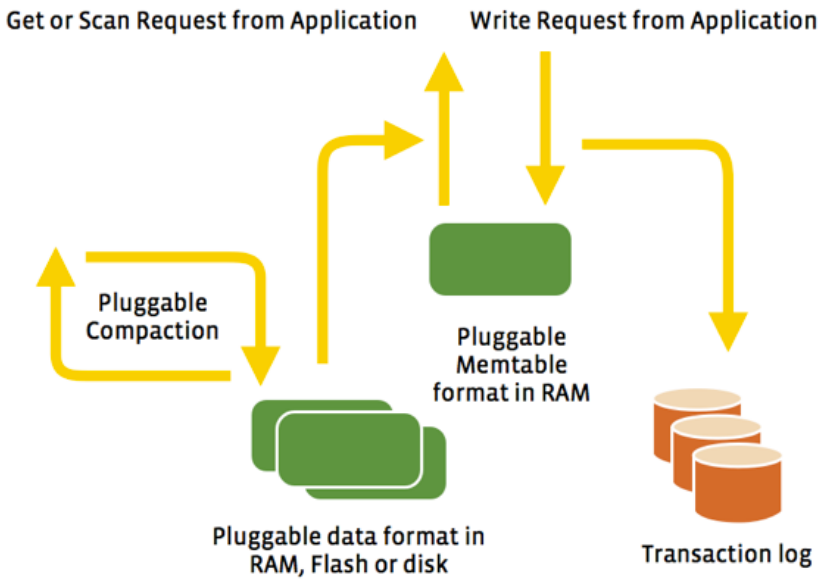
3. RocksDB has a flexible architecture to allow for innovation
RocksDB code is manageable and easy to extend. For example, we've added a merge operator that makes some updates write-only rather than read-modify-write, where "read" and "write" imply potential storage reads and writes. This reduces the amount of IO for some write-intensive workloads.

4. RocksDB supports IO-bound, in-memory, and write-once workloads
An IO-bound workload is one where the database size is much larger than memory and there are frequent reads from storage. An in-memory workload is one where the database fits entirely in memory and can still use storage to persist the database after changes. A write-once workload is one where the majority of the keys are written once or insert-only without updates. Today, RocksDB supports an IO-bound workload optimally. There has been a lot of work to make RocksDB better for in-memory support, but much work remains and we are just starting to figure out what support is needed for write-once.

RocksDB isn't a distributed database--rather the focus is on making an efficient, high-performance, single-node database engine.

Architecture of RocksDB
RocksDB is a C++ library that persistently stores keys and values. Keys and values are arbitrary byte streams, and keys are stored in sorted sequences. New writes occur to new places in the storage, and a background compaction process eliminates duplicates and processes delete markers. Data is stored in the form of a log-structured merge tree. There is support for atomically writing a set of keys into the database, and backward and forward iterations over the keys are supported.

RocksDB is built using a "pluggable" architecture, which makes it easy to replace parts of it without impacting the overall architecture of the system. We believe this architecture will allow users to tune RocksDB for different workloads and different hardware.



For example, you can plug in various compression modules (snappy, zlib, bzip, etc.) without changing any RocksDB code. This helps when setting up different compression algorithms for data in different workloads. Similarly, an application can plug in its own compaction filter to process keys during compaction. For example, one of our applications uses it to implement an "expiry-time" for keys in the database. RocksDB also has pluggable APIs so

that an application can design a custom data structure to cache its writes inside the database. One example of this is prefix-hash, where a portion of the key is hashed and the remainder of the key is arranged in the form of a B-tree. The implementation of a storage file is also pluggable, so an application can design its own format for storage files.

RocksDB currently supports two styles of compaction: “level style” compaction and “universal style” compaction. These two styles offer flexible tradeoffs with respect to read amplification, write amplification, and space amplification. Compactions are inherently multi-threaded so that large databases can sustain high update rates.

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

Performance
RocksDB software can fully utilize the IOPS offered by flash storage, making it perform faster than LevelDB across random read, write, and bulk uploads. We've seen it perform 10 times faster for a pure random write workload and a bulk upload, as well as 30% faster for a pure random read workload. You can see all our performance benchmark results for server-side workloads on flash storage here.

We found that LevelDB's single-threaded compaction process did not work well for certain server workloads, and that RocksDB outperforms LevelDB for these IO-bound workloads. In our measurements, we saw frequent write-stalls with LevelDB that caused significant 99-percentile latency. Our testing also found that mmap-ing a file into the OS cache introduced performance bottlenecks for reads. We saw that applications were not able to derive the full potential of the flash hardware because of data bandwidth bottleneck caused by LevelDB's high write-amplification. By increasing write rate and lowering write-amplification, we were able to avoid many of these issues and improve performance with RocksDB.

Typical workloads RocksDB is suitable for
RocksDB can be used by applications that need low-latency database accesses. Some examples include:

1. A user-facing application that stores the viewing history and state of users of a website.
2. A spam-detection application that needs fast access.
3. A graph-search query that needs to scan a data set in real time.
4. An app that needs to query Hadoop in real time.
5. A message-queue that supports a high number of inserts and deletes.

Our use cases for RocksDB have grown tremendously, and we have close to a petabyte of data across different applications being managed by RocksDB today. We're excited to release RocksDB to the community and hope people will find it as useful as we have.

Open-sourcing the code
Our code is now live at http://github.com/facebook/rocksdb, and we hope that software programmers and database developers will use, enhance, and customize RocksDB for their own use cases. We’re looking forward to hearing your feedback and continuing to improve RocksDB and add more features. Check out our RocksDB Facebook Group to join the conversation.

ปอ อีกแบรนด์คือ Babara Basic และ เสื้อผ้า แฟชั่น สำหรับคนอ้... See More
See Translation

ร้านแมลงปอ ! เสื้อผ้าแฟชั่น จำหน่ายขายส่งเสื้อผ้า เสื้อแฟชั่นราคาส่ง

ขายส่ง ! เสื้อผ้าแฟชั่น จำหน่ายขายส่งเสื้อผ้า เสื้อแฟชั่นราคาส่ง เสื้อผ้า...

YOUTUBE.COM

April 15 at 9:54pm


Radu Hambasan Is RocksDB using Snappy as its default compression engine?
January 11 at 12:57pm


غريب الدار
August 25, 2014 at 1:39pm


Vinnie Falco Good stuff
August 21, 2014 at 4:17pm


Jbetewi Gara Jbetewi Cooll.....
January 17, 2014 at 10:03pm


Shi Yudi great
December 10, 2013 at 12:34am


Adam Iskandar wtf is this
December 8, 2013 at 8:07pm


Pax Zoe Favored i need help with a tecnichal question .
December 7, 2013 at 9:51am


Qiu Billow 真心niubility
See Translation
November 23, 2013 at 12:05am


Nathan Duke Judas Priest, I read this headline and thought they were basing their new app on Adobe Flash. Almost swallowed my tongue. "Facebook's latest open source effort: a flash-powered database called RocksDB"
November 22, 2013 at 12:43am


Karthik Rajasekaran RocksDB rocks!
November 21, 2013 at 3:45pm


Leslie Mashadza This amazing, i wish to member of this team, this great
November 21, 2013 at 11:52am

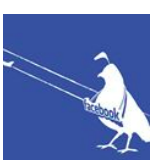

Siddharth Kulkarni Distributed Key Value stores have been getting better and better at performance. At Facebook's scale, this looks like a good solution. I would like to know though, which parts of facebook still run a Relational Database?
November 21, 2013 at 10:30am

2 Replies


Aravind S Murthy Pure Amalagamation of storage and db
November 21, 2013 at 10:24am


Opay Ashraf https://www.facebook.com/groups/Engineer.4.ever/
November 21, 2013 at 10:17am


Kaiypov Emil ANTI-MILITARY "SPAM" Dear Facebook Engineering Team https://www.facebook.com/Engineering! Please support Global truce - Pause all wars anti-war initiative! Best wishes from Bishkek, Kyrgyzstan!!! Launch of the First Anti-military satellite. ...
See More



Facebook Engineering
Computers/Technology

1,287,429 Likes
4,645 talking about this

January 8 at 1:47pm


Without Engineers, Science Is Just Philosophy https://www.facebook.com/pages/Without-Engineers-Science-Is-Just-Philosophy/491380210895062?ref=hl
December 7, 2013 at 2:08am