Lighting Memory-Mapped Database (LMDB)

LEONARD MWANGI[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: lmwangi@iu.edu

paper 2 April 12, 2017

LMDB is one of the most recent in-memory key-value databases, it has shown performance not matched by many other key-value databases and a unique capability to manage and reclaim unused space. We'll examine the database, it's architecture, features and use cases that makes it ideal for self-contained application in need a local small-footprint database.

Keywords: Cloud, I524, LMDB, BDB, B+tree

https://github.com/cloudmesh/sp17-i524/tree/master/paper2/S17-IO-3013/report.pdf

INTRODUCTION

Lightning Memory-Mapped Database (LMDB) is a high-performance transaction non-relational database that stores data in the form of key-value store [1] while using memory-mapped file capabilities to increase I/O performance. Designed to solve multiple layers of configuration and caching issues inherent on Berkeley DB (DBD) design [2], LMDB fixes caching problems with a single, automatically managed cache which is controlled by the host operating system [3]. The database footprint is small enough to fit in an L1 cache [4] and its designed to always append data at the end of the database (MDB_APPEND) thus there is never a need to overwrite data or do garbage collection. This design protects the database from corruption in case of a system crash or need to carry out intensive recovery process.

LMDB is built on combination of multiple technologies dating back to 1960s. Memory-Mapped files or persistent object concept was first introduced by Multics in mid 1960s as single-level storage (SLS) [5] which provided distinction between files and process memory this technology incorporated in LMDB architecture. LMDB also utilizes B+tree architecture which provides a self-balancing tree data structure making it easy to search, insert and delete data due to its sorting algorithm. LMDB specifically utilizes the append-only B+tree code written by Martin Hedenfalk [6] for OpenBSD ldapd implementation. The architecture is also modeled after BDB API and perceived as it's replacement.

ARCHITECTURE

The following section provides an overview of LMDB architecture and the technologies involved.

Language

LMDB is written in C but the API supports multiple programming languages with C and C++ supported directly while other languages like Python, Ruby, Erlang amongst others supported using wrappers [7].

B+tree

The append-only B+tree architecture ensures that that the data is never overwritten thus every time modification is required, a copy is made and changes made to the copy which is in turn written to a new disk space. To reclaim the freed space, LMDB uses a second B+tree which keeps track of pages that have been freed thus keeping the database size fairly the same. This is a major architectural advantage compared to other B+tree based databases.

Figure 2 Regular B+tree architecture.

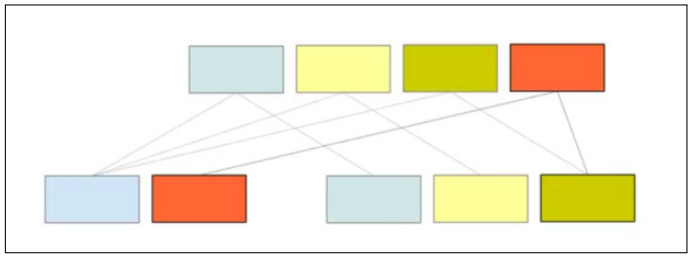

Fig. 1. For every new leaf-node created, a root-node is created

Figure 2 LMDB B+tree architecture.

Transaction

The database is architected to support single-writer and many-readers per transaction [8] ensuring no deadlock within the database. No memory allocation (malloc) or memory-to-memory copies (memcpy) required by the database thus ensuring that locks do not occur during

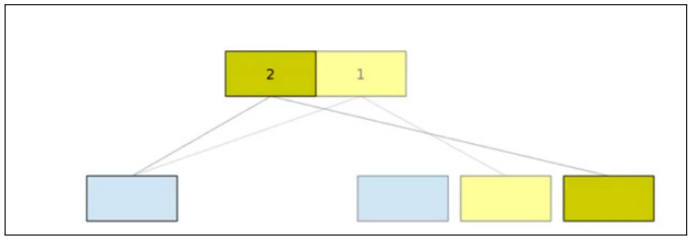

Fig. 2. LMDB uses two root-nodes for valid leaf-nodes as shown in diagram above. Unused leaf-nodes do not have root-nodes tied to them which signifies unused page that can be reclaimed

IMPLEMENTATION

LMDB implementation is supported by both Windows and Unix operating systems. It is provided in two variants which are automatically selected during installation. CFFI variant supports PyPy for CPython version 2.6 or greater and C extensions that support CPython versions 2.5 through 2.7 and version 3.3 and greater. For ease of adoption, both versions have the same interface. This installation guide will focus on Python based implementation for both operating systems. At the moment 32-bit and 64-bit binaries are provided for Python 2.7. Future binaries are expected to support all versions of Python.

Installation

Installation and configuration process for LMDB.

```
#Unix based Systems
pip install lmdb
#or
easyinstall lmdb
#Create and open and environment
 mdb_env_create() mdb_env_open()
 #defines the directory path and must
 exist before being used.
 #mdb_nosubdir option can be used if no
 directory path needs to be passed.
#Create transactions after opening the
# environment
 mdb_env_create() mdb_env_open()
 #defines the directory path and must
 exist before being used.
 #mdb_nosubdir option can be used if no
 directory path needs to be passed.
#Open database for the transaction
#mdb_dbi_open() #NULL value can be passed
    if only a single database
    will be used in the environment.
#mds_create flag must be specified to
     create a named database.
#mdb_env_set_maxdbs() defines the maximum
    number of databases the
    environment will support.
#mdb_get() and #mdb_put() can be
    used to store a single key/value pairs,
    if more transactions are needed ver
    then cursors are required.
#mdb_txn_commit() is required to
    commit the data in the environment
```

FEATURES

LMDB has the following features that are not in a regular key/value store databases:

1. Explicit Key Types Key comparisons in reverse byte order as well as native binary integer supported, this goes beyond the regular key/value string comparisons and memory-to-memory copy.

2. Append Mode Useful when bulk loading the data which is already a sorted format otherwise it would lead to data corruption.

3. Fixed Mapping Data can be stored in a fixed memory address where applications connecting to the database will see all the objects with the same address.

4. Hot Backups Since LMDB uses MVCC, a hot backup can be carried while the database is live because transactions are self-consisted from beginning to the end.

LICENSING

LMDB is licensed under OpenLDAP public license, a BSD style licensing.

ALTERNATIVES

There are several alternatives to LMDB, focusing on key-value databases, some of the leading alternatives include the following:

SQLite3

One of the alternatives to LMDB, an in-process database designed to be embedded in the applications rather than having its own server. The main comparison with LMDB is they are both in-memory databases but SQLite3 is also a relational database with structured data in form of tables where is LMDB is not a relational database [9].

Oracle Berkeley DB (BDB)

A key-value database provided in form of software libraries offers an alternative to LMDB. LMDB is actually molded from DBD thus structure and architecture is almost the same. BDB is considered to be fast, flexible, reliable and scalable database and has the ability to access both key and sequential data. BDB does not require a stand-alone server and is directly integrated to the application [2].

Google LevelDB

LevelDB is an on-disk key-value open source database developed by google fellows and inspired by BigTable memtable/sstable architecture. One of the main differences from LMDB is LevelDB utilizes disk for storage rather than memory which can have negative impact on performance due to disk seeks [10]. Also, being an on-disk database, it is susceptible to corruption especially when there is system crash. LevelDB is licensed under BSD Licensing model [11].

Kyoto Cabinet

Provided in form of libraries, is an on-disk key-value database with B+tree and hash tables architecture. Kyoto Cabinet is written in C++ and has APIs for other languages [12]. Disk management has been identified as an issue for Kyoto Cabinet [13]. It's also licensed under General Public Licensing.

USE CASE

NoSQL Data Stores

Due to its small footprint and great performance, LMDB is an ideal candidate for NoSQL data stores. The data store is widely used for caching, queue-ing, tasks distribution and pub/sub to enhance performance. Being an in-memory database, LMDB is a great candidate for these stores and is currently being used as Redis data store [13].

Mobile Phone Database

In the wake of smart phones, the need for mobile applications to collect and store data has grown exponentially. Most of the mobile applications store this data in cloud requiring them to connect back and forth in order to facility user's request. Having a local lightweight self-contained database on the device has become an attractive solution to enhance user experience with performance and effectiveness. LMDB fills in this gap due to its small footprint, performance and ability to reuse storage thus making it a favorable mobile phone database.

Postfix Mail Transfer Agent (MTA) Database

Postfix is an SMTP Server that provides first layer of spambots and malware defense to the end users. Having the ability to track and keep history of the scans and identified threats at a fast rate is paramount for the success of Postfix. Postfix uses LMDB adapter to provide access to lookup tables and make data available to the application reliably [14].

CONCLUSION

Self-managing, small footprint and well performing database is a critical to many client side applications. These applications are better suited when they can offload storage management to the database application and gain great performance. LMDB has shown these capabilities by managing how data is written and claiming unused storage. It's great performance and small footprint stages it well for many of the application. In comparison to other databases in its class, LMDB benchmarks [15] shows its capability to be the leading database on in self-contained application databases. Also, being available as an OpenLDAP public licensing and having wrappers for different languages makes it easier to port to already existing applications.

ACKNOWLEDGEMENTS

This research was done as part of course "I524: Big Data and Open Source Software Projects" at Indiana University. I thank Professor Gregor von Laszewski and associate instructors for their support throughout the course.

REFERENCES

[1] Aerospike, "What is a key-value store?" WebPage. [Online]. Available: http://www.aerospike.com/what-is-a-key-value-store

[2] K. B. Margo Seltzer, "Berkeley db," WebPage, Aug. 2012. [Online]. Available: http://www.aosabook.org/en/bdb.html

[3] H. Chu, "Lightning memory-mapped database," WebPage, Nov. 2011. [Online]. Available: https://en.wikipedia.org/wiki/Lightning_Memory-Mapped_Database

[4] WhatIs, "L1 and l2," WebPage, Apr. 2005. [Online]. Available: http://whatis.techtarget.com/definition/L1-and-L2

[5] Wikipedia, "Multics," WebPage, Feb. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Multics

[6] J. Mayo, "btest.c," WebPage, Jan. 2012, originary written by Martin Hedenfalk. [Online]. Available: https://github.com/OrangeTide/btree/blob/master/btest.c

[7] Symas Corporation, "Wrappers for other languages," WebPage. [Online]. Available: https://symas.com/offerings/lightning-memory-mapped-database/wrappers/

[8] H. Chu, "Lightning memory-mapped database manager (lmdb)," Dec. 2015. [Online]. Available: http://www.lmdb.tech/doc/

[9] D. Hellmann, "sqlite3 – embedded relational database," WebPage, Jan. 2017. [Online]. Available: https://pymotw.com/2/sqlite3/

[10] Basho Technologies, Inc., "Leveldb," WebPage. [Online]. Available: http://docs.basho.com/riak/kv/2.2.1/setup/planning/backend/leveldb/

[11] Wikipedia, "Leveldb," WebPage, Mar. 2017. [Online]. Available: https://en.wikipedia.org/wiki/LevelDB

[12] fallabs, "Kyoto cabinet: a straightforward implementation of dbm," WebPage, Mar. 2011. [Online]. Available: http://fallabs.com/kyotocabinet/

[13] B. Desmond, "Second strike with lightning!" WebPage, May 2013. [Online]. Available: http://www.anchor.com.au/blog/2013/05/second-strike-with-lightning/

[14] H. Chu, "Postfix lmdb adapter," WebPage. [Online]. Available: http://www.postfix.org/lmdb_table.5.html

[15] Symas Corp, "Database microbenchmarks," WebPage, Sep. 2012. [Online]. Available: http://www.lmdb.tech/bench/microbench/