# 背景

RocksDB是FaceBook起初作为实验性质开发的，旨在充分实现快存上存储数据的服务能力。由Facebook的Dhruba Borthakur于2012年4月创建的LevelDB的分支，最初的目标是提高服务工作负载的性能，最大限度的发挥闪存和RAM的高度率读写性能。

Key和value是任意大小的字节流支持原子的读和写。除此外，RocksDB深度支持各种配置，可以在不同的生产环境（纯内存、Flash、hard disks or HDFS）中调优，RocksDB针对多核CPU、高效快速存储（SSD)、I/O bound workload做了优化，支持不同的数据压缩算法、和生产环境debug的完善工具。

RocksDB的主要设计点是在快存和高服务压力下性能表现优越，所以该db需要充分挖掘Flash和RAM的读写速率。 例如360开源的Pika， 是由360 DBA 和基础架构组联合开发的类 Redis 存储系统，完全支持 Redis 协议，用户不需要修改任何代码，就可以将服务迁移至 Pika，Pika底层存储引擎用的就是Rocksdb。

RocksDB需要支持高效的point lookup和range scan操作，需要支持配置各种参数在高压力的随机读、随机写或者二者流量都很大时性能调优，基于LSM树数据结构( log-structured merge-tree)，由C++编写并官方提供C、C++、Java(官方提供的称为RocksJava)三种语言的API，社区提供了不少第三方API，如python、go等。

尽管RocksDB不是一个SQL 数据库，但是有facebook有修改了代码的MyRocks存储引擎作为MySQL的存储引擎。和其他的NoSQL类似，RocksDB不提供关系型数据模型、不支持SQL查询，没有直接对辅助索引(secondary indexes)支持。

# 概述

参考：

<https://blog.csdn.net/weixin_43618070/article/details/102317769>

## 简介

[RocksDB](https://github.com/facebook/rocksdb) 是由Facebook基于LevelDB开发的一款提供键值存储与读写功能的LSM-tree架构引擎。用户写入的键值对会先写入磁盘上的WAL (Write Ahead Log)，然后再写入内存中的跳表（SkipList，这部分结构又被称作MemTable）。LSM-tree引擎由于将用户的随机修改（插入）转化为了对WAL文件的顺序写，因此具有比B树类存储引擎更高的写吞吐。

内存中的数据达到一定阈值后，会刷到磁盘上生成SST文件(Sorted String Table)，SST又分为多层（默认至多6层），每一层的数据达到一定阈值后会挑选一部分SST合并到下一层，每一层的数据是上一层的10倍（因此90%的数据存储在最后一层）。

Rocksdb目前已经运用在许多知名的项目中，例如TiKV，MyRocks，CrockRoach等。

### ColumnFamily

RocksDB允许用户创建多个ColumnFamily ，这些ColumnFamily各自拥有独立的内存跳表以及SST文件，但是共享同一个WAL文件，这样的好处是可以根据应用特点为不同的ColumnFamily选择不同的配置，但是又没有增加对WAL的写次数。

写操作先写WAL，再写memtable，memtable达到一定阈值后切换为Immutable Memtable，只能读不能写。后台Flush线程负责按照时间顺序将Immu Memtable刷盘，生成level0层的有序文件(SST)。后台合并线程负责将上层的SST合并生成下层的SST。Manifest负责记录系统某个时刻SST文件的视图，Current文件记录当前最新的Manifest文件名。 每个ColumnFamily有自己的Memtable， SST文件，所有ColumnFamily共享WAL、Current、Manifest文件，用户可以基于RocksDB构建自己的column families。很多应用程序把RocksDB当做库(libary),尽管他提供server或者CLI接口。

### Memtable

可插拔memtable，RocksDB的memtable的默认实现是一个skiplist。skiplist是一个有序集，当工作负载使用range-scans并且交织写入时，这是一个必要的结构。然而，一些应用程序不交织写入和扫描，而一些应用程序根本不执行范围扫描。对于这些应用程序，排序集可能无法提供最佳性能。因此，RocksDB支持可插拔的API，允许应用程序提供自己的memtable实现。

开发库提供了三个memtable：skiplist memtable，vector memtable和前缀散列（prefix-hash）memtable。Vector memtable适用于将数据批量加载到数据库中。每个写入在向量的末尾插入一个新元素; 当它是刷新memtable到存储的时候，向量中的元素被排序并写出到L0中的文件。前缀散列memtable允许对gets，puts和scans-within-a-key-prefix进行有效的处理。

### SSTFile(SSTTable)

RocksDB在磁盘上的file结构sstfile由block作为基本单位组成，一个sstfile结构由多个data block和meta block组成，其中data block就是数据实体block，meta block为元数据block，其中data block就是数据实体block，meta block为元数据block。

sstfile组成的block有可能被压缩(compression)，不同level也可能使用不同的compression方式。sstfile如果要遍历block，会逆序遍历，从footer开始。RocksDB是一个嵌入式的K-V（任意字节流）存储。所有的数据在引擎中是有序存储，可以支持Get(key)、Put（Key）、Delete（Key）和NewIterator()。RocksDB的基本组成是memtable、sstfile和logfile。memtable是一种内存数据结构，写请求会先将数据写到memtable中，然后可选地写入事务日志logfile。logfile是一个顺序写的文件。当内存表溢出的时候，数据会flush到sstfile中，然后这个memtable对应的logfile也会安全地被删除。sstfile中的数据也是有序存储以方便查找。

## 特点

### 优点

1. LevelDB是一个持久化存储的KV系统，和Redis这种内存型的KV系统不同，LevelDB不会像Redis一样狂吃内存，而是将大部分数据存储到磁盘上。
2. LevleDB在存储数据时，是根据记录的key值有序存储的，就是说相邻的key值在存储文件中是依次顺序存储的，而应用可以自定义key大小比较函数。
3. LevelDB支持数据快照（snapshot）功能，使得读取操作不受写操作影响，可以在读操作过程中始终看到一致的数据。
4. LevelDB还支持数据压缩等操作，这对于减小存储空间以及增快IO效率都有直接的帮助。

### 缺点

rocksdb是通过wal来保证数据的持久性的，当rocksdb出现问题当机后，可以通过重做wal来使rocksdb恢复到当机前的状态。但是这里存在两个问题：

1、当你为了读性能把memtable设置的足够大时，WAL也可能变得很大（Flush频率下降），此时如果发生当机，rocksdb需要足够长的时间来恢复。

2、如果机器硬盘出现损坏，wal被破坏，那么会出现数据损坏（这里没有double write机制？）。即使你做了Raid，也需要很长时间来恢复数据。

因此可用性是个大问题。

另外由于LSM架构，rocksdb的读性能存在问题。

怎么解决？

为了一致性，大部分的解决方案都是将一致性协议置于rocksdb之上，每份数据通过一致性协议提交到多个处于不同机器的rocksdb实例中，以保证数据的可靠性和服务的可用性。

## LevelDB

RocksDB与LevelDB对比：

1. 增加了column family，这样有利于多个不相关的数据集存储在同一个db中，因为不同column family的数据是存储在不同的sst和memtable中，所以一定程度上起到了隔离的作用。
2. 采用了多线程同时进行compaction的方法，优化了compact的速度。
3. 增加了merge operator，优化了modify的效率。
4. 将flush和compaction分开不同的线程池，能有效的加快flush，防止stall。
5. 增加了对write ahead log(WAL)的特殊管理机制，这样就能方便管理WAL文件，因为WAL是binlog文件。

# 架构

# 原理

## 写流程

rocksdb写入时，直接以append方式写到log文件以及memtable，随即返回，因此非常快速。  
 memtable/immute memtable触发阈值后，flush到Level0 SST，Level0 SST触发阈值后，经合并操作(compaction)生成level 1 SST，level1 SST合并操作生成level 2 SST，以此类推，生成level n SST。

## 读流程

按照memtable --> Level 0 SST–> Level 1 SST --> … -> Level n SST的顺序读取数据。这和记录的新旧顺序是一的。因此只要在当前级别找到记录，就可以返回。

# 应用场景

参考：

<https://www.jianshu.com/p/3302be5542c7>

RocksDB的典型场景（低延时访问）:

1. 需要存储用户的查阅历史记录和网站用户的应用
2. 需要快速访问数据的垃圾检测应用
3. 需要实时scan数据集的图搜索query
4. 需要实时请求Hadoop的应用
5. 支持大量写和删除操作的消息队列