LevelDB 学习交流-Part One

2011.09.23

淘宝解伦

OUTLINE

- 背景简介
- 代码组织
- 主体架构
- 读写特性
- 问题讨论

What Is LevelDB

- LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values.
  - A Fast And Lightweight KV Storage Library , Not A DataBase

- Reference
  - http://code.google.com/p/leveldb/
  - http://leveldb.googlecode.com/svn/trunk/doc/index.html
  - http://leveldb.googlecode.com/svn/trunk/doc/impl.html

Three Tags

- NoSQL Movement
  - KV Store
- Open Source
  - New BSD License
- Made In Google
  - Authors

    - also develop GFS\MapReduce\Bigtable\Protocol Buffers
    - More Info
      - http://research.google.com/people/jeff
      - http://research.google.com/people/sanjay/

Birth for HTML5

- IndexedDB
  - HTML 5 客户端存储 Object storage
  - History: Cookies -> Web storage -> SQL Database
  - 已有执行标准之间妥协的产物
  - More Info
    - http://www.html5rocks.com/en/tutorials/indexeddb/todo/
    - http://nparashuram.com/ttd/IndexedDB/index.html
    - http://soft.chinabyte.com/database/270/11200270.shtml

Leveldb Vs BigTable

- 关系和区别

Compare	LevelDB	BigTable
Implement	LSM Storage	
Operation Mode	Embedded DB	DB Manage Server
Topology	One Machine	Distributed Server
Data Model	KV Store	Column Store
Schema	No Schema	Semi-Schema Free

Storage Category

- Embedded Store
  - DB: SQLite、 BerkeleyDB、 innondb
  - KV: tokyo/kyoto cabinet、 riak bitcask
- Storage Server
  - RDBMS
  - NoSQL
    - tc&tt、kc&kt 、flare
    - bigtable、 habase、hypertable
    - mongodb、couchdb、orientdb
    - dynamo 、voldmort、 cassandra
    - neo4j、flockdb

Source Code


8
如果你的生命只剩下8分钟
源代码
SOURCE CODE
8月30日 拯救未来
它的创意如此之有趣，
以致让你无暇思考。
这是《偷天情缘》和
《盗梦空间》的结合体
《波斯王子》
杰克·吉伦哈尔
《恩眼》
米歇尔·莫娜汉
《狐儿盗》
维拉·法米加

Code Dir.

- util---------common cache、crc、hash
- port--------multiple os support
- include----c/ c++ api
- table------- block、sstable reader/writer
- db ----------memtable、commit log、snapshot、compaction

LevelDB Architecture

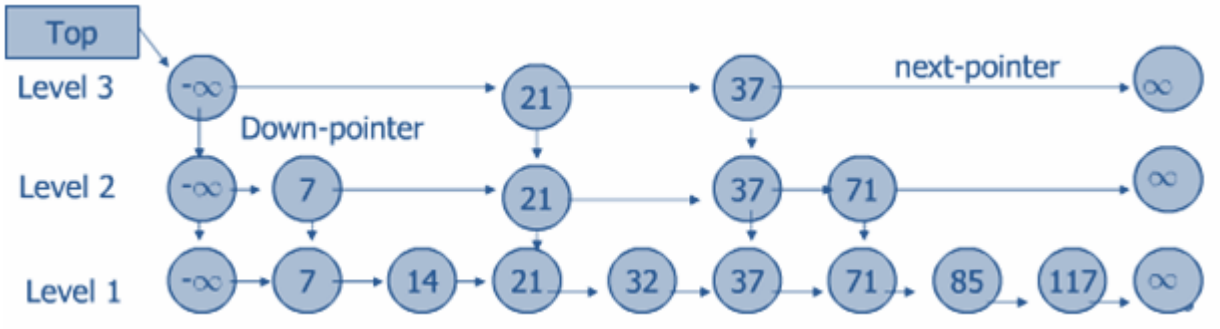
- Log Structured Merge Tree (LSM)
  1. 更新记Commit Log，写到in-memory table；
  2. Memory数据延迟落地到磁盘，不断打patch；
  3. 定期将磁盘数据sstable 在后台进行Compact.

  – 充分考虑磁盘的特性，将随机IO写变为顺序IO，牺牲随机读来优化写

  – Reference http://academic.research.microsoft.com/Paper/299876.aspx

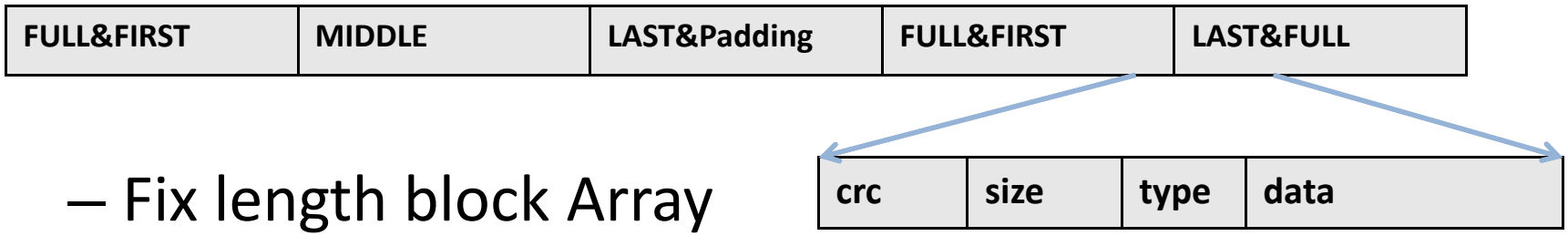
MEMTABLE

- Dynamic Data Struct
  - Order by (Key + Sequence)
  - frozen + active
  - Skiplist implemention
    - 写操作是insert到list,没有删除list node操作
    - 这种场景指针操作顺序保证单写多读线程安全



LOG

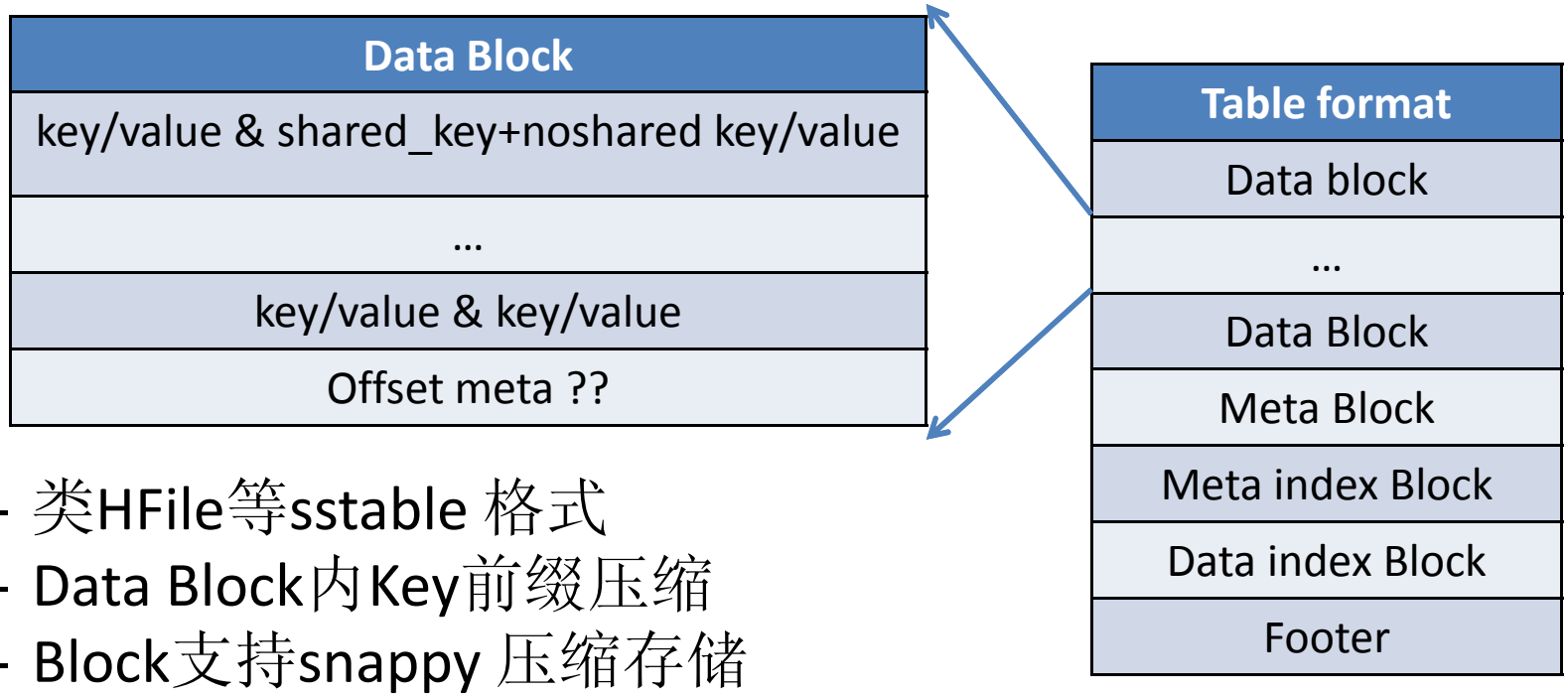
- Commit Log File Format



  - Fix length block Array
    - Pros
      - 可以过滤出错记录，跳跃到正确的BLOCK
      - Find next FULL or FIRST type record
    - Cons
      - 不支持压缩存储，空间有少量浪费（trailer）
  - 每个定长block包含一条或者多条日志记录

SSTABLE

- Static Table File Format



  - 类HFile等sstable 格式
  - Data Block内Key前缀压缩
  - Block支持snappy 压缩存储
  - 只有一个meta block

COMPACTION

- Multiple Level Compaction
  - 此level 非彼level (skiplist)
  - Input + Output
    - Level 0: 无序多路归并
    - Level n: 有序两路归并
  - New数据从Level 0逐渐向Level N搬迁
  - 每层的新数据总量达到10^i时触发
  - 同一key的多次更新、部分删除被合并

Compaction Analogy

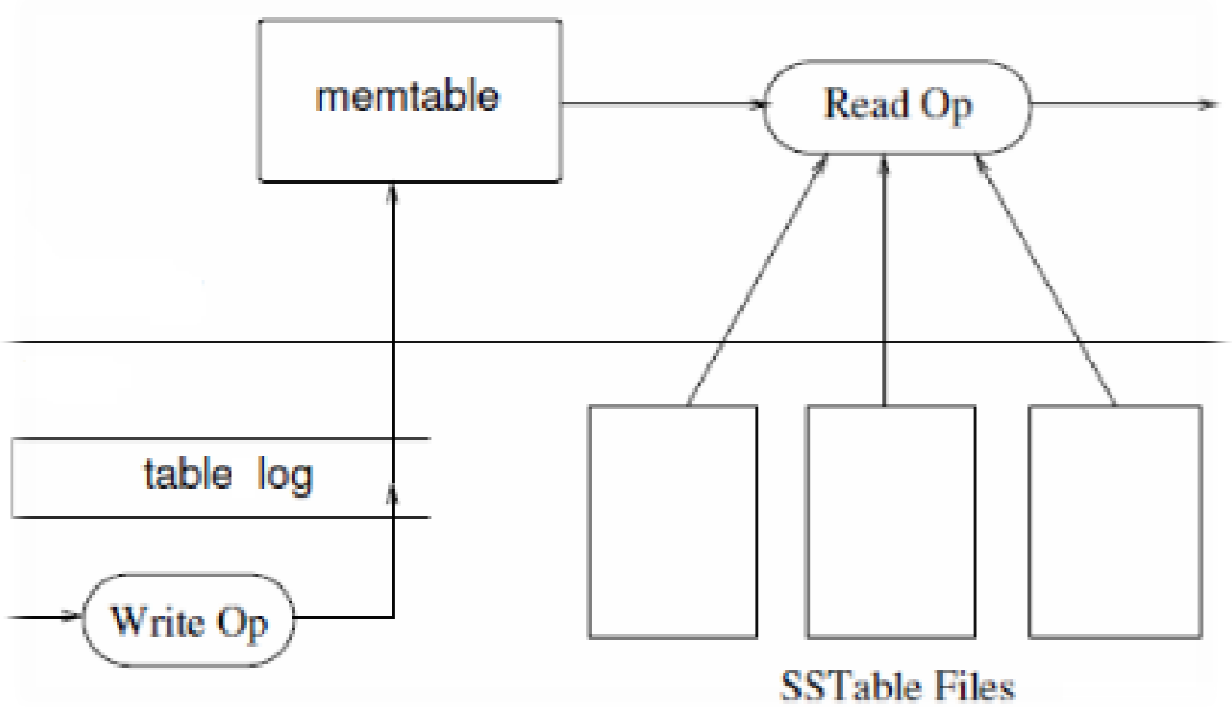
BigTable	LevelDB	
Minor Compaction	Memtable -> Level 0	Level 0 Compaction
Merge Compaction	Level i + level i+1 -> Level i+1	Level 1 ~ N Compaction
Major Compaction	No	

- Compaction BG Task
  - 属于IO密集型任务对于正常的随机顺序读有较大影响
  - 触发条件选择、带宽占比、速度控制都至关重要

Multi-Level Data

- Multi-level SSTable Data Layout
  - Level 内部
    - Level 0 sst 内部有序， sst之间无序，key有重叠
    - Level N sst内部有序， sst之间有序，key无重叠
  - Level 之间
    - 新旧数据，Patch关系

Write / Read



Write

- 顺序/随机写
  - 顺序记commit Log + 更新到active Memtable
  - 顺序写场景的compaction进行IO优化

- 纠结中权衡
  - LevelDB将随机写转为顺序写，牺牲了随机读性能的同时又利用策略控制写速率，甚至阻塞写以防止随机读latency太差。

Random Read

- 随机读
  - 从新到旧反向Read数据
    - 结合KV特点，短路式查询
    - Query Path: active_mem -> frozen_mem -> (last->first) Level 0 sst -> Level i sst -> Level i+1 sst
    - 最坏情况下需要Level + 1 次随机IO
    - 查询在range范围内并不存在的key很杯具

Sequential Read

- 顺序扫描
  - 2 Memtables + 多Level归并
    - Level 0所有sstable，其他Level部分sstable归并
    - 大部分操作是对下层sstable数据的顺序扫描
    - 最坏情况下需要合并Memtable 和所有level的数据
    - Level 0 sstable越多，level层次越多，性能越差？

ACID 特性

- 没有提供事务相关接口，需要上层封装
  - 原子性（Atomic）
    - Write Ahead Log
    - commit log要么成功，要么失败成为bad record
  - 一致性（Consistency）
    - Not Relational
  - 隔离性（ Isolation ）
  - 持久性（Durability）
    - 可配置的 sync方式 Not flush every write

ACID 特性

- 隔离性（Isolation）
  - 第三级隔离性可重复读
    - 先写log后更新Memtable未提交事务不会读到
    - 由sequence可以保证两次读Memtable一致性
    - 如何保证读写线程安全
      - 写写线程：由log writer condition锁控制
      - 读写静态数据无需同步
      - 读写动态数据主要包括
        - » cache ： 多线程安全
        - » memtable ： 单线程写多线程读

Cache & Arena

- Cache Sharding
  - 按key做分片，分片数固定
  - 每个分片是功能独立的cache (hashtable + LRU)
  - 在分片内部加锁降低锁冲突

- Memtable Arena
  - 内存池，只负责分配一次性回收
  - 定长 block + 变长的block 链表
    - 如需新block而待分配内存超过block_size/4时，直接分配相应大小的block

Performance

- Pros.
  - 高性能的随机顺序写
  - SSD性能优异，容易Scale Up
- Cons.
  - 随机读性能没有优势
  - 读写比率和频率需适用
  - 大Value性能较差

- More Info
  - http://leveldb.googlecode.com/svn/trunk/doc/benchmark.html
  - https://github.com/m1ch1/mapkeeper/wiki/MySQL-vs.-LevelDB
  - http://blog.basho.com/2011/07/01/Leveling-the-Field/

LeveDB Discuss

- LevelDB 定位
  - IndexedDB是客户端存储
    - 胖客户端是趋势
    - 不关心内存管理
      - 只有Memtable 使用了Arena内存池
      - 怀疑会有类似Redis内存碎片问题
  - 互联网服务器应用是否适用
    - 用户对写性能并不敏感，对读操作延时相反
    - 而LSM牺牲了随机读的性能，但写盘方式适合SSD
    - SSD恰可弥补随机读性能问题

Optimize Discuss

- 关于优化的讨论
  - increase Memtable Size
    - 增加随机读命中memtable的概率
    - 减少level 0的文件数
      - 提高Scan顺序扫描以及compaction的多路归并性能
    - 降低compaction频率，降低对读稳定性影响
  - Log sync Strategy (Bad Record Skip)
  - Multiple Disk Support When Compaction
  - Add Bloom Filter as sstable size increase
  - Less Level for speed up Read

Acknowledgement

Thanks!
欢迎指正讨论 @淘宝解伦