Security Level:

蟑螂数据库

程广卫 @华为技术有限公司

www.huawei.com

开源软件能力中心

开源软件
研究/跟踪

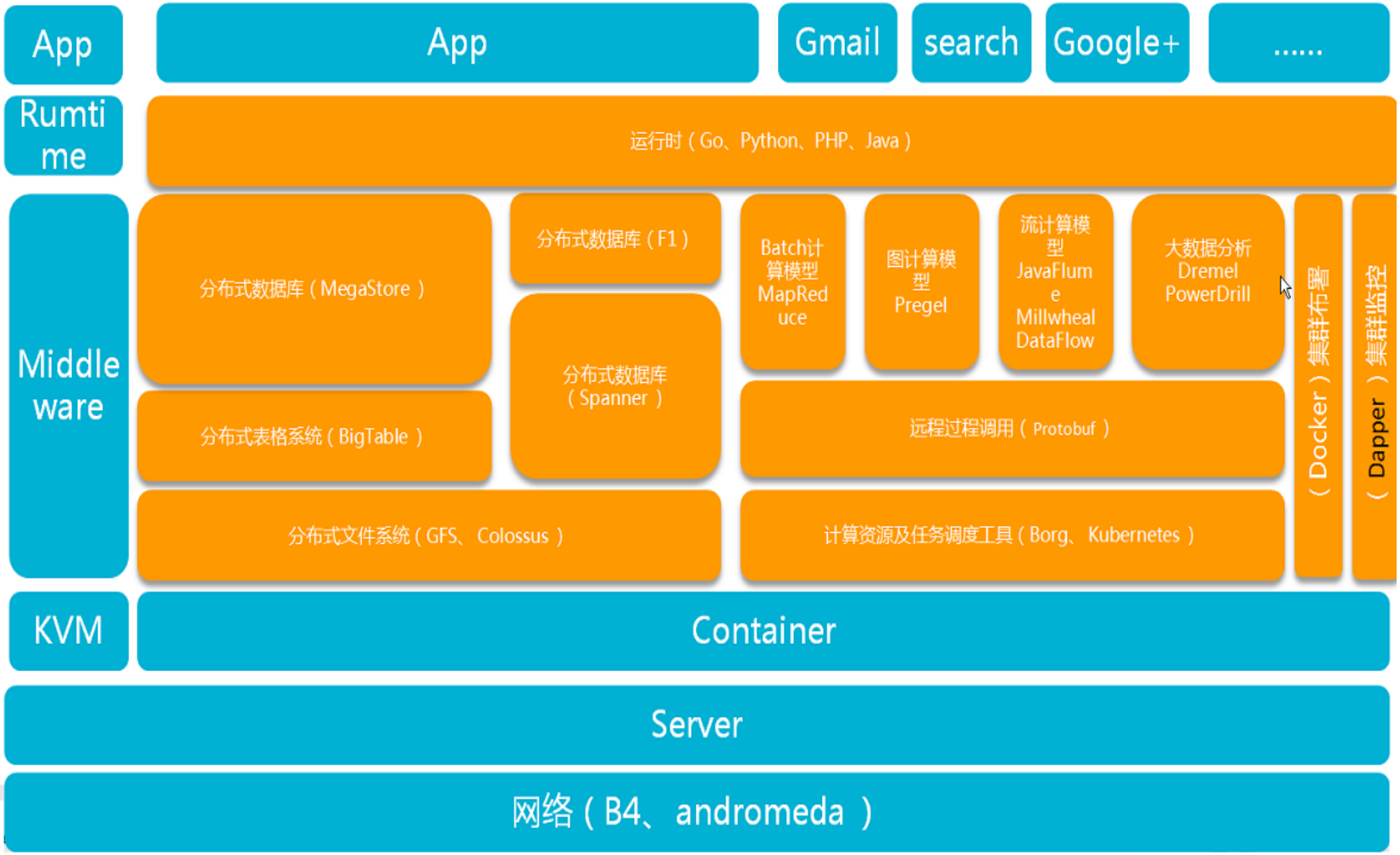
开源软件
开发/贡献

Committer

Google 业务和技术发展

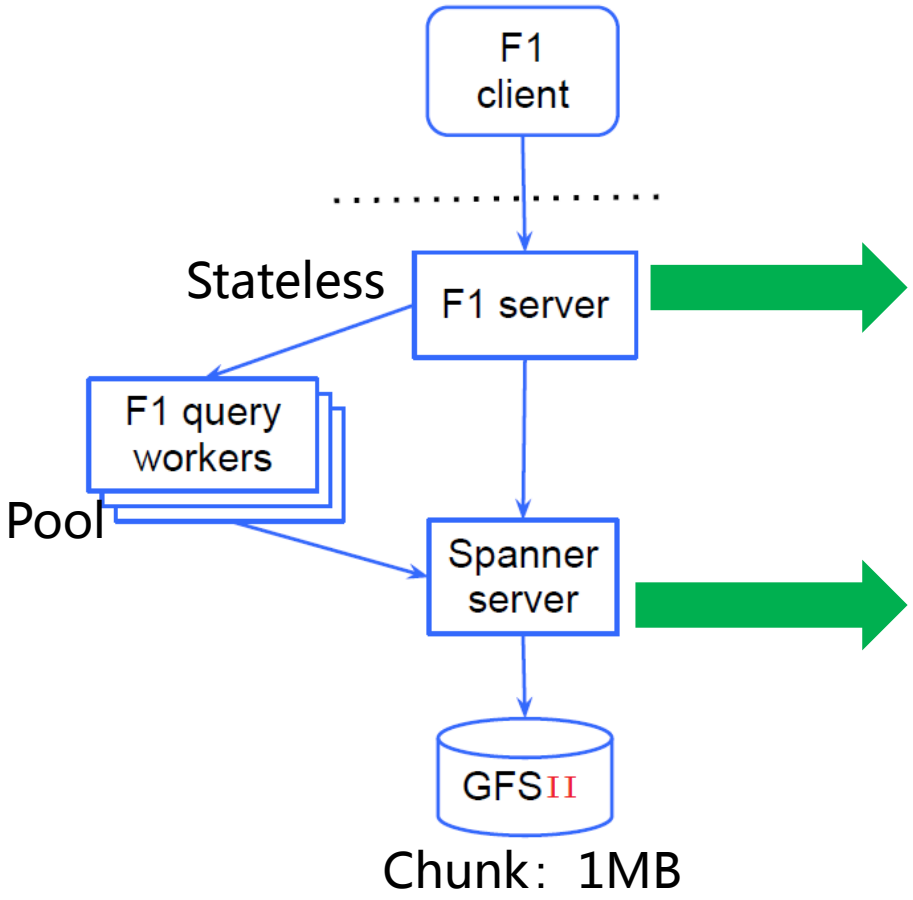
1997-2003年 分布式+批处理	2003-2008年 数据中心 as 计算机	2008-现在 实时&搜索习惯的变更
Goole文本搜索、图片搜索、地图等 **业务规模**：网页搜索数达到80亿，8.8亿图片 **数据中心**：租用为主	搜索向专有领域扩展：新闻/财经/专利等 向社交领域扩展：Blogger/google+ **业务规模**：一万亿个独立网址、直至150+种语言翻译 **数据中心**：开始投资构建自己的数据中心	搜索向实时性、数据分析后的推送扩展：实时搜索、GoogleNow、Knowledge Graph等 **业务规模**： **数据中心**：全球13个数据中心，管理近200万台计算机
海量数据的搜集、存储 快速响应用户的搜索请求	多数据中心的数据管理、数据中心管理针对WEB2.0应用的计算模型的变化 快速响应用户的搜索请求	用户信息/搜索习惯/位置信息的综合分析 数据中心管理能力/存储能力/计算能力/分析能力向用户开发 实时响应用户的搜索请求
MapReduce(2004) GFS Bigtable(2006)	MegaStore(2011)	Dataflow(2014) Spanner(2012) F1(2013) Mesa(2014)

Google系统架构


App
App
Gmail
search
Google+
.....
Rumtime
运行时（Go、Python、PHP、Java）
Middleware
分布式数据库（MegaStore）
分布式数据库（F1）
Batch计算模型MapReduce
图计算模型Pregel
流计算模型JavaFlume Millwheal DataFlow
大数据分析Dremel PowerDrill
（Docker）集群布署
（Dapper）集群监控
分布式表格系统（BigTable）
分布式数据库（Spanner）
远程过程调用（Protobuf）
分布式文件系统（GFS、Colossus）
计算资源及任务调度工具（Borg、Kubernetes）
KVM
Container
Server
网络（B4、andromeda）

Google Spanner

源于Bigtable，Megastore的替代者，F1的存储引擎



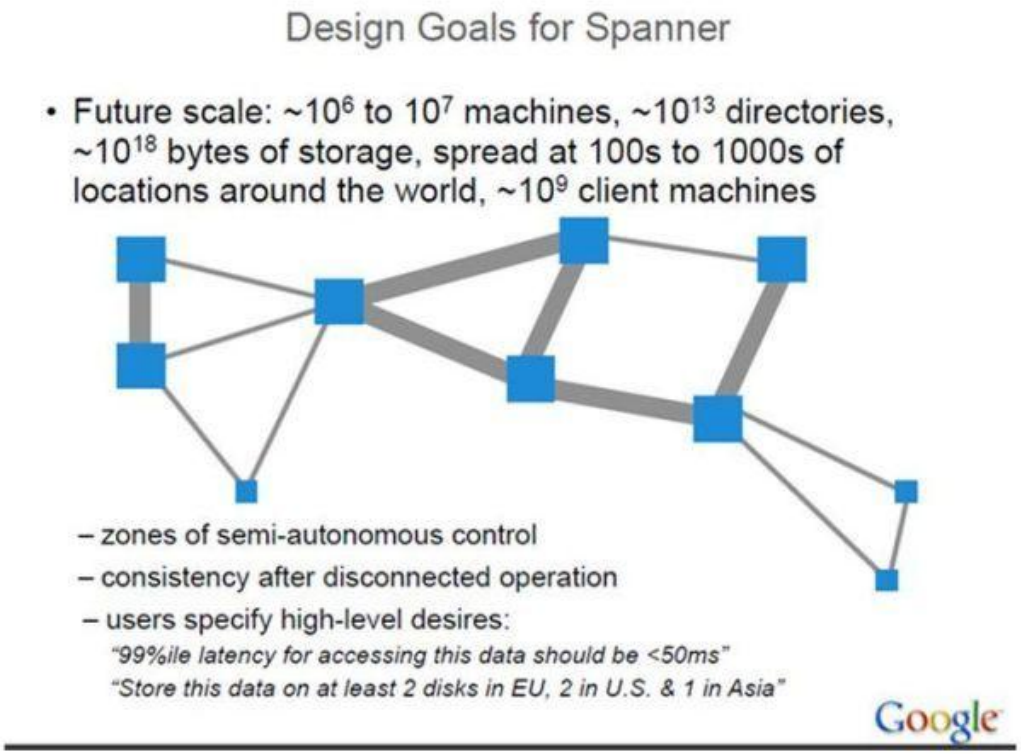
- 关系型 schema
  - 扩展数据类型支持
  - 无阻塞更改schema
- *一致性索引*
- *并行读，支持SQL 和Map-Reduce*
- 7×24高可用（甚至数据中心挂掉）
- 事务提交延迟50-100ms
- 读延迟5-10ms
- 高吞吐量

- 全球分布式存储
- 跨数据中心同步复制
- 透明Sharding和数据移动
- 支持事务
  - 单原子写&多次读
  - 单机事务
  - 分布式事务（两阶段提交）
- 快照读

Google Spanner 特点

可扩展的
基于时间戳的多版本
全球分布式
同步复制
应用可控副本存储

Spanner能 做到这些，离不开一个用GPS和原子钟实现的时间API。这个API能将数据中心之间的节点时间同步精确到10ms以内。



Spanner 主要技术

- 基于时间戳的多版本并发控制（MVCC）
- 应用可控的数据多副本跨数据中心存储
- TrueTime api

Spanner Server结构

- 时间戳进入Key值存储
  - 新旧版本数据共存
  - 历史版本数据用于快照
  - 需要定期清理历史数据

(Key: string, timestamp: int64) ->string。

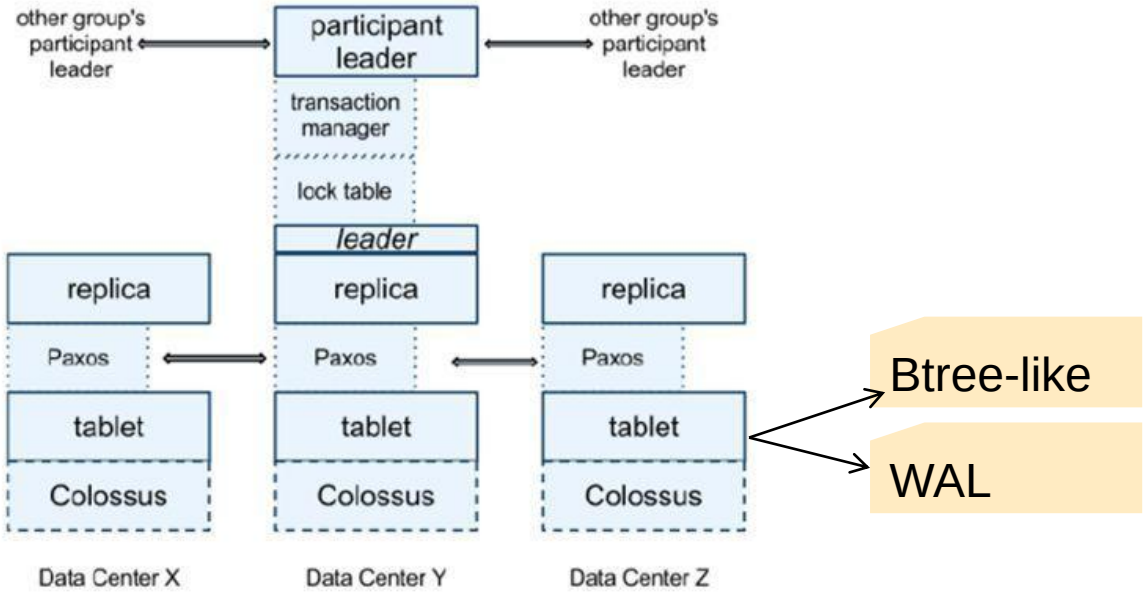


Figure 2: Spanserver software stack.

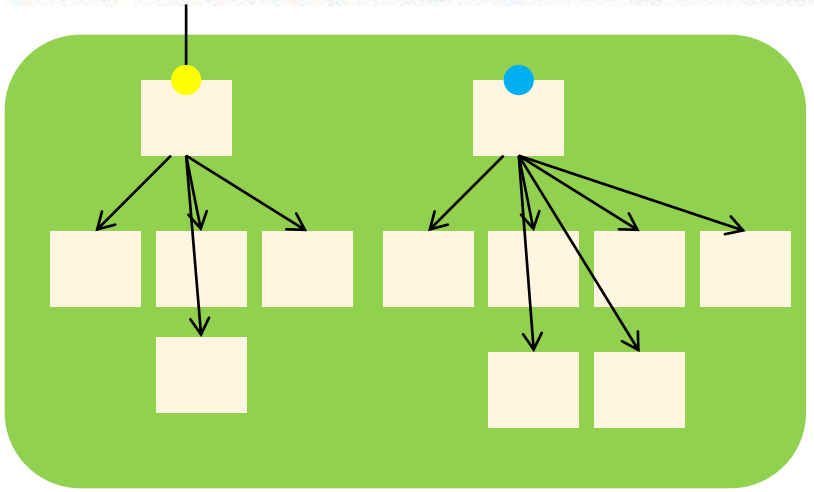
Datacenter:Clossus:Tablet:Paxos
1:1:N(100-1000):N

Spanner TrueTime

- 物理时钟
  - GPS
  - 原子钟
- 多Master
  - 互备
  - 互相校准
- 时间戳带误差

Method	Returns
TT.now()	*TTinterval*: [*earliest*, *latest*]
TT.after(*t*)	true if *t* has definitely passed
TT.before(*t*)	true if *t* has definitely not arrived

Table 1: TrueTime API. The argument *t* is of type *TTstamp*.



Spanner MVCC并发控制

Operation	Concurrency Control	Replica Required
Read-Write Transaction	pessimistic	leader
Read-Only Transaction	lock-free	leader for timestamp; any for read
Snapshot Read, client-provided timestamp	lock-free	any
Snapshot Read, client-provided bound	lock-free	any

- 读写事务
- 只读事务
- 快照读，客户端提供时间戳
- 快照读，客户端提供时间范围

读写事务：**会将所有的写操作先缓存起来**，在Commit的时候一次提交

读事务：首先要指定一个读事务时间戳。还需要了解在这个读操作中，需要访问的所有的读的Key。

CockroachdB使命（Mission）

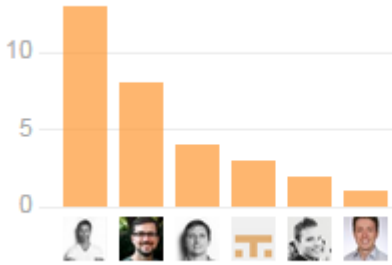
打造开源版本 F1

CockroachDB 的灵感来自于一份 Google 的研究论文，“Spanner”的大型系统及F1分布式数据库。

CockroachDB 并没有尝试复制 Spanner 最不寻常的理念——用原子钟来让全球各地的数据中心时间同步。考虑到大多数线上应用都没有达到 Google 的规模，他们或许不需要这样的功能。真正需要的是是有一种稳定可靠的方式来让数据自动复制和同步到各个数据中心的服务器里，这样就算一个数据中心倒下了，应用还能正常运行，这也是 CockroachDB 的目标。

项目现状

- 官方网站，http://cockroachdb.org/
- 代码托管：https://github.com/cockroachdb
- 项目统计
  - 2000+人加星
  - 236人订阅邮件列表 遍布全球各地，包括苹果/Google等知名公司人员



核心成员介绍

Andrew Bonventre

最出名的 Photoshop 开源替代产品 GIMP 的的联合创始人 Spencer Kimball 和 Peter Mattis 曾帮助开发 Google 的大型文件存储系统，也就是 Colossus；设计文档作者 Ben Darnell 曾参与过 Google Reader 开发，而 Andy Bonventre 则参与过 Chrome 和 Google Tasks 的开发。
Peter Mattis 是Kimball同学及同事

工作经历

Forward Deployed Engineer
Palantir Technologies
2014 年 8 月 – 现在 (1 个月) | 美国 大纽约地区

Palantir

Head of Product Engineering
Poptip
2012 年 9 月 – 2014 年 8 月 (2 年) | 美国 大纽约地区

1 个项目

Head of Mobile Development
Stamped
2011 年 7 月 – 2012 年 3 月 (9 个月)

Within approximately four months, I implemented the iOS client app of Stamped, which was featured on the App Store and chosen as one of Mashable's "15 Best Mobile Apps of 2011".

Software Engineer
Google
2006 年 5 月 – 2011 年 6 月 (5 年 2 个月)

Launched
- Chrome Extensions for the Mac
- Google Forms
- Google Tasks
- Google New (re

Tobias Schottdorf

工作经历

IT Consultant, Software Engineer
IT Freelancer
2013 – 现在 (1 年)

consulting and full stack web engineering. I love nodeJS and the neat stuff that comes with it (think angularJS, meteor), eat map-reduce for breakfast and have a vivid interest in using the JS stack to make your mobile development platform independent where it makes sense. I "speak" a bunch of other languages and know that creating a good product is a function of minds and not keyboards.

Technical Lead
Shippo
2014 年 3 月 – 2014 年 6 月 (4 个月) | 美国 旧金山湾区

Backend development (REST-based API on Python/Django/PostgreSQL/Redis/Celery stack, some gevent/Tornado/nodeJS/Ruby). Focus on scalability.
During Batch8@500Startups.

1 个组织

jiangmingyang

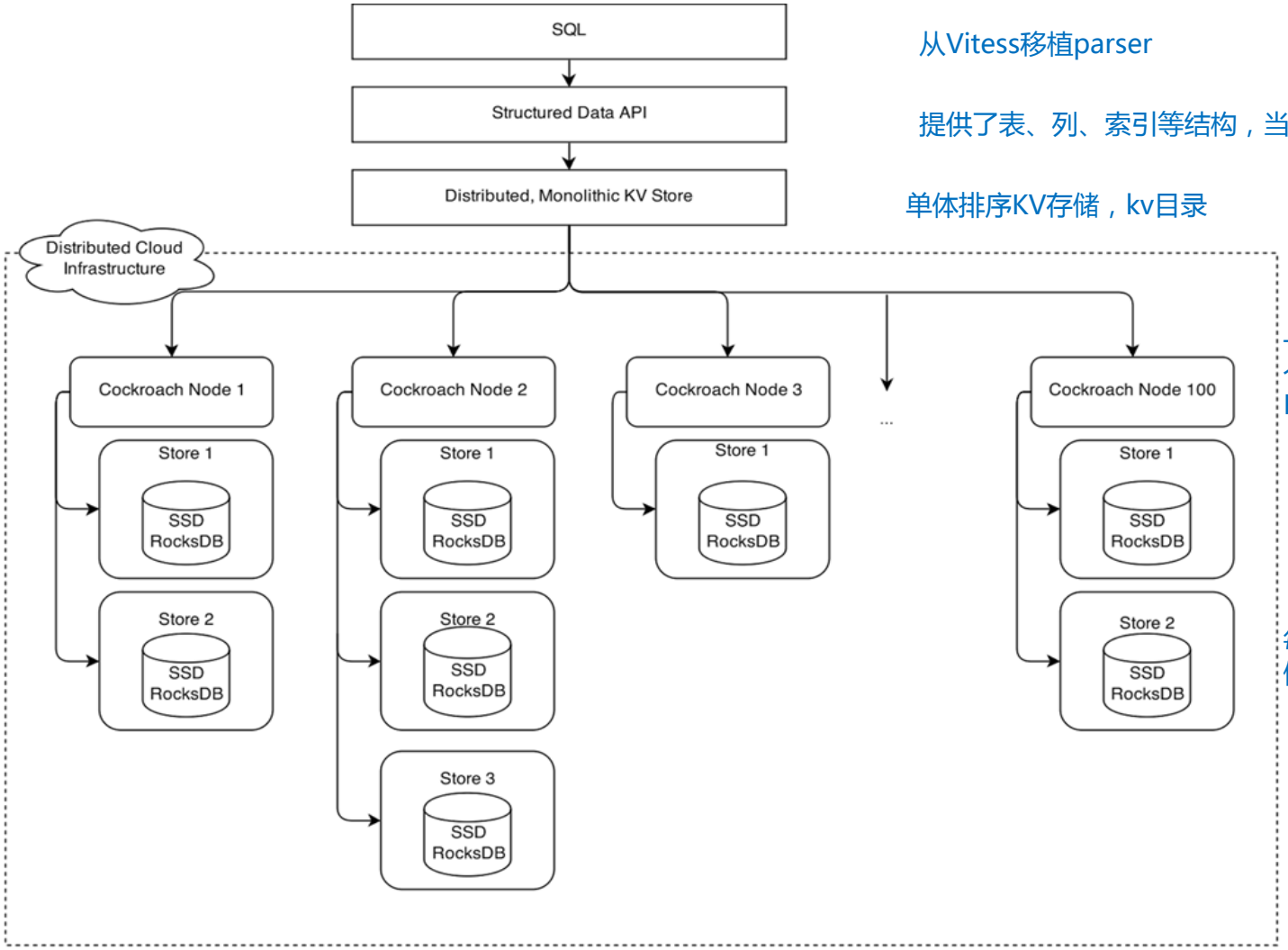
Spencer Kimball

Ben Darnell

项目使用的其他开源项目

- 开发语言：GoLang
- Docker：支持Docker部署
- Etcd：基于Raft实现跨数据中心的MutliRaft
- RockDB：高效K/V存储
- ……

总体架构



从Vitess移植parser

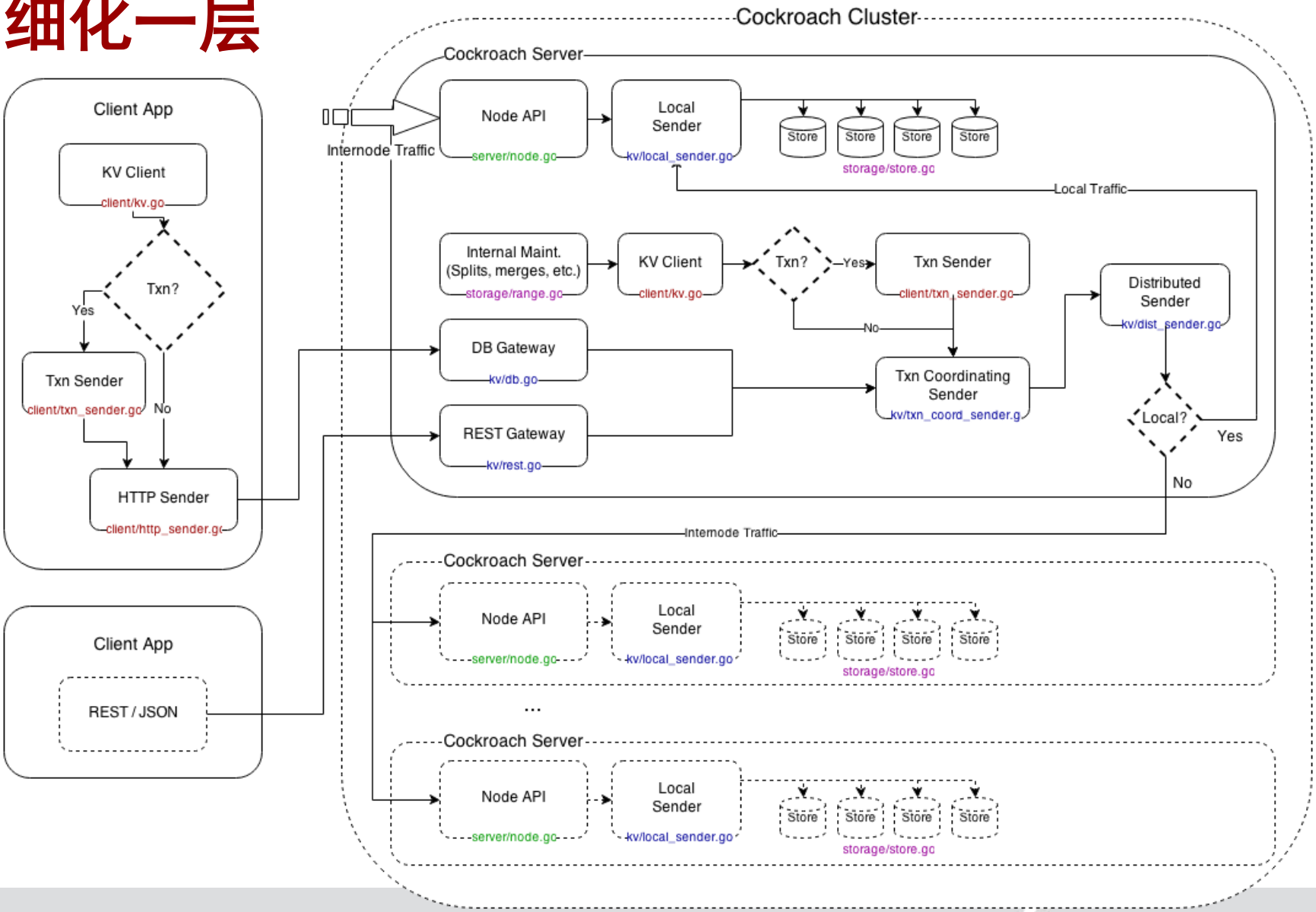
提供了表、列、索引等结构，当前不成熟

单体排序KV存储，kv目录

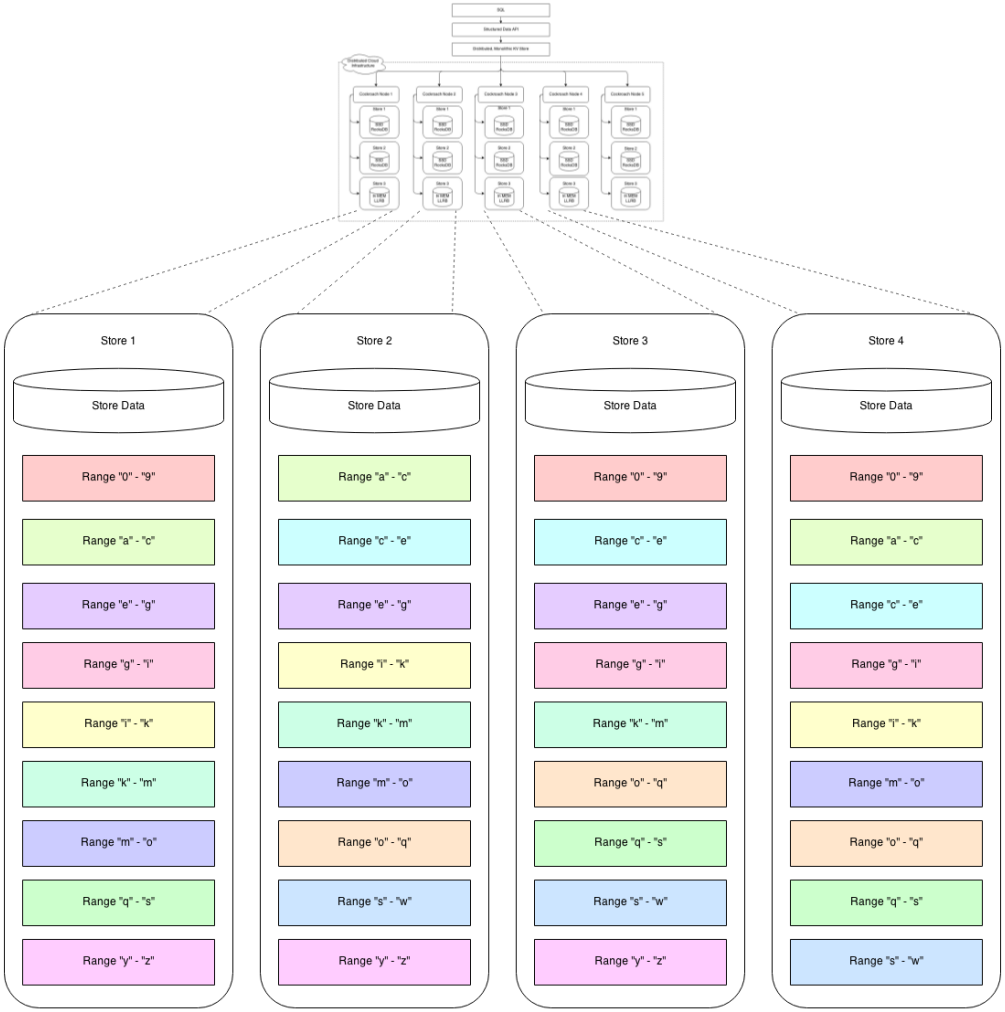
一个Node对应多个Store，即RocksDB实例

每个Store中存储多个Range

细化一层


Cockroach Cluster
Cockroach Server
Client App
KV Client
client/kv.go
Txn?
Yes
No
Txn Sender
client/txn_sender.go
HTTP Sender
client/http_sender.go
Internode Traffic
Node API
server/node.go
Local Sender
kv/local_sender.go
Store
Store
Store
Store
storage/store.go
Local Traffic
Internal Maint. (Splits, merges, etc.)
storage/range.go
KV Client
client/kv.go
Txn?
Yes
Txn Sender
client/txn_sender.go
No
Distributed Sender
kv/dist_sender.go
DB Gateway
kv/db.go
REST Gateway
kv/rest.go
Txn Coordinating Sender
kv/txn_coord_sender.g
Local?
Yes
No
Internode Traffic
Cockroach Server
Node API
server/node.go
Local Sender
kv/local_sender.go
Store
Store
Store
Store
storage/store.go
...
Cockroach Server
Node API
server/node.go
Local Sender
kv/local_sender.go
Store
Store
Store
Store
storage/store.go
Client App
REST / JSON

基于Range前缀的数据分布



- Cockroach对上层暴露一个已排序的分布式KV Map
- 业务数据和系统内部数据（事务、Schema等）一同存放于KV Map，使用不同的Key前缀进行区分隔离
- 数据使用Range Partition。每个Range缺省64MB。当Range的数据量和请求量超过或低于阈值时，会进行Range分裂或合并
- 支持Range Partition是支持上层SQL层的基础，如果使用传统的Hash、DHT等方式，则无法有效提供范围查询的能力，只适合进行并行统计类计算
- 数据支持多副本存储，放置位置可由策略决定，Range是数据复制的基本单位

两级Range Metadata

- 两级元数据分别使用\0\0meta1、\0\0meta2作为前缀
- Range元数据的Value其实是RangeDescriptor，包括：
  - Raft_id、start_key、end_key、replicas[]（nodeid、storeid、range_id）
- Range0的地址信息使用Gossip传递，其他所有Range的Metadata均存储在Cockroach中
- 查询某个key（如key1）所在的位置时
  - 在Range0所在的meta1中查找“\0\0meta1<key1>”对应的meta2 range（如右图中，小于lastkeyN-1的key的meta2元数据都在Range0中，大于其的都在Range1中）
  - 在meta2对应的Range中，查找“\0\0meta2<key1>”对应的Range
  - 通过得到的RangeDesc得到Range对应的nodeid、storeid等，并从Gossip中获取nodeid对应的节点地址
- 在数据量小于16TB时，Range数量小于2^{18}，这样meta1和meta2可以保存在一个Range中
- 理论数据容量：每个Range默认64MB（2^{26}），一个Range的Metadata为256Bytes（2^{8}），这样一个Metadata Range最多负责2^{26-8}个Range，然后两级Metadata可以负责2^{18+18}个Range，即$2^{18+18+26}$Bytes，即2^{62}Bytes——4EB

Range 0

Key	Value
\0\0meta1<lastkeyN-1>	Range 0
\0\0meta1\xff	Range 1
\0\0meta2<lastkey1>	Range 1
\0\0meta2<lastkey2>	Range 2
\0\0meta2<lastkey3>	Range 3
…	…
\0\0meta2<lastkeyN-1>	Range 262,143

Range 1

Key	Value
\0\0meta2<lastkeyN>	Range 262,144
\0\0meta2<lastkeyN+1>	Range 262,145
…	…
\0\0meta2\xff	Range 500,000
…	…
<lastkey1>	<lastvalue1>

Range 2

Key	Value
…	…
<lastkey2>	<lastvalue2>

Range 3

Key	Value
…	…
<lastkey3>	<lastvalue3>

…

Range 262,144

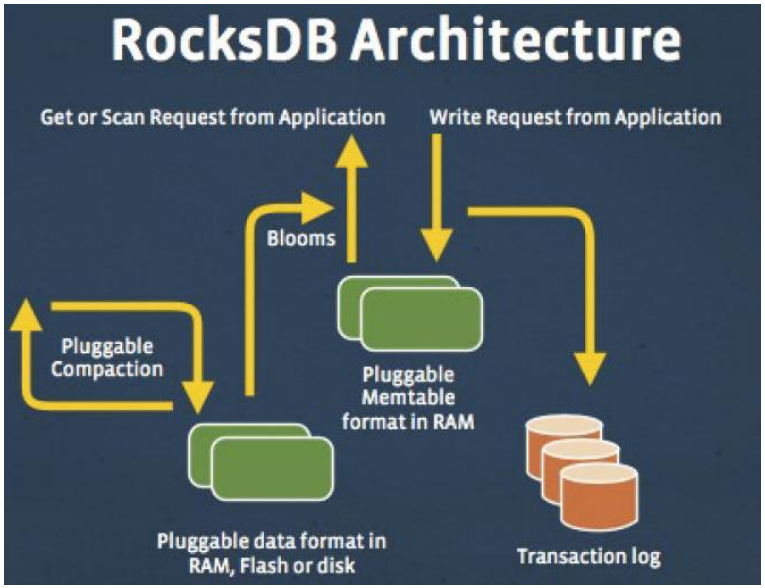
Key	Value
…	…
<lastkeyN>	<lastvalueN>

Range 262,145

Key	Value
…	…
<lastkeyN+1>	<lastvalueN+1>

基于RocksDB的高性能数据读写

- Facebook开发，基于LSM的嵌入式KV存储引擎，C++开发，初始版本是LevelDB1.5
- 设计目标是**充分发挥RAM、Flash的性能**，高度可配置调优（纯内存、Flash、硬盘、HDFS）；



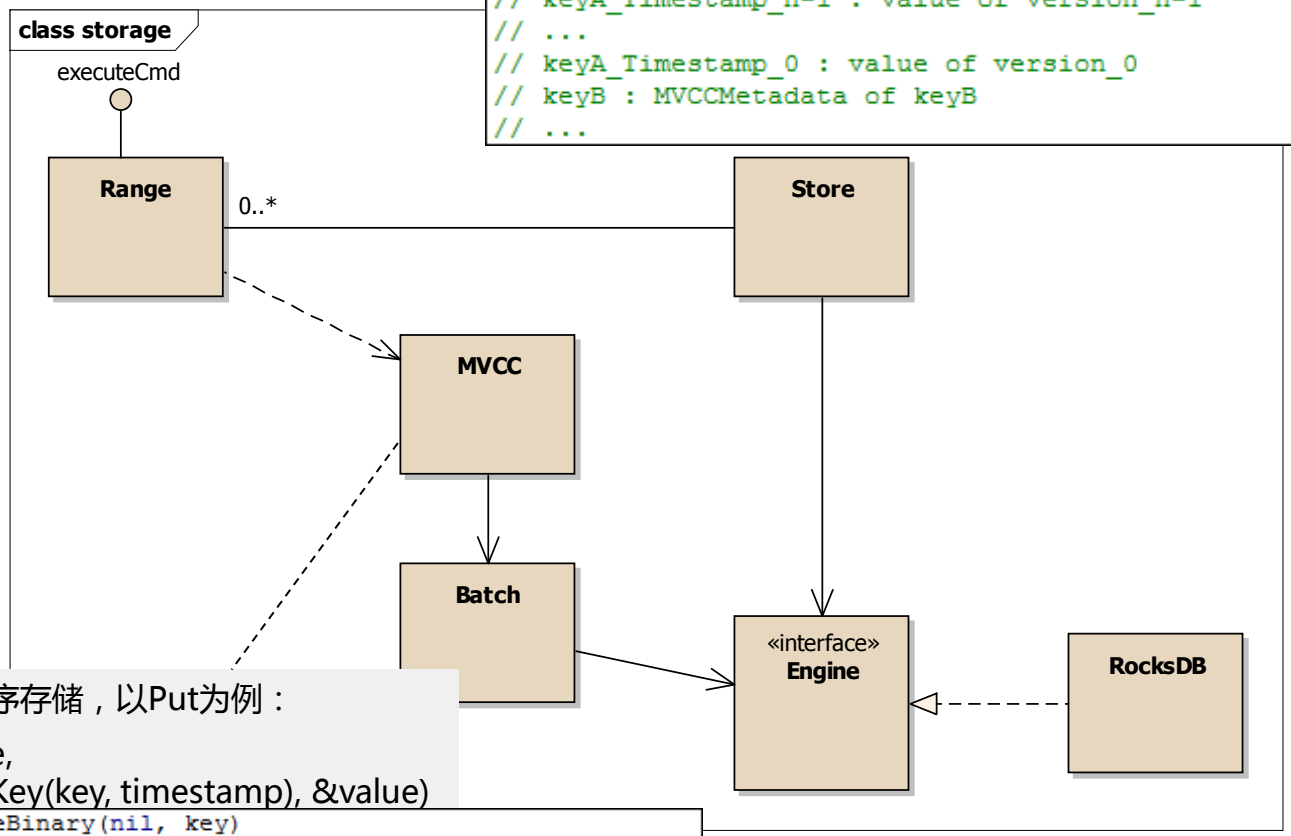
- RocksDB中三个基础元素：**memtable**、**sstfile**、**logfile**，类似HBase中的概念
- sstfile文件可以设置在内存或者硬盘上。每个sstfile保存一个bloomfilter，用于确定该文件中是否存在查找的Key，以进行遍历加速
- memtable是内存数据结构，新的写入会插入到memtable中，并可选的写logfile（WAL）
- 当memtable满后，刷到一个sstfile中，此时会进行一次compaction精简操作；而且系统也会周期的进行compaction

- **高性能**：随机写（17Kops/s vs 1.6Kops/s），随机读（126Kops/s vs 83kops/s）
- **关键模块可插拔**：memtable（ SkipList / Vector / HashSkipList / HashLinkList /...）、sstfile(BlockBasedTable/ PlainTable/ Cuckoo Hashing based SST/...)、compaction(LevelStyle/ UniversalStyle/...)均可自定义，以适应不同的数据特征及读写负载

- 单版本存储，为了实现免锁读写，还需要封装多版本能力
- RocksDB只支持单实例内多值原子操作，需要封装跨实例分布式事务能力

数据多版本机制

```
// The values of multiple versions for the given key should
// be organized as follows:
// ...
// keyA : MVCCMetatata of keyA
// keyA_Timestamp_n : value of version_n
// keyA_Timestamp_n-1 : value of version_n-1
// ...
// keyA_Timestamp_0 : value of version_0
// keyB : MVCCMetadata of keyB
// ...
```



利用了RocksDB本身的排序存储，以Put为例：

- PutProto(mvcc.engine, MVCCEncodeVersionKey(key, timestamp), &value)

```
k := encoding.EncodeBinary(nil, key)
k = encoding.EncodeUint64Decreasing(k, uint64(timestamp.WallTime))
k = encoding.EncodeUint32Decreasing(k, uint32(timestamp.Logical))
return k
```

- PutProto(mvcc.engine, MVCCEncodeKey(key), ne

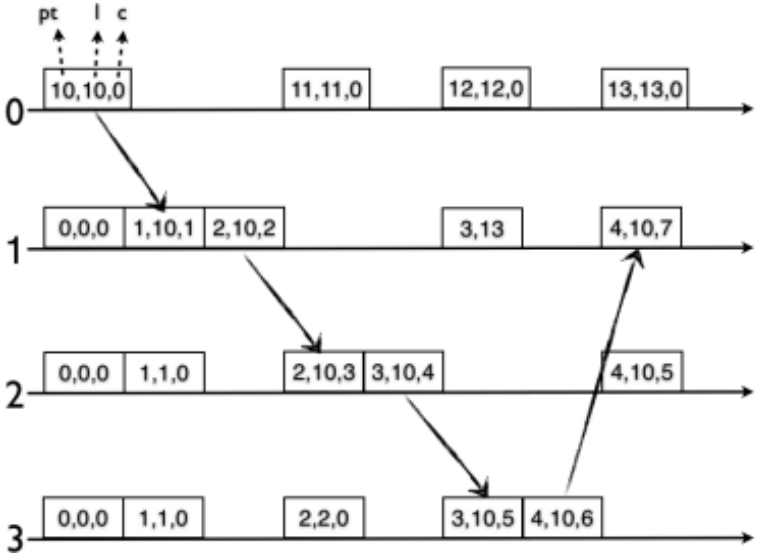
```
return encoding.EncodeBinary(nil, key)
```

分布式事务机制

- 两种隔离级别：
  - Snapshot Isolation （借助数据多版本实现SI，但有Write Skew问题）
  - SSI（缺省级别）
- 免锁，SSI级别也不会加锁，只是事务中止率更高
- 动态时间戳：HLC（混合逻辑时钟——本地）
- 动态优先级：避免长事务饿死，未见其他分布式事务机制使用
- 无中心事务管理节点，在底层数据Map中存储事务状态信息

HLC混合逻辑时钟

- 每个Server维护自己的时钟戳，包括物理时间戳walltime，以及逻辑时间戳logical。比较时，先比较walltime，walltime相等的话，再比较logical。
- 节点间互相交互时（心跳、读写等）或本地操作需要时间戳时，综合本地物理时钟、远端时间戳，来更新自身时间戳
- 优点：
  - 无中心时间戳提供节点，易于实现跨数据中心
  - 无特殊硬件要求（原子钟、GPS）



Initially $l.j := 0; c.j := 0$

Send or local event

$l'.j := l.j;$
$l.j := max(l'.j, pt.j);$
If $(l.j = l'.j)$ then $c.j := c.j + 1$
Else $c.j := 0;$
Timestamp with $l.j, c.j$

设置walltime为原值、本地物理时钟中的最大值

如果本地物理时钟小于时间戳中的walltime，则logical+1；否则logical为0

Receive event of message m

$l'.j := l.j;$
$l.j := max(l'.j, l.m, pt.j);$
If $(l.j = l'j = l.m)$ then $c.j := max(c.j, c.m) + 1$
Elseif $(l.j = l'.j)$ then $c.j := c.j + 1$
Elseif $(l.j = l.m)$ then $c.j := c.m + 1$
Else $c.j := 0$
Timestamp with $l.j, c.j$

设置walltime为原值、收到值、本地物理时钟值中的最大值

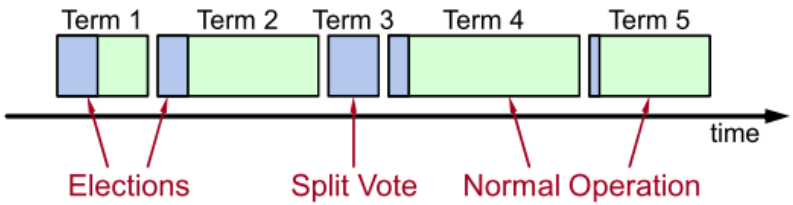
并根据三者的大小关系，设置logical值，如果物理时钟较大，则设置logical为0

Why Raft?

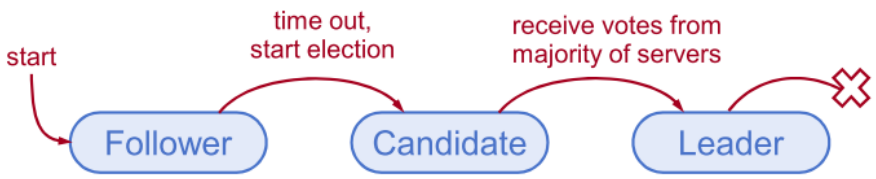
- 基于分布式一致性算法的数据复制的基本原理是将WAL在各节点形成复制状态机，而后各节点可以按序重放WAL以得到相同的数据（Gray和Lamport在2004年提出）
- Raft出现之前，分布式一致性领域的事实标准是Paxos，有大量针对Paxos的分支版本：如FastPaxos、MultiPaxos等
- 但问题在于
  - Paxos算法本身难以理解（Raft的作者花了一年时间学习）
  - Paxos算法描述与实际系统的需求之间存在着巨大的Gap，...，最终系统将基于一个未经证明的协议，（Chubby作者）
  - 可用的Paxos库极少（libpaxos...）
  - 现实应用的分布式一致性系统（Zookeeper、Chubby等）并不严格遵循Paxos算法

- Raft的情况
  - 与Paxos等价
  - 简单易理解
  - 已经有大量的、各种语言的Raft库（Github上已有25+）
  - 已经有大型开源项目开始应用Raft（RamCloud、etcd）
  - 出身不错（Stanford）

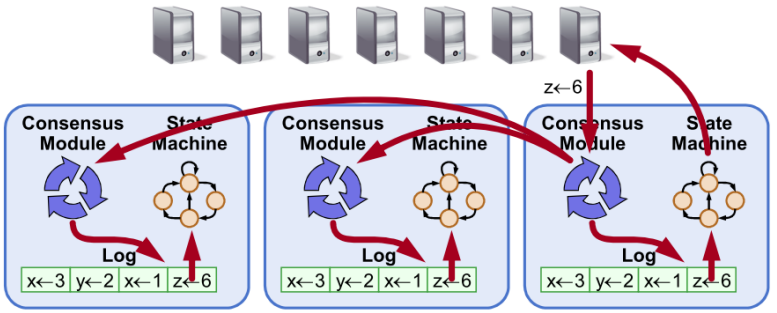
Raft协议概述



- 时间被划分为Term，每个Term开始时进行选主，之后一段时间内保持此Leader，Leader定期向其他节点进行心跳



- 如果Follower一段时间内未收到心跳，则将自己升级为Candidate，并向其他节点发送消息，如果从大多数节点收到了确认消息，则将自己升级为Leader



- Client的读写操作都需要由Leader节点处理，Follower的角色是保证数据可靠。
- 在Cockroach中，如果容忍读取旧数据，则可以读取非Leader副本。

分布式数据库对比

	Oracle RAC	DB2 pureScale	Google Spanner(F1)	Open Source CockroachDB
读写控制	MVCC 读写不冲突	锁机制 读写冲突	‘MVCC’ 读写不冲突	‘MVCC’ 读写不冲突
多版本机制	时间戳+活跃事务列表	无	物理时间戳	逻辑时间戳
时间戳	绝对物理时间戳，可转换LSN	无	绝对物理时间戳，包含误差	Hybrid Logical Clock
可靠性	集群+介质恢复 存储可靠性	集群+介质恢复 存储可靠性	软件定义复制，不依赖存储可靠性	软件定义复制，不依赖存储可靠性
可伸缩性	差(4节点以下)	好（最多128节点）	极好（百万级别节点）	极好（百万级别节点）
分布式	同一机框(架)	同数据中心	跨数据中心	跨数据中心
数据规模	TB	TB	EB	EB
接口	SQL	SQL	SQL & M/R	SQL

Thank you

www.huawei.com