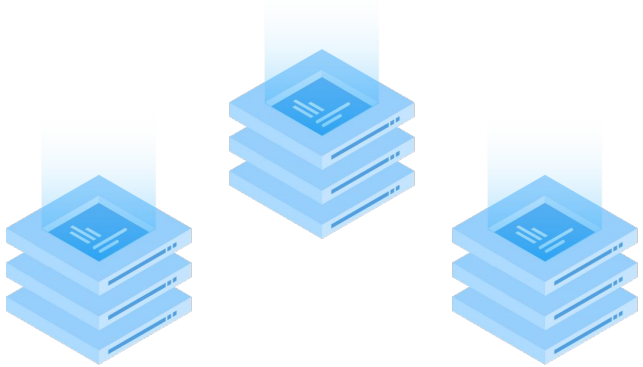
分布式数据库发展趋势和新思路

黄东旭
CTO @ PingCAP

PingCAP TiDB



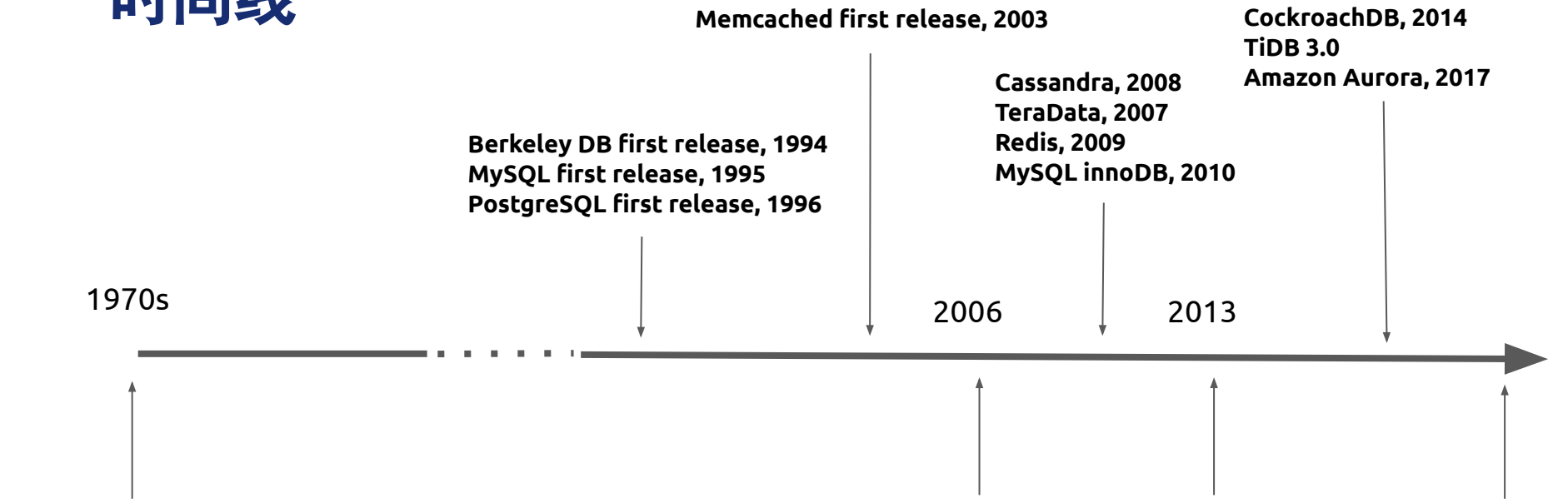
PingCAP

主流的分布式数据库设计模式

常见的分布式数据库流派

- 一代系统：数据库中间件
- 二代系统：NoSQL 数据库
- 三代系统：
  - Google Spanner 及其类似的 NewSQL (TiDB 3.0, CockroachDB)
  - AWS Aurora 及其类似架构的云数据库
- 四代系统：HTAP 数据库（以 TiDB 4.0 为代表）
- 未来？

时间线


Memcached first release, 2003
CockroachDB, 2014
TiDB 3.0
Amazon Aurora, 2017
Cassandra, 2008
TeraData, 2007
Redis, 2009
MySQL innoDB, 2010
Berkeley DB first release, 1994
MySQL first release, 1995
PostgreSQL first release, 1996
1970s
2006
2013
E.F Codd's paper is published
Oracle, 1972
BigTable paper publish by
Google, 2006
Dynamo paper publish by
Amazon, 2007
Hadoop project kick-off,
2005
Spanner paper by
Google, 2012
Snowflake Inc, 2012
Databricks, 2013
HTAP
TiDB 4.0

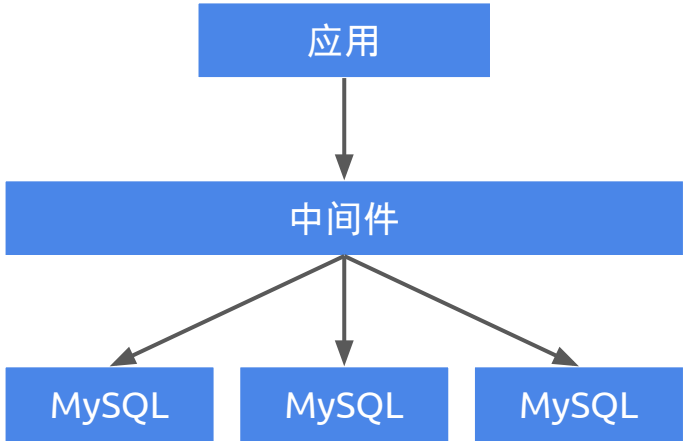
PingCAP

数据库中间件

两种实现模式：

- 业务层手动分库分表（手动）
- 通过中间件指定 Sharding Key 的模式水平分表（半自动）

常见分片Key选择: user id、城市、时间
常见分片规则：哈希分片、范围分片



数据库中间件的优缺点

优点：

- 架构相对简单
- 单机事务延迟低，应用充分适配后（全单机事务）可以在维持原有延迟的基础上获得 TPS 的线性提升
- 对底层的关系型数据库改动不大，运维经验可以复用

缺点：

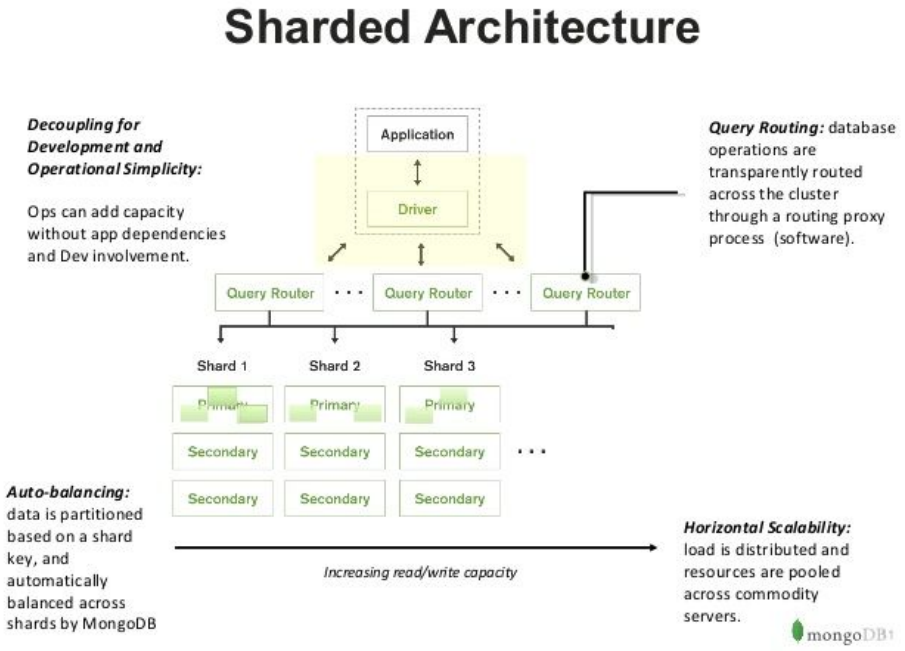
- 只适用于简单业务，同时对业务有侵入性，需要改造
- 很难支持跨分片的高性能复杂查询
- 分布式事务性能损失
  - 扩展问题：为什么会有性能牺牲？
- 水平扩展能力差，只能按照选定的分片规则横向扩展
- 大型集群运维困难
  - 表结构变更（DDL）操作困难，容易出现失误
  - 只能按照分片进行维护（备份、恢复等操作）

NoSQL - Not Only SQL

- 放弃高级 SQL 能力，追求强水平扩展能力（反过来意味着业务兼容的成本高）
  - 放弃复杂 SQL 查询
  - 放弃分布式事务（ACID）
- 通常的数据访问模型：
  - 键值对 (Key-Value)
  - 文档型 (Document)
- 代表系统：
  - MongoDB
  - CouchBase
  - Cassandra
  - HBase
- 在互联网行业比较活跃：2006 ~ 2012

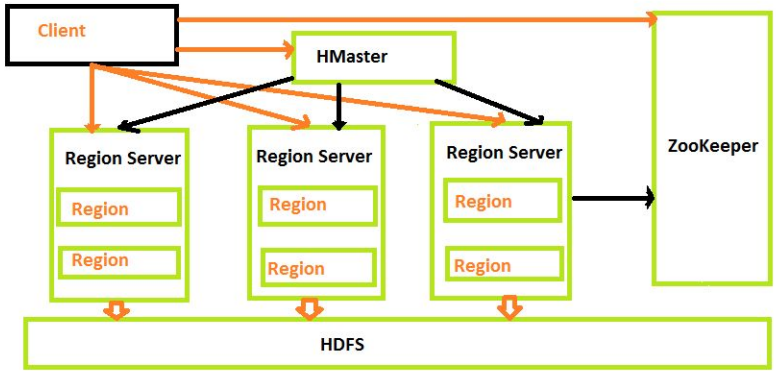
典型系统分析：MongoDB

- 文档型数据库
- 仍然是通过选择 Sharding Key 的方式进行分片
  - Range 或 Hash 分片
- 优点：
  - Schema-less，对文档型数据比较友好
- 缺点：
  - 跨分片聚合能力差
  - Rebalance 过程中会占用大量带宽
  - 无跨分片事务



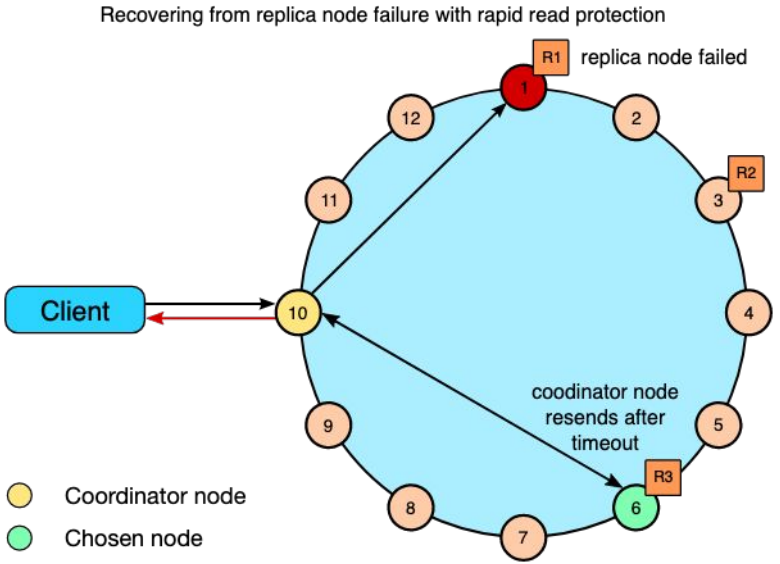
典型系统分析：HBase

- Google BigTable 的开源实现
- 扩展能力构建在分布式文件系 统（HDFS）之上
- 分布式表格系 统
- 优点：
  - 依赖 HDFS 的横向扩展能力，基本可以做到无限的水平扩展
  - Region Server 用于管理数据分片
    - 支持动态水平切分
    - *初步*做到热点转移
- 缺点：
  - 无事务支持
  - 依赖 HDFS 提供存储能力，多一层抽象，性能损失
  - Hadoop 体系运维复杂度高



典型系统分析：Cassandra

- Amazon Dynamo 论文实现
- 分布式 KV 数据库
- 优点：
  - 提供在*单 KV 操作*上的多种一致性模型
    - 强一致 or 最终一致性 可配置
    - 并不是 ACID 事务
  - 扩展性强
- 缺点：
  - KV 模型过于简单，对业务侵入性大
  - 运维比较复杂
  - 国内社区和支持缺乏



NoSQL 系统的通用优缺点

优点：

- 水平扩展能力强
- 针对特殊类型数据效果好，可以作 为关系型数据库的很好补充
- 对于一致性要求不强的场景，可能会有更好的性能

缺点：

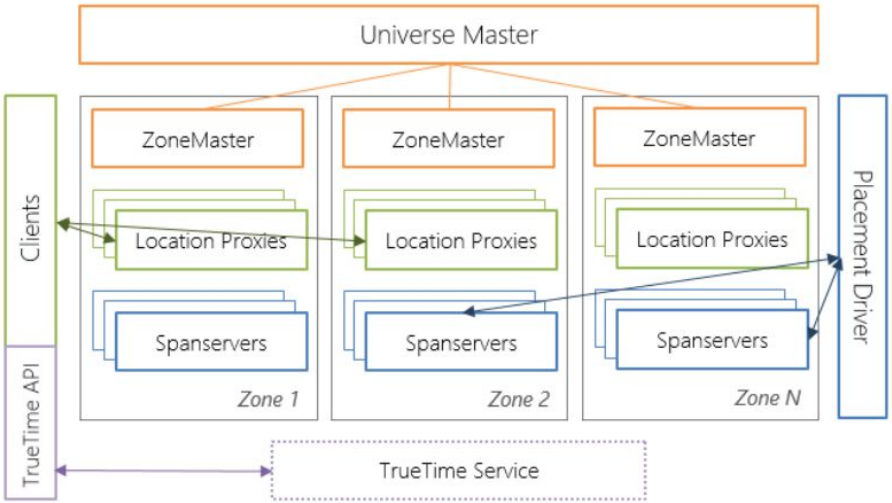
- 由于不支持 SQL 业务需要进行较大的改造
- 普遍无法支持事务，强一致场景比较难实现
- 复杂查询受限

第三代分布式数据库 NewSQL

- 两种流派
  - 以 Google Spanner 为代表的 Shared Nothing 架构
  - 以 AWS Aurora 为代表的 Shared Everything 架构

Shared Nothing 流派

- 特点：
  - 无限的弹性水平扩展
  - 强 SQL 支持（几乎不需要妥协，无需指定分片策略，系统自动扩展）
  - 和单机数据库一样的事务支持
  - 跨数据中心故障自恢复级别的高可用能力
- 代表产品
  - Google Spanner
  - TiDB 3.0 及之前版本
  - CockroachDB



Shared Nothing 流派

- 优点：
  - 无限水平扩展，没有容量上限，对海量数据场景友好
  - 对金融级别的一致性 ACID 事务支持，业务改动代价小
  - 故障自恢复的高可用能力，运 维省事
  - SQL 能力强，和单机数据库的体验基本一致
    - 例如：TiDB 支持 MySQL 的绝大多数语法和网络协议 (覆盖度超过 92%)
  - 无限扩展的吞吐能力（和业务负载有关）
- 缺点：
  - 并非 100% 兼容传统数据库的语法
  - 对于一些极端场景（秒杀），延迟不如单机数据库（跨机，跨地域的分布式事 务必然带来更高延迟）
    - 例子：小 Query 平均 latency：2ms（单机）vs 5ms（分布式）

Shared Everything 流派

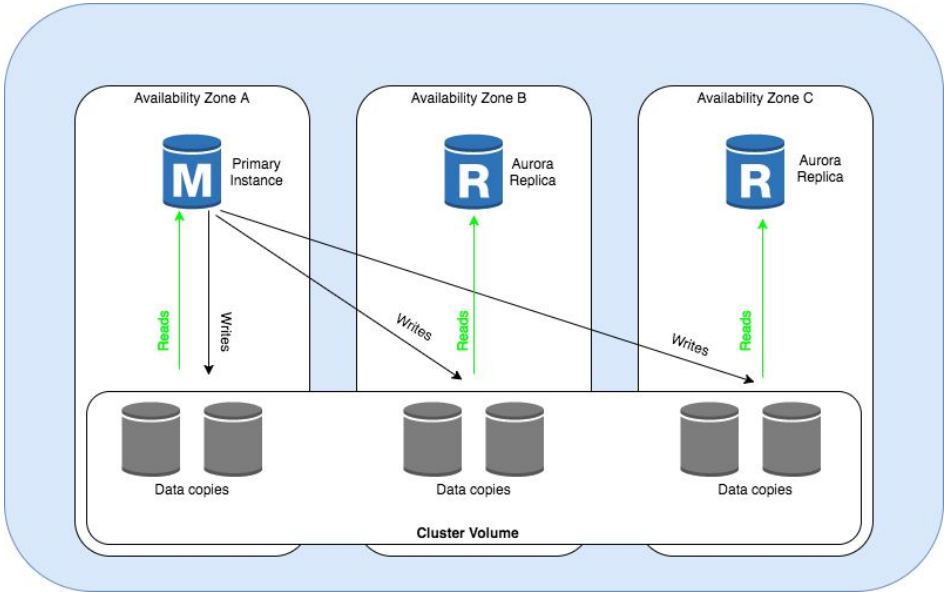
代表产品：AWS Aurora、阿里云 PolarDB

"Cloud-Native"？

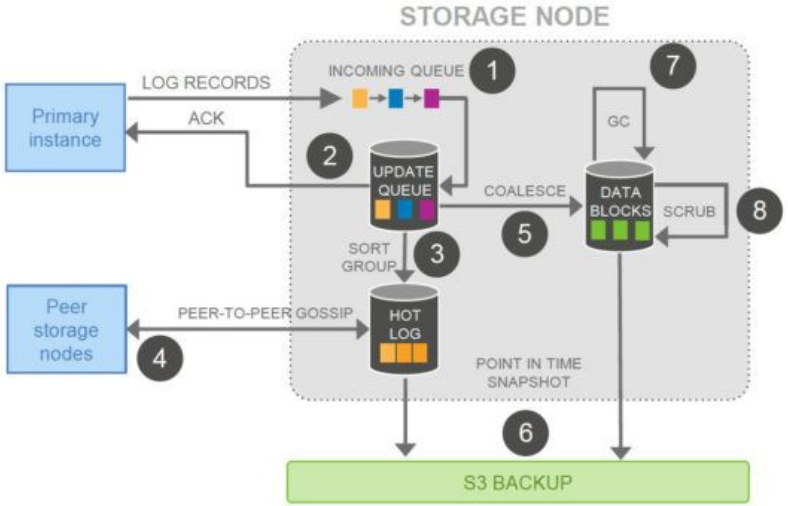
- 通常由公有云提供
  - 为什么？

存储计算分离

- 无状态 SQL 计算节点
  - 计算节点通常直接复用 MySQL，但是不存储数据
- 远程存储（数据文件层）



IO traffic in Aurora (storage node)



IO FLOW

1. Receive record and add to in-memory queue
2. Persist record and ACK
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new data block versions
6. Periodically stage log and new block versions to S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

OBSERVATIONS

All steps are asynchronous

Only steps 1 and 2 are in foreground latency path

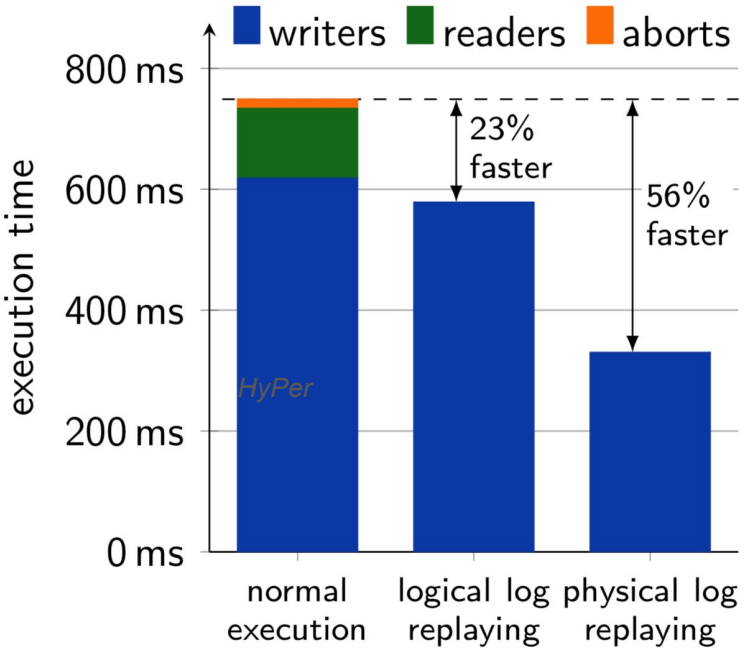
Input queue is **46X** less than MySQL (unamplified, per node)

Favor latency-sensitive operations

Use disk space to buffer against spikes in activity

PingCAP

为什么 Aurora 会比基于 Binlog 的主从 MySQL 复制快？



- 更少的 IO 路径
- 更小的网络包

Shared Everything 流派

优点：

- 易用，100% 兼容 MySQL，业务兼容性好（最大优势）
- 对一致性要求不高的场景，读可以水平扩展（但是有上限）

缺点：

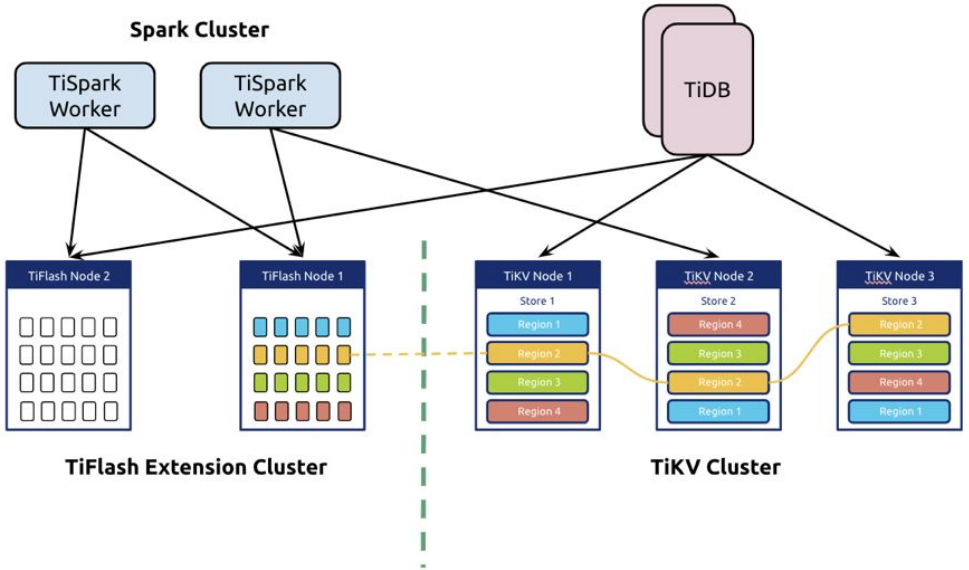
- 本质上还是**单机数据库，如果要支持大数据量，仍然需要分 库分表**
- 内存和容量不匹配，在 单表数据量大后，性能抖 动严重
- 跨机的事务一致性问题（存在同步延迟）
- 分析能力受限于 单点，几乎没有分布式 OLAP 能力
    - 在 AWS Aurora 中，在从库上的大 OLAP 查询可能会拖慢主 库

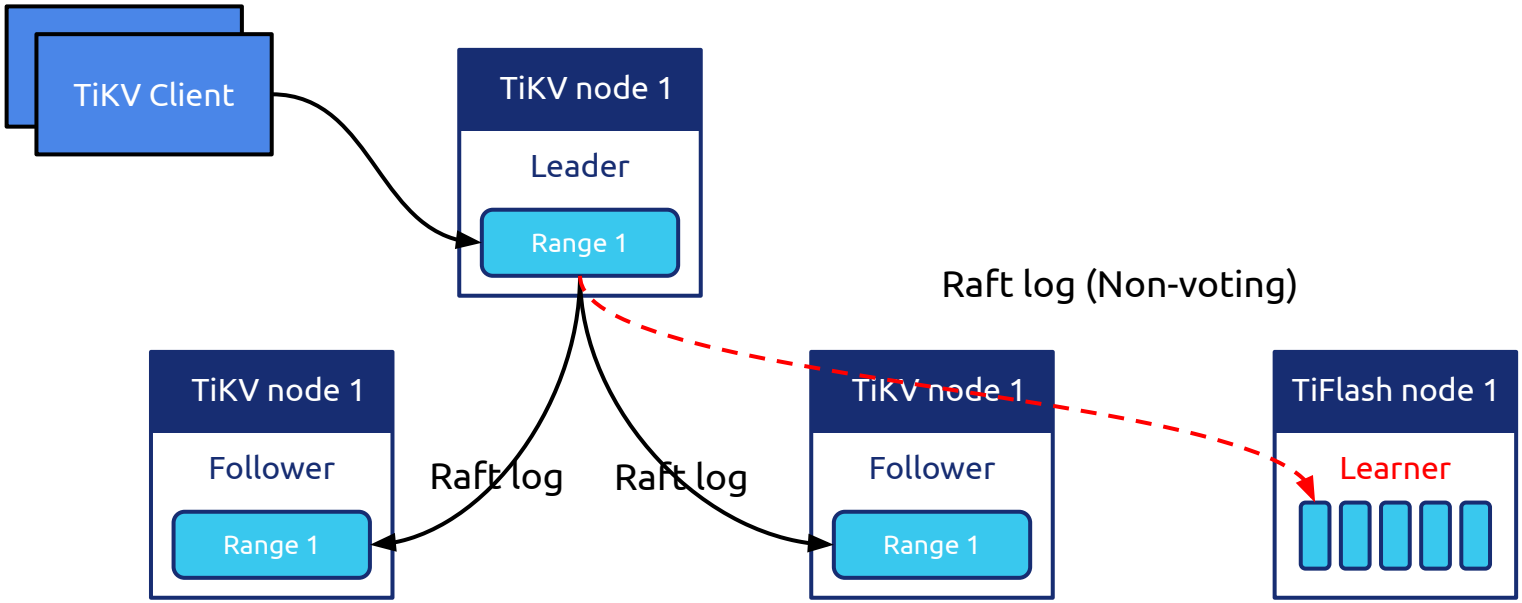
第四代系统：分布式 HTAP 数据库

- HTAP 的定义：Hybrid Transactional/Analytical Processing，混合事务分析处理
- 分布式 HTAP 的标准
  - 业务透明的**无限水平扩展能力**
  - 分布式事务的支持
  - 多数据中心故障自恢复的高可用能力
  - 提供高性能的分析能力
    - 提供列式存储能力
  - 在混合负载下，**实时 OLAP** 分析**不影响** OLTP 事务
- 目前业界仅有 TiDB 4.0 能达到上述的要求
  - TiDB + TiFlash (TiDB 的列存扩展)

分布式 HTAP 数据库：TiDB (with TiFlash)

- 为什么能实现 OLAP 和 OLTP 的彻底隔离，互不影响？
  - 存储和计算彻底分离
  - 列式存储（适用于 OLAP）以副本扩展的形式存在
  - 通过 Multi Raft 架构进行日志级别的复制同步，业务层完全无感知
- 扩展性依托 TiDB 的分布式架构，能做到水平扩展
  - 数据同步不会成为瓶颈
    - 面向实时分析设计，不需要额外的技术栈从数据库同步到实时数仓



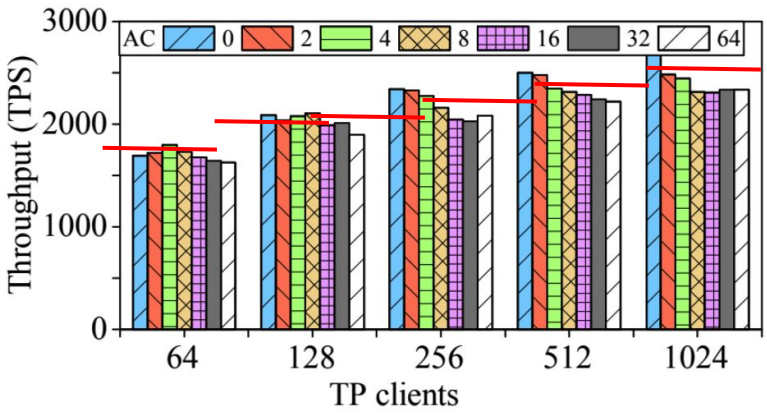

TiKV Client
TiKV node 1
Leader
Range 1
Raft log (Non-voting)
TiKV node 1
Follower
Range 1
Raft log
Raft log
TiKV node 1
Follower
Range 1
TiFlash node 1
Learner

分布式 HTAP 数据库：TiDB + TiFlash

- 右表是一个航空公司 实时航班起降的分析 场景测试
  - 约 1.8亿条记录
  - https://github.com/Percona-Lab/ontime-airline-performance
- TiDB + TiFlash 的测试结果
  - https://zhuanlan.zhihu.com/p/106688537

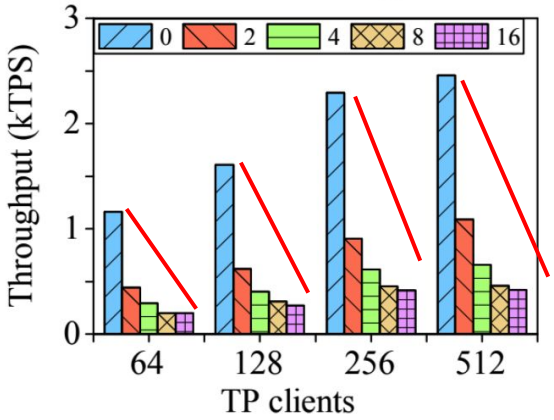
	TiDB + TiFlash	MySQL 5.7.29	Greenplum 6.1	Mariadb Columnstore 1.2.5	Spark 2.4.5 + Parquet	Oracle 12.2.0.1
Q1	0.508	290.340	4.206	1.209	2.044	88.53
Q2	0.295	262.650	3.795	0.740	0.564	76.05
Q3	0.395	247.260	2.339	0.583	0.684	74.76
Q4	0.512	254.960	2.923	0.625	1.306	74.75
Q5	0.184	242.530	2.077	0.258	0.627	67.44
Q6	0.273	288.290	4.471	0.462	1.084	134.08
Q7	0.659	514.700	9.698	1.213	1.536	147.06
Q8	0.453	487.890	3.927	1.629	1.099	165.35
Q9	0.277	261.820	3.160	0.951	0.681	76.5
Q10	2.615	407.360	8.344	2.020	18.219	127.29

Experiment - CH-benCHmark (TiDB vs MemSQL)



(a) Throughput of OLTP

TiDB

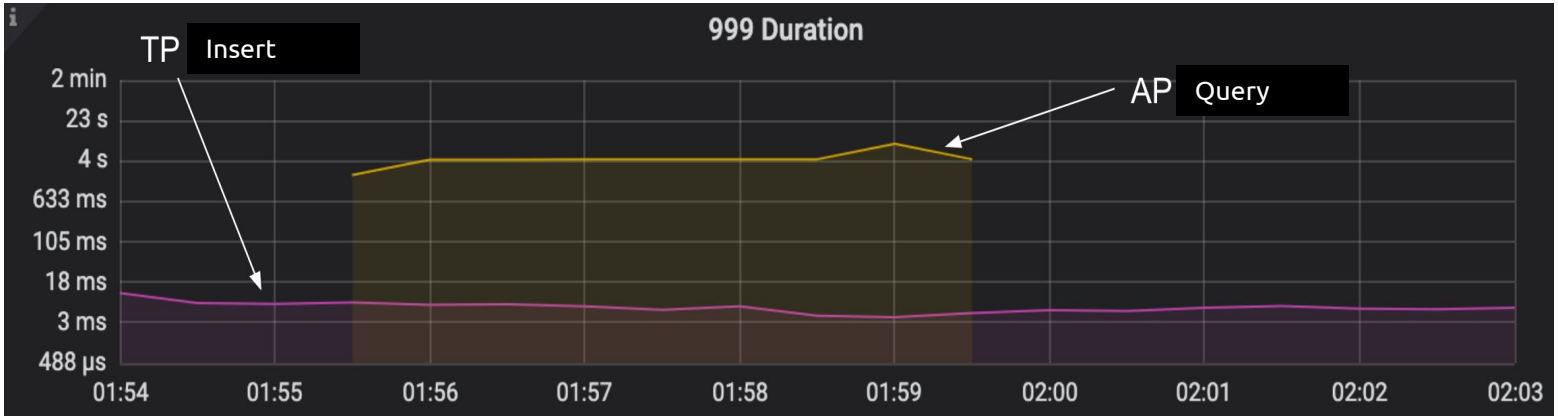


(a) The throughput of OLTP

MemSQL

PingCAP

A Real-world Case



Smooth write latency with analytical queries on/off.

PingCAP

未来在哪里？

PingCAP

我们先看几个例子

Snowflake: Elastic Data Warehouse on Cloud

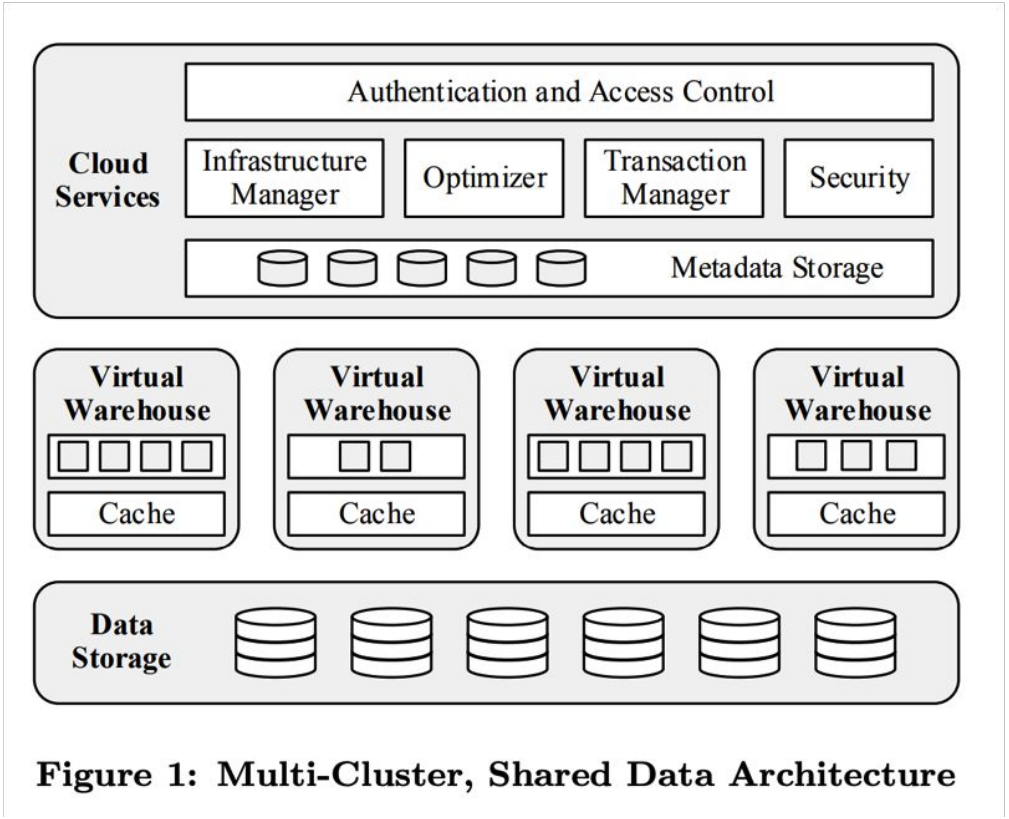
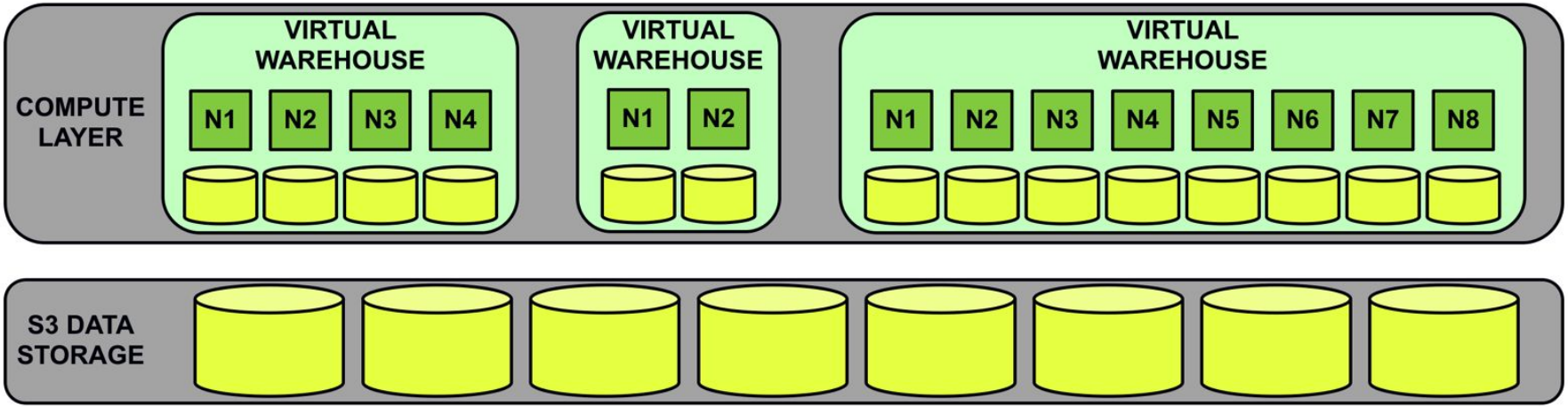
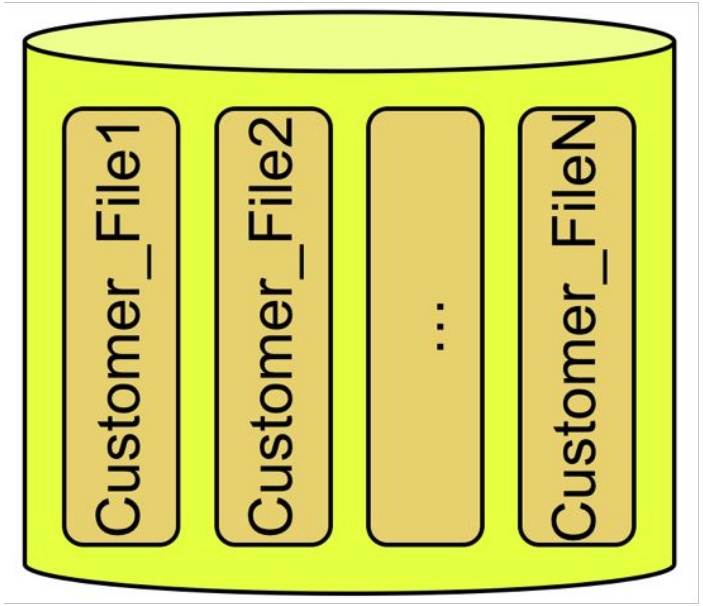


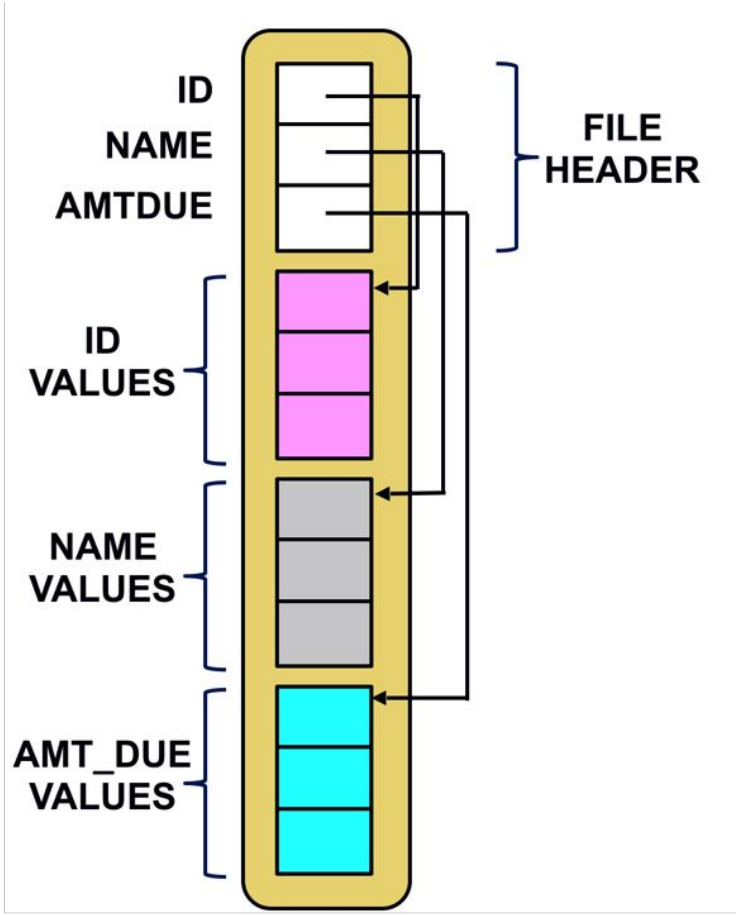
Figure 1: Multi-Cluster, Shared Data Architecture

PingCAP


COMPUTE LAYER
VIRTUAL WAREHOUSE
N1
N2
N3
N4
VIRTUAL WAREHOUSE
N1
N2
VIRTUAL WAREHOUSE
N1
N2
N3
N4
N5
N6
N7
N8
S3 DATA STORAGE

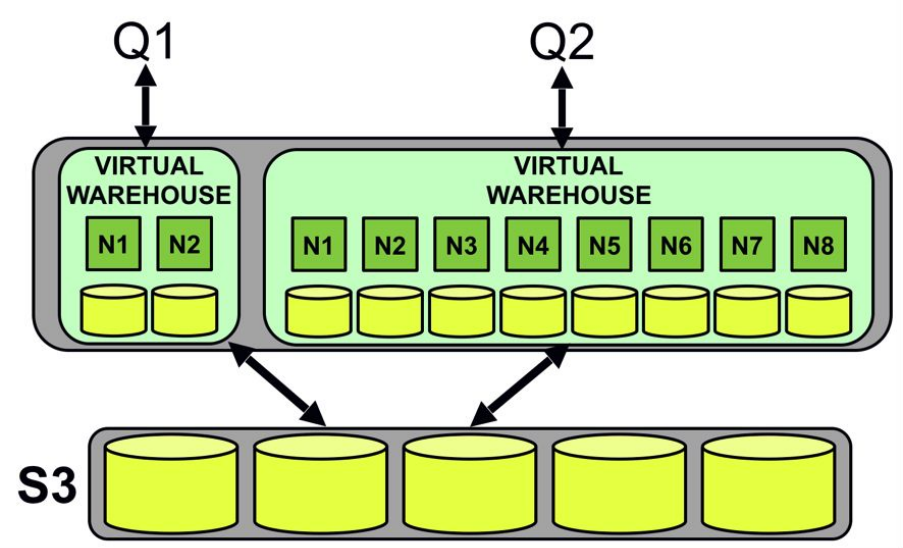


~10MB 一个文件，只追加



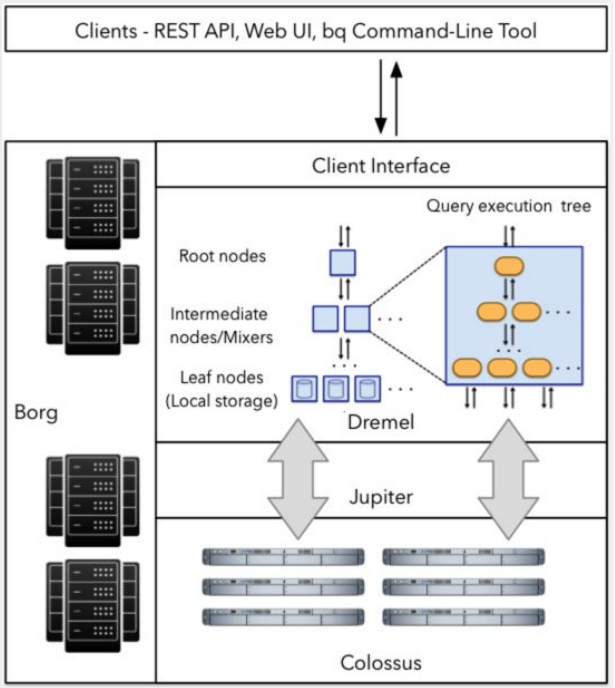
列式存储格式

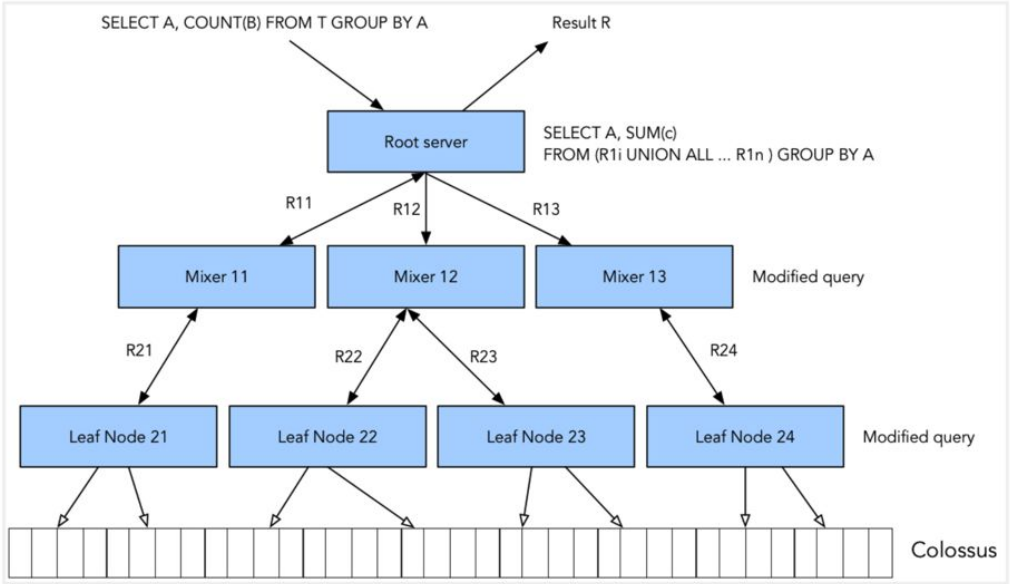
PingCAP



真正的闪光点：
同一份数据可以分配不同的计算资源进行计算

Google BigQuery





PingCAP

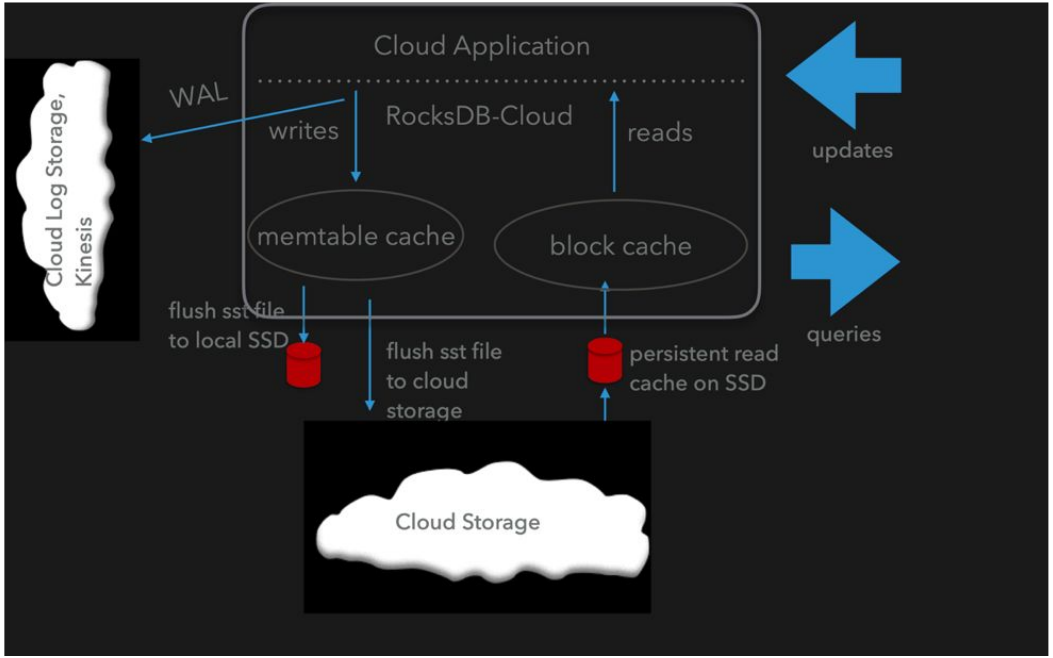
Google BigQuery

1 Petabit/sec of total bisection bandwidth

$40000/month for 2000 dedicated slots （Borg 分配）

Storage: $0.02/GB/month

RockSet: RocksDB(LSM-Tree) on Cloud Stoage



RocksDB-Cloud was created by RockSet to run RocksDB with backup and failover in the cloud. It writes SST files first to SSD but then flushes them to S3. It stores the WAL in AWS Kinesis. They have a read replica that downloads SST files from S3 and applies the WAL from Kinesis.

AWS Kinesis has ~10x the write latency of an EBS Volume. Kinesis is designed for high throughput rather than low latency. RockSet might be sacrificing durability and not waiting for the Kinesis write to sync. Kinesis is also fairly expensive. But note that this is how RockSet is achieving cross-AZ replication since Kinesis can be accessed cross-AZ.

云上的数据库的趋势？

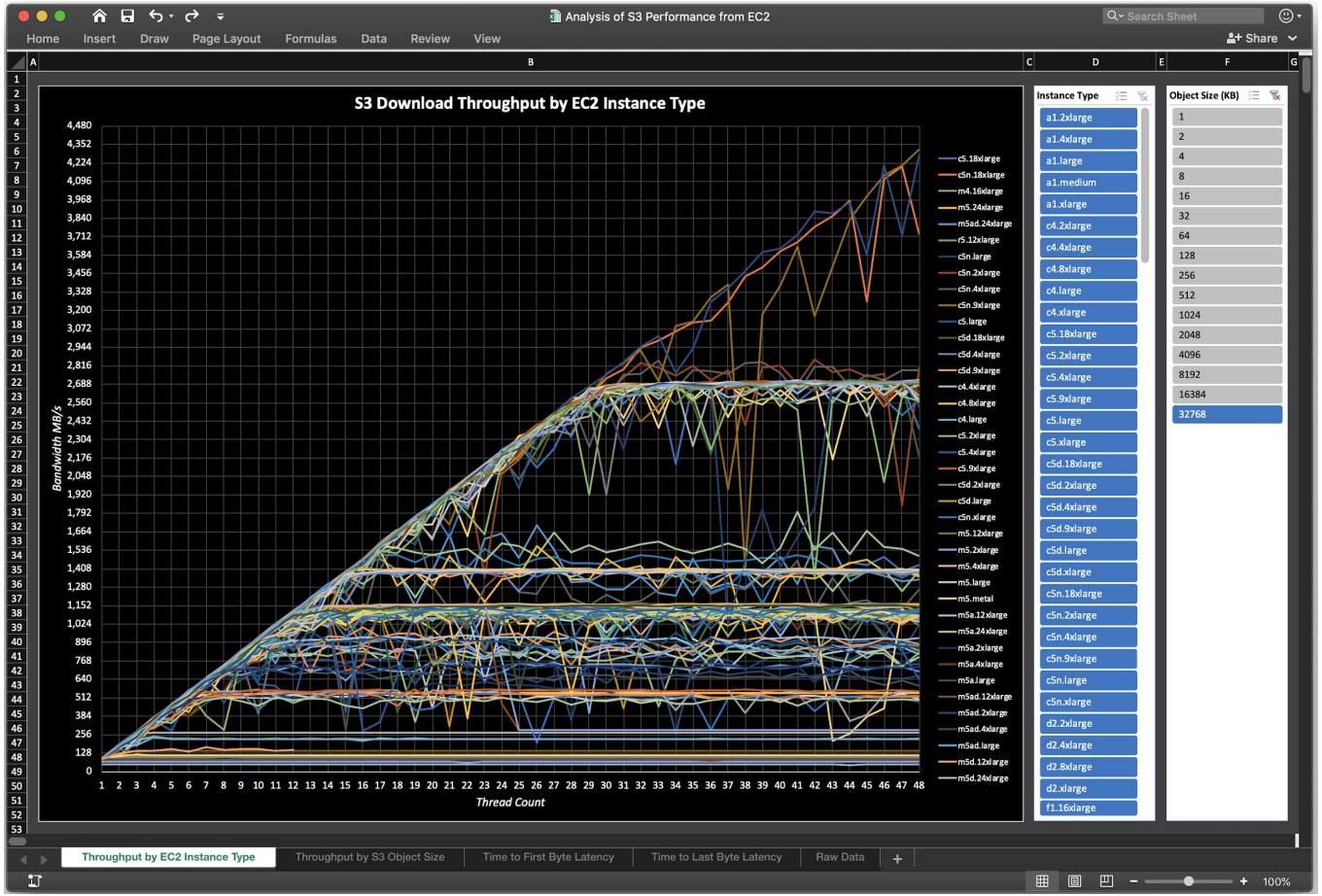
1. 天然分布式
2. 构建在云提供的标准服务之上，尤其是*S3* or EBS
3. 架构中充分利用弹性能力

关键是存储

为什么 S3 是关键?

1. 划算，EBS： $0.125/GB-month plus $0.065/PIOPS-month，S3：flat $0.023 per GB per month
2. 稳定，S3 provides 99.999999999% durability
3. 线性扩展的吞吐能力（见下页）
4. 天然跨云（每个云都有 S3 API 的对象存储服务）

缺点：Latency


Analysis of S3 Performance from EC2
Search Sheet
Home
Insert
Draw
Page Layout
Formulas
Data
Review
View
Share
S3 Download Throughput by EC2 Instance Type
Bandwidth MB/s
Thread Count
Instance Type
a1.2xlarge
a1.4xlarge
a1.large
a1.medium
a1.xlarge
c4.2xlarge
c4.4xlarge
c4.8xlarge
c4.large
c4.xlarge
c5.18xlarge
c5.2xlarge
c5.4xlarge
c5.9xlarge
c5.large
c5.xlarge
c5d.18xlarge
c5d.2xlarge
c5d.4xlarge
c5d.9xlarge
c5d.large
c5d.xlarge
c5n.18xlarge
c5n.2xlarge
c5n.4xlarge
c5n.9xlarge
c5n.large
c5n.xlarge
d2.2xlarge
d2.4xlarge
d2.8xlarge
d2.xlarge
f1.16xlarge
Object Size (KB)
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
Throughput by EC2 Instance Type
Throughput by S3 Object Size
Time to First Byte Latency
Time to Last Byte Latency
Raw Data
100%

PingCAP

如何解决 Latency 问题

一些野路子仅供大家参考。。。。

1. Local SSD/disk as cache
2. Log is data (参考 RockSet 通过 Kinesis 降低写入延迟)
3. Zero-copy data cloning

PingCAP

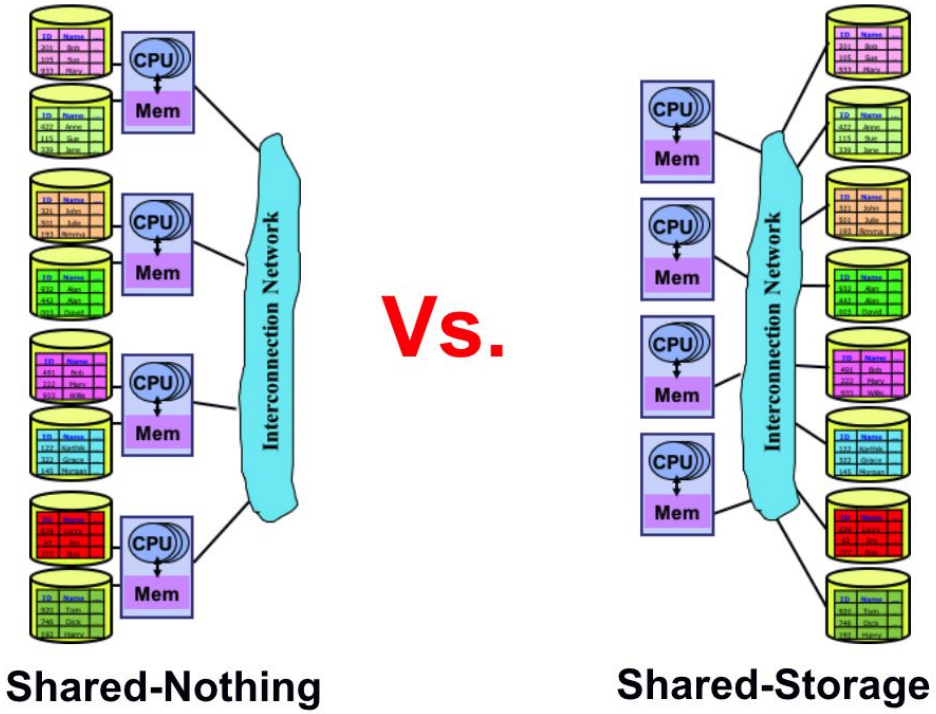
上面的例子的共同点？

PingCAP

为啥都是数据仓库？

Shared Nothing 架构的优势
在于 Data locality

对于 Point Lookup 可能只需要从客户端的一次rpc

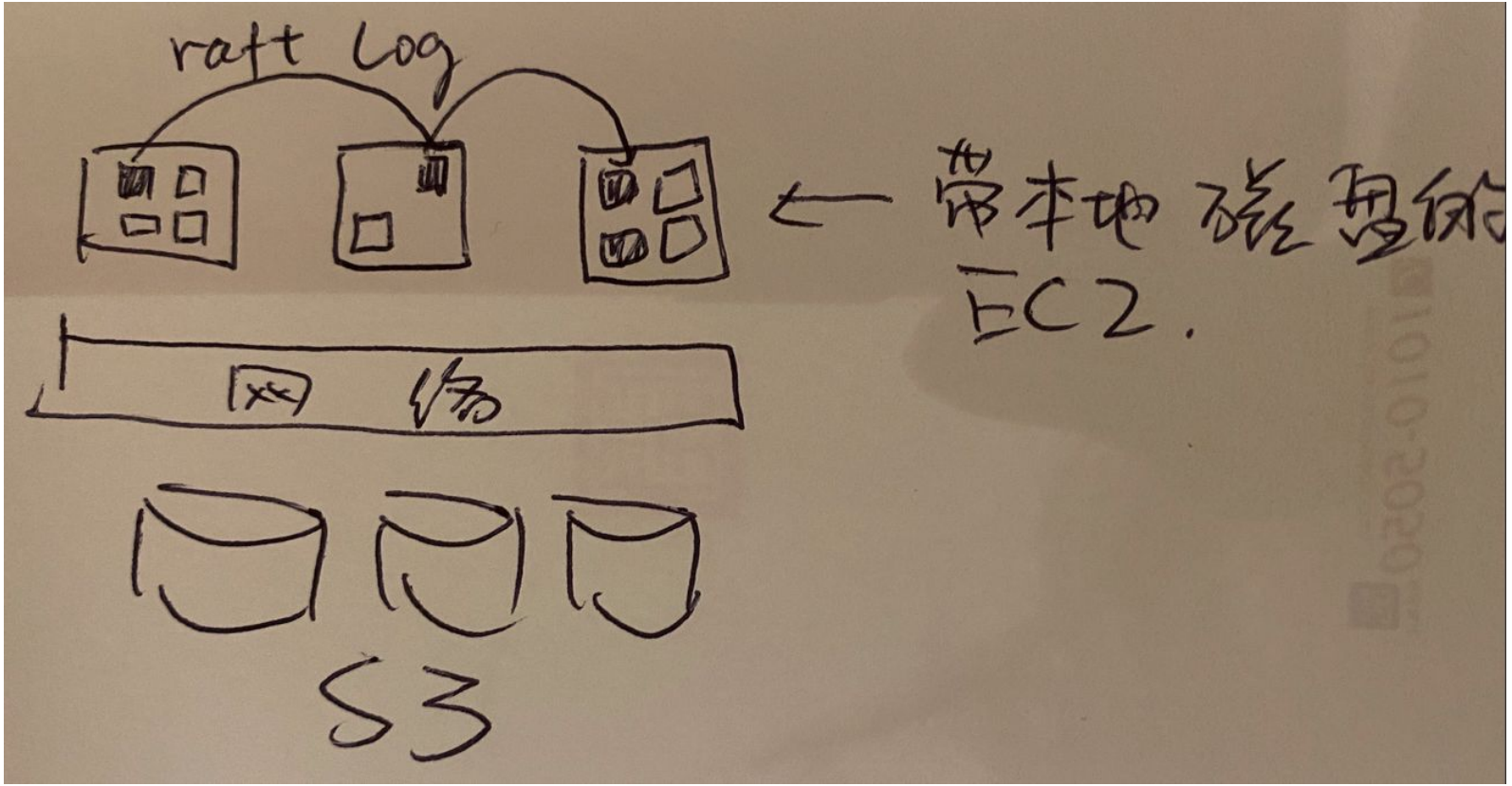


PingCAP

你说没关系，反正大力出奇迹
（计算存储分离）

PingCAP

何必那么极端？
比如，如果我的计算节点带一点状态呢？


raft Log
带本地磁盘的
EC2.
网络
S3

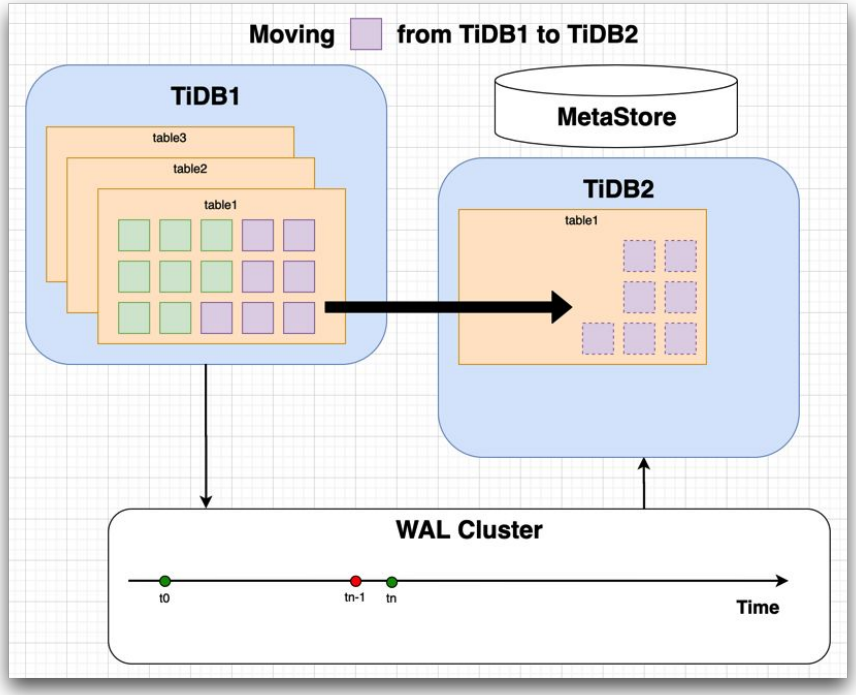
优点：

拥有对实时业务的数据-计算亲和力（传统数据库的优化很多可以用了）

不需要关系复杂的数据迁移逻辑（数据存储仍然在S3）

缺点：

计算节点拥有一部分状态，Failover 需要处理日志回放



还有很多值得研究的问题

1. 如何最高效的利用本地磁盘（缓存比例，LRU 策略），以及和性能的关系

2. S3 的网络性能、吞吐和计算节点配比的关系，尤其对于复杂查询的 Reshuffle

3. 计算复杂度和计算节点、机型的关系是什么？如何用数学表达?

...

当然这些都只是开始

1. Serverless
2. 按需转换数据形态（AI-Driven）

PingCAP

Thank You !

公众号: PingCAP