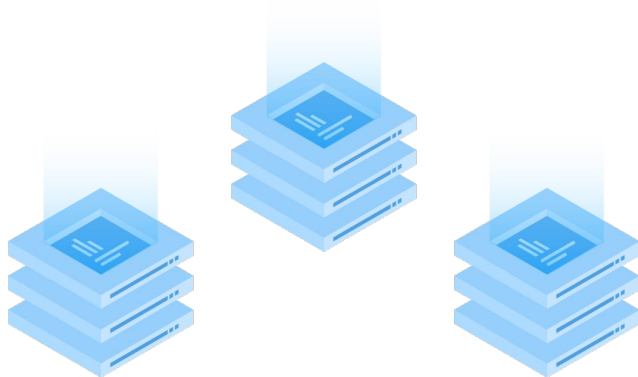


分布式数据库发展趋势

黄东旭

CTO @ PingCAP



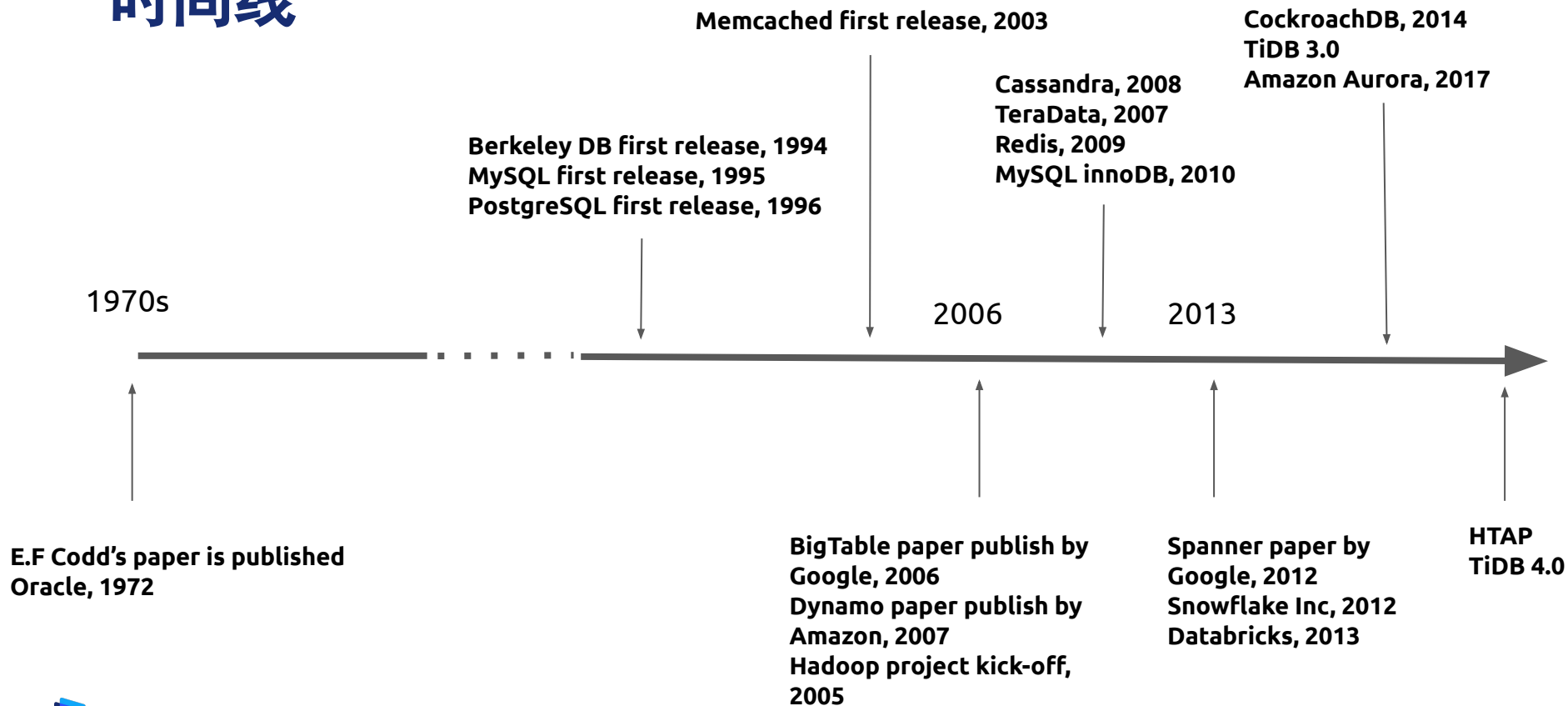
主流的分布式数据库设计模式



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
- 新一代趋势: HTAP 数据库 (以 TiDB 4.0 为代表)

时间线



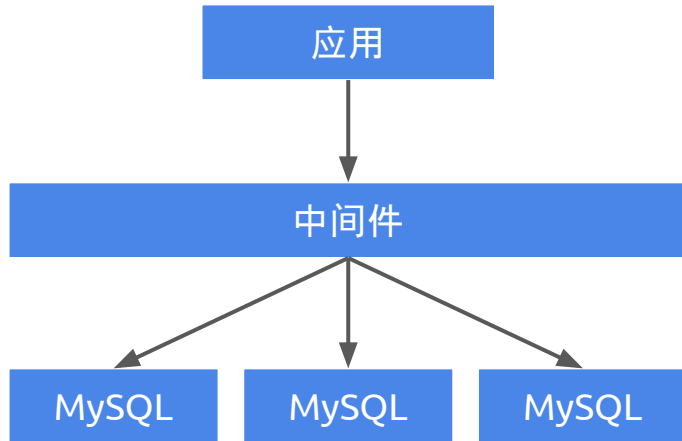
数据库中间件

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
- 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)

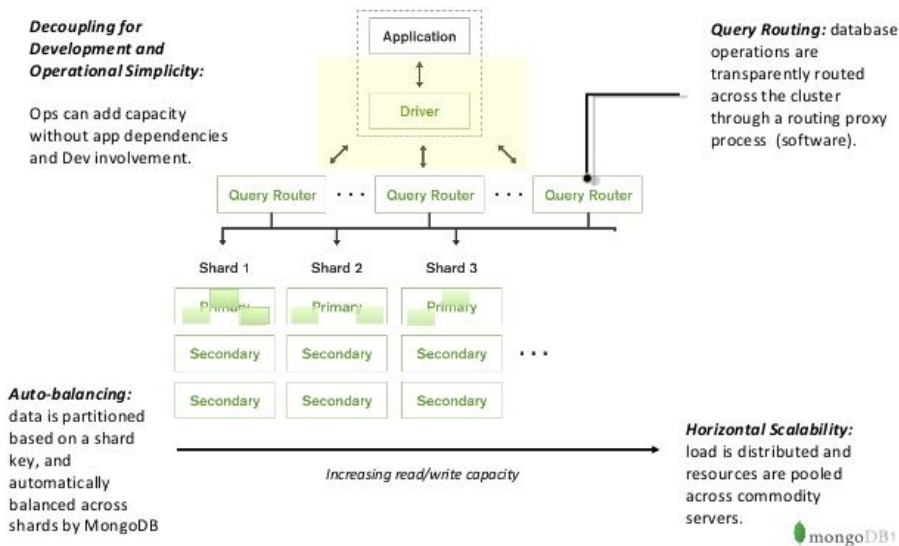
NoSQL - Not Only SQL

- 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

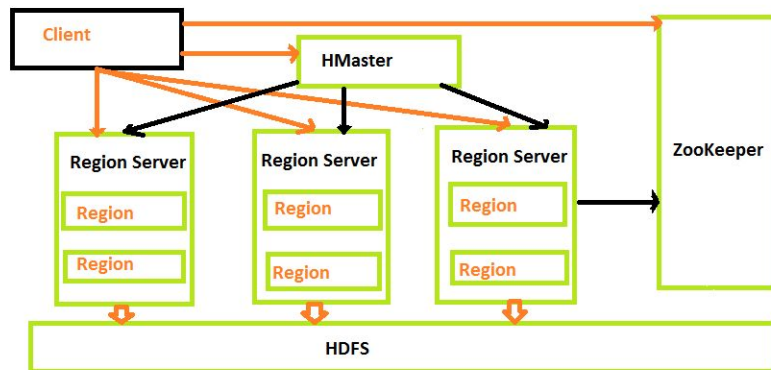
- 文档型数据库
- 仍然是通过选择 Sharing Key 的方式进行分片
 - Range 或 Hash 分片
- 优点:
 - Scheme-less, 对文档型数据比较友好
- 缺点:
 - 跨分片聚合能力差
 - Rebalance 过程中会占用大量带宽
 - 无跨分片事务

Sharded Architecture



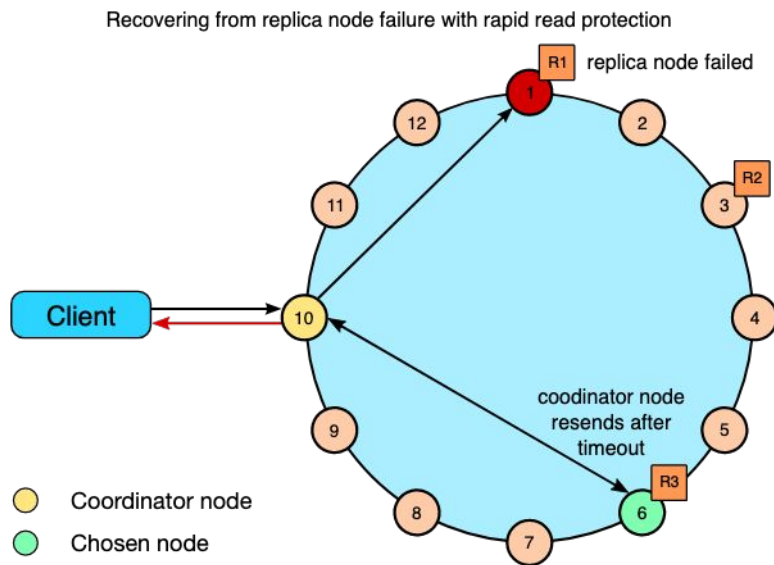
典型系统分析:HBase

- 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：

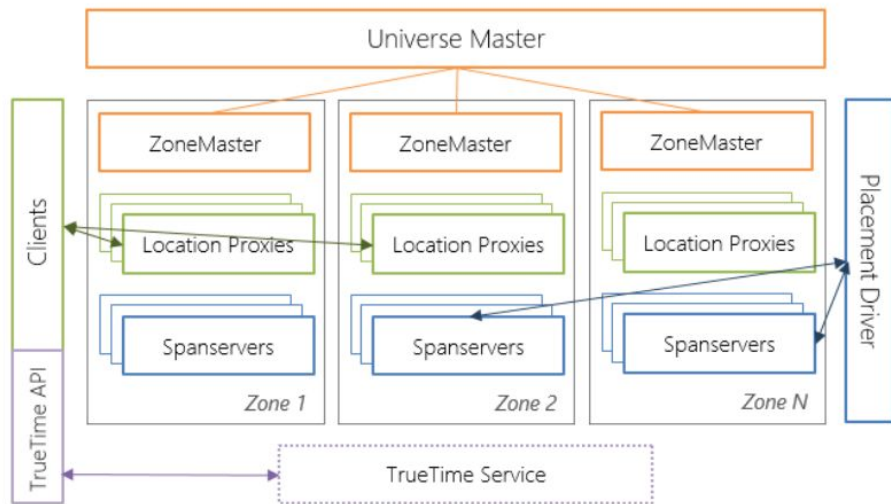
- 由于不支持 SQL 业务需要进行较大的改造
- 普遍无法支持事务，强一致场景比较难实现
- 复杂查询受限

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: 小事务平均 latency: 2ms (单机) vs 5ms (分布式)

Shared Everything 流派

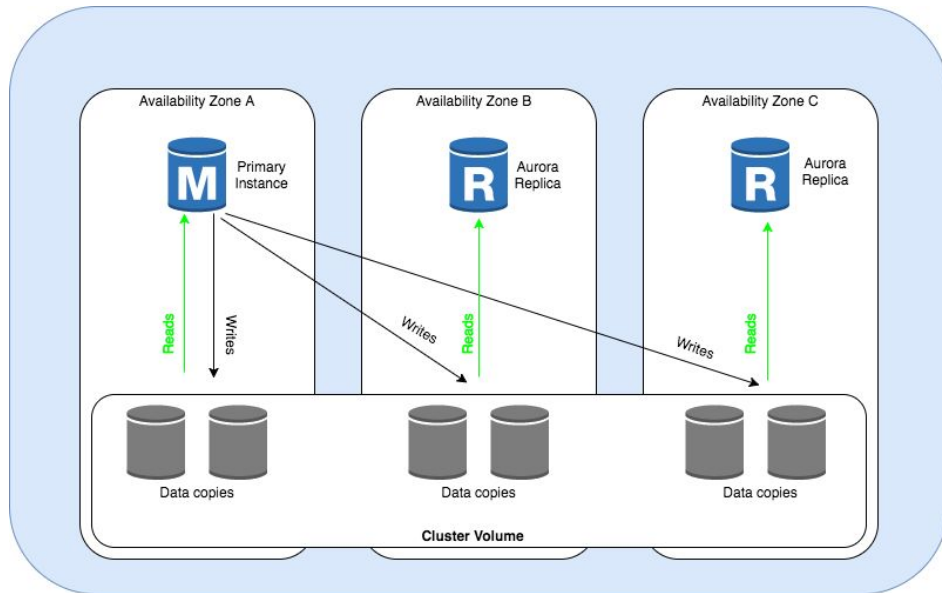
代表产品: AWS Aurora、阿里云 PolarDB

Cloud-Native

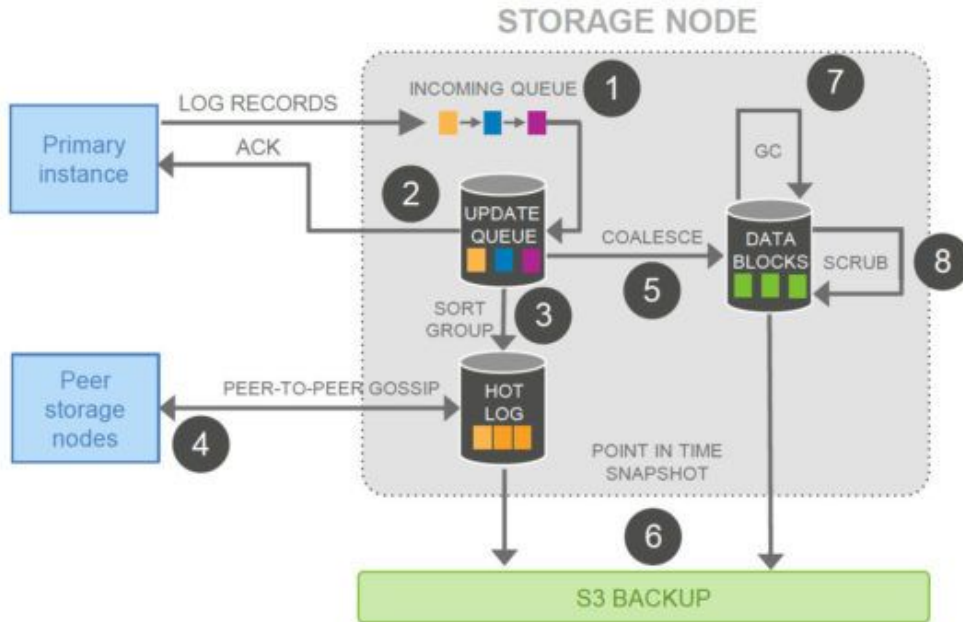
- 通常由公有云提供
 - 为什么？

存储计算分离

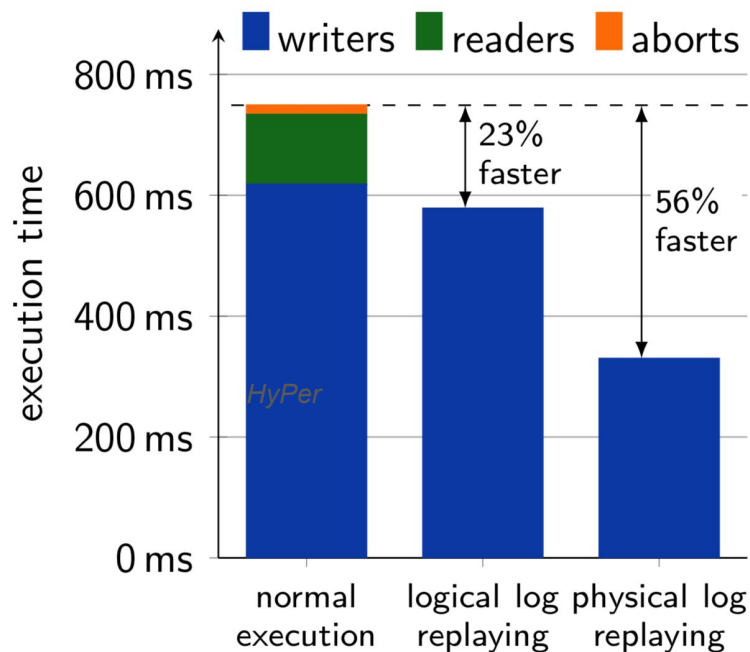
- 无状态 SQL 计算节点
 - 计算节点通常直接复用 MySQL, 但是不存储数据
- 远程存储(数据文件层)



IO traffic in Aurora (storage node)



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

新一代分布式数据库方向



下一代系统: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)

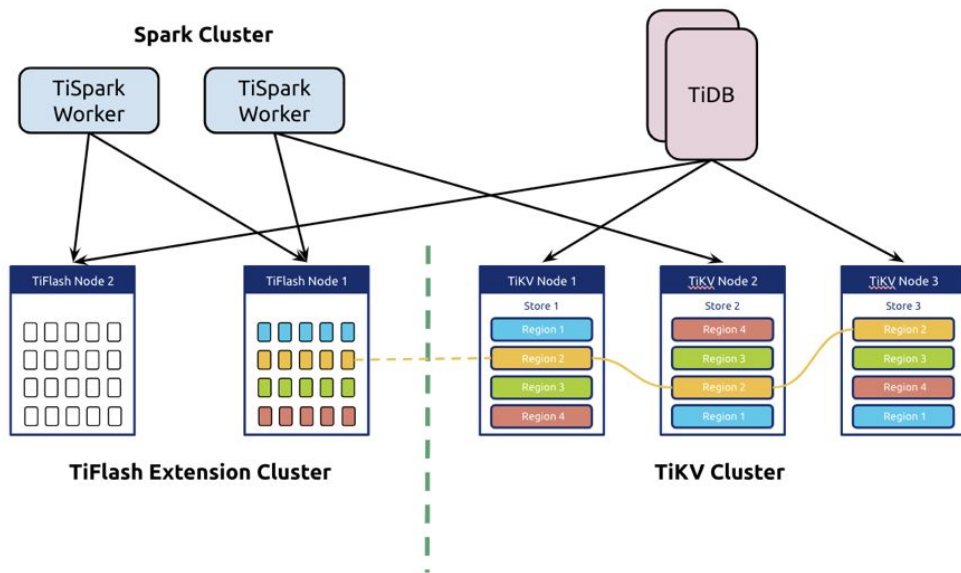
分布式 HTAP 数据库: TiDB + TiFlash

- 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

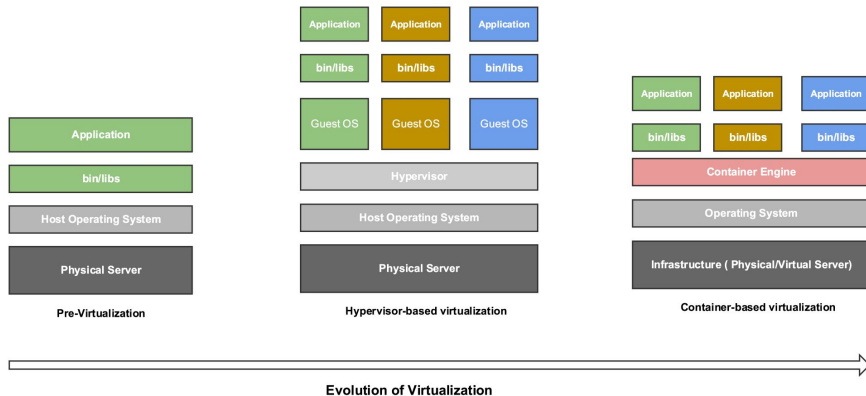
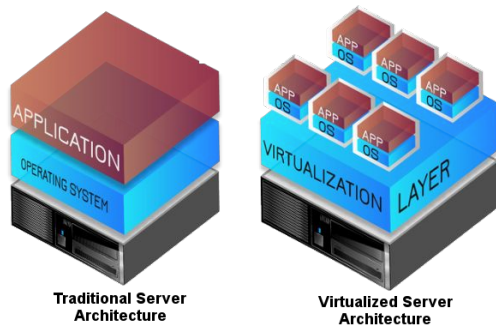
分布式 HTAP 数据库:TiDB + TiFlash

- 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



未来的数据库的趋势？

- 功能上以 HTAP 为基础
- 面向云特性设计
- 智能化
- 平台化

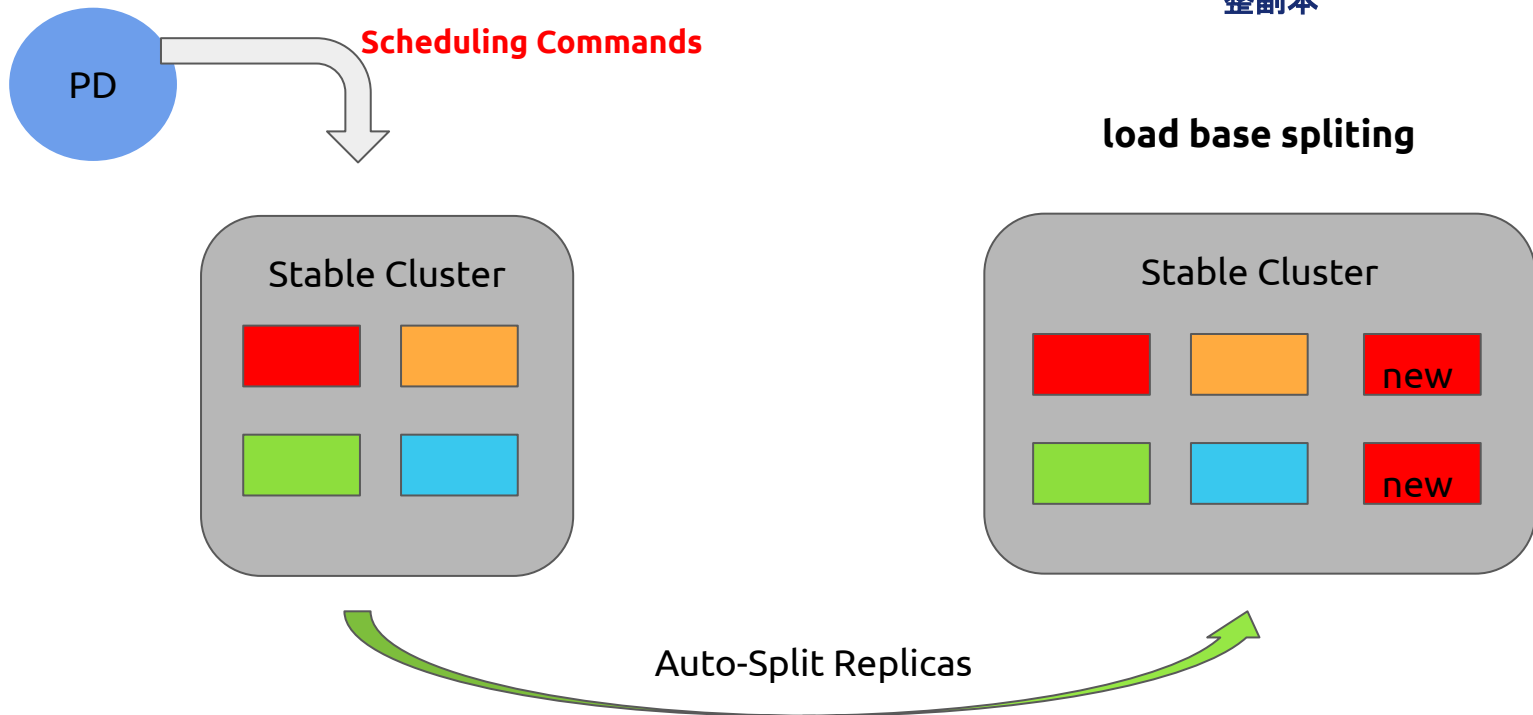


未来的数据库的趋势？

- 数据库是否能够
 - 真正的根据 Workload 自动的决定存储介质？
 - 实时的跟踪热点并弹性的伸缩？
 - 跨数据中心, 跨地域做到真正的高可用和根据 业务的数据智能分布？
 - ...
- 这一切背后的基础是：
 - 调度能力

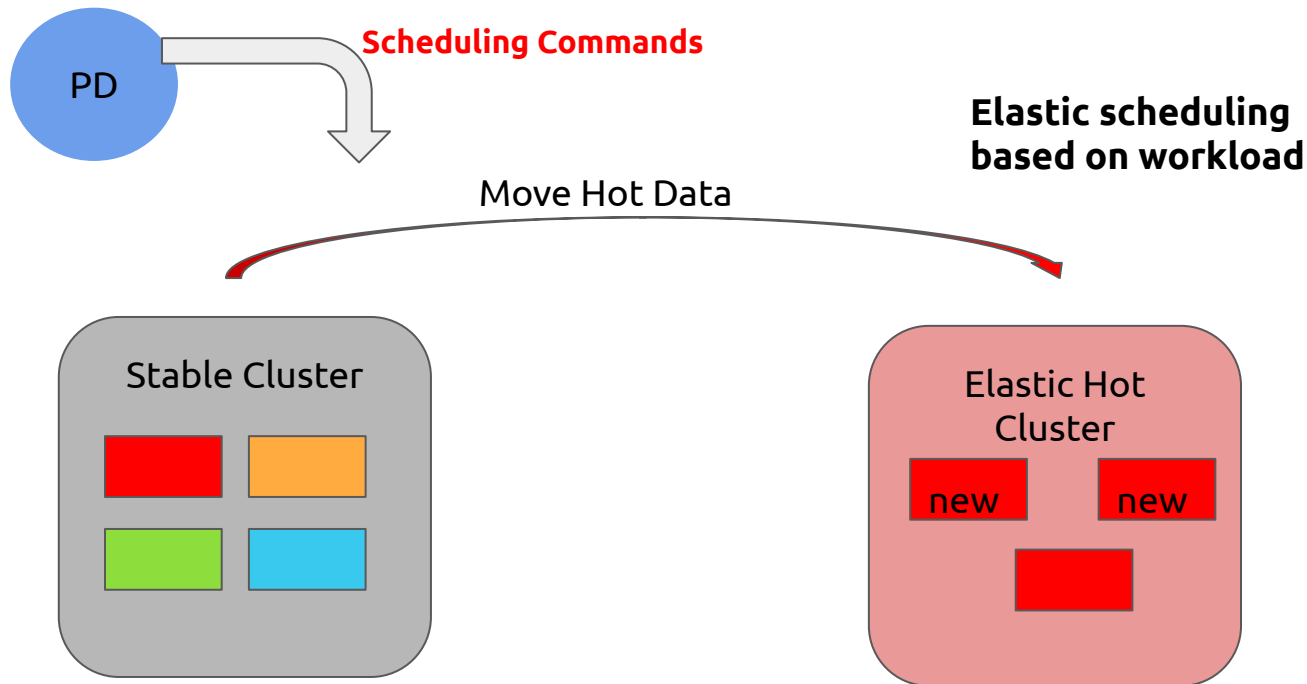
未来数据库的核心能力 - 调度能力

基于负载的分裂均衡及调整副本



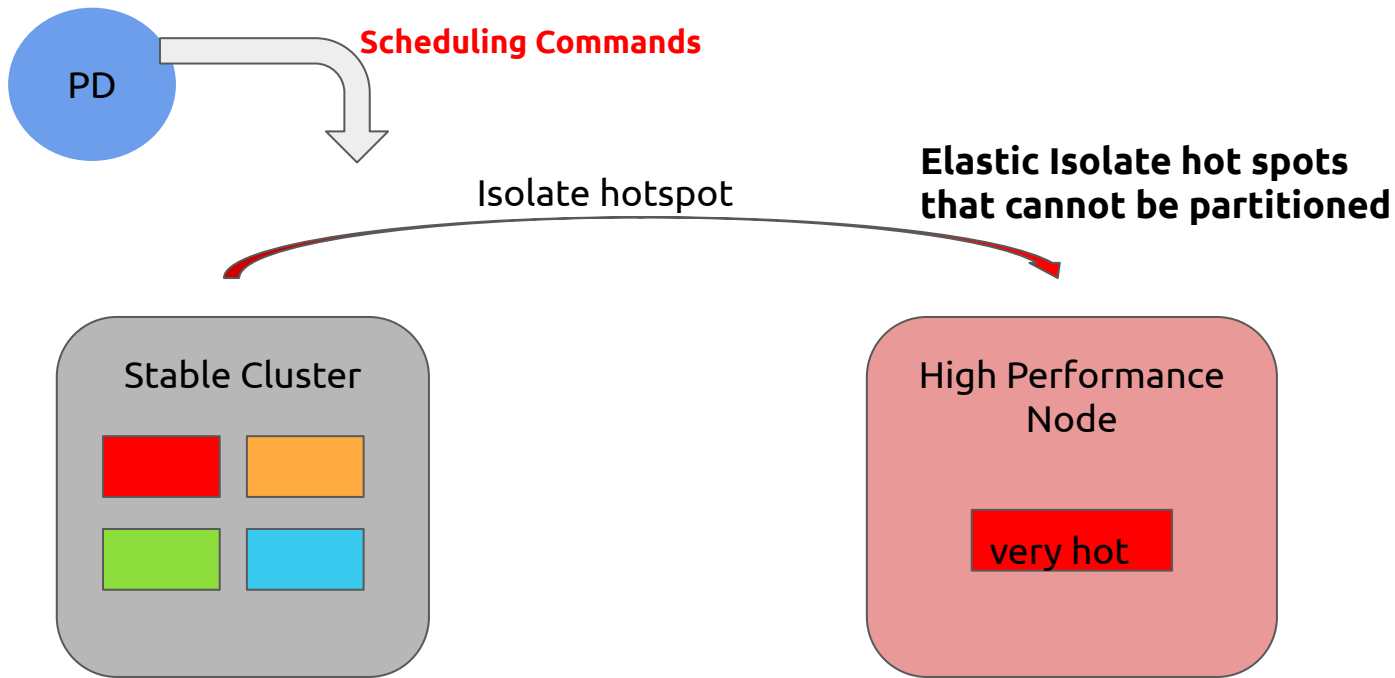
未来数据库的核心能力 - 调度能力

基于负载的节点自动扩充



未来数据库的核心能力 - 调度能力

主动热点隔离

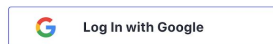


平台化

- Database-As-A-Service
- 多租户
- 可视化管控
- 基于云



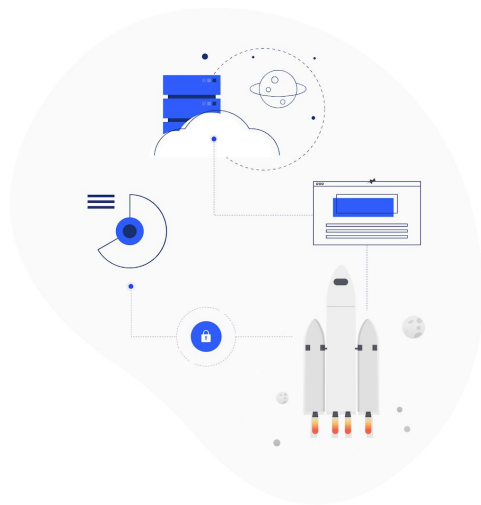
Sign in



or



email/password sign-in is not supported yet, please sign in with google



[illegible]

Thank You !

