ArchSummit

全球架构师峰会（北京）2014

NoSQL数据库的事务机制实现

王涛

Why does it matter?

关系型数据库支持事务！

可是我必须用NoSQL的怎么办？

选择一个支持事务的NoSQL？

还是自己实现一套事务机制？

神马是事务？

- 保障数据可靠地原子性操作
- 一个事务中多个操作要么同时成功，要么同时失败
- 工厂之间的货物转移是一个典型的事务操作

-1

关系型数据库中的事务

T-SQL:

```
DECLARE @TranName VARCHAR(20);
SELECT @TranName = 'MyTransaction';

BEGIN TRANSACTION @TranName;
USE AdventureWorks2012;
DELETE FROM AdventureWorks2012.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;

COMMIT TRANSACTION @TranName;
GO
```

NoSQL是否需要支持事务？

- 仅有少量的NoSQL支持事务！
  - VoltDB
  - RavenDB
  - SequoiaDB
  - MarkLogic

- 很多人认为：需要事务功能的业务就去用MySQL
  - 如果我的业务既需要非结构化存储，又需要原子性操作呢？
- 应用程序自己来实现事务吧！
  - 事务是数据库紧耦合的功能，如何在应用层实现ACID？
- NoSQL支持事务（ACID）是未来的趋势
  - 不支持事务的NoSQL会大大缩小其应用场景

什么样的业务需要事务？

- 电商产品货架
  - 不同类型的产品数据结构迥异
  - 购物车查询需要关联几十个不同的表
  - 使用NoSQL弱化产品结构表的需求
  - 购物车一键购买多个产品
  - 购买失败需要回退整个操作

为什么很多NoSQL不支持事务？

- 大部分NoSQL都是为了“性能”而设计
- 很多NoSQL是为了满足某种特定需求而生，而大部分情况下，这个“需求”并不包含事务
- 对于Facebook等一些互联网公司，数据一致性并非核心，极少量的数据丢失或不一致可以容忍或手工修复
- 分布式系统中事务机制复杂，大部分NoSQL数据库过于年轻

NoSQL支持ACID是未来趋势

- 业界需要分布式系统是一大趋势
- NewSQL从关系型数据库基础开始，做减法以提供分布式功能的支持
- NoSQL从最简分布式框架开始，做加法已提供更多的数据访问存储功能
- NoSQL数据库尝试支持ACID
- 分布式数据库会渐渐与传统数据库统一

分布式已经成为业界的趋势

NoSQL
从基础架构开始，不断添加功能
特点：将SQL执行引擎独立于存储引擎

NewSQL
从传统关系型理论开始弱化ACID
特点：SQL执行引擎与存储绑定

最终目标：两者融合

支持事务的基本要素

- A　　原子性
  - 多个操作要么都成功，要么都失败
- C　　一致性
  - 只有合法的数据才能写入数据库
  - 每次读和写的数据必须保持一致
- I　　隔离性
  - 当多任务同时访问一个数据时，不破坏数据的正确性和完整性
  - 事务的修改必须与其他并行事务的修改互相独立
- D　　持久性
  - 事务提交后，结果必须得到固化

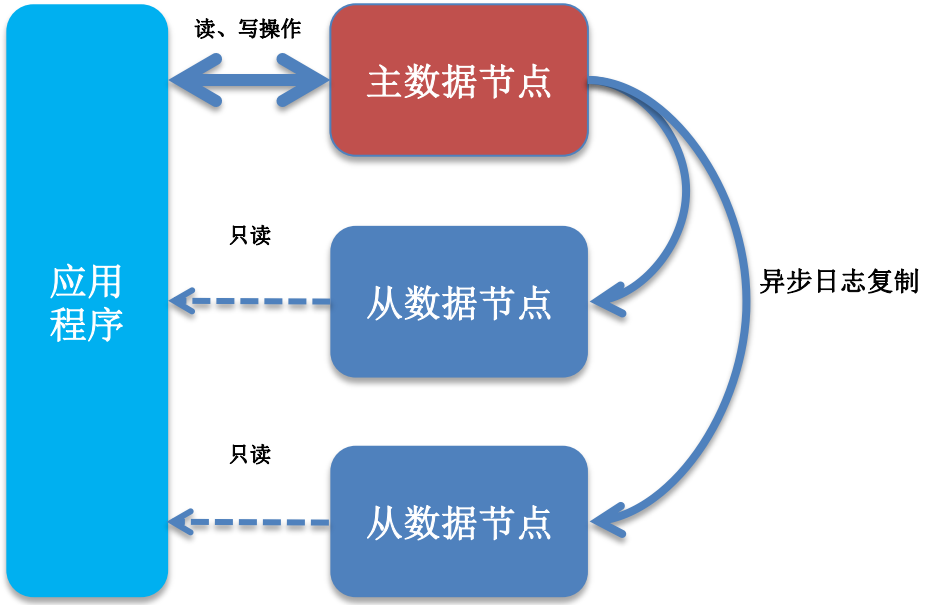
集群ACID最基本前提：单节点支持ACID

原子性和隔离性与传统做法类似

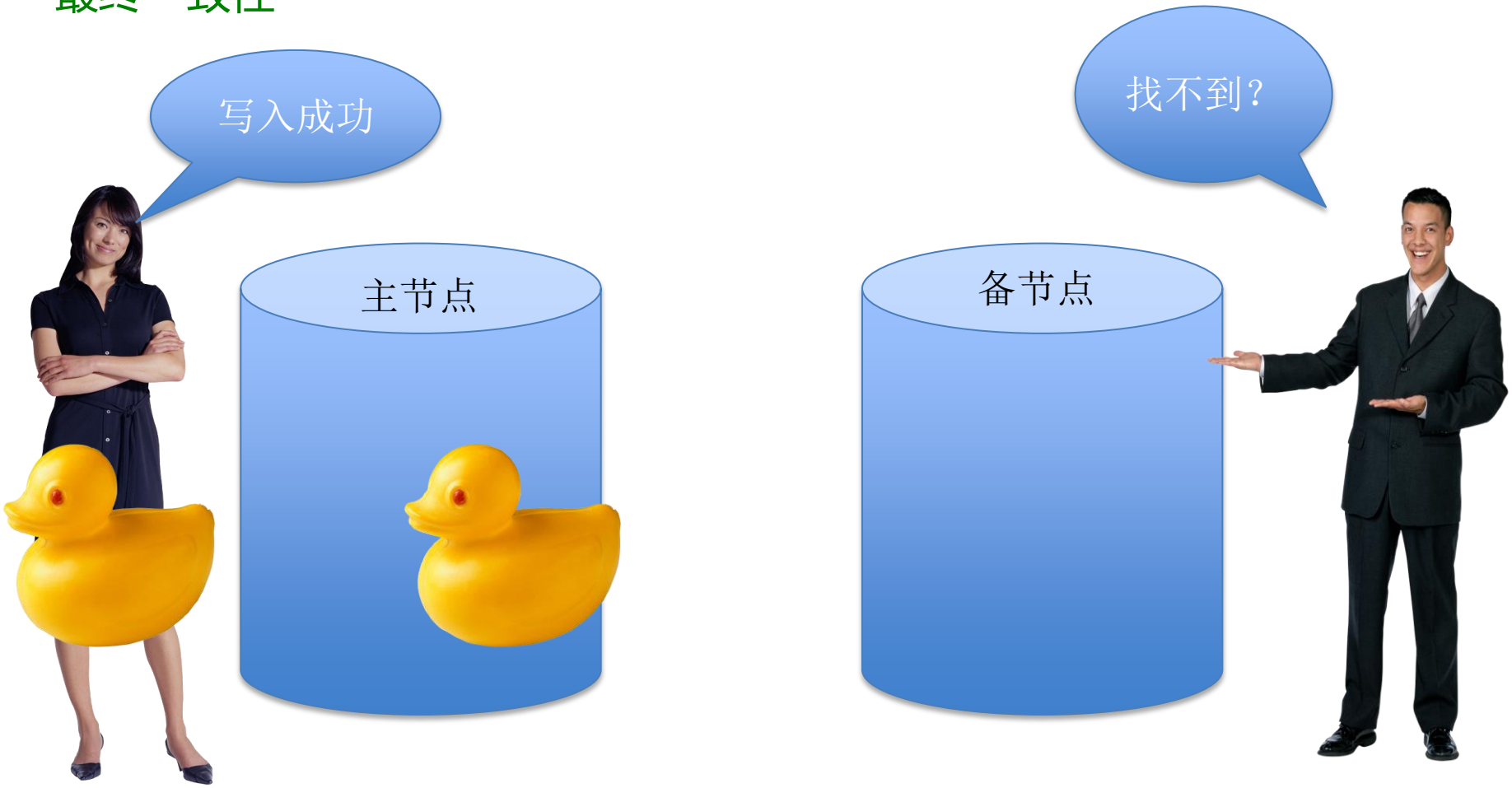
- 原子性
  - 事务提交回滚机制
  - 提交日志形成事务操作的反向链表
  - Redo+Undo
- 隔离性
  - 锁机制
  - 表锁、行锁
  - 乐观锁、悲观锁
  - MVCC

一致性（主从模型）

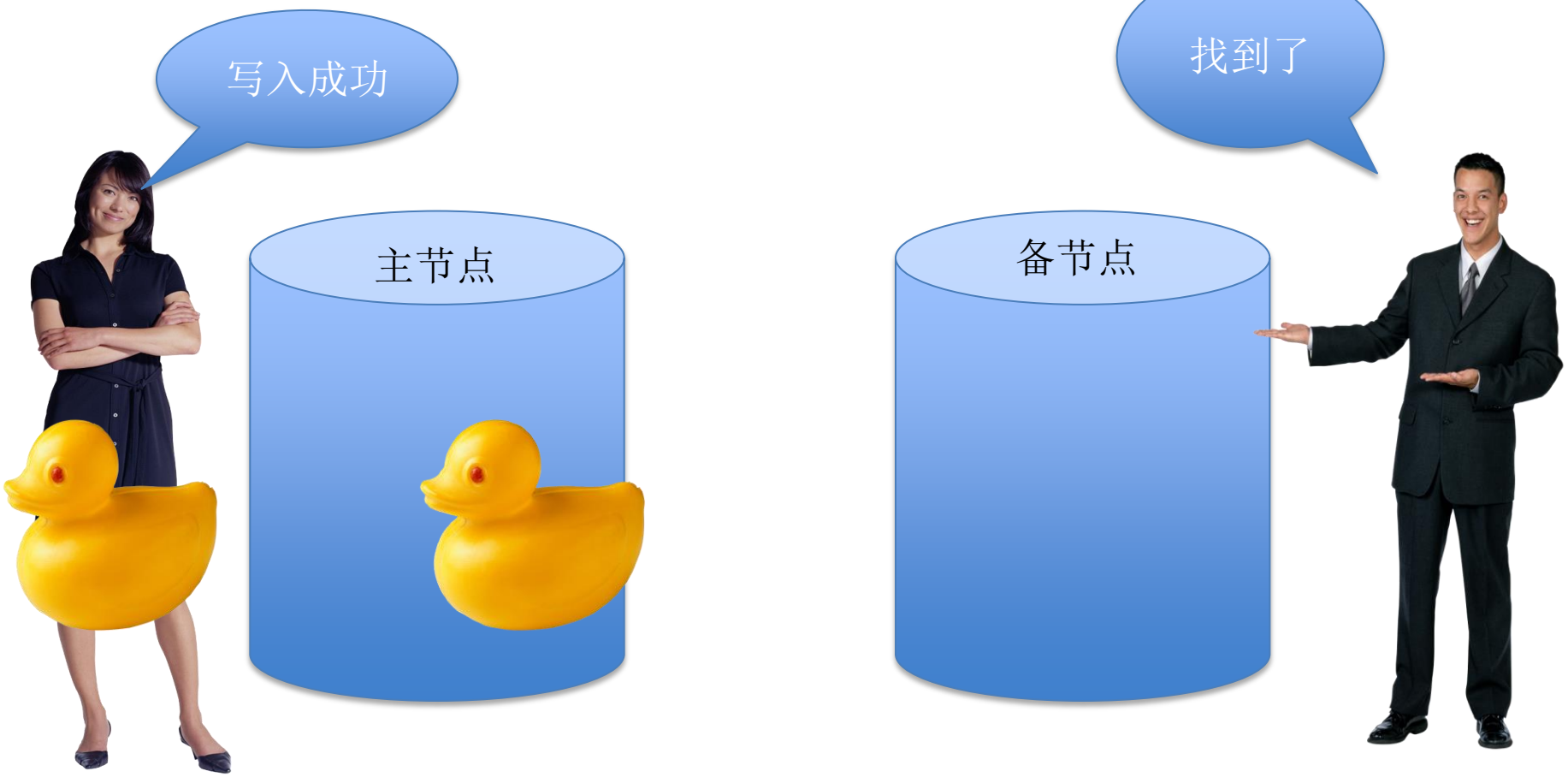
- 传统的一致性：保证正确的数据进入数据库
  - 利用唯一索引、Constraint、主外键、触发器等机制保障
- 分布式系统下的一致性：读写数据保持一致
  - 最终一致性 v.s. 强一致性
- 分布式系统中如何保证强一致？
  - R+W>N
  - 读写从主节点发生
  - 数据同步策略保证持久性



最终一致性



强一致性



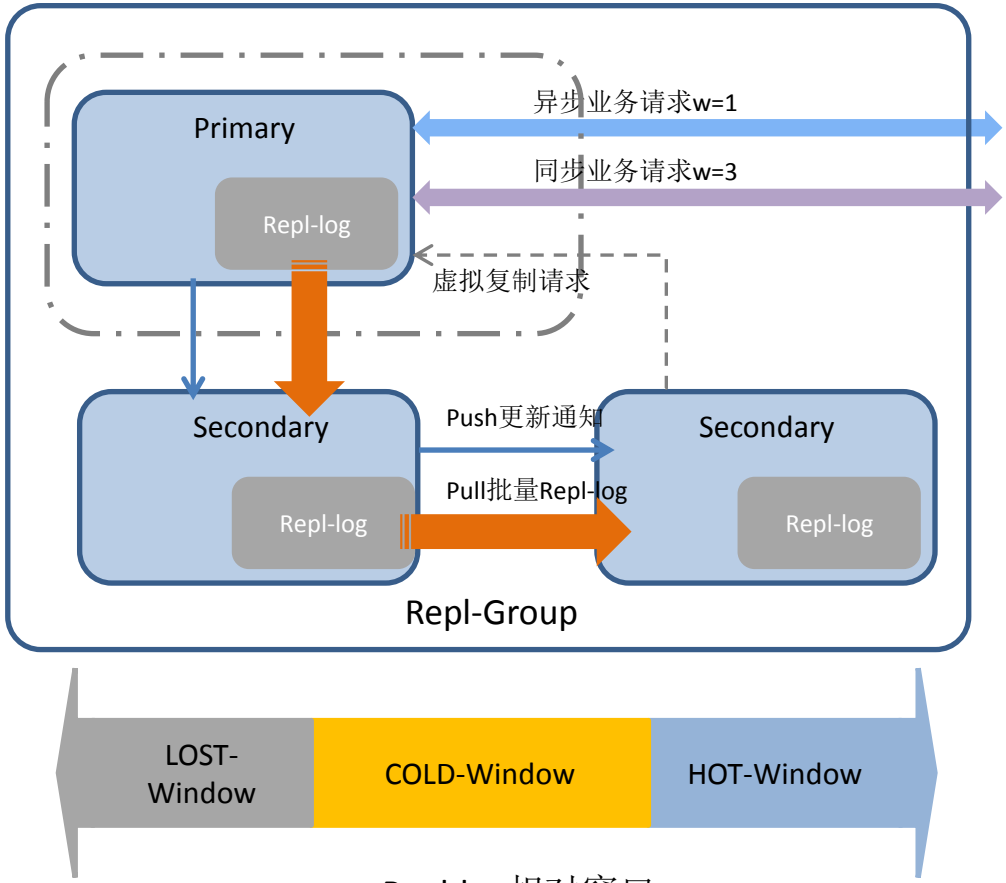
一致性对比

- **最终一致性**
- 优势
  - 速度快
  - 读写分离
  - 大部分情况下工作良好
- 适合场景
  - 对数据精确性要求不高
  - 允许少量的数据丢失
  - 对性能要求较高
  - 网络不稳定

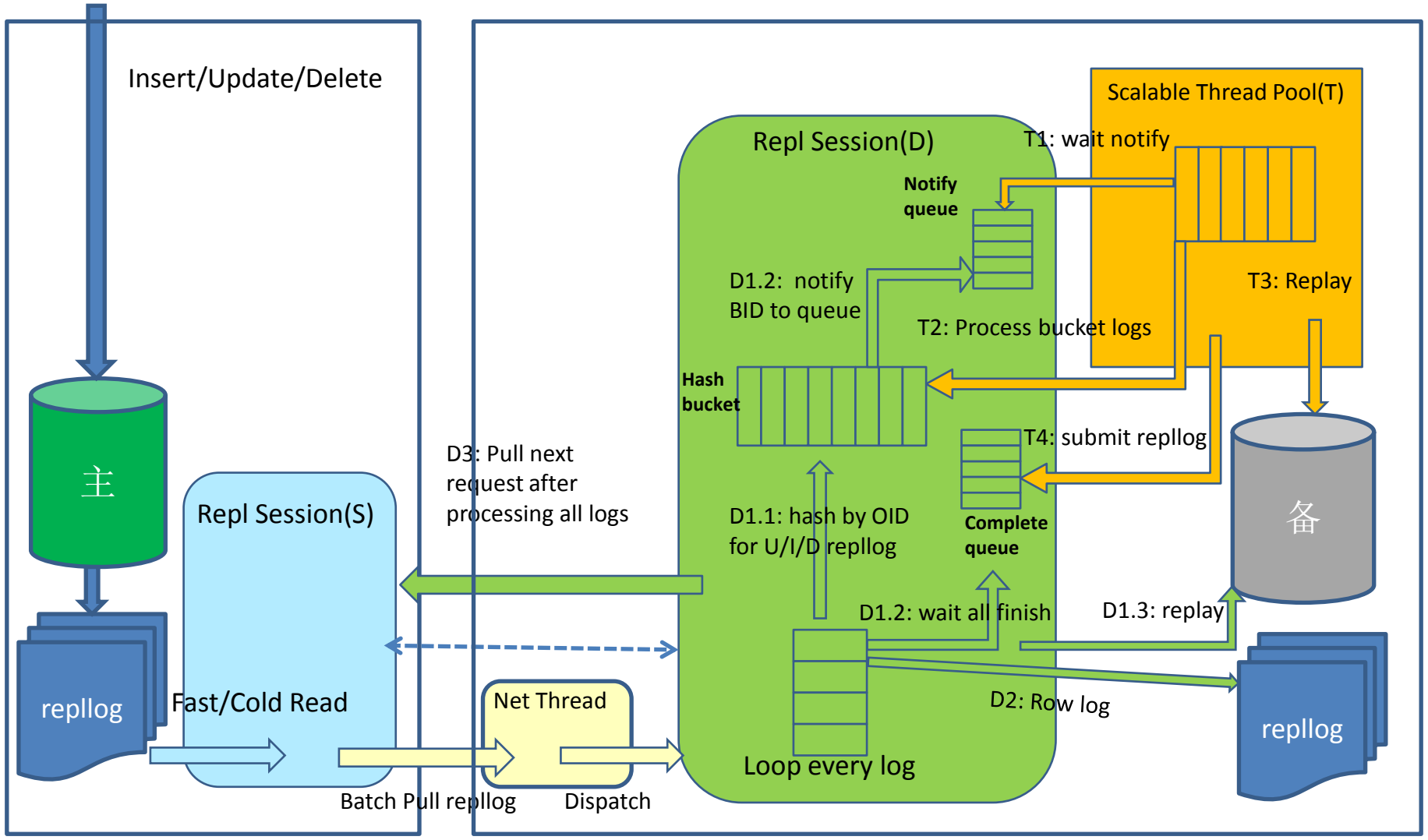
- **强一致性**
- 优势
  - 数据安全可靠
- 适合场景
  - 对数据精确性要求高
  - 不允许数据丢失
  - 对性能要求一般
  - 对可靠性要求较高

一致性：主从机制

- 最经典的数据复制技术
  - DB2 HADR
  - Oracle DataGuard
  - MySQL binlog复制
- 主节点读写提供传统强一致的保障
- 从节点只读提供脏读隔离级别
- 控制备节点响应策略可以控制持久性
- 控制读写同步比例来控制一致性



主备同步机制 – 并行Replay机制



影响一致性的因素

一致性因素	读主节点	读备节点
W=N	强一致，无数据丢失，事务操作符合传统关系型数据库模型	强一致，无数据丢失，读操作不受隔离级别影响（脏读）
W!=N	强一致，极端情况下可能发生数据丢失，事务操作符合传统关系型数据库模型	最终一致，可能发生读取不到写入的数据，极端情况下可能发生数据丢失

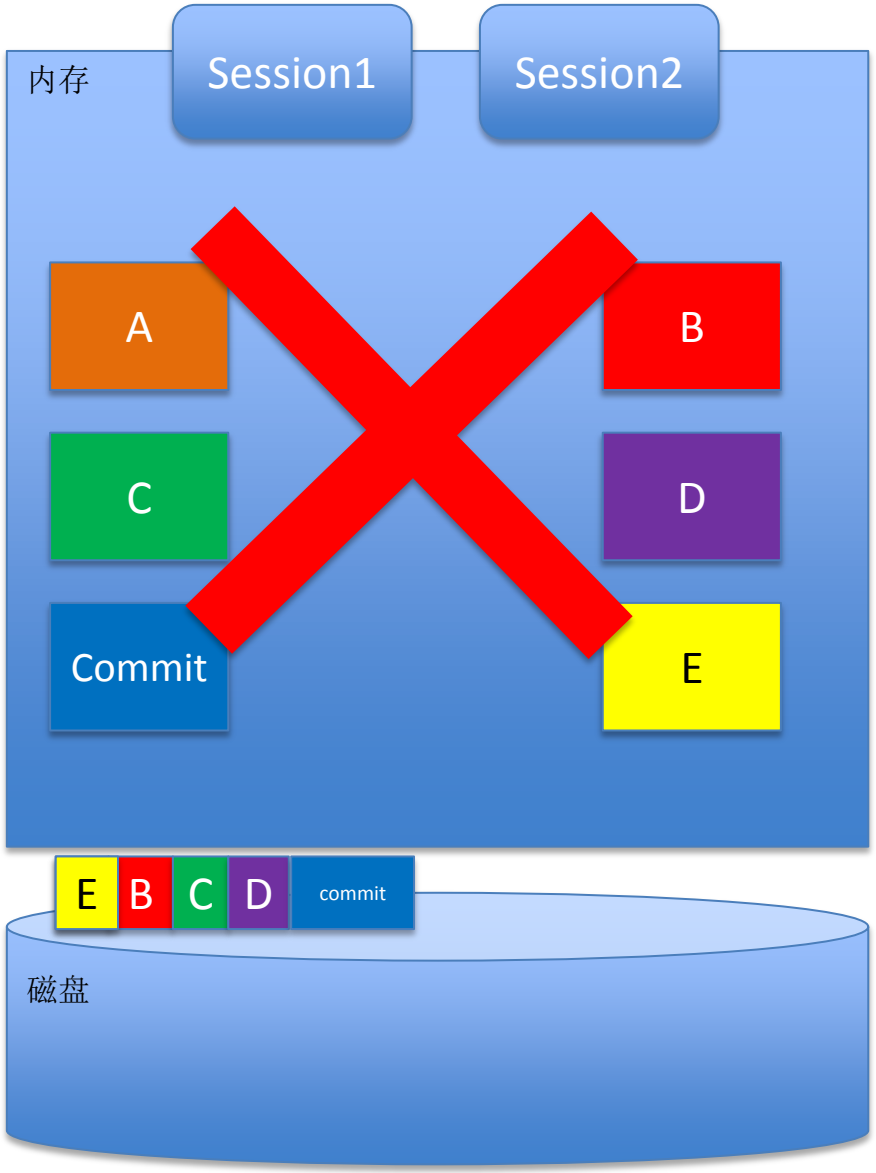
持久性因素	备节点响应策略	数据可靠性
物理同步	数据在备节点写入事务日志	全部节点宕机不会造成数据丢失，但会损失性能
逻辑同步	数据在备节点处理但未写入事务	主备节点同时宕机可能会造成数据丢失，数据查询强一致
半同步	数据在备节点成功接收但还未处理	备节点宕机可能会造成数据丢失，最终一致性
异步	数据不需要被备节点感知	主节点宕机可能会造成数据丢失

持久性

- 全部确认提交的数据必须被持久化
- 断电、崩溃等操作不会造成已确认提交的数据丢失
- 传统数据库使用Redo、Undo机制实现

持久性：Redo+Undo

- 日志中记录了数据在每个数据页的操作
- 通过顺序回放数据页操作，数据库可以重新执行全部操作
- 前提：
  - 日志文件不丢失数据
  - 数据页无损坏（一般由外接存储保障）

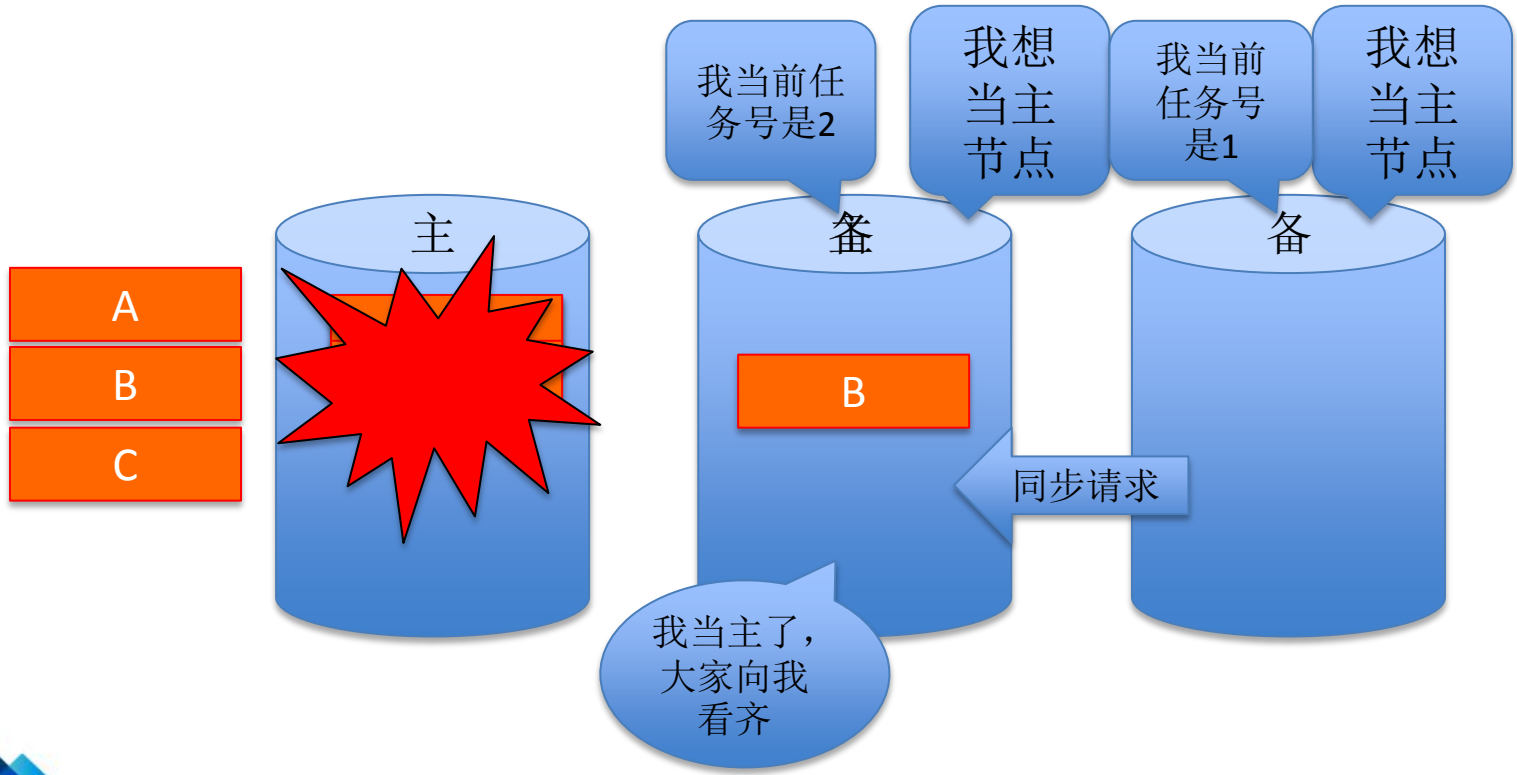


集群下持久性

- PC服务器内置磁盘在断电的情况下可能会丢失数据或造成数据损坏
  - 一些数据页来不及从控制器写入磁盘
  - 数据页之间造成元数据不一致
- 在非极端情况下（所有节点同时崩溃或掉电），可以通过备节点恢复数据
- 前提：
  - 提交操作必须确保写入备节点
  - 备节点数据在崩溃恢复时需完整拷贝入需恢复节点
  - 恢复过程中所有主节点操作需同步入恢复节点

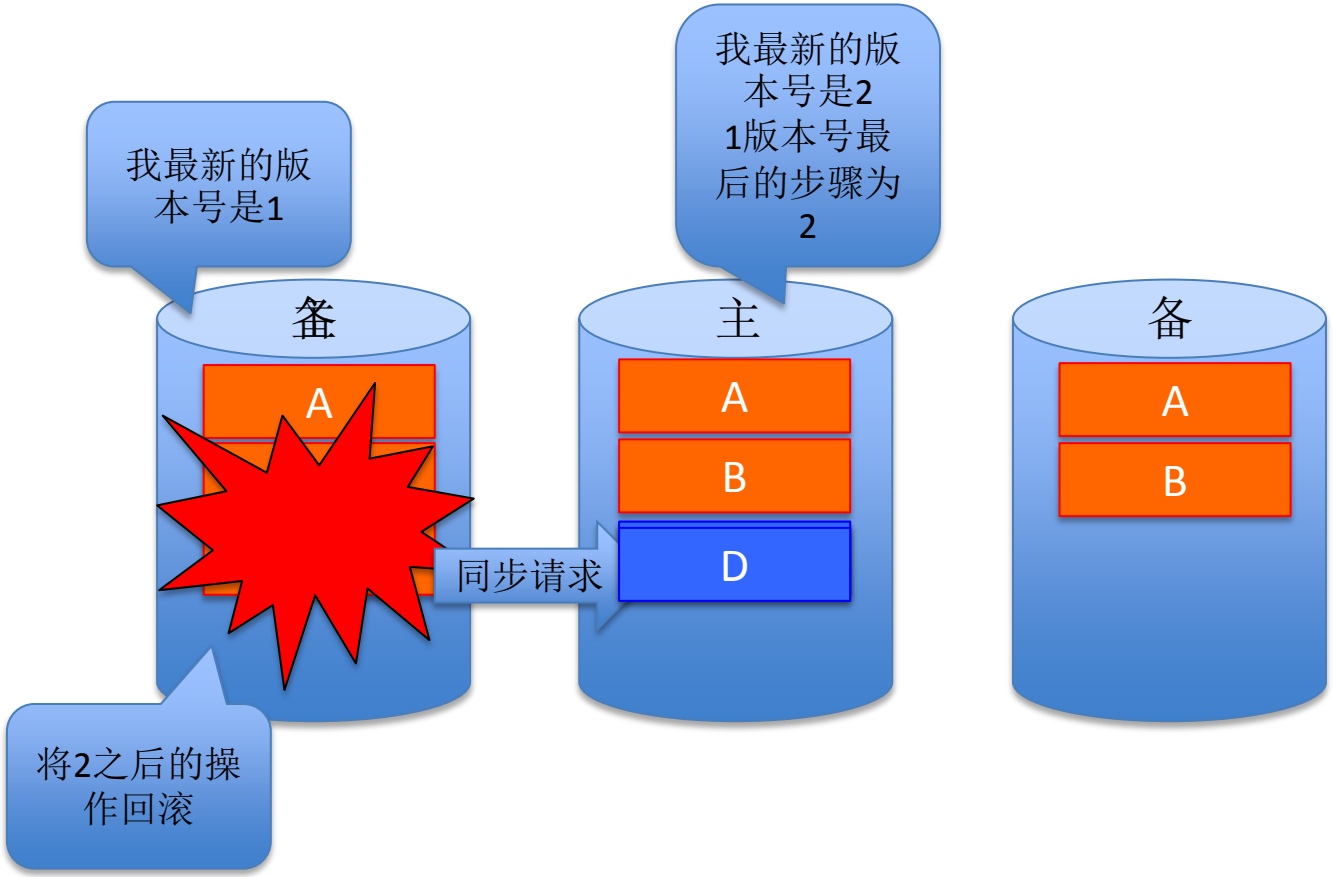
重新选举时的一致性要素

- 在原主节点宕机，原备节点生主后，如何在其他备节点之间做到数据一致性？
- 如果事务开启，重新选举机制会回滚未完成事务



重新加入集群时的一致性要素

- 当原主节点恢复，如何将该节点与当前的主节点同步？



分布式事务机制

- 二段提交
- 协调节点首先发起预提交
- 当所有数据节点响应成功后，进行统一提交

有疑问事务的处理方式（Indoubt transaction）

- 如果在预提交阶段后数据节点出现故障，会造成事务处于疑问模式
  - 如果节点在提交请求发生后执行了提交操作，但是无法返回结果，意味着事务应当成功
  - 如果节点在提交请求发生前故障，意味着事务应当失败
  - 但是协调者无法判断当时真正的状态告知其他节点

- 手工解决疑问模式
  - 提交
  - 回滚

CAP原则对NoSQL事务的影响

- 传统数据库：C+A
  - 机制成熟
  - 数据稳定可靠
- 内存数据库/缓存：A+P
  - 事务无法保证在极端情况下的可靠性
  - 可以用来进行简单的提交回滚操作，并在正常情况下可以正常工作
- 持久化存储NoSQL：C+P
  - 和传统数据库相比一致性模型引入了网络不稳定性的问题

SequoiaDB提供了NoSQL的事务功能

- 文档类NoSQL数据库
  - 与MongoDB为同类型数据库
- 100%完全开源
  - 引擎：AGPL协议
  - 客户端：Apache协议
- 已获得启明创投的A轮千万美元级别融资

Demo

Thanks!

王涛
SequoiaDB联合创始人
MP: +86 18617391323
Email: taoewang@sequoiadb.com