分区（Partitioning）与复制（Replication）

本节内容

- **理解：**
  - 为什么需要分区与复制
  - 两者解决的问题差异
- **掌握：**
  - 常见分区策略
  - 一致性与可用性权衡
- **能够：**
  - 设计基础分布式数据架构

单机系统的瓶颈

- **存储容量有限**
- **CPU/内存有限**
- **单点故障（Single Point of Failure）**

- 👉 **问题：**
  - 数据太多怎么办?
  - 机器挂了怎么办?

两个核心解决思路

- **两个核心解决思路**
- **数据太大**
  - 分区（Partitioning）
- **容易宕机**
  - 复制（Replication）

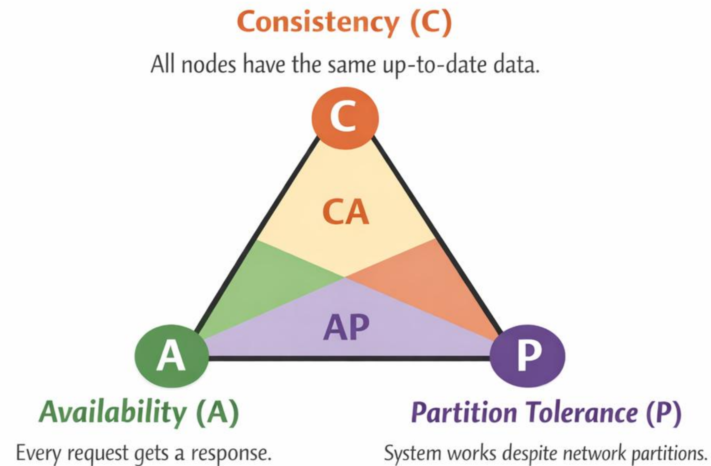
分区（Partitioning）

- 将数据拆分到多个节点
- 每个节点存储部分数据
- Horizontal Scaling（水平扩展）

不要和CAP中的“Partition”混淆

- **Network Partition**
- **划分为相对独立的子网**
- **设计/单独优化/网络设备故障**

- **CAP中的P指网络分区**
- **分布式软件必须设计为分区兼容**
  - 即使网络被分割后
  - 它仍然能正常工作



分区解决的问题

- **数据规模扩展（Big Data）**
- **吞吐提升（Parallelism）**
- **降低单机压力**

- **例如：**
  - 用户数据按ID分布到不同服务器

分区的核心挑战

- 如何分配数据?
- 如何查询数据?
- 如何避免数据倾斜（skew）？

分区策略

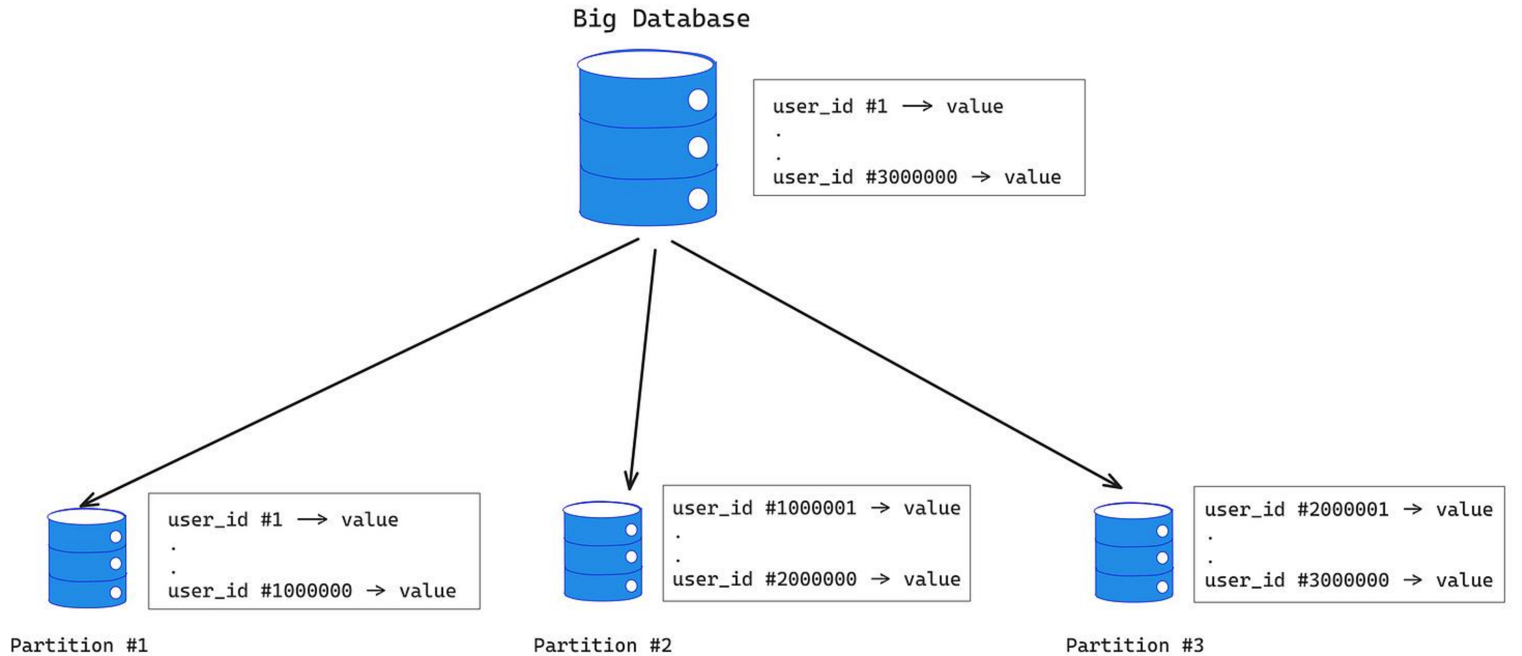
- 键范围分区（Range Partitioning）
- 哈希分区（Hash Partitioning）

范围分区（Range Partitioning）

- 数据分布基于某个范围内的值
- 每个分区独立管理

范围分区（Range Partitioning）

- **Partition 1:**
  - user_id from 1 to 1,000,000
- **Partition 2:**
  - user_id from 1,000,001 to 2,000,000
- **Partition 3:**
  - user_id from 2,000,001 to 3,000,000



What is key based vs range based

特点

- **支持范围查询（range scan）**
- **数据局部性好**

- **容易数据倾斜（热点问题）**
- **例如：**
  - Key是时间
  - 每天一个分区

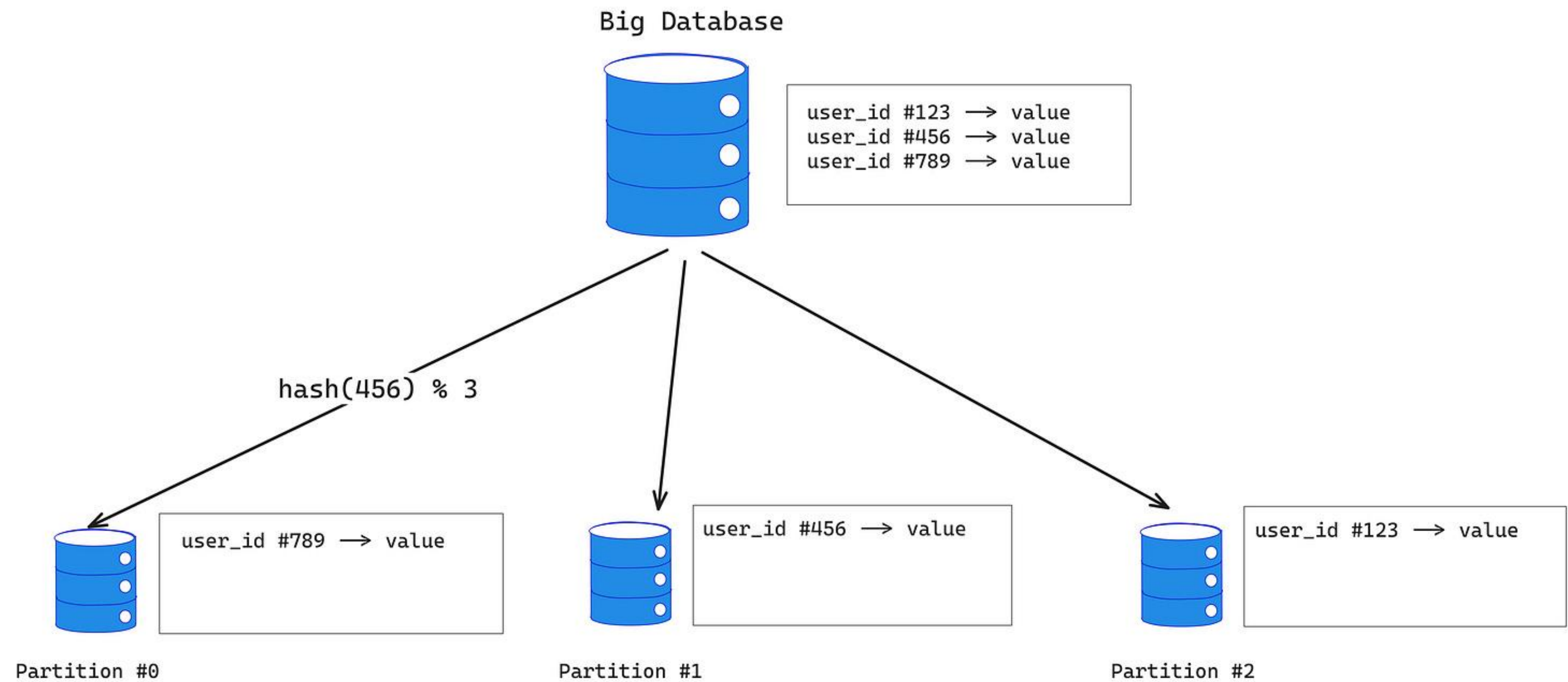
典型应用

- HBase
- Bigtable

哈希分区

- 对 key 进行 hash 运算后分配节点
- hash(user_id) % N

- **User with user_id 123**
  - —> hash(123) = 84362
  - —> 84362 % 3 = 2.
  - The record goes to Partition 2.
- **User with user_id 456**
  - —> hash(456) = 19371
  - —> 19371 % 3 = 1.
  - The record goes to Partition 1.
- **User with user_id 789**
  - —> hash(789) = 55219
  - —> 55221 % 3 = 0.
  - The record goes to Partition 0.



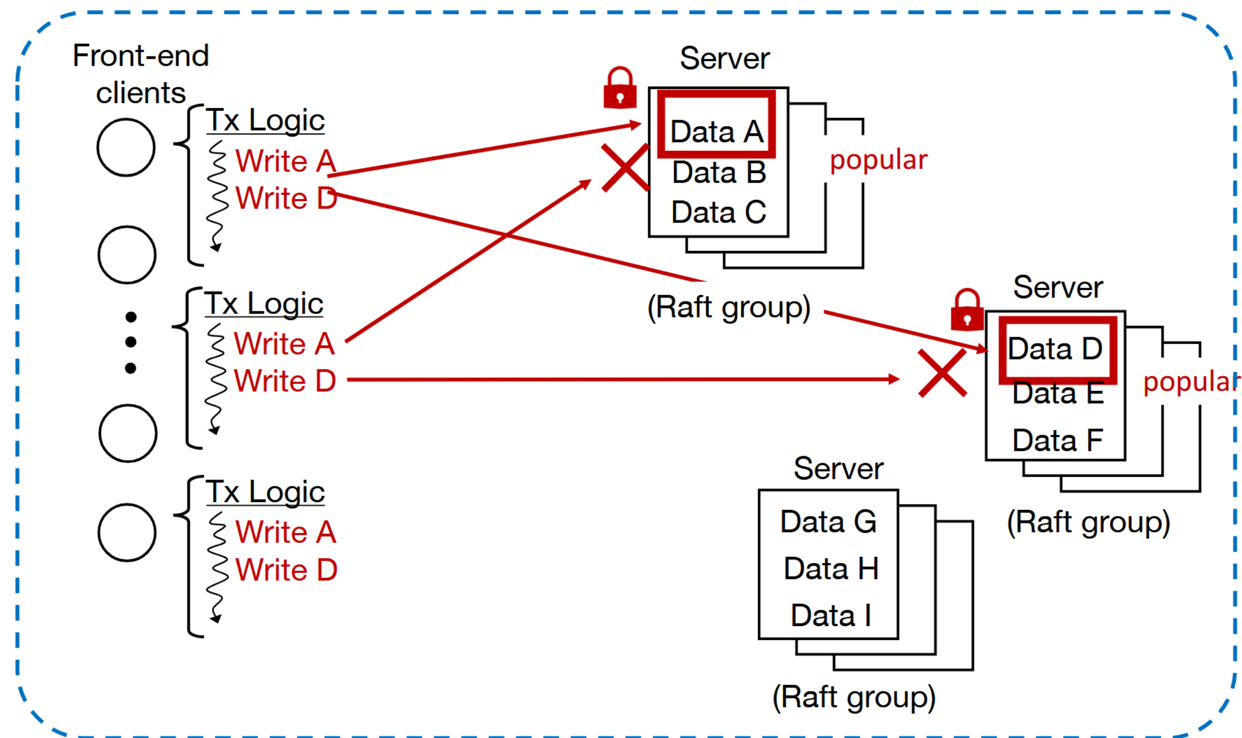
[What is key based vs range based]

特点

- 数据均匀分布
- 避免热点
- 不支持范围查询
- 扩容困难（rehash）

负载倾斜

- **数据/访问分布不均**
- **导致单个/少数分区负载过高**



Accelerating Skewed Workloads With Performance Multipliers in the TurboDB Distributed Database

Jennifer Lam

Jeffrey Helt Wyatt Lloyd Haonan Lu

Princeton University University at Buffalo

热点

2026
豆包过年
除夕AI总互动19亿次
陪大家热热闹闹过除夕

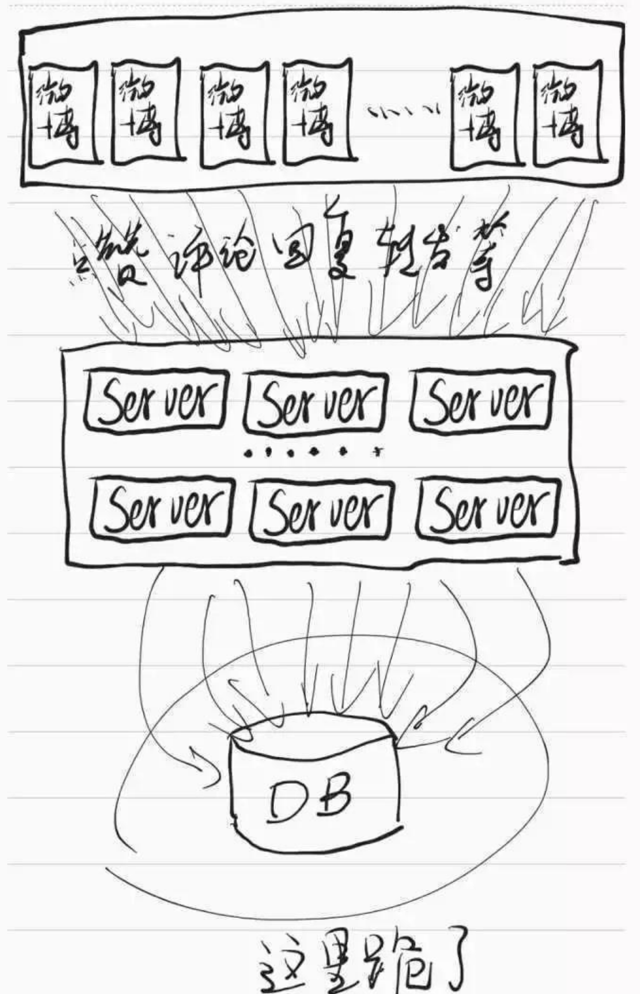
天猫双11
2025
天猫双11
全年最优惠
10月20日晚8点正式开卖

红包大战
千问
文心一言
腾讯元宝

M鹿M
44分钟前 来自 vivo X20全面屏手机
大家好，给大家介绍一下，这是我女朋友@关晓彤
19万 27万 58万

关晓彤
43分钟前 来自 HUAWEI nova 2 Plus
哎呀嘛，咔咔的！@M鹿M
1万 3万 19万

微博客服 + 关注
10-8 12:32 来自 微博 weibo.com
#微博公告#目前客户端无法正常刷新、评论等多个页面无法正常显示的问题，工程师已在排查（具体怎么造成的，大家心里也都有数），给您带来的不便敬请谅解，修复进度请您关注官方账号最新动态。
微博帮助 help.weibo.com #微博公告#



日	一	二	三	四	五	六
例 购票日期 乘车日期	一月 19 2月2日	大寒 20 2月3日	21 2月4日	22 2月5日	23 2月6日	24 2月7日
25 2月8日	26 2月9日	27 2月10日 北小年票	28 2月11日 南小年票	29 2月12日	30 2月13日	31 2月14日
二月 1 2月15日 廿八票	2 2月16日 除夕票	3 2月17日 春节票	立春 4 2月18日 初二票	5 2月19日 初三票	6 2月20日 初四票	7 2月21日 初五票
8 2月22日 初六票	9 2月23日 初七票	北小年 10 2月24日	南小年 11 2月25日	12 2月26日	13 2月27日	班 14 2月28日
休 廿八 15 3月1日	休 除夕 16 3月2日	休 春节 17 3月3日 元宵节票	休 雨水 18 3月4日	休 初三 19 3月5日	休 初四 20 3月6日	休 初五 21 3月7日
休 初六 22 3月8日	休 初七 23 3月9日	24 3月10日	25 3月11日	26 3月12日	27 3月13日	扫码关注 中国铁路微信

·新线开通·余票查询·失物查找·重点旅客服务预约·获取更多数路资讯 一键抵达

《人民铁道》报业有限公司移动传播中心出品

鹿晗和关晓彤干崩了微博

处理热点

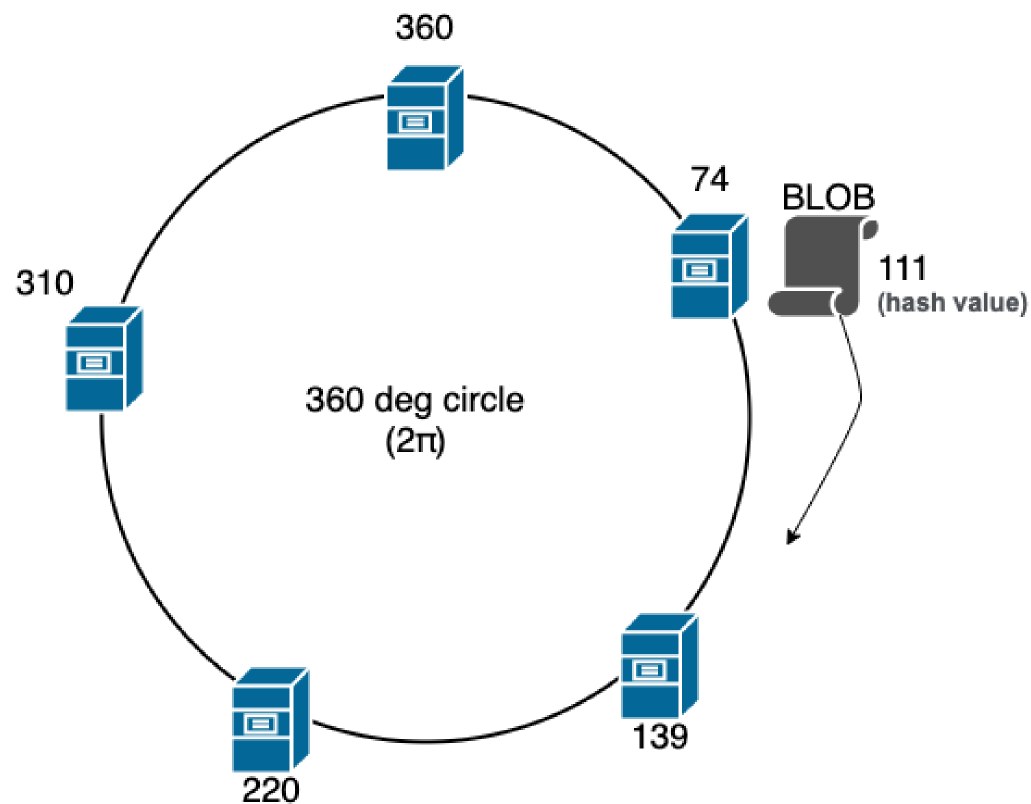
- **识别热点**
- **限流**
  - 瞬间流量进行限制
  - 放入消息队列异步处理
- **读写分离**
- **多级缓存**
- **随机**
  - Key上添加随机数前缀/后缀
  - 拆分成多个子Key分布
  - 聚合，去除随机

重分区

- 处理倾斜
- 经过一段时间，数据分布变了
- 手动
- 自动

一致性哈希（扩展）

- **整个哈希空间:**
  - 映射到一个**环（0 ~ 2^32）**
- **节点（Node）：**
  - 通过 hash 映射到环上的点
- **数据（Key）：**
  - 也映射到环上



一致性哈希（扩展）

- **解决节点增减问题**
  - 新节点插入环中
  - 只影响局部数据
  - 不需要全量 rehash（这是最大优势）
- **在动态节点环境下，实现最小数据迁移的分区方案**
- **常用于分布式缓存**
- **DynamoDB**
- **Cassandra**

路由

- **客户端请求到哪里去?**
- **重分区后，如何规划新的路由?**

- **客户端缓存路由表**
- **传到一个路由中间层**
  - 中间层再分发
- **允许客户端请求发到任何节点**
  - 节点自己执行
  - 传到对应的期望的节点

什么是复制

- **将同一数据存储在多个节点**

- **冗余（Redundancy）**
- **对抗失败**

复制解决的问题

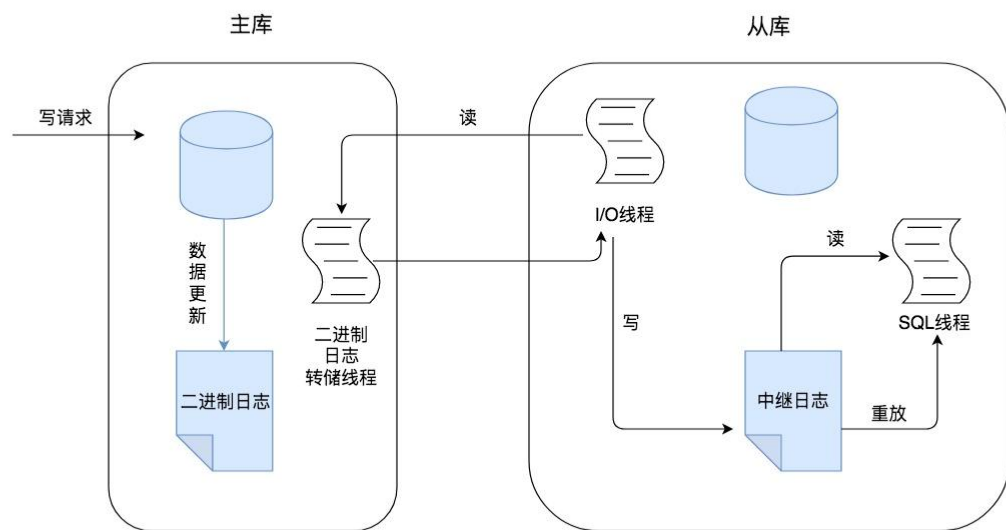
- **高可用（High Availability）**
- **容错（Fault Tolerance）**
- **提高读取性能**

复制策略

- **主从复制（Leader-Follower）**
- **多主复制（Multi-Leader）**
- **无主复制（Leaderless）**

主从复制

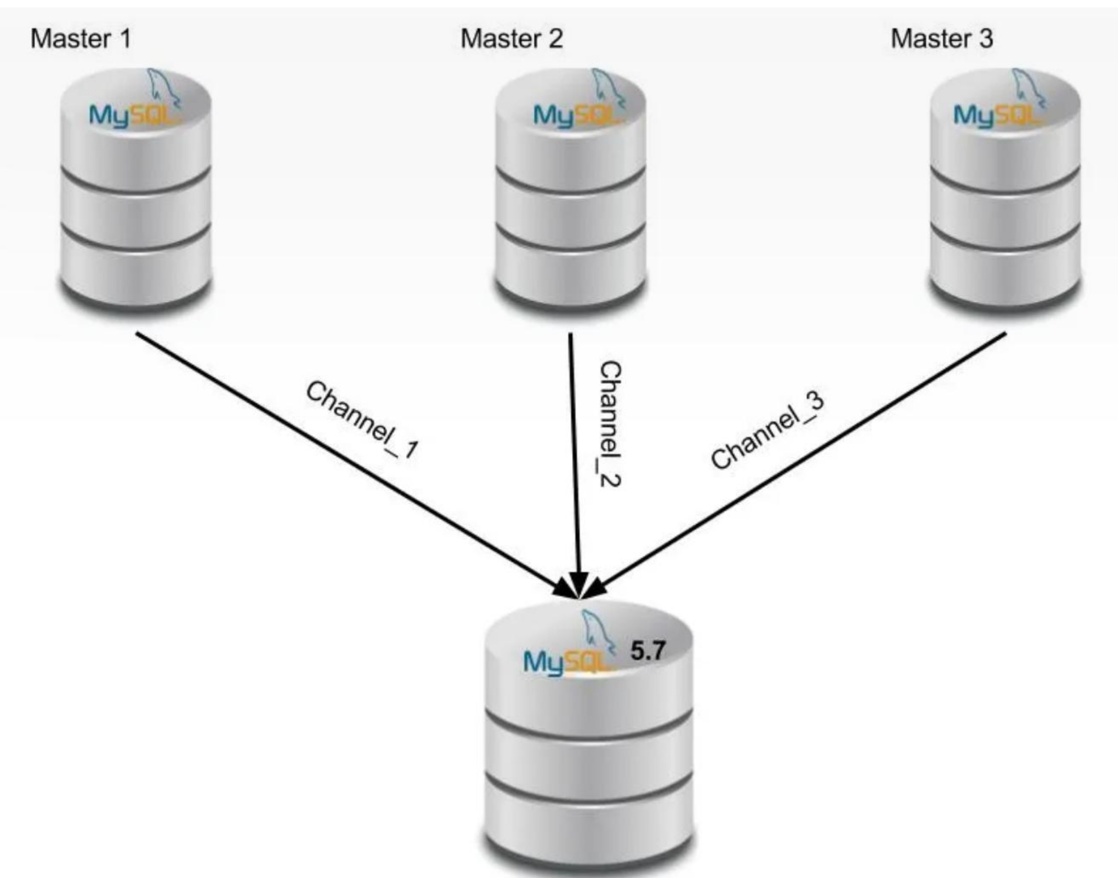
- 一个主节点（写）
- 多个从节点（读）
- 简单
- 主节点瓶颈



mysql主从复制原理

多主复制（Multi-Leader)

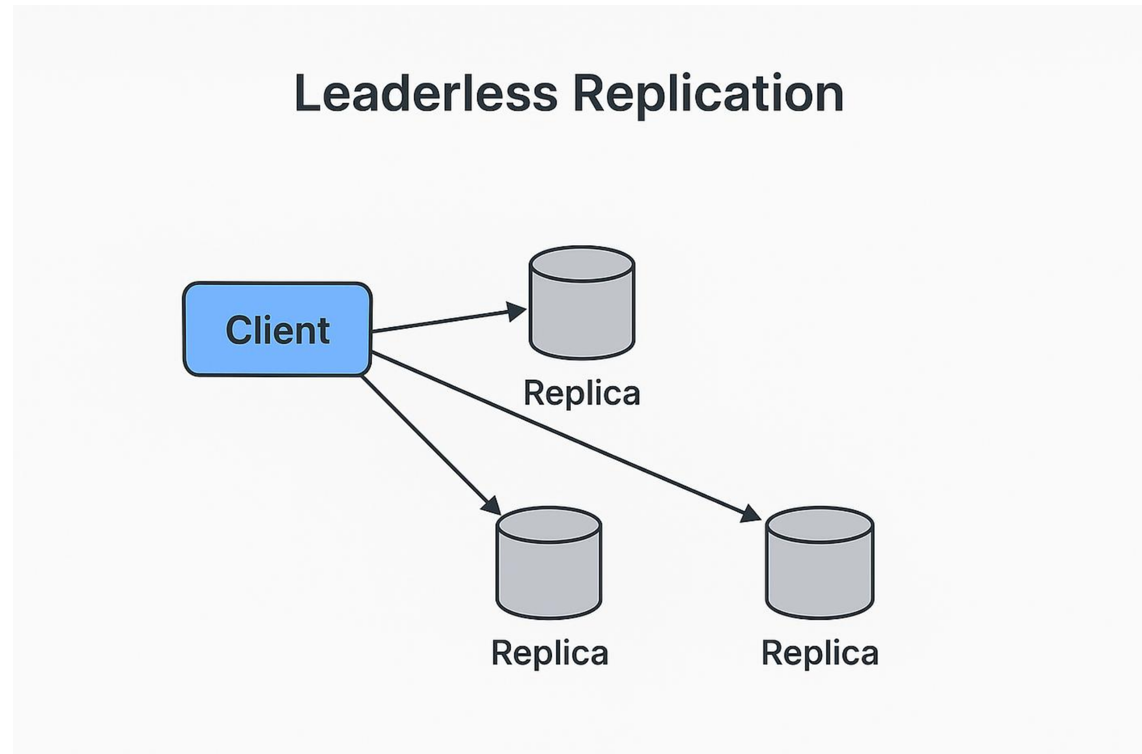
- **有多个节点充当主节点**
- **一组主节点上进行写（更新和插入）**
- **数据有多个副本**
- **问题：**
  - 并发写冲突
  - 需要有合适的解决策略



https://www.mydbops.com/blog/disabling-multi-source-replication-in-mysql-5-7

无主复制

- 无主复制
- 不部署在固定主节点（集合）
- 每个副本都可独立地接受写



[Leaderless Replication: Inside Dynamo-Style Datastores]

解决冲突策略（自动）

- **Conflict-free Replicated Data Types (CRDTs)**
  - 可以在网络中的多个节点上复制的数据结构
  - 副本可以独立和并发地更新
  - 不需要在副本之间进行协调
  - 在数学上总是可以解决可能出现的不一致问题
- **系统案例：Riak**

Conflict-free Replicated Data Types

Nuno Preguiça
DI, FCT, Universidade NOVA de Lisboa and NOVA LINCS, Portugal

Carlos Baquero
HASLab / INESC TEC & Universidade do Minho, Portugal

Marc Shapiro
Sorbonne-Université & Inria, Paris, France

20 February 2018

- **Mergeable persistent data structures**
  - 能否像 Git 管理代码一样管理“数据结构”?
  - 把“版本控制 + merge”能力引入任意数据结构
  - Data Structure
    + Version Control
    + Merge Semantics

Mergeable persistent data structures

Benjamin Farinier[1], Thomas Gazagnaire[2] and Anil Madhavapeddy[2]

1: ENS Lyon
benjamin.farinier@ens-lyon.fr

2: University of Cambridge
thomas.gazagnaire@cl.cam.ac.uk
anil.madhavapeddy@cl.cam.ac.uk

容忍失败节点

- **Quorum（法定人数）机制**
- **N = 2f + 1**
  - N个节点，最多容忍 f 个节点失败
  - Raft、Paxos
- **N = f + 1**
  - N个节点，最多容忍 f（N-1） 个节点失败
  - 仅保证可用性（Availability）
  - 无法保证一致性（Consistency）
- **W + R > N**
  - W（写确认数） + R（读确认数） > N

- **故障检测**
  - 心跳
  - 超时
- **自动切换**
  - 主节点失败 → 选举新主
- **数据同步策略**
  - 同步
  - 异步

一致性问题

- **复制一致性（Replication Consistency）**
  - 数据副本之间的语义保证
- **内部机制**
- **一致性模型，是一整个谱系（spectrum）**
  - Strong Consistency (Linearizability)
  -
  - Sequential Consistency
  -
  - Causal Consistency
  -
    - Session Guarantees
    - ├── Read-your-writes
    - └── Monotonic reads
  -
  - Eventual Consistency
- **不要与CAP的“Consistency”混淆**
  - 它们相关联
  - 但又不是一回事
- **CAP中的“C”**
  - 外部观察模型
  - 连续的Strong Consistency (Linearizability)
  - system ≈ single-copy system
  - 看起来和单机一样

Strong Consistency

- **定义**
  - 每个操作
  - 看起来像在某个全局时间点瞬时发生
- **等价于**
  - 单副本语义
  - 满足 real-time order
- **保证**
  - 读一定返回**最新写**
  - 所有客户端看到**相同顺序**
  - respects wall-clock order
- **不允许**
  - stale read
  - 重排序

Sequential Consistency

- **定义**
  - 所有操作存在一个全局顺序
  - 但不要求符合真实时间
- **保证**
  - 所有节点看到**相同操作顺序**
  - 每个进程内部顺序保持
- **不保证**
  - 不保证最新写
  - 不保证 real-time order

T1: write(x=1)
T2: read(x) -> 0 ← allowed（如果全局顺序把读排在写前）

Causal Consistency

- **定义**
  - 只保证**有因果关系的操作顺序一致**
- **因果关系来源**
  - 程序顺序（program order）
  - 读→写（read-from）
  - 传递性（transitive closure）
- **保证**
  - 因果顺序一致
  - 无因果序的->允许乱序
- **不保证**
  - 全局顺序
  - 最新写

Session Guarantees

- **Causal consistency**
  - 有关联，但不完全等价
- **Read-your-writes**
    - 你写过的数据，你之后一定能读到
  - 例如：
    - x=0 → write(x=1) → read(x) >= 1
    - 同一客户端
- **Monotonic Reads**
  - 一旦读到某个值，之后不会读到更旧的值
  - 例如：
    - x=0 → write(x=1) → read(x) >= 1
    - 若第一次读到 x=1
    - 后续不会读到 x=0
- **不保证**
  - 一定是最新值
  - 写后可见

Eventual Consistency

- **定义**
  - 如果没有新写入，所有副本最终会收敛
- **保证**
  - 收敛
- **不保证**
  - 何时一致
  - 顺序一致
  - 读到最新值

t0: write(x=1)

t1: A 读到 0
t2: B 读到 1
t3: 所有节点最终变成 1

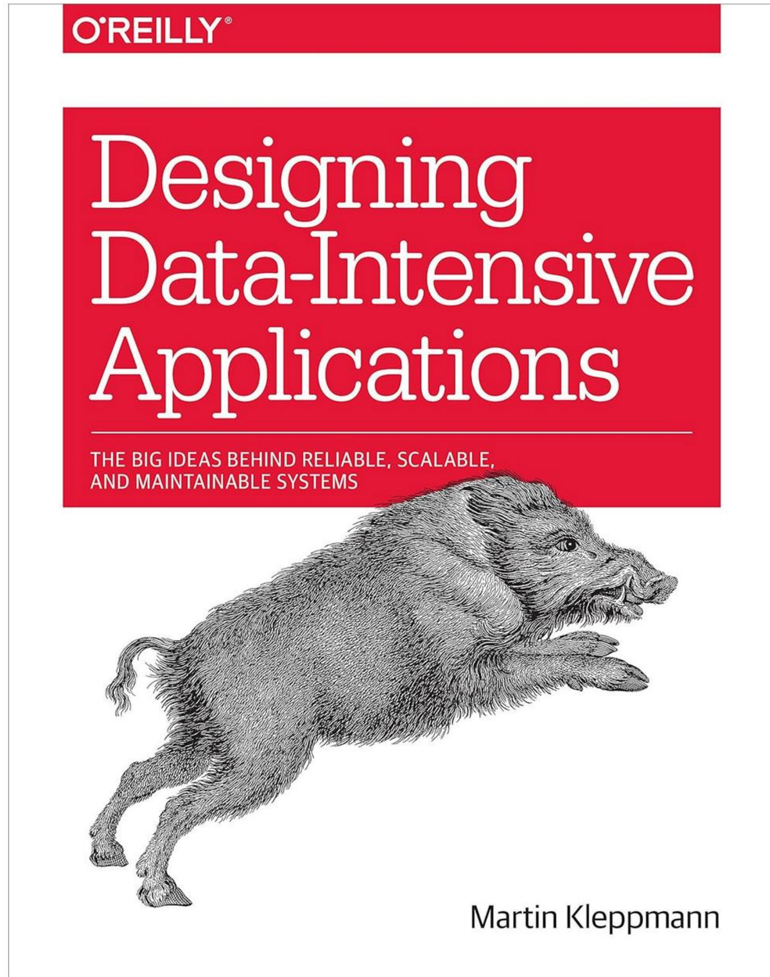
不同的一致性模型

模型	是否全局顺序	是否实时顺序	是否保证最新读	是否允许乱序
Strong	✅	✅	✅	❌（强、时间）
Sequential	✅	❌	❌	❌（无时间）
Causal	❌（非因果部分）	❌	❌	✅（非因果部分）
• RYW	❌	❌	✅（仅自己）	✅（非因果部分）
• Monotonic	❌	❌	❌	✅（非因果部分）
Eventual	❌	❌	❌	✅

核心总结

- **分区：**
  - 解决“数据太多”
- **复制：**
  - 解决“系统不可靠”
- **组合使用：**
  - 构建大规模分布式系统
- **分区与复制，它们一般会同时出现**
  - 思考：为什么？

参考



- **Designing Data-Intensive Application**
  - C5，C6