



分布式存储

(第五部分)

李洪亮

lihongliang@jlu.edu.cn

云计算和网络安全实验室

吉林大学 计算机科学与技术学院



- 概论 (4)
- 分布式系统基础 (8)
 - 分布式系统发展
 - 高性能计算和数据中心（架构与管理）
- 分布式存储技术 (9)
- 分布式存储系统 (27)
 - 网络文件系统 (NFS)
 - 分布式文件系统 (HDFS, GFS)
 - 分布式键值对存储 (Dynamo)
 - 分布式表格系统 (BigTable)
 - 分布式对象存储 (S3)





- 内容提要
 - Dynamo简介与架构
 - 数据分布
 - 一致性与复制
 - 容错和负载分配
 - 读写流程与应用

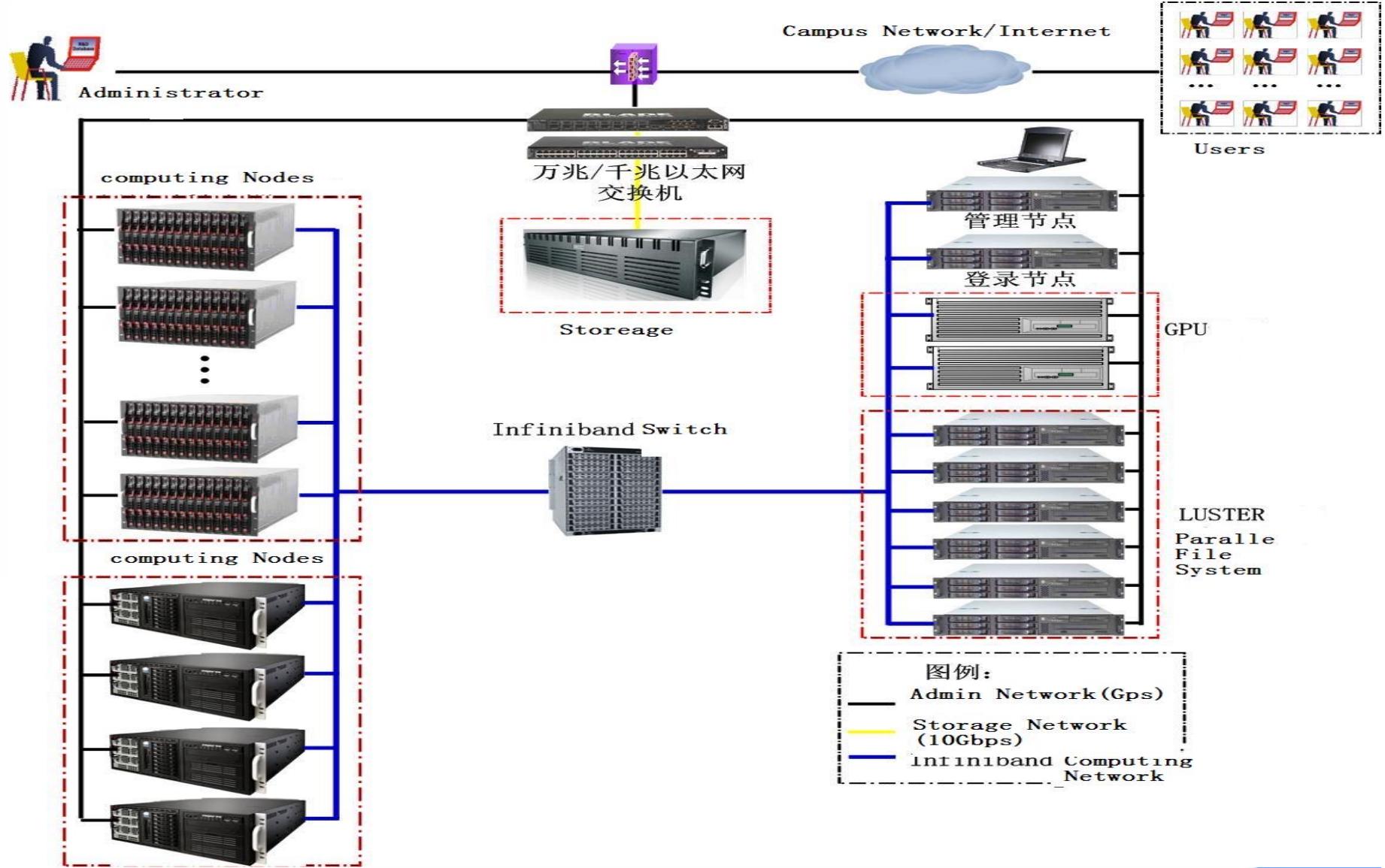


背景(1)

- 分布式系统频繁出现故障，可能造成数据损坏甚至服务中断
- 分布式系统底层拓扑结构复杂，如异构系统和复杂网络拓扑结构(多数据中心)
- 如何在此类大规模分布式系统中权衡**容错性、一致性和可用性**？



吉林大学高性能计算中心

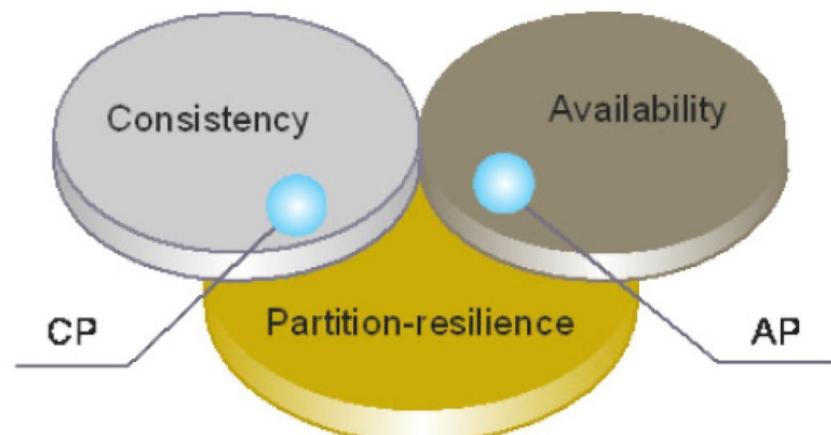




背景(2)

- 一致性：并发操作的结果与串行操作结果一致
- 可用性：读写延迟小
- 分区容错性：部分节点故障的情况下能持续提供服务

- [Grey Box] Same up-to-date data on each node
- [Grey Box] Minimum downtime, quick access to data
- [Yellow Box] No unresponsiveness, fault-tolerance

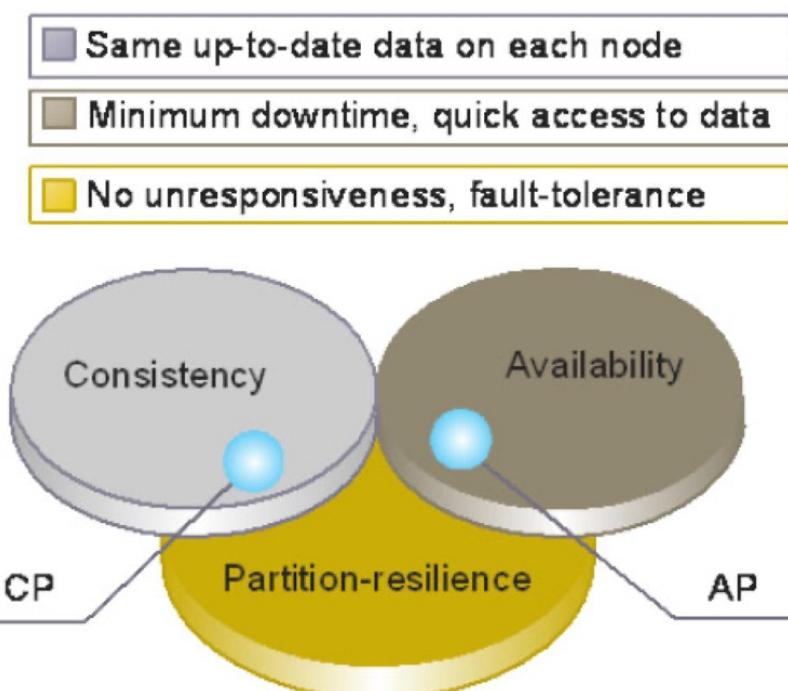




背景(3)

- CP(一致性/容错性): NoSql(Hbase/Bigtable), HDFS
- AP(可用性/容错性): DynamoDB, GFS
- CA(一致性/可用性): MySQL/Oracle, NFS

CAP 理论认为分布式系统只能兼顾其中的两个特性，即出现 CA、CP、AP 三种情况。





背景(3)

- 需要一个分布式存储系统：
 - 可扩展
 - 支持 key-value形式数据和操作
 - 高可用性 (牺牲一致性)
 - 保证服务质量 Service Level Agreements (SLA)



系统假设

- 简单的读写操作(Simple query model), 单个key-value增/删/改/查
- 数据有一个唯一的ID(Key)
- 大多数Amazon的服务都需要这样一个简单的查询操作, 不是必须要关系模型
- 存储对象一般较小(数据小于1MB)



一致性保证(ACID)

- 应用中发现符合ACID的分布式存储系统往往
可用性较差
- 放宽一致性要求，来提高系统的可用性(*high availability*)



其他假设

- 假设系统运行环境没有入侵危险
- 不考虑安全性问题(如授权和认证等)

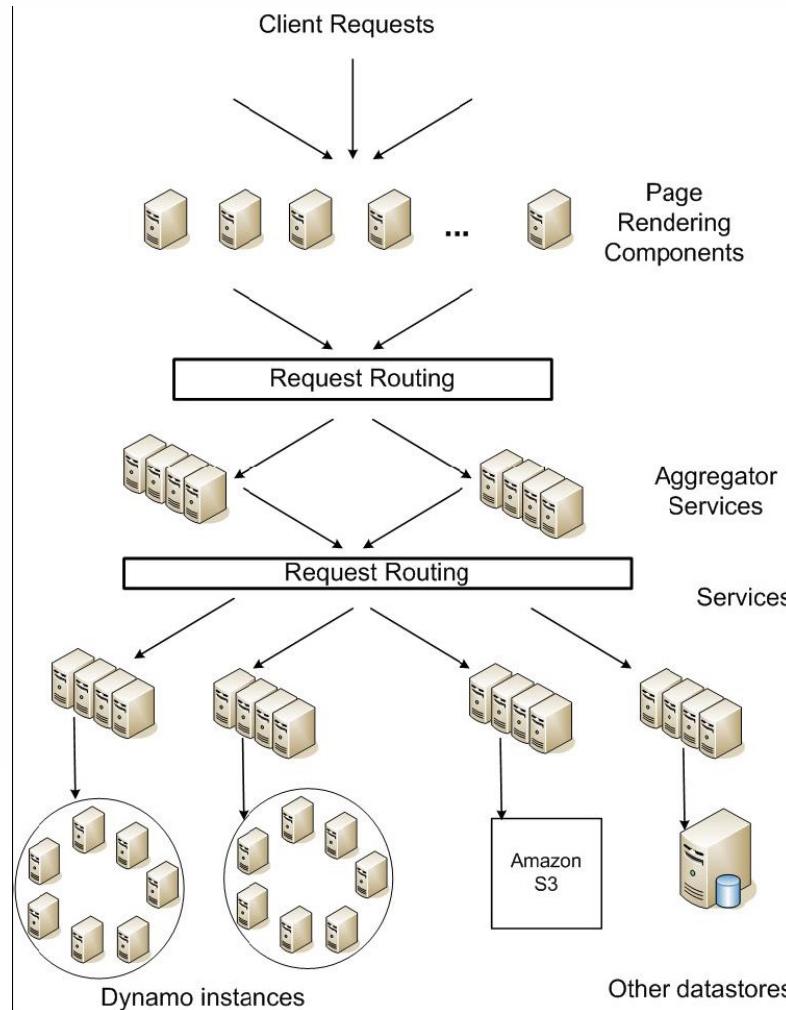


服务质量SLA

- 服务可以根据指定的时间限制来提供服务
 - bounded deliver time
 - 基于截止时间的工作流程
- 例子
 - 服务保证能在99.9%的情况下，在300ms内提供响应，支持最大并发度为每秒500个请求。



面向服务质量的架构



Dynamo: Amazon's Highly Available Key-value Store, SOSP07



设计思路(1)

- 牺牲强一致性换取可用性
 - *Sacrifice strong consistency for availability*
- 较弱的一致性导致数据冲突，怎么办？！
- 读的时候解决数据冲突，而不是写的时候来解决
- 支持数据随时可写“always writeable”.



设计思路(2)

- 支持按需扩展
- 对称系统架构(P2P)
 - 系统中所有节点担负相同的责任
- 去中心化
 - 避免中心化带来的潜在瓶颈问题
- 异构性和动态性
 - 支持异构机器进出系统
- 支持随时可写“always writeable”
 - 符合大规模并发写的需求



设计问题及解决方案

问 题	采取的技术
数据分布	改进的一致性哈希（虚拟节点）
复制协议	复制写协议（Replicated-write protocol, NWR 参数可调）
数据冲突处理	向量时钟
临时故障处理	数据回传机制（Hinted handoff）
永久故障后的恢复	Merkle 哈希树
成员资格及错误检测	基于 Gossip 的成员资格和错误检测协议





- 内容提要
 - Dynamo简介与架构
 - **数据分布**
 - 一致性与复制
 - 容错和负载分配
 - 读写流程与应用



哈希分布

- 哈希：根据数据的某种特性计算哈希值，建立哈希值与位置的映射关系
- 建立哈希值和节点之间的映射关系
- 服务器上下线， N 发生变化
 - 数据映射发生变化，大范围数据迁移
- 一致性哈希DHT



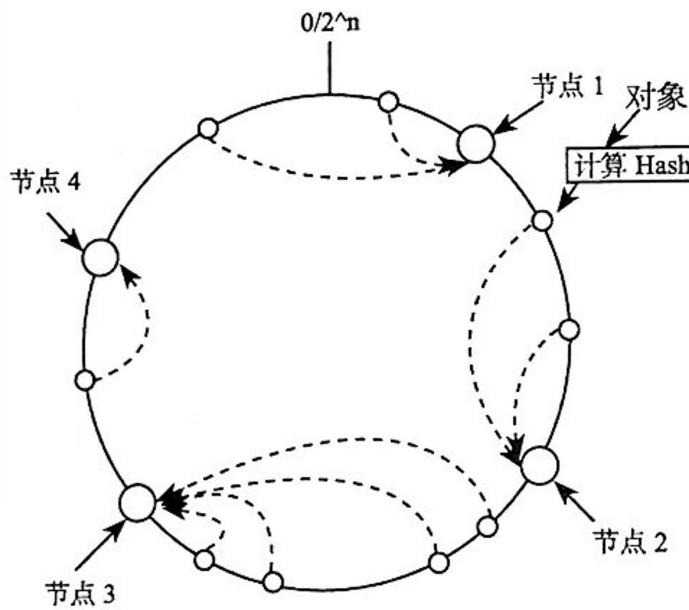
一致性哈希(Distributed Hash Table, DHT)

- 给系统中每个节点分配一个随机token
- token构成一个逻辑上的哈希环
- 存放数据时先计算key值的哈希
- 存放于顺时针方向第一个大于或等于该哈希值token的节点
- 优点：节点上下线仅影响邻居节点

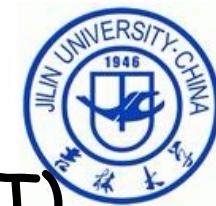


一致性哈希(Distributed Hash Table, DHT)

- 一致性哈希算法：
 - 求每个服务器hash值，分配到 $0-2^n$ 的环上
 - 同样的方法求待存储数据的主键hash值，放于环上
 - 从数据映射的位置顺时针查找第一个服务器



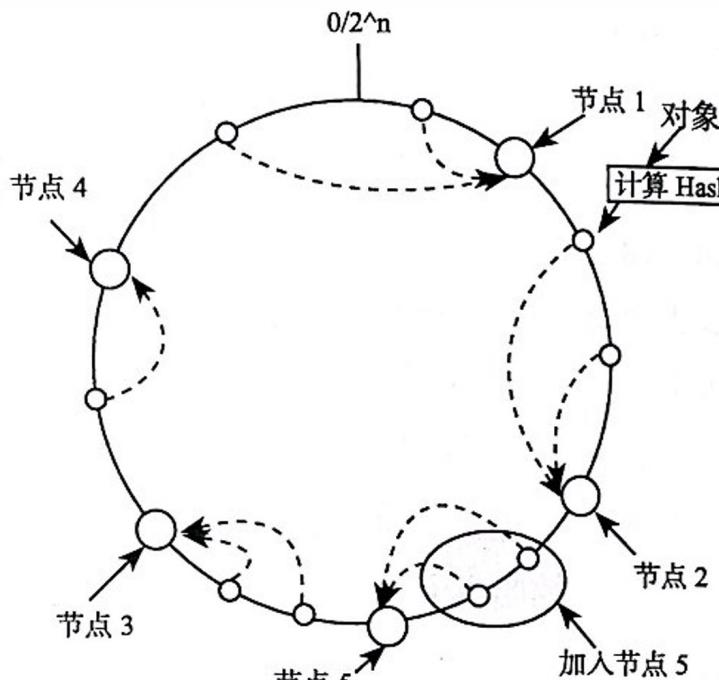
一致性哈希算法



一致性哈希(Distributed Hash Table, DHT)

- 添加服务器：

- 按照一致性哈希方法调整部分数据的存储位置
- 避免了大范围数据迁移



添加节点示意图



异构系统带来的问题

- 异构服务器的服务能力不同
 - 一致性hash平均分配负载
 - 服务能力差异导致总体性能下降（水桶效应）
- 虚拟节点——改进的一致性哈希
 - 设置虚拟节点，每个虚拟节点对应一个token
 - 物理节点可以根据服务能力承载多个虚拟节点

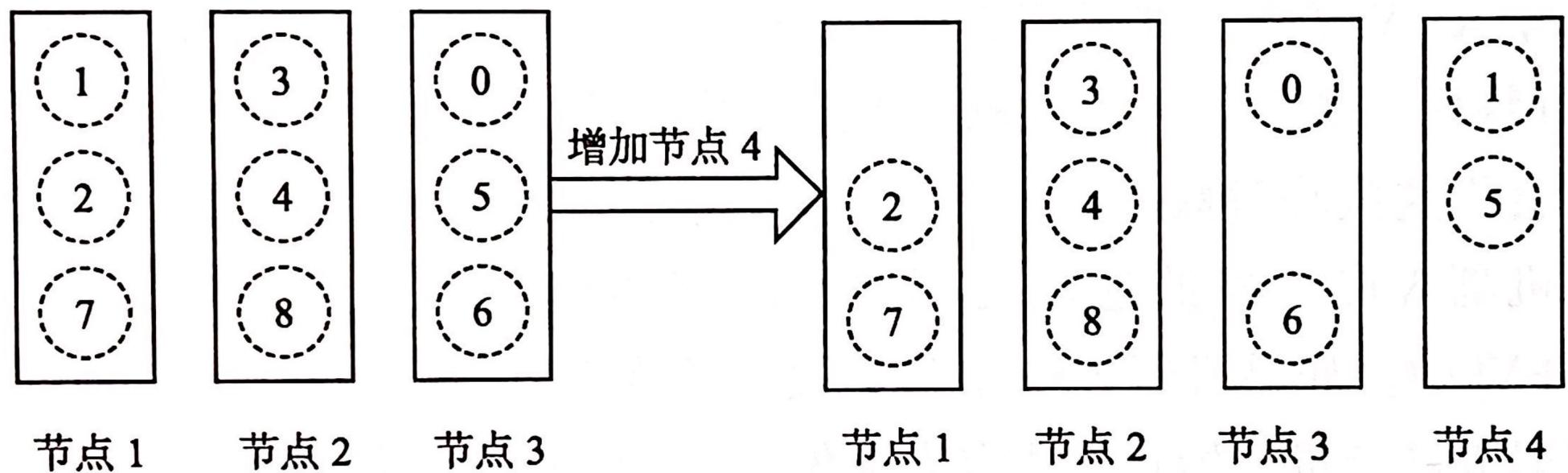


虚拟节点的好处

- 一个物理节点失效时，其承担的虚拟节点可以被分配给其他可用物理节点
- 一个物理节点上线时，可以从其他物理节点调整一定数量的虚拟节点负载到新加入的物理节点
- 虚拟节点的本质：分离负载映射和资源映射，实现存储负载细分



虚拟节点举例





虚拟节点的信息更新问题

- 如何记录服务器在环中的位置?
 - $O(1)$: 每台服务器记录前后邻居位置, 查找时间复杂度 $O(N)$
 - $O(N)$: 每台服务器维护所有主机信息, 查找时间为 $O(1)$ *
 - $O(\log N)$: 每台服务器维护 $\log N$ 主机信息(路由表), 查找时间为 $O(\log N)$



虚拟节点的信息更新问题

- 如何记录服务器在环中的位置
 - $O(N)$: 每台服务器维护所有主机信息, 查找时间为 $O(1)$, 追求性能!
- 信息更新如何达成一致?
- Gossip协议



Gossip协议

- 系统中节点加入或删除情况普遍
- 如何保证每个节点存储最新的环信息？
- 每个节点按间隔时间(1s)执行Gossip协议
 - 1) 从其他节点中任选一个与之交换信息
 - 2) A告诉B其管理的所有节点版本(Down/up)
 - 3) B告诉A哪些版本比较旧，将B的新版本数据发送给A
 - 4) A将B中比较旧的节点数据发送给B，同时按照B发来的节点数据更新本地数据
 - 5) B收到A发来的数据，更新本地数据



节点上下线

- 种子节点负责 **节点上线**信息更新
- 每个节点定期同种子节点交换信息
- 每个节点定期同所有节点交换信息
- 超过一定时间无响应的节点，被认为已经下线



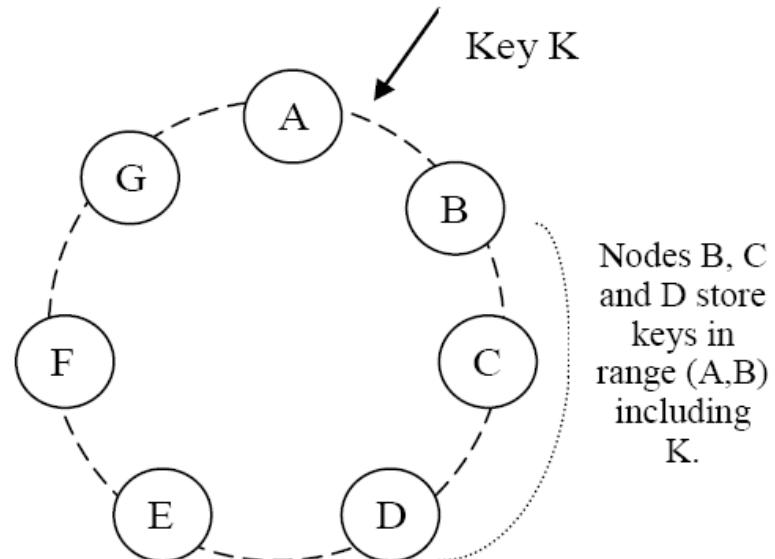


- 内容提要
 - Dynamo简介与架构
 - 数据分布
 - **一致性与复制**
 - 容错和负载分配
 - 读写流程与应用



数据副本

- 为应对容错需求，创建数据副本。
- 假设数据存储N份，所属节点是J，则数据存储在 $J, J+1, \dots, J+N-1$ 节点上
- Preference list：存储某个key数据副本的节点集合





读写可用性保证(NWR机制)

- 要求多副本“随时可写”（不能使用分布式锁）
- 随时写导致不一致的情况下如何保证读有效？
- 假设N个副本，NWR机制：
 - 最少R个成功读操作，则读成功
 - 最少W个成功写操作，则写成功

满足 $W+R > N$ ，就保证不超过一台机器故障时，可以读到至少一份可用数据



读写可用性保证(NWR机制)

- 按需调整NWR配置
 - 重视读效率: $W=N, R=1$ (同步复制)
 - 重视写性能: $W=1, R=N$
 - 读写平衡: $N=3, W=2, R=2$
 - 可损失一些更新: $W=1, R=1, N=3$
- 存在问题: 多个节点存储信息不一致(更新顺序)



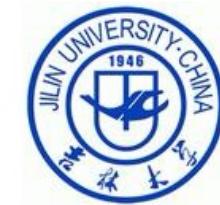
数据版本

- “随时可写”带来的弱一致性问题
 - 写操作put()可能在数据更新在所有节点完成之前返回（更新顺序/完成先后）
 - get()可能返回一个数据的多个版本
- 挑战：每一个数据对象有多个不同的修改记录版本
- 方案：**向量时钟**



向量时钟

- 如何解决无法使用分布式锁带来的多版本问题
- 记录各个版本
- 一个数据的每个版本都附带一个向量时钟
 - 向量时钟 (node, counter) 对
 - 节点, 版本号(计数器, 初始为0, 每次+1)

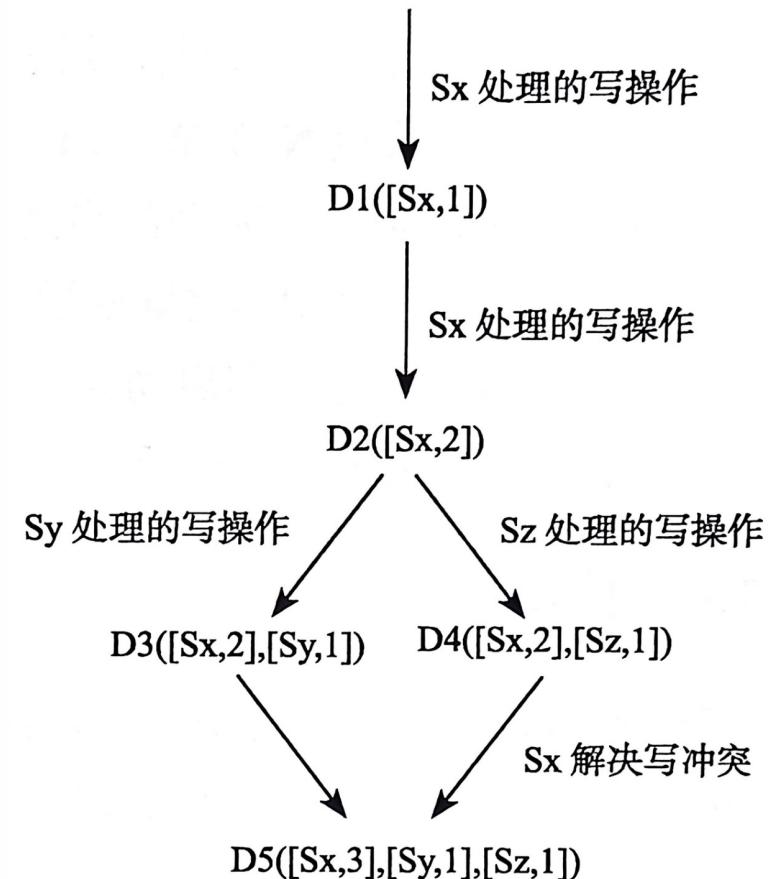


向量时钟

- 如果第一个分支节点上的向量时钟早于第二个分支的，则第一个分支可以被忽略

- S_x, S_y, S_z 三个副本

- D5: 读处理冲突后的状态





向量时钟

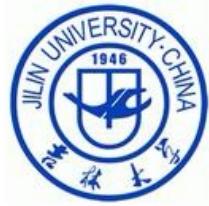
- 当系统出现故障的时候，preference list以外的机器也可能处理写操作
- 具有最新时间戳的数据状态会被存储
- 向量时钟可设置存储上限(如10个)，来节约空间



存在的问题

- 多个版本冲突的解决
 - 并发写：记录向量时钟，不处理冲突
 - 并发读：需要处理冲突
 - 冲突解决
 - Last write wins最后有效原则(依赖同步时钟)
 - 用户层解决
 - 只能保证最终统一(更新顺序可能不一致)





- 内容提要
 - Dynamo简介与架构
 - 数据分布
 - 一致性与复制
 - 容错和负载分配
 - 读写流程与应用



错误和异常

• Google某数据中心运行故障

发生频率	故障类型	影响范围
0.5	数据中心过热	5分钟之内大部分机器断电，一到两天恢复
1	配电装置（PDU）故障	大约500到1000台机器瞬间下线，6小时恢复
1	机架调整	大量告警，500~1000台机器断电，6小时恢复
1	网络重新布线	大约5%机器下线超过两天
20	机架故障	40到80台机器瞬间下线，1到6小时恢复
5	机架不稳定	40到80台机器发生50%丢包
12	路由器重启	DNS和对外虚IP服务失效约几分钟

发生频率	故障类型	影响范围
3	路由器故障	需要立即切换流量，持续约1小时
几十	DNS故障	持续约30秒
1000	单机故障	机器无法提供服务
几千	硬盘故障	硬盘数据丢失



错误和异常

- 临时性错误：短时间可以恢复，如网络延迟
- 永久性错误：持续时间长，如硬盘设备错误



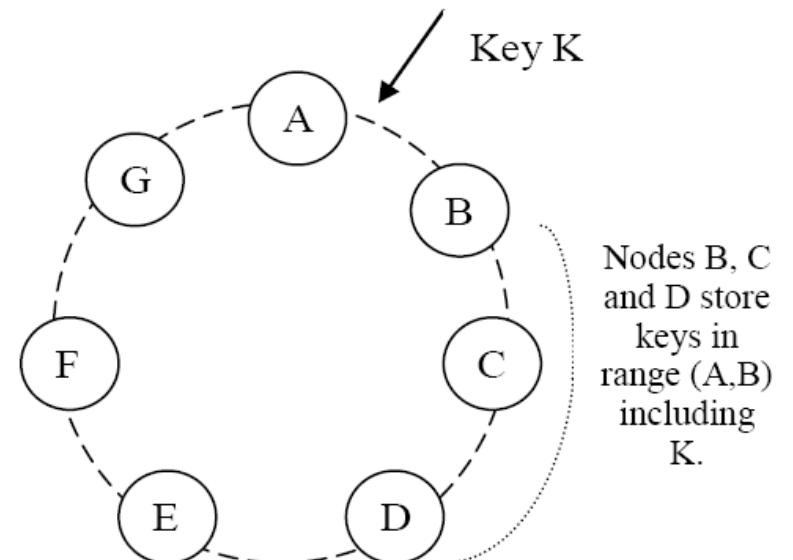
临时错误处理

- N个健康副本，才能保证读到最新的数据
 - 保证N个副本上读R个副本，写W个副本。
 - 合理设置R和W，以满足 $R+W > N$
 - 出错情况下不满足NWR的条件，怎么办？
- Sloppy Quorum 宽松的计数
 - 数据写入到临时节点
 - 保证写操作的可用性
- 数据回传技术
 - 节点恢复后，回传数据



宽松计数和数据回传

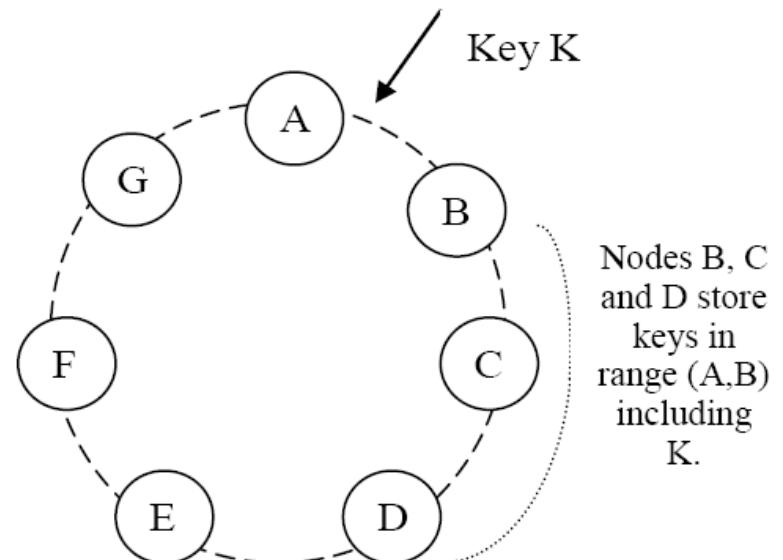
- 数据副本 $I, I+1, \dots, I+N-1$
- $I+j$ 出错
- 新的写入数据转移到 $I+N$
- 如果 $I+j$ 重启成功， $I+N$ 通过 Gossip 发现
- $I+N$ 回传数据到 $I+j$





永久错误

- 如果 $I+j$ 长时间得不到恢复
- $I+N$ 可能也会面临错误
- 可能造成临时接管的 $I+j$ 数据同时丢失
- 数据副本不一致
- $I+j$ 永久失效：
 - $I+N$ 长期节点接管
 - 从其他节点同步数据





节点之间的数据同步

- 如何解决数据副本不一致？**节点间数据同步**
- 数据同步面临的问题：
 - 如何查找出不一致的数据
 - 减小数据传输量(减小开销)

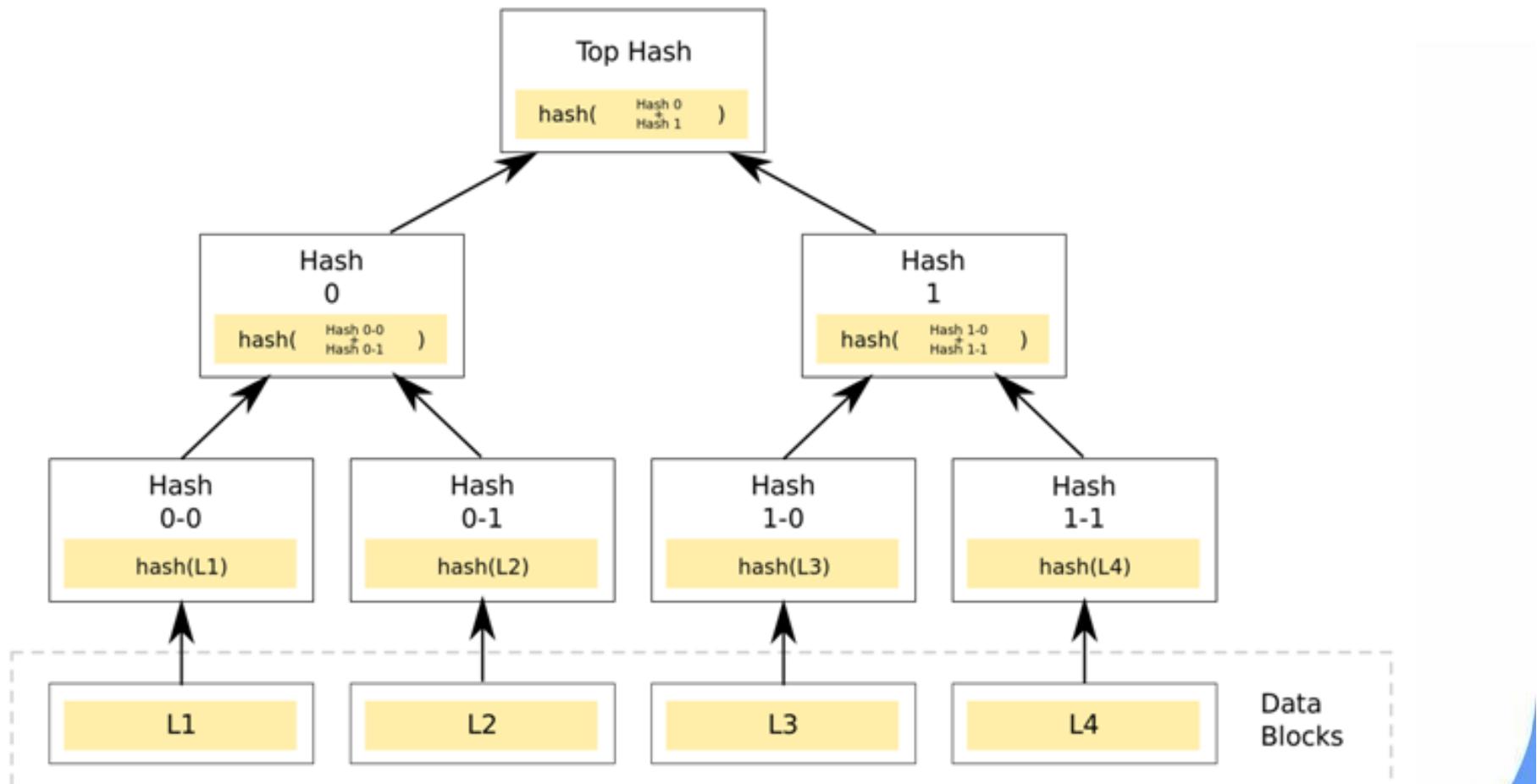


Merkle树同步

- 叶子节点对应数据Value值(值哈希)
- 非叶子节点对应多数据(其子节点组合值的哈希)
- 每个物理节点（服务器）（为每个虚拟节点）维护一棵Merkle树，负责虚拟节点的key值范围索引
- 机器之间传输Merkle树信息，检测Merkle树结构，找到不一致的数据，再同步数据。



Merkle树同步





Merkle树同步讨论

- 优点：
 - 节点之间传输数据量小(先检查索引树)
 - 检测效率高(检测到不一致, 按需同步文件)
 - 检测速度快(树遍历、按需检测)
- 缺点：
 - 与虚拟节点挂钩, Key重新分布需要重算Merkle树

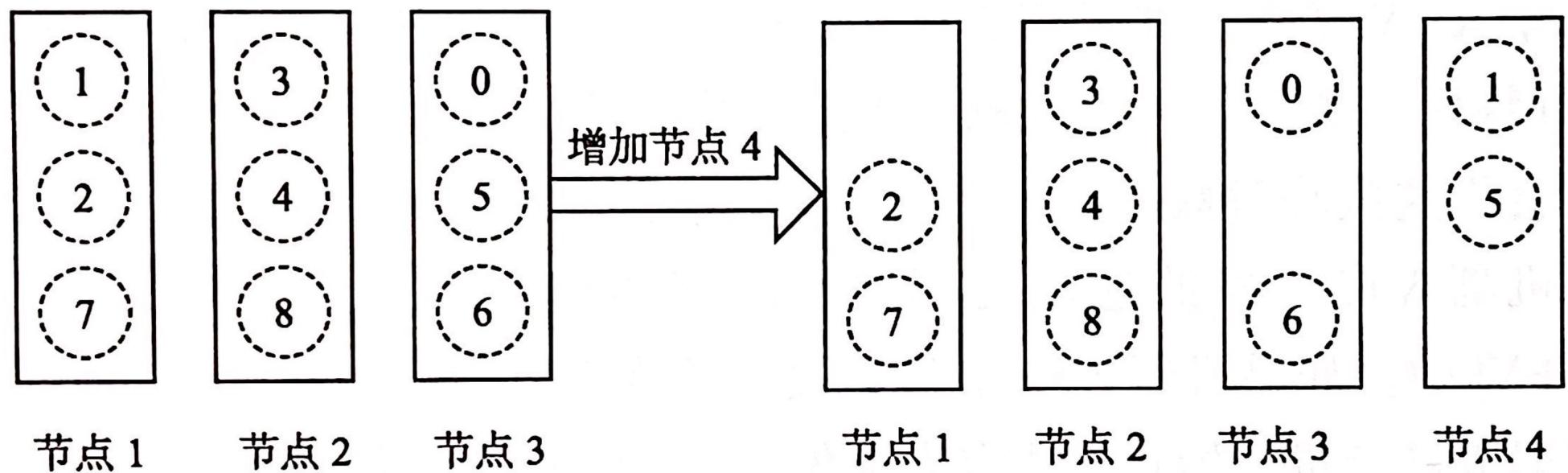


负载分配

- 数据倾斜：某一范围内值多于其他范围(不同虚拟节点负担不同)
- 如何给物理节点分配虚拟节点(token)是关键
 - 随机法、数据范围等分+随机



虚拟节点举例





随机负载分配

- 分配方法：
 - 根据配置性能给每个物理节点随机分配S个token(虚拟节点)
- 优点：
 - 只要S足够大(虚拟节点够小)，负载均衡效果较好
- 缺点：
 - 新节点加入可能导致token重划分，进而需更新Merkle树(数据范围临近有序)



数据范围等分+随机分配

- 分配方法：
 - 将数据哈希空间分为 $Q=M*S$ (M台物理机，每台机器S个虚拟节点)
 - 每台机器随机选择S个分割点作为Token
 - 每台机器维护每个数据范围的Merkle树
- 优点：
 - 先确定数据范围，再细分token，负载均衡较好
 - 数据上下线扫描部分数据进行同步
- 缺点：
 - 提前确定数据范围(哈希空间)不容易





- 内容提要
 - Dynamo简介与架构
 - 数据分布
 - 一致性与复制
 - 容错和负载分配
 - 读写流程与应用



选择(副本)服务节点的策略

- 由客户端决定服务节点(直接)
 - 客户端存储分区信息(元数据)
 - 节省中转代价，访问延迟低
- 协调者选取服务节点(代理)*
 - 依据每个节点的负载情况
 - 客户端与服务端隔离性好



读写可用性保证(NWR机制)

- 要求多副本“随时可写”（不能使用分布式锁）
- 随时写导致不一致的情况下如何保证读操作有效？
- 假设N个副本，NWR机制：
 - 最少R个成功读操作，则读成功
 - 最少W个成功写操作，则写成功

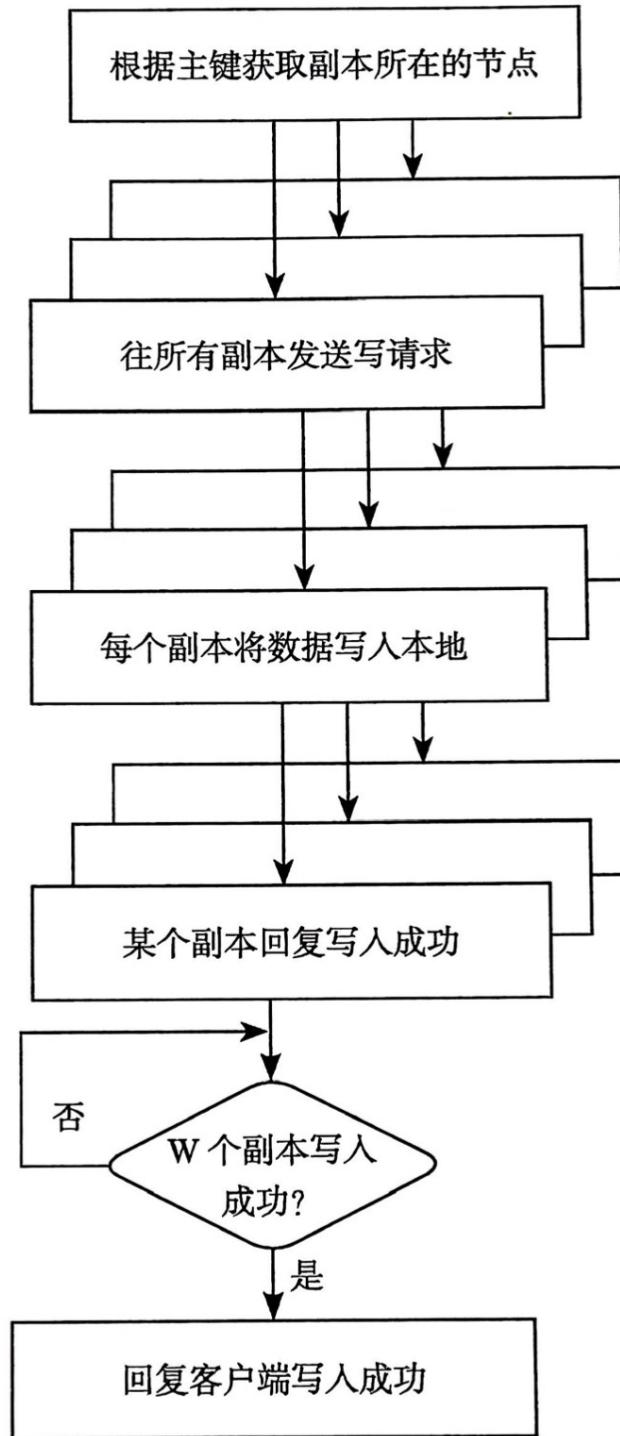
满足 $W+R > N$ ，就保证不超过一台机器故障时，可以读到至少一份可用数据



写流程

1. 根据一致性哈希计算副本位置
2. 其中一个副本作为协调者
3. 协调者并发地向所有副本发送请求
4. 每个副本接收数据，写入成功回复协调者
5. 协调者反复重试，直至 $W-1$ 个其他副本成功写数据
6. 协调者写成功后向客户端回复
7. 协调者会继续尝试将数据发送给其他副本

写数据流程



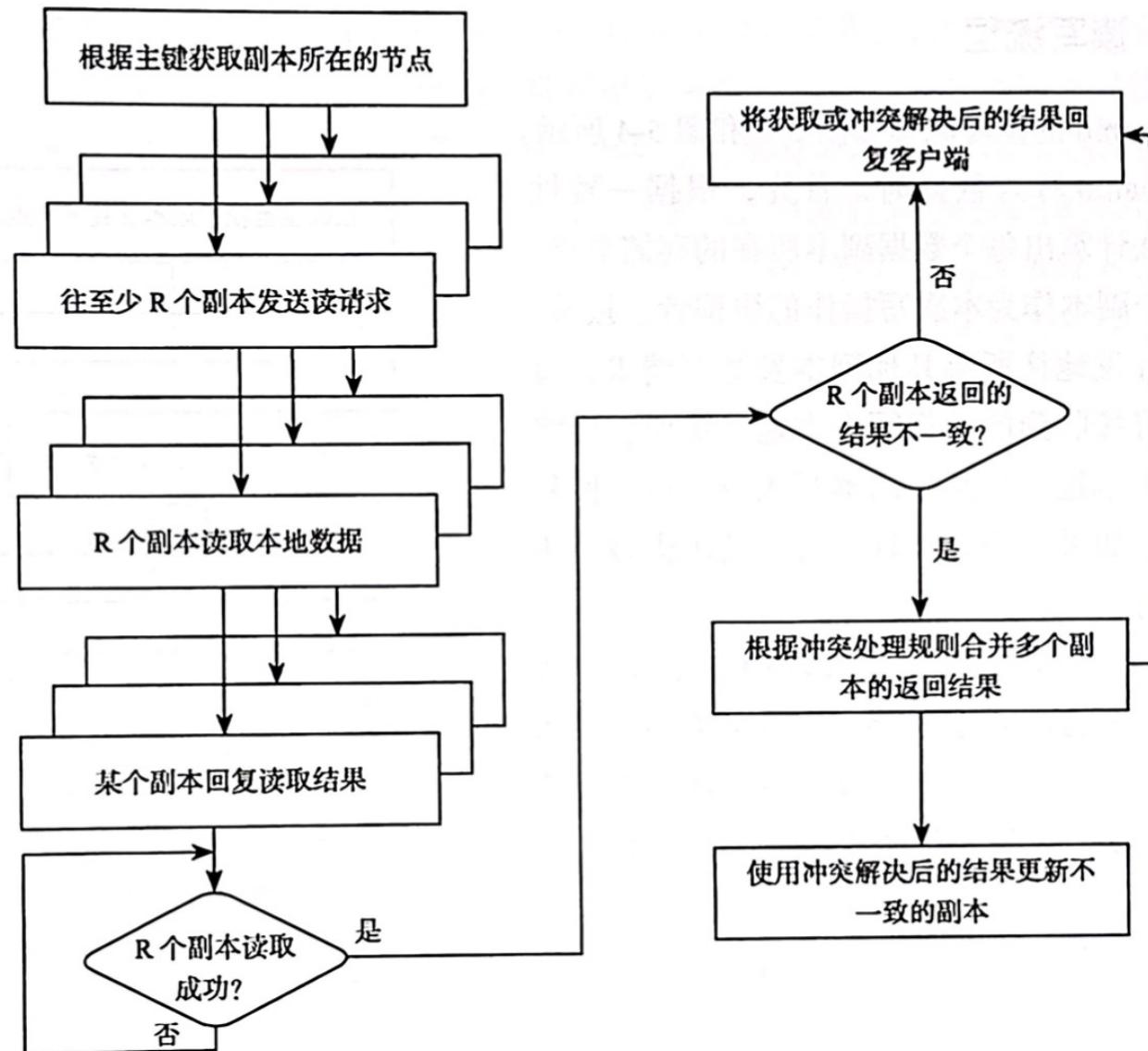


读流程

1. 根据一致性哈希计算副本位置
2. 其中一个副本作为协调者
3. 协调者根据负载策略选择 R 个副本
4. 并发地向其他副本发送请求
5. 每个副本读取本地数据，回复协调者结果
6. 协调者反复重试，直至 $R-1$ 个其他副本成功读数据
7. 协调者读数据
8. 解决冲突，将结果返回客户端(最新时间戳)
9. 修正不一致的副本(离线)



读数据流程





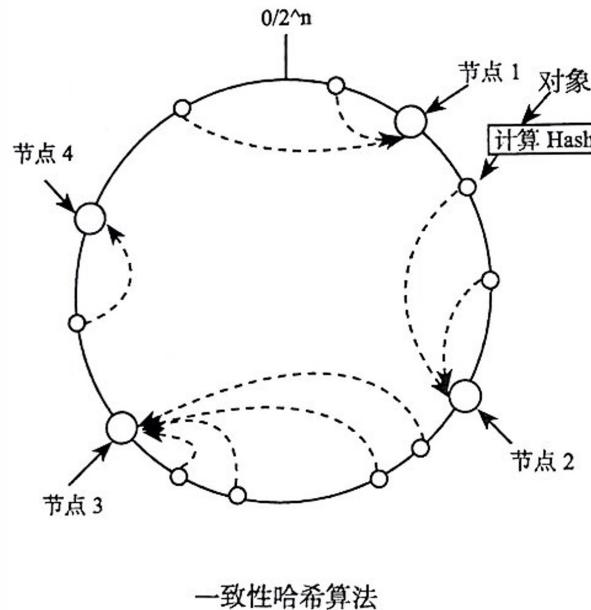
设计问题及解决方案

问 题	采取的技术
数据分布	改进的一致性哈希（虚拟节点）
复制协议	复制写协议（Replicated-write protocol, NWR 参数可调）
数据冲突处理	向量时钟
临时故障处理	数据回传机制（Hinted handoff）
永久故障后的恢复	Merkle 哈希树
成员资格及错误检测	基于 Gossip 的成员资格和错误检测协议



系统优点

- 访问性能
 - Key-value形式查询
 - 哈希性能
 - 访问延迟低
- 可扩展性
- 负载均衡
- 设计完美主义

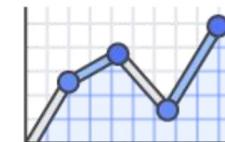


一致性哈希算法

Fast, Consistent Performance



Highly Scalable



Fully Managed



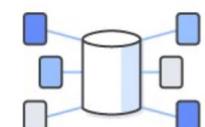
Event Driven Programming



Fine-grained Access Control



Flexible





系统缺点

- 数据记录容量小(UTF-8, 400K)
- 查询功能较单一, 不支持关系(Join)
- **一致性较差**
 - 写顺序(乱序)
 - 并发操作结果难预期
- 哈希维护代价



- 概论 (4)
- 分布式系统基础 (8)
 - 分布式系统发展
 - 高性能计算和数据中心（架构与管理）
- 分布式存储技术 (9)
- 分布式存储系统 (27)
 - 网络文件系统 (NFS)
 - 分布式文件系统 (HDFS, GFS)
 - 分布式键值对存储 (Dynamo)
 - 分布式表格系统 (BigTable)
 - 分布式对象存储 (S3)



- Thanks!

Hongliang Li

lihongliang@jlu.edu.cn

College of Computer Science and Technology
Jilin University