

如何理解高可用数据复制原理

原创 疾风先生 小坤探游架构笔记 2025年06月08日 20:01 广东

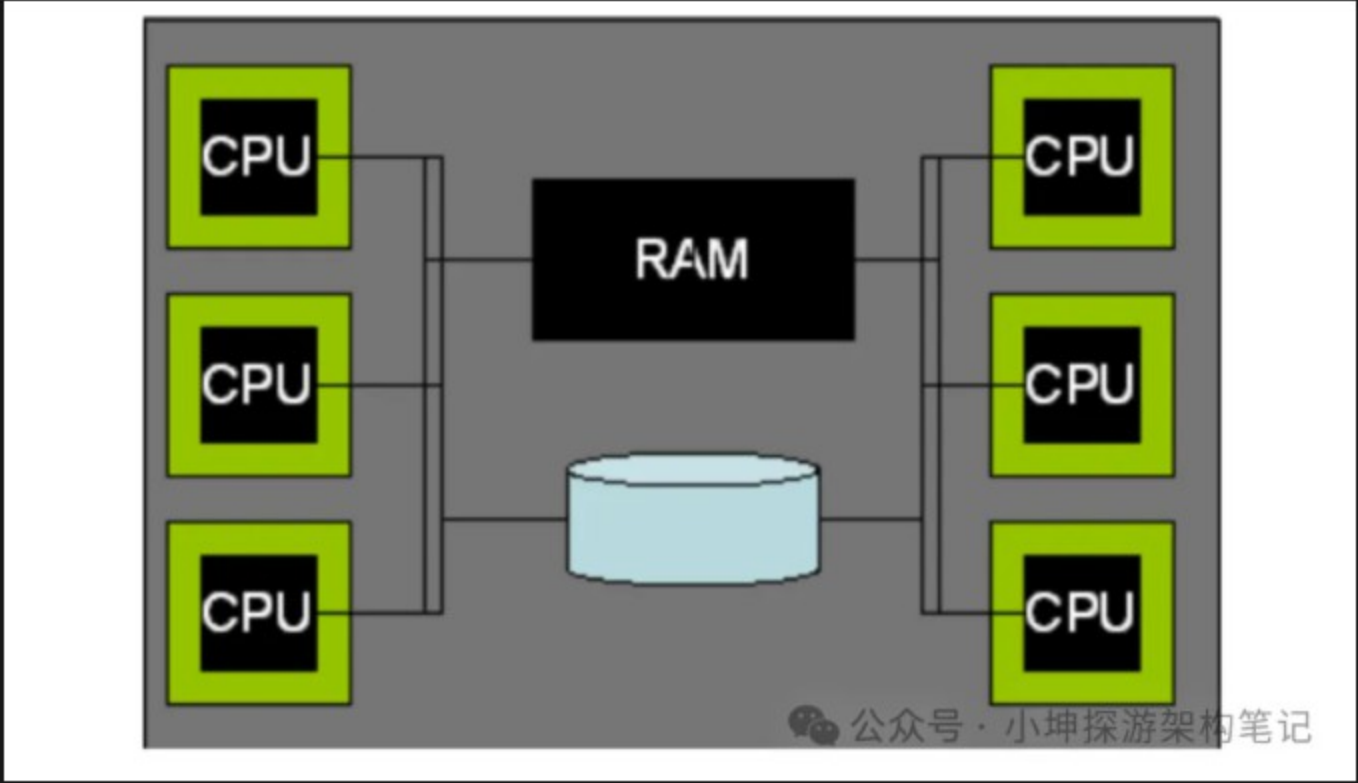
点击上方小坤探游架构笔记可以订阅哦

一谈到复制技术,相信我们大部分都有一个认知,那就是实现数据存储的高可用,其实进行数据复制也不仅仅是实现高可用,同时也是边缘加速以及提升读性能的一个技术手段,今天我就来讲述下复制技术原理,也是作为学习过程中的一个笔记记录.

共享存储与无共享存储架构

同样我们关注一项技术还是需要先了解过往的背景,现在我们可以先思考下为什么数据库数据需要分布在多台机器上呢?

首先我们先来看单机的共享内存架构,所有处理器都能以大致相同的性能访问相同的RAM和磁盘,这种架构相当标准,如下所示:

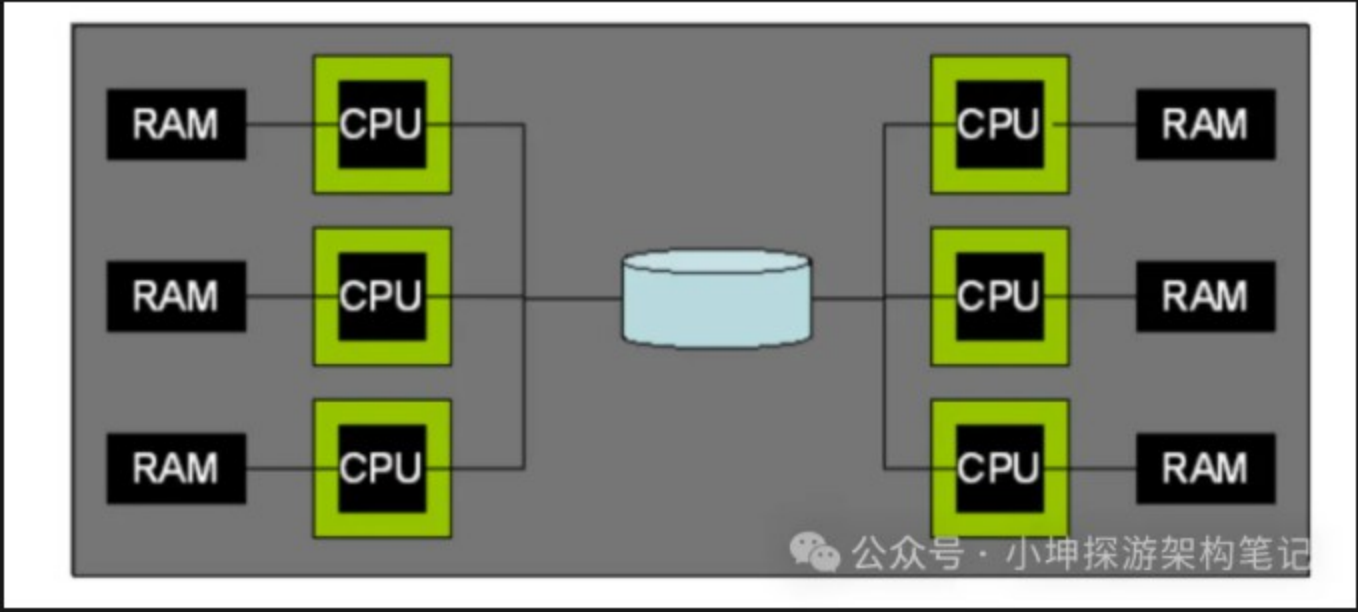


在上述架构中如果我们要扩展到更高的负载,最简单的方式就是购买更高的机器配置,将多个CPU、内存芯片以及磁盘块在一个操作系统下组合起来,快速的互连组件允许任意 CPU 访问内存或磁盘的任何部分。在这种共享内存架构中,所有组件都可视为一台单一机器.

共享内存方法的问题在于成本增长速度超过线性比例:一台拥有两倍 CPU、两倍内存和两倍磁盘容量的机器,其成本通常远高于两台普通机器的成本之和。而且由于瓶颈的存在,规模两倍的机器未必能处理两倍的负载。即成本与扩展的局限性.

其次是有限的故障容忍能力,即使具备热插拔组件也存在单点问题,换言之如果节点发生宕机就全部不可用. 即有限的故障容忍能力,不具备高可用.

上述我们讲到是共享内存架构,那如果是共享磁盘的方法呢? 对于这类方法我们会有一个很熟悉的应用场景,那就是数仓,可以通过彼此相互独立的CPU以及内存将数据通过互联网的方式连接共享磁盘,即:

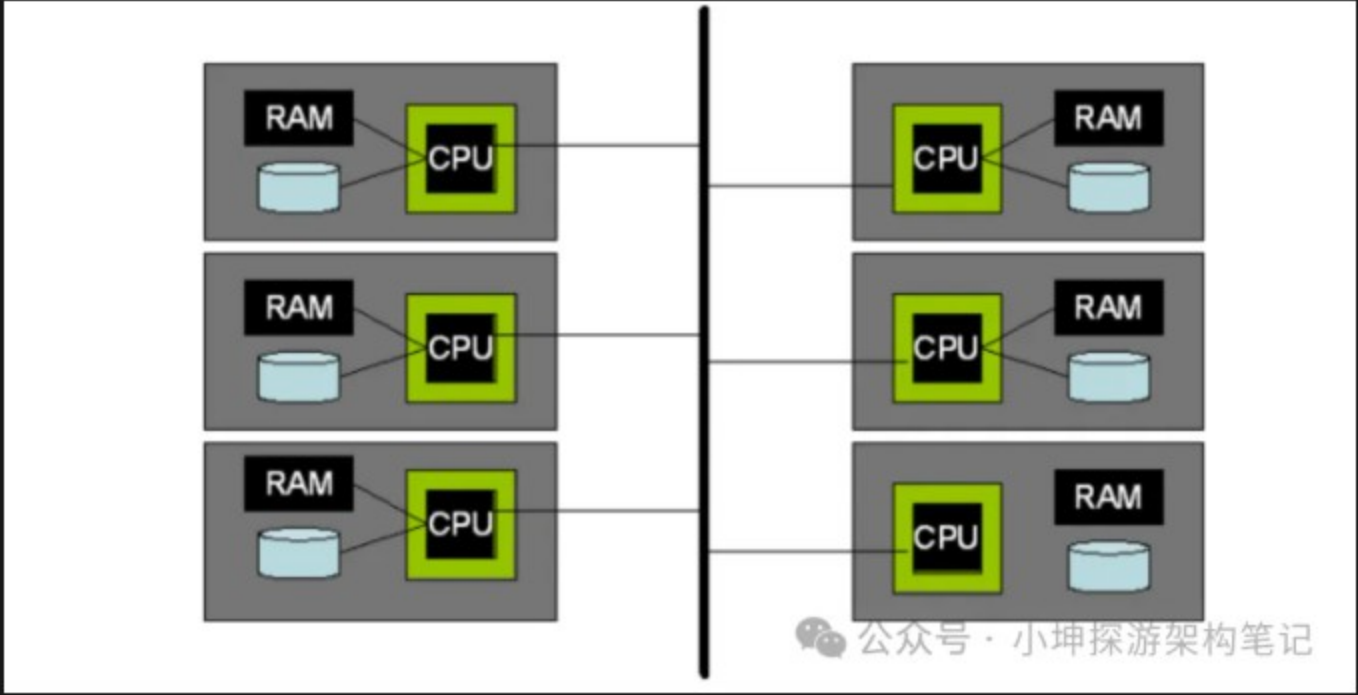


看到这里我们可能会想共享磁盘的方法相比共享内存的方式在扩展性以及故障容忍性更好,其一是我们通过加相同的CPU以及RAM机器来提升计算性能,其二是每台机器彼此独立,相比共享内存架构,如果机器发生故障,其他机器仍然能够访问磁盘数据.

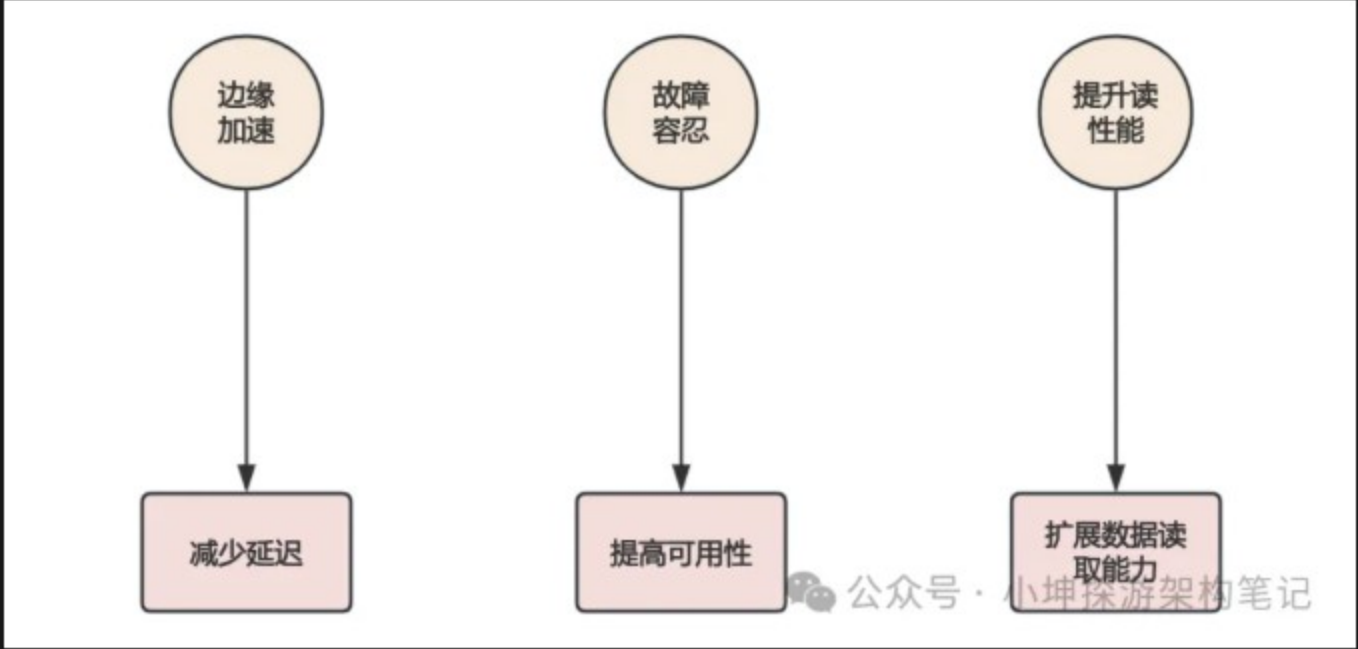
但共享磁盘架构也存在单点问题,即如果数据在到达存储子系统之前发生硬件或者软件故障导致数据被损坏,即使采用RAID或者其他冗余技术,被损坏的数据也将会被冗余存储,那么这个时候所有机器节点访问的数据的时候都受到影响.同样是有限故障容忍,存在单点问题.

除此之外,如果数据需要在多台机器进行交互协调,由于数据是共享的,那么必然存在锁争用问题,竞争和锁定开销会限制共享磁盘方法的可扩展性。

基于扩展性、高可用以及成本问题,我们需要将数据分布在多台机器上,每个机器都彼此有自己独立的CPU、内存以及磁盘数据,彼此之间互不共享,彼此通信通过网络互联组成,即:



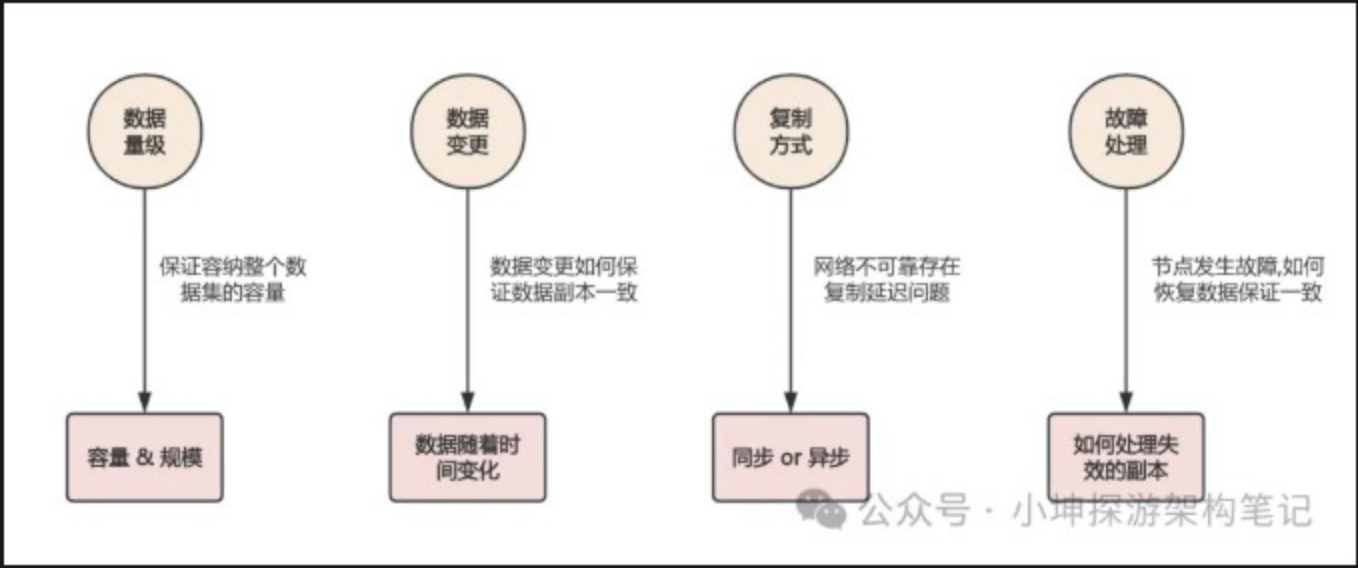
无共享架构能够在扩展性、高可用以及成本等相比共享存储层面有一定的优势,这个时候我们的数据由单台变成了多台分布,此时我们再来回答数据为啥需要分布在多台机器上,主要原因有:



这也是我们进行复制的原因,何为复制? 复制就是通过网络连接的多台机器保留相同的数据副本,通过数据复制一是可以让我们系统在离用户更近的地理位置上加速访问减少延迟; 二是即使部分节点发生故障也能保证系统稳定运行; 三是通过数据副本冗余可以将数据读取分散到其他机器,提升数据查询的扩展能力.

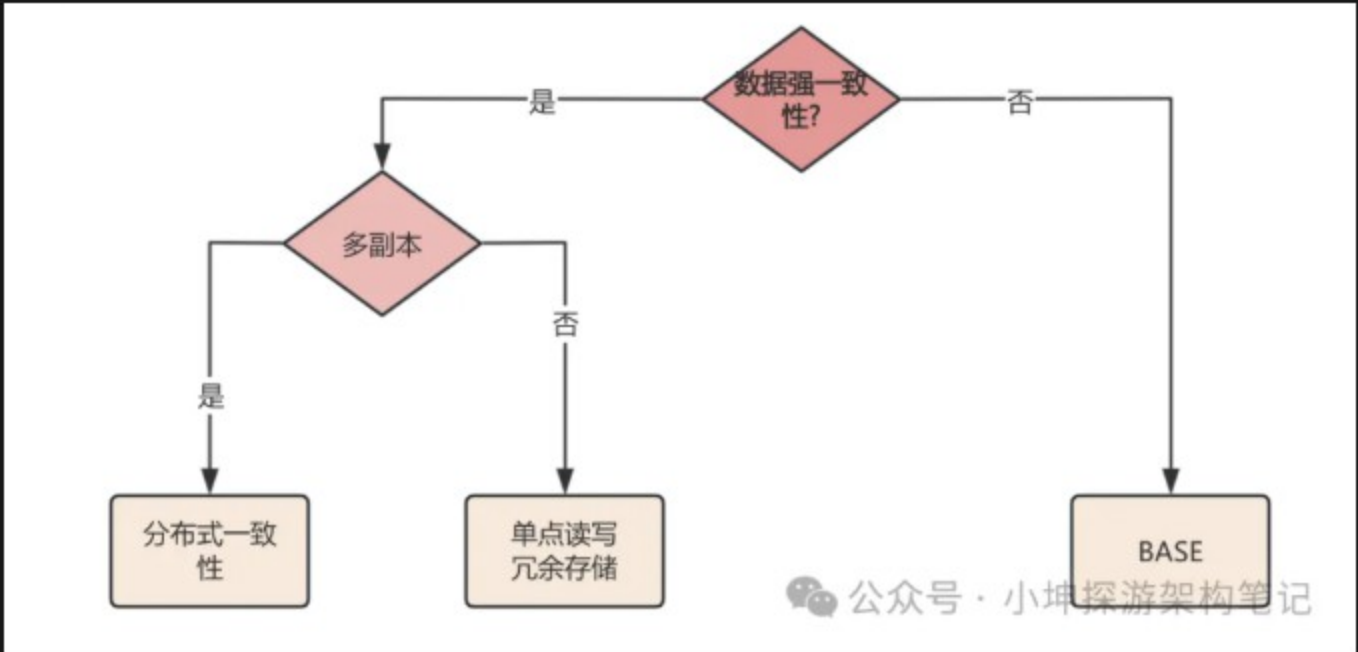
数据复制面临的问题

在一个数据无共享架构中,如果我们要进行数据复制,那么我们会面临以下几个问题:



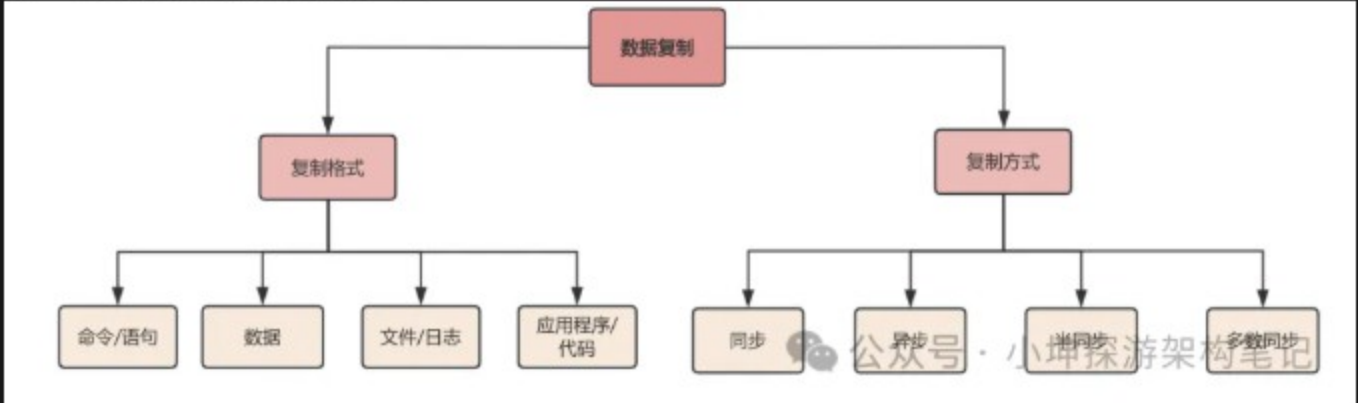
- 对于数据量级的扩展,我们一般采用的手段就是分片技术,采用分片主要从两个层次考虑, 其一是数据容量,比如用数据容量有100gb,而我们单台节点不足以存储100gb,那么就扩展一台机器来分担; 其次是规模,比如1000w份数据, 每份数据存储可能1byte, 可能也就是10gb, 其实单节点是足以存储, 这个时候扩展一台机器更多是想将数据分散, 分担流量提升性能的目的.因此我们的数据复制是建立在每台机器能够容纳整个数据集的副本基础上.
- 数据复制最大的难点就是要捕获数据变更并复制到对应的数据副本节点上,一般有三种复制方式, 即single-leader(单领导者)、multi-leader(多领导者/数据分片/多数据中心)以及leaderless(无领导者)复制.那么如何保证数据副本的一致性?
- 数据复制是通过网络进行通信的,也就是变成发送数据复制节点与接收数据节点之间的通信,在计算机层面节点/程序之间通信要么是同步要么异步,因此数据复制应当采用哪种复制方式呢? 同时由于网络不可靠导致复制延迟如何保证数据副本一致性?
- 如果数据副本失效, 即不可用, 该采取什么样的策略与措施呢? 数据如何追赶到其他数据副本保持一致呢?

看到上述的问题都有一个共性,那就是数据一致性问题,我觉得我们都不陌生了,既然是数据一致性问题,这个时候我们可以采用之前的一个框架来辅助我们进行决策:



数据复制原理

在存储高可用架构中,数据复制其实就是实现数据冗余的落地方式,而我们要进行数据复制就需要有对应的复制格式以及复制方式.即:



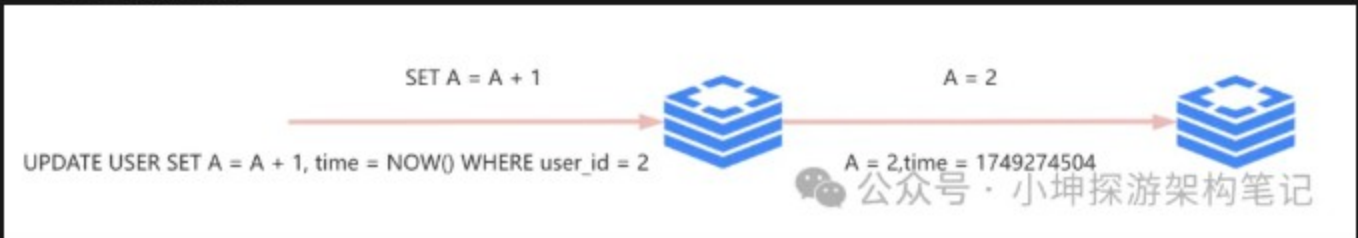
数据复制格式

- 基于命令/语句的复制



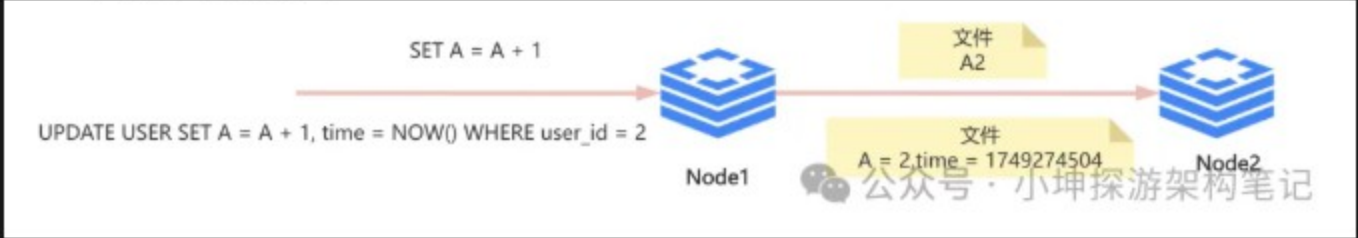
在增量复制的场景下,也许我们可以采用上述的复制格式,因为实现简单,复制数据量小,但是这种方式存在数据安全问题.为什么呢?比如像上述的SQL,使用函数NOW()来获取时间和日期将会和Node1节点产生不一致;其二是执行命令不确定是否存在前置数据依赖,即无法保证语句执行的顺序性,那就有可能产生数据结果不一致.这种的不一致是安全性问题,需要我们手动修复解决的,那么如何避免呢?那就是使用确定性的值替换对应的不确定性命令/函数操作.这个时候我们就有了基于数据复制的格式.

- 基于数据复制



同样适用于增量复制场景,实现简单,能够保证数据安全性,但存在复制流量很大.为什么流量会很大?比如上述SQL语句,原本我是复制一条SQL语句即可,现在我是直接复制数据值,那么一条SQL改动影响了多少行那么我就得复制多少行过去.但是基于数据复制有时候考虑的一些特殊情况,比如迁移上云可能需要数据脱敏处理,一般我们更多采用基于日志/文件的格式进行复制.

- 基于日志的复制格式

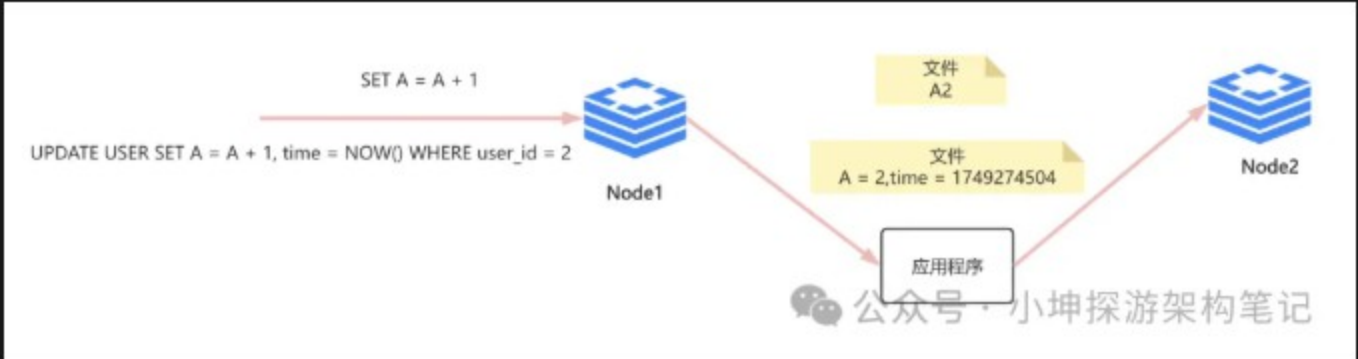


基于文件的复制格式在实现上比较复杂,同样能够保证数据一致性,同时复制的时候数据是随着时间变化着的,复制流量也会很大.

基于日志的复制方式一般有两种,一是存储引擎的WAL方式,不论是LSM还是B树结构,都是追加字节序列,能够包含对数据库的所有写入操作,因此可以同步到另一个副本上进行数据重放构建;然而它的不足就是与存储引擎强相关,依赖于存储引擎执行恢复,因为WAL日志包含具体哪些磁盘块对应的哪些字节被更改的信息,使得复制与存储引擎紧密耦合.试想下如果Node1节点的存储引擎版本升级,Node2并没有升级的话,中间复制协议的版本就可能存在差异.

另外一种基于日志复制的方式是基于逻辑日志复制,如果是mysql的话那么基于逻辑日志是一个二进制日志,也可以理解为基于行复制,主要是解决与存储引擎解耦,这种更具备向后兼容性.即使是不同的存储引擎版本也不影响到复制协议,甚至是不同存储引擎也能够实现日志复制,比如搜索引擎需要依赖mysql同步数据,那么就可以基于逻辑日志复制到ES中写入倒排索引提供搜索召回查询.

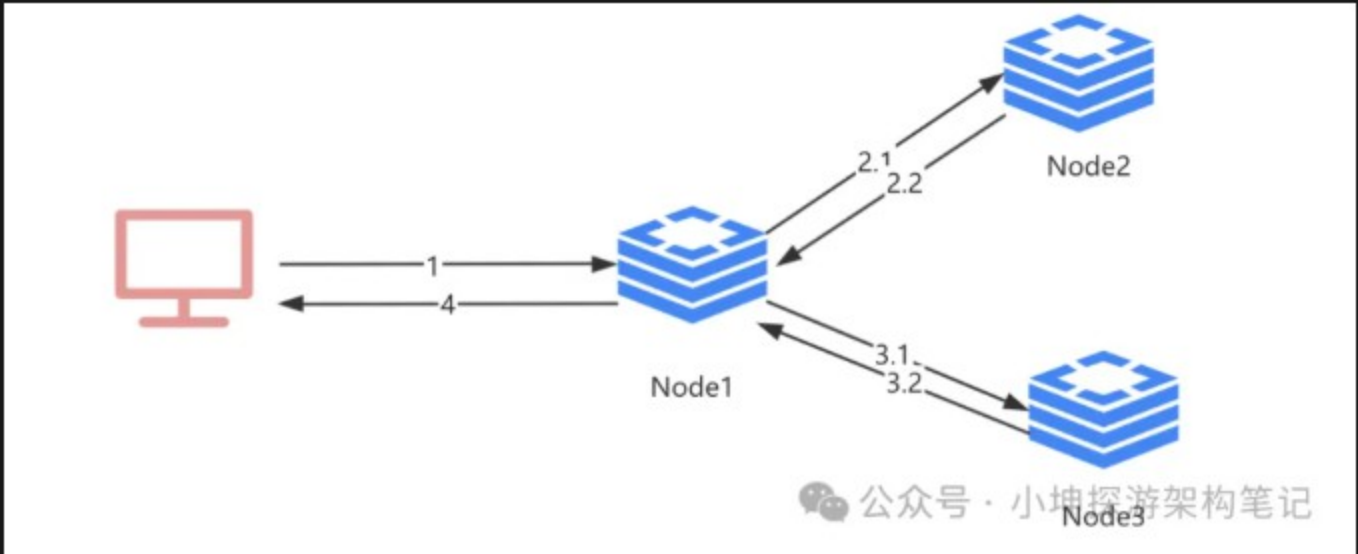
• 基于应用程序/代码复制



一般采用这种复制方式可能是要做异构数据存储,需要汇总不同数据来源进行加工处理重新存储到新的存储介质以满足对应的需求.当然在数据库层面也存在基于应用程序的复制,比如触发器和存储过程.

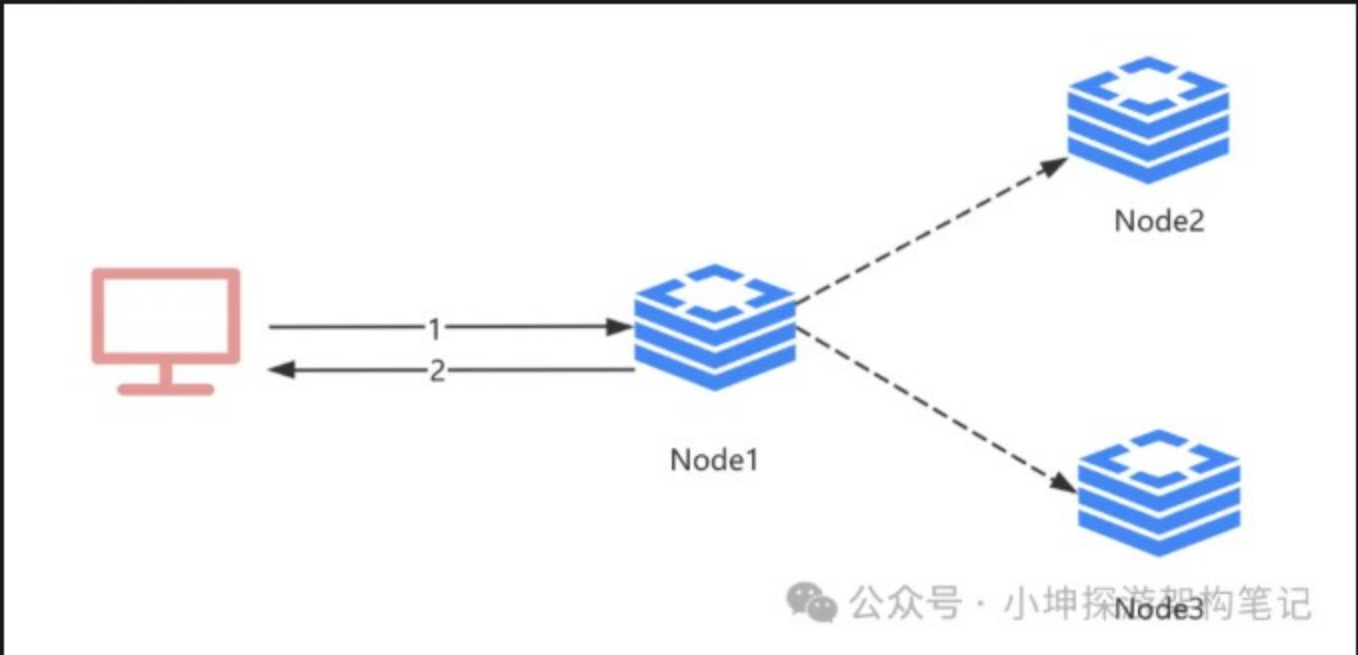
数据复制方式

• 同步复制



在上述同步复制架构中,我们能够实现最强一致性,即读己之所写,也就是客户端写入一个数据值,再次读取就能够读取到最新值.但是由于Node1节点需要等待Node2以及Node3节点的数据同步,如果当Node2以及Node3节点发生不可用的时候,Node1将无法提供客户端写入操作,因此故障容忍度比较低.即我们所说的可用性低.

• 异步复制

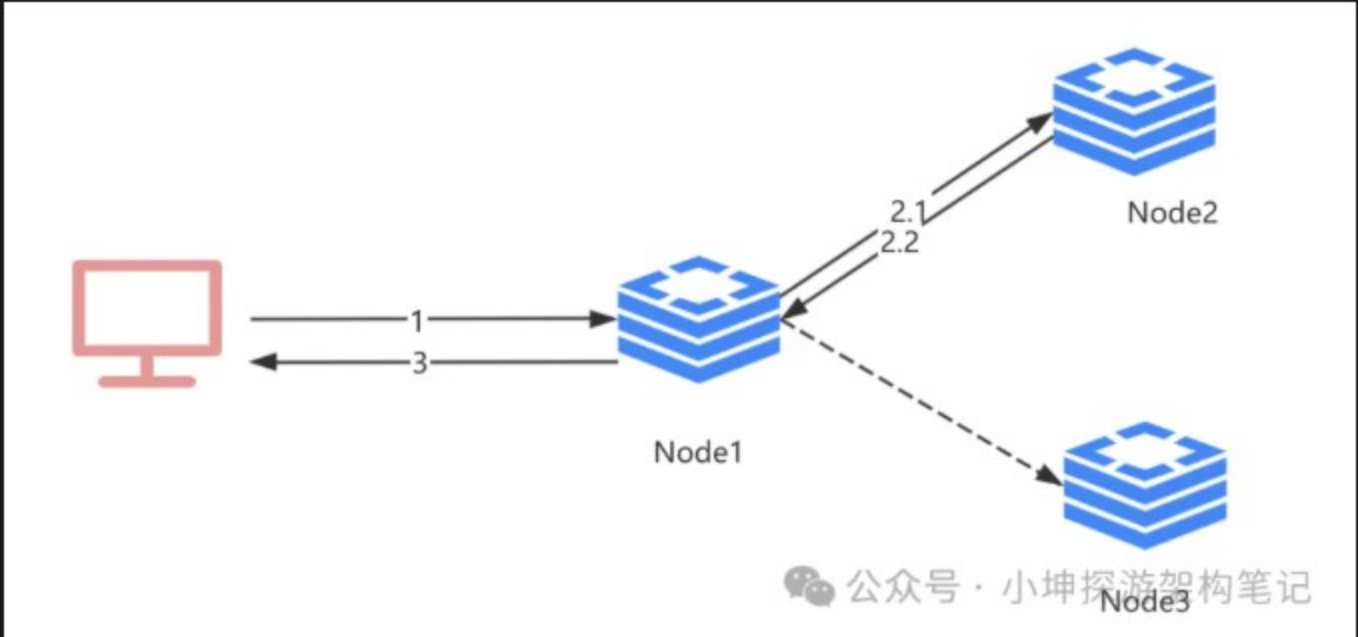


相比同步复制,我们的Node1接受客户端写入数据并持久化到本地,同时向Node2以及Node3发起异步数据复制方式,这个时候Node1不需要等待Node2以及Node3的回复,这个时候写入性能高,即使Node2或者Node3节点发生不可用也能够继续对外提供服务,因此故障容忍度相比同步复制高.

但并不是没有代价,客户端再次读取数据有可能是读取到旧值,其次由于异步复制,Node1节点是不知道其他节点是否与自己存储的数据是一致的,如果Node1节点不可用需要从Node2以及Node3中进行选举,如果我们选举Node2作为顶替Node1节点,但是Node2节点此时并未完全同步到Node1节点已存储的数据,那么就会出现我们所谓数据丢失现象.数据丢失与我们设计的初衷是相悖的,我们想要最终能够追赶上Node1节点数据,那么这个时候就会采取同步与异步复制之间的折衷方案,即部分采用同步复制,部分采用异步复制.

• 半同步复制

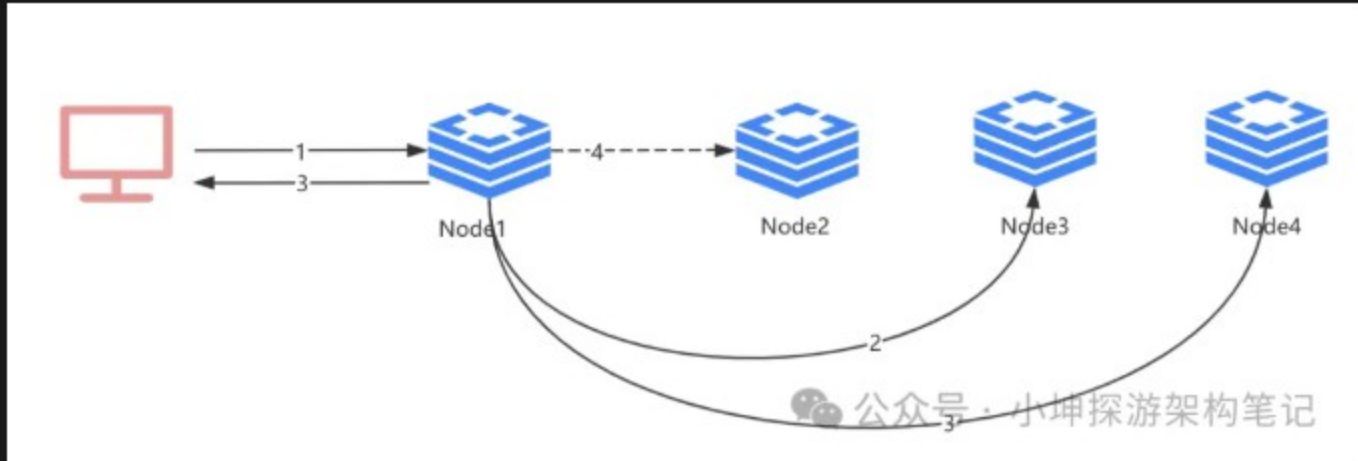
既然我们想让数据副本能够保证与Node1节点一致,那么我们就在同步与异步复制方式之间进行取舍,即Node1节点同步复制到Node2节点,而Node3节点则是异步方式进行复制,即如下:



那么这个时候如果Node1节点发生不可用,我们直接用Node2节点顶上,这个时候就能够保证我们的数据不丢失.但是我们把读写请求都落到了单个节点上,无法充分利用其他数据副本节点资源,如果我既想让客户端写入就读取到最新数据,又想充分利用资源节点呢?

- 多数复制

这个时候我们会采用大多数复制方式来实现,如下:




这个时候我们就可以实现数据强一致性,同时可用性以及故障容忍度也较高,还能充分利用冗余节点资源,但是它没有缺点吗? 肯定是有代价, 一是写入性能低, 因为需要等待集群半数以上节点返回才响应给客户端; 二是实现复杂,尤其是实现共识算法一致性,需要在数据同步过程保证其安全性属性,而且还会面临不可靠时钟以及网络问题,Node1节点需要就相同的数据复制提案保证原子性,实现成本比较高.

总结

关于数据复制原理,其一是复制格式,对于复制数据的格式我们要考虑到数据安全性问题,如果损坏了数据的原有安全属性,我们要实现冗余数据副本的目标就不功自破;其次是数据复制方式,需要基于一致性模型根据系统实际情况进行Trade-Off.

你好,我是疾风先生, 主要从事互联网搜广推行业, 技术栈为java/go/python, 记录并分享个人对技术的理解与思考, 欢迎关注我的公众号, 致力于做一个有深度,有广度,有故事的工程师,欢迎成长的路上有你陪伴,关注后回复greek可添加私人微信,欢迎技术互动和交流,谢谢!



小坤探游架构笔记

10年后端技术架构设计 | AI工程化基础建设 & 存储架构设计 & 性能优化 | 前网易、斗鱼、L...
52篇原创内容

公众号



疾风先生

喜欢作者

分布式架构 · 目录

< 上一篇
如何理解高可用架构设计原理

下一篇 >
如何理解分布式领导者复制算法模型

个人观点，仅供参考