

Dengshenyu

Code and Coffee

HOME / ARCHIVE / CATEGORY / ABOUT /

分布式存储系统基础

APR 9, 2017

最近读了杨传辉的《大规模分布式存储系统：原理解析与架构实践》，这本书写的很好，涉及的知识点枚不胜数。本篇对于其中的分布式存储系统基础知识做些整理，以飨诸君。

目录

- [基本概念](#)
 - [异常](#)
 - [服务器宕机](#)
 - [网络异常](#)
 - [磁盘故障](#)
 - [超时](#)
 - [一致性](#)
 - [衡量指标](#)
 - [性能](#)
 - [可用性](#)
 - [一致性](#)
 - [可扩展性](#)
- [数据分布](#)
 - [哈希分布](#)
 - [顺序分布](#)
 - [负载均衡](#)
- [复制](#)
 - [复制的概述](#)
 - [一致性与可用性](#)
- [容错](#)
 - [故障检测](#)

- [故障恢复](#)
- [可扩展性](#)
 - [同构系统](#)
 - [异构系统](#)
- [分布式协议](#)
 - [两阶段提交协议](#)
 - [Paxos协议](#)
 - [Paxos与2PC](#)
- [跨机房部署](#)

分布式存储系统首先要面对的问题就是数据分片，即将数据均匀地分布到多个存储节点。另外，为了保证可靠性和可用性，需要将数据复制多个副本，这就带来了多个副本的数据一致性问题。

大规模系统的重要目标是节省成本，因而只能采用性价比较高的PC服务器。这些服务器性能很好，但是故障率很高，要求系统能够在软件层面实现自动容错。当存储节点出现故障时，系统能够检测出来，并将原有的数据和服务迁移到集群中其他正常工作的节点。

基本概念

异常

在分布式存储系统中，往往将一台服务器或者服务器上运行的一个进程称为一个节点，节点与节点之间通过网络互联。然而，服务节点是不可靠的，网络也是不可靠的，它们之间通信可能会出现各种异常。

服务器宕机

引发服务器宕机的原因有很多，例如内存错误、服务器停电等等。服务器宕机可能随时发生，当发生宕机时，节点无法正常工作。服务器重启后，节点将失去所有的内存信息。

因此，设计存储系统时需要考虑如何通过读取持久化介质（如机械硬盘、固态硬盘）中的数据来恢复内存信息，从而恢复到宕机前的某个一致的状态。

网络异常

引发网络异常的原因可能是消息丢失、消息乱序或者网络包数据错误。有一种特殊的网络异常称为“网络分区”，即集群的所有节点被划分为多个区域，每个区域内部可以正常通信，但是区域之间无法通信。例如，某分布式系统部署在

两个数据中心，由于网络调整，导致数据中心之间无法通信，但是，数据中心内部可以正常通信。

磁盘故障

磁盘故障可以分为两种情况：磁盘损坏和磁盘数据错误。磁盘损坏时，将会丢失存储上面的数据，因而，分布式存储系统需要考虑将数据存储到多台服务器，即使其中一台服务器磁盘出现故障，也能从其他服务器上恢复数据。对于磁盘数据错误，往往可以采用校验和机制来解决，这样的机制既可以在操作系统层面实现，又可以在上层的分布式存储系统层面实现。

超时

由于网络异常的存在，分布式系统中请求结果存在“三态”的概念，即“成功”、“失败”、“超时”（未知状态）。“成功”和“失败”指客户端请求明确收到服务器的返回值；而“超时”指客户端发出了一个请求但是没有收到回复，但客户端不能简单地认为服务端处理失败，因为有可能服务端已经处理成功了，但在返回结果时出现了网络异常或宕机。

对于超时（未知）状态，有两种处理思路：1）不断读取之前操作的状态来验证rpc操作是否成功；2）将操作设计为“幂等”的，也就是说，操作执行一次与执行多次的结果相同。

一致性

由于异常的存在，分布式存储系统设计时往往将数据冗余存储多份，每一份称为一个副本（replica）。这样，当一个节点出现故障时，可以从其他副本上读取数据。可以认为，副本是分布式存储系统容错技术的唯一手段。

由于多个副本的存在，如何保证副本之间的一致性是整个分布式系统的理论核心。

可以从两个角度理解一致性：第一个角度是客户端，即客户端读写操作是否符合某种特性；第二个角度是存储系统，即存储系统的多个副本是否一致，更新的顺序是否相同等等。

首先定义如下场景，这个场景包含三个组成部分：

- 存储系统：存储系统可以理解为一个黑盒子，它为我们提供了可用性和持久性的保证。
- 客户端A：客户端A主要实现从存储系统write和read操作。

- 客户端B和客户端C：客户端B和客户端C是独立于A，并且B和C也相互独立，它们同时也实现对存储系统的write和read操作。

从客户端的角度看，一致性包含如下三种情况：

- 强一致性：假如A先写入了一个值到存储系统，存储系统保证后续A，B，C的读取操作都将返回最新值。
- 弱一致性：假如A先写入了一个值到存储系统，存储系统不能保证后续A，B，C的读取操作是否能够读取到最新值。
- 最终一致性：最终一致性是弱一致性的一种特例。假如A首先写入一个值到存储系统，存储系统保证如果后续没有写操作更新同样的值，A，B，C的读取操作“最终”都会读取到A写入的值。“最终”一致性有一个“不一致窗口”的概念，它特指从A写入值，到后续A，B，C读取到最新值得这段时间。

最终一致性的描述比较粗略，其他常见的变体如下：

- 读写（Read-your-writes）一致性：如果客户端A写入了最新值，那么A的后续操作都会读取到最新值。但是其他用户（比如B或者C）可能要过一会才能看到。
- 会话（Session）一致性：要求客户端和存储系统交互的整个会话期间保证读写一致性。如果原有会话因为某种原因失败而创建了新的会话，原有会话和新会话之间的操作不保证读写一致性。
- 单调读（Monotonic read）一致性：如果客户端A已经读取了对象的某个值，那么后续操作不会读取到更早的值。
- 单调写（Monotonic write）一致性：客户端A的写操作按顺序完成，这就意味着，对于同一个客户端的操作，存储系统的多个副本需要按照与客户单相同的顺序完成。

从存储系统的角度看，一致性主要包含如下几个方面：

- 副本一致性：存储系统的多个副本之间的数据是否一致，不一致的时间窗口等；
- 更新顺序一致性：存储系统的多个副本之间是否按照相同的顺序执行更新操作。

衡量指标

评价分布式存储系统有一些常用的指标，下面分别介绍。

性能

常见的性能指标有：系统的吞吐能力（throughput）以及系统的响应时间（latency）。其中，系统的吞吐能力指系统在某一时间段可以处理的请求总数，通常用每秒处理的读操作数（QPS，Query Per Second）或者写操作数（TPS，Transaction Per Second）来衡量。系统的响应时间，指从某个请求发出到接收到返回结果消耗的时间，通常用平均延时或者99.9%以上请求的最大延时来衡量。

这两个指标往往是矛盾的，追求高吞吐的系统，往往很难做到低延迟；追求低延迟的系统，吞吐量也会受到限制。因此，设计系统时需要权衡这两个指标。

可用性

系统的可能性（availability）是指系统在面对各种异常时可以提供正常服务的能力。系统的可用性可以用系统停服务的时间与正常服务的时间的比例来衡量，例如某系统的可用性为4个9（99.99%），相当于系统一年停服务时间不能超过 $365 * 24 * 60 / 10000 = 52.56$ 分钟。系统可用性往往体现了系统的整体代码质量以及容错能力。

一致性

前面已经说明了系统的一致性。一般来说，越是强的一致性模型，用户使用起来越简单。

可扩展性

系统的可扩展性（scalability）指分布式存储系统通过扩展集群服务器规模来提高系统存储容量、计算量和性能的能力。随着业务的发展，对底层存储系统的性能需求不断增加，比较好的方式就是通过自动增加服务器提高系统的能力。理想的分布式存储系统实现“线性可扩展”，也就是说，随着集群规模的增加，系统整体性能与服务器数量呈线性关系。

数据分布

分布式系统区别于传统单机系统在于能够将数据分布到多个节点，并在多个节点之间实现负载均衡。数据分布的方式主要有两种，一种是哈希分布，如一致性哈希，代表系统为Amazon的Dynamo系统；另一种方法是顺序分布，即数据按照主键整体有序，代表系统为Google的Bigtable系统。Bigtable将一张大表根据主键切分为有序的范围，每个有序的范围是一个子表。

哈希分布

哈希取模的方法很常见，其方法是根据数据的某一种特征计算哈希值，并将哈希值与集群中的服务器建立映射关系，从而将不同哈希值得数据分布到不同的服务器上。

如果哈希函数的散列特性很好，哈希方式可以将数据比较均匀地分布到集群中去。然而，找出一个散列特性很好的哈希函数是很难的。举个例子，如果按照主键散列，那么同一个用户id下的数据可能被分散到多台服务器，这会使得一次操作同一个用户id下的多条记录变得困难；如果按照用户id散列，容易出现“数据倾斜”问题，即某些大用户的数据量很大，无论集群的规模有多大，这些用户始终由一台服务器处理。

处理大用户问题一般有两种方式，一种方式是手动拆分，即线下标记系统中的大用户，并根据这些大用户的数据量将其拆分到多台服务器上。这相当于在哈希分布的基础上针对这些大用户特殊处理；另一种方式是自动拆分，即数据分布算法能够动态调整，自动将大用户的数据拆分到多台服务器上。

传统的哈希分布算法还有一个问题：当服务器上线或者下线时，N值发生变化，数据映射完全被打乱，几乎所有的数据都需要重新分布，这将带来大量的数据迁移。

一种思路是不再简单地将哈希值和服务器个数之间做除法取模映射，而是将哈希值与服务器的对应关系作为元数据，交给专门的元数据服务器来管理。访问数据时，首先计算哈希值，再查询元数据服务器，获得该哈希值对应的服务器。这样，集群扩容时，可以将部分哈希值分配给新加入的机器并迁移对应的数据。

另一种思路就是采用一致性哈希算法。算法思想如下：给系统中每个节点分配一个随机token，这些token构成一个哈希环。执行数据存放操作时，先计算Key（主键）的哈希值，然后存放到顺时针方向第一个大于或者等于该哈希值得token所在的节点。一致性哈希的优点在于节点加入/删除时只影响到在哈希环中相邻的节点，而对其他节点没影响。

顺序分布

哈希散列破坏了数据的有序性，只支持随机读操作，不能够支持顺序扫描。顺序分布在分布式表格系统中比较常见，一般的做法是将大表顺序划分为连续的范围，每个范围称为一个子表，总控服务器负责将这些子表按照一定的策略分配到存储节点上。

例如，用户表（User表）的主键范围为1~7000，在分布式存储系统中划分为多个子表，分别对应数据范围1~1000，1001~2000，...，6001~7000。某些系统只有根表（Root表）一级索引，在Root表中维护用户表的位置信息，即每个用户子表存放在哪个存储节点上。为了支持更大的集群规模，Bigtable这样的系统将索引分为两级：根表以及元数据表（Meta表），由Meta表维护User表的位置信息，而Root表维护Meta表的位置信息。

顺序分布与B+树数据结构比较类似，每个子表相当于叶子节点，随着数据的插入和删除，某些子表可能变得很大，某些变得很小，数据分布不均匀，系统设计时需要考虑子表的分裂与合并。

负载均衡

分布式存储系统的每个集群中一般都有一个总控节点，其他节点为工作节点，由总控节点根据全局负载信息进行整体调度。系统运行过程中需要不断地执行迁移任务，将数据从负载较高的工作节点迁移到负载较低的工作节点。

工作节点通过心跳包（Heartbeat，定时发送）将节点负载相关的信息，如CPU，内存，磁盘，网络等资源使用率，读写次数及读写数据量发送给总控节点。总控节点计算出工作节点的负载以及需要迁移的数据，生成迁移任务放入迁移队列中等待执行。

分布式存储系统中往往会存储数据的多个副本，其中一个副本为主副本，其他副本为备副本，由主副本对外提供服务。迁移备副本不会对服务造成影响，迁移主副本也可以首先将数据的读写服务切换到其他备副本。整个迁移过程可以做到无缝，对用户完全透明。

复制

复制的概述

为了保证分布式存储系统的高可靠和高可用，数据在系统中一般存储多个副本。当某个副本所在的存储节点出现故障时，分布式存储系统能够将服务切换到其他副本，从而实现自动容错。分布式存储系统通过将复制协议将数据同步到多个存储节点，并保证多个副本的数据一致性。

同一份数据的多个副本往往有一个副本为主副本（Primary），其他副本为备副本（Backup），由主副本将数据复制到备副本。复制协议分为两种，强同步复制以及异步复制。二者的区别在于用户的写请求是否需要同步到备副本才可

以返回成功。假如备副本不止一个，复制协议还会要求写请求至少需要同步到几个备副本。

主副本将写请求复制到其他备副本常见的做法是同步操作日志（Commit Log），主副本首先将操作日志同步到备副本，备副本回放操作日志，完成后通知主副本。等这些操作完成后再通知客户端写成功。这种协议称为强同步协议。强同步协议提供了强一致性，但是，如果备副本出现问题将阻塞写操作，系统可用性较差。

操作日志的原理很简单：为了利用磁盘的顺序读写特性，将客户端的写操作先顺序写入磁盘中，然后应用到内存中。当服务器宕机重启时，只需要回放操作日志就可以恢复内存状态。为了提高系统的并发能力，系统会积攒一定的操作日志再批量写入到磁盘中，这种技术称为成组提交。

如果每次服务器出现故障都需要回放所有的操作日志，效率是无法忍受的，检查点（checkpoint）正是为了解决这个问题。系统定期将内存状态以检查点文件的形式dump到磁盘中，并记录检查点时刻对应的操作日志回放点。检查点文件创建成功后，回放点之前的日志可以被垃圾回收，以后如果服务器出现故障，只需要回放检查点之后的操作日志。

强同步复制和异步复制都是基于主副本的复制协议（Primary-based protocol）。这种方法要求在任何时刻只能有一个副本为主副本，由它来确定写操作之间的顺序。如果主副本出现故障，需要选举一个备副本称为新的主副本，这步操作称为选举，经典的选举协议为Paxos协议。

一致性和可用性是矛盾的，强同步复制协议可以保证主备副本之间的一致性，但是备副本出现故障时，也可能阻塞存储系统的正常写服务，系统的整体可用性受到影响；异步复制的可用性相对较好，但是一致性得不到保障，主副本出现故障还有数据丢失的可能。

除了基于主副本的复制协议，分布式存储系统还可能使用基于写多个存储节点的复制协议（Replicated-write protocol）。比如Dynamo系统中的NWR复制协议，其中N为副本数量，W为写操作的副本数，R为读操作的副本数。NWR协议中不再区分主和备，客户端根据一定的策略往其中的W个副本写入数据，读其中的R个副本。只要 $W+R>N$ ，可以保证读到的副本中至少有一个包含了最新的更新。

一致性与可用性

来自Berkerly的Eric Brewer教授提出了一个著名的CAP理论：一致性（Consistency），可用性（Availability）以及分区可容忍性（Toleration of network Partition）三者不能同时满足。

- 一致性：读操作总能读取到之前完成的写操作结果。
- 可用性：读写操作始终能够成功。
- 分区可容忍性：系统能够容忍由于机器故障、网络故障、机房停电等异常情况所造成的网络分区。

在分布式系统中，分区可容忍性总是要满足的，因此一致性和可用性不能同时满足。存储系统设计时需要在一致性和可用性之间权衡，在某些场景下，不允许丢失数据，在另外一些场景下，极小的概率丢失部分数据是允许的，可用性更加重要。例如，Oracle数据库的DataGuard复制组件包含三种模式：

- 最大保护模式（Maximum Protection）：即强同步复制模式，写操作要求主库先将操作日志（数据库的redo/undo日志）同步到至少一个备库才可以返回客户端成功。这种模式保证即使主库出现无法恢复的故障，比如硬盘损坏，也不会丢失数据。
- 最大性能模式（Maximum Performance）：即异步复制模式，写操作只需要在主库上执行成功就可以返回客户端成功，主库上的后台线程会将重做日志通过异步的方式复制到备库。这种方式保证了性能和可用性，但是可能丢失数据。
- 最大可用性模式（Maximum Availability）：上述两种模式的折衷。正常情况下相当于最大保护模式，如果主备之间的网络出现故障，切换为最大性能模式。

容错

随着集群规模越来越大，故障发生的概率也越来越大，大规模集群每天都有故障发生。容错是分布式存储系统涉及的重要目标，只有实现了自动化容错，才能减少人工运维成本，实现分布式存储的规模效应。

首先，分布式存储系统需要能够检测到机器故障，例如通过租约（Lease）协议实现。接着，需要能够将服务复制或者迁移到集群中的其他正常服务的存储节点。

故障检测

容错处理的第一步是故障检测，心跳是一种很自然地想法。假设总控机A需要确认工作机B是否发生故障，那么总控机A每隔一段时间，比如1秒，向工作机B发送一个心跳包。如果一切正常，机器B将响应机器A的心跳包；否则，机器A

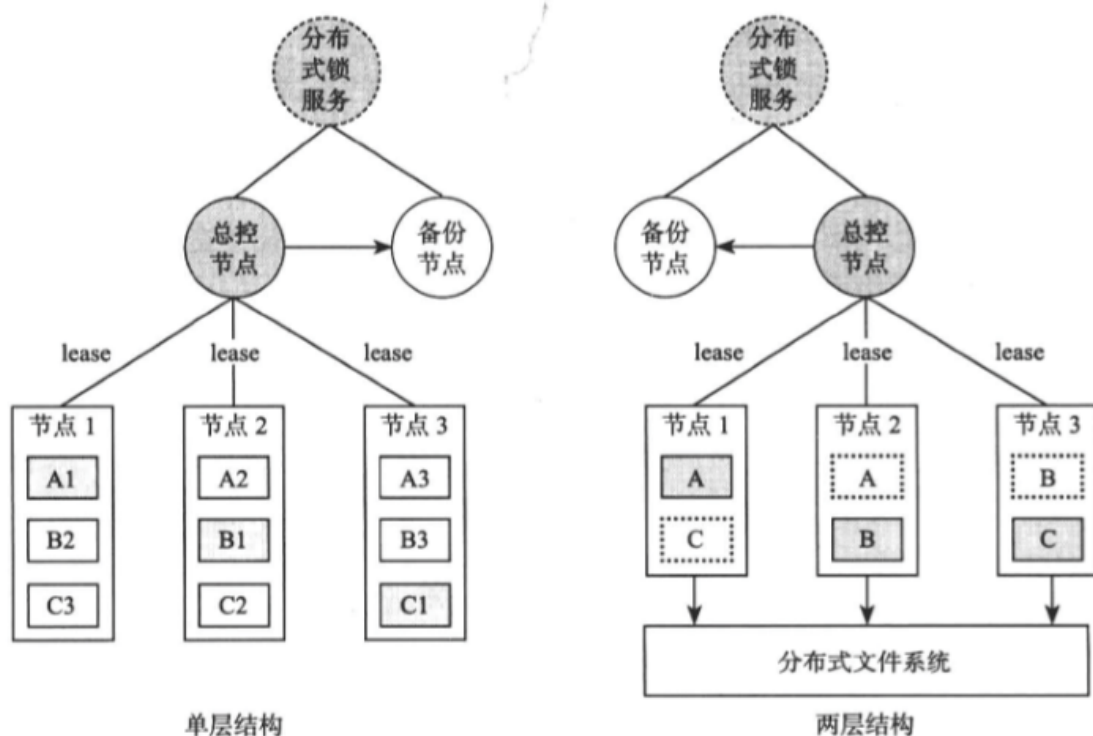
重试了一定次数后认为机器B发生了故障。但是，机器A收不到机器B的心跳并不能确保机器B发生故障并停止了服务，比如可能是A和B之间出现网络问题导致A收不到回复。由于在机器A“认为”机器B发生故障后，往往需要将它上面的服务迁移到集群中的其他服务器，为了保证强一致性，需要确保机器B不再提供服务。

这里的问题是机器A和机器B之间需要对“机器B是否应该被认为发生故障且停止服务”达成一致。我们可以通过租约（Lease）机制进行故障检测，机器A可以通过机器B发放租约，机器B持有的租约在有效期内才允许提供服务，否则主动停止服务。机器B的租约快要到期的时候向机器A重新申请租约。正常情况下，机器B通过不断申请租约来延长有效期，当机器B出现故障或者与机器A之间的网络发生故障时，机器B的租约将过期，从而机器A能够确保机器B不再提供服务，机器B的服务可以被安全地迁移到其他服务器。

故障恢复

当总控机检测到工作机发生故障时，需要将服务迁移到其他工作节点。常见的分布式存储系统分为两种结构：单层结构和双层结构。大部分系统为单层结构，在系统中对每个数据分票维护多个副本；只有类Bigtable系统为双层结构，将存储和服务分为两层，存储层对每个数据分片维护多个副本，服务层只有一个副本提供服务。单层结构和双层结构的故障恢复机制有所不同。

单层结构和双层结构如下图所示：



单层结构的分布式存储系统维护了多个副本，例如副本个数为3，主备副本之间通过操作日志同步。如上图所示，某单层结构的分布式存储系统有3个数据分片A、B、C，每个数据分片存储了三个副本。其中，A1，B1，C1为主副本，分别存储在节点1，节点2以及节点3.假设节点1发生故障，总控节点选择一个最新的副本（比如A2或者A3）来替换A1成为新的主副本并提供写服务。

两层结构的分布式存储系统会将所有的数据持久化写入底层的分布式文件系统，每个数据分片同一时刻只有一个提供服务的节点。如上图所示，某双层结构的分布式存储系统有3个数据分片，A、B和C。它们分别被节点1，节点2和节点3所服务。当节点1发生故障时，总控节点将选择个工作节点，比如节点2，加载A的服务。由于A的所有数据都存储在共享的分布式文件系统中，节点2只需要从底层的分布式文件系统读取A的数据并加载到内存中。

可扩展性

同构系统

同构系统将存储节点分为若干组，组内的节点服务完全相同的数据，其中有一个节点为主节点，其他节点为备节点。由于同一个组内的节点服务相同的数据，这样的系统称为同构系统。如下图所示。

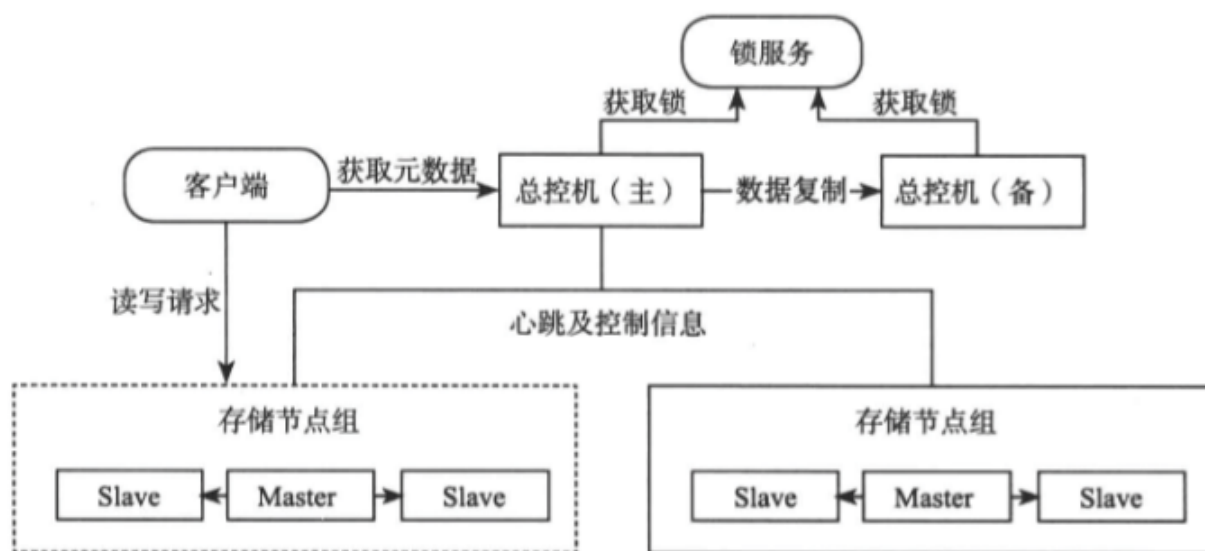


图 3-8 同构系统

同构系统的问题在于增加副本需要迁移的数据量太大，假设每个存储节点服务的数据量为1TB，内部传输带宽限制为20MB/s，那么增加副本拷贝数据需要的时间为 $1\text{TB}/20\text{MB}=50000\text{s}$ ，大约十几个小时，由于拷贝数据的过程中存储节点再次发生故障的概率很高，所以这样的架构很难做到自动化，不适合大规模分布式存储系统。

异构系统

大规模分布式存储系统要求具有线性可扩展性，即随时加入或者删除一个或者多个存储节点，系统的处理能力与存储节点的个数成线性关系。为了实现线性可扩展性，存储系统的存储节点之间是异构的。

异构系统将数据分为很多大小相近的分片，每个分片的多个副本可以分布到集群的任何一个存储节点。如果某个节点发生故障，原有的服务将由整个集群而不是某几个固定的存储节点来恢复。

如下图所示，系统中有五个分片（A，B，C，D，E），每个分片包含三个副本，如分片A的三个副本分别为A1，A2以及A3。假如节点1发生永久性故障，那么可以从剩余的节点中任意挑选健康的节点来增加A，B以及E的副本。由于整个集群都参与到节点1的故障恢复过程，故障恢复时间很短，而且集群规模越大，优势越明显。

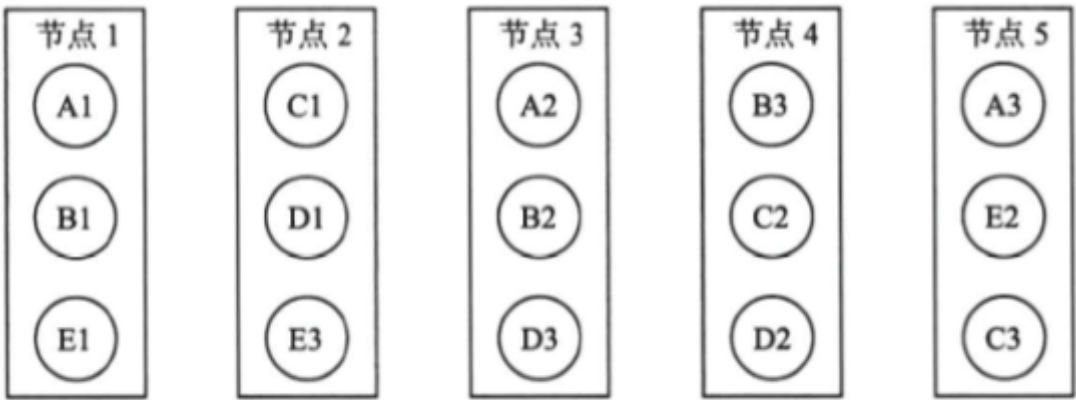


图 3-9 异构系统

分布式协议

分布式系统涉及的协议很多，例如租约，复制协议，一致性协议，其中以两阶段提交协议和Paxos协议最具有代表性。两阶段提交协议用于保证跨多个节点操作的原子性，也就是说，跨多个节点的操作要么在所有节点上全部执行成功，要么全部失败。Paxos协议用于确保多个节点对某个投票（例如哪个节点成为主节点）达成一致。

两阶段提交协议

两阶段提交协议（Two-phase Commit，2PC）经常用来实现分布式事务，在两阶段提交协议中，系统一般包含两类节点：一类为协调者（coordinator），

通常一个系统中只有一个；另一类为事务参与者（participants），一般包含多个。顾名思义，两阶段提交协议由两个阶段组成，如下所述：

- 阶段1：请求阶段（Prepare Phase）。在请求阶段，协调者通知事务参与者准备提交或者取消事务，然后进入表决过程。在表决过程，参与者将告知协调者自己的决策：同意（事务参与者本地执行成功，但没有提交）或者取消（事务参与者本地执行失败）。
- 阶段2：提交阶段（Commit Phase）。在提交阶段，协调者将基于第一个阶段的投票进行决策：提交或者取消。当且仅当所有的参与者同意提交事务协调者才通知所有的参与者提交事务，否则协调者通知所有的参与者取消事务。参与者在接收到协调者发来的消息后将执行相应的操作。

两阶段提交协议可能面临两种故障：

- 事务参与者发生故障。给每个事务设置一个超时时间，如果某个事务参与者一直不响应，到达超时时间后整个事务失败。
- 协调者发生故障。协调者需要将事务相关信息记录到操作日志并同步到备用协调者，假如协调者发生故障，备用协调者可以接替它完成后续的工作。如果没有备用协调者，协调者又发生了永久性故障，事务参与者将无法完成事务而一直等待下去。

Paxos协议

Paxos协议用于解决多个节点之间的一致性问题。多个节点之间通过操作日志同步数据，如果只有一个节点为主节点，那么，很容易确保多个节点之间操作日志的一致性。考虑到主节点可能出现故障，系统需要选举出新的主节点。Paxos协议正是用来实现这个需求。只要保证多个节点之间操作日志的一致性，就能够在这些节点上构建高可用的全局服务，例如分布式锁服务，全局命名和配置服务等。

为了实现高可用，主节点往往将数据以操作日志的形式同步到备节点。如果主节点发生故障，备节点会提议自己成为主节点。这里存在的问题是网络分区的时候，可能会存在多个备节点提议（Proposer，提议者）自己成为主节点。Paxos协议保证，即使同时存在多个proposer，也能够保证所有节点最终达成一致，即选举出唯一的主节点。

大多数情况下，系统只有一个proposer，他的提议也总是会很快被大多数节点接受。步骤如下：

- 1) 批准（accept）：Proposer发送accept消息要求所有其他节点（acceptor，接受者）接受某个提议值，acceptor可以接受或者拒绝。

2) 确认 (acknowledge) : 如果超过一半的acceptor接受, 意味着提议值已经生效, Proposer发送acknowledge消息通知所有的acceptor提议生效。

当出现网络或者其他异常时, 系统中可能存在多个Proposer, 他们各自发起不同的提议。这里的提议可以是一个修改操作, 也可以是提议自己成为主节点。如果proposer第一次发起的accept请求没有被acceptor中的多数派批准 (例如与其他proposer的提议冲突), 那么, 需要完整地执行一轮Paxos协议。过程如下:

1) 准备 (prepare) : Proposer首先选择一个提议序号n给其他的acceptor节点发送prepare消息。Acceptor收到prepare消息后, 如果提议的序号大于他已经回复的所有prepare消息, 则acceptor将自己上次接受的提议回复给proposer, 并承诺不再回复小于n的提议。

2) 批准 (accept) : Proposer收到了acceptor中的多数派对于prepare的回复后, 就进入批准阶段。如果在之前的prepare阶段acceptor回复了上次接受的提议, 那么, proposer选择其中序号最大的提议值发给acceptor批准; 否则, proposer生成一个新的提议值发给acceptor批准。Acceptor在不违背他之前在prepare阶段的承诺的前提下, 接受这个请求。

3) 确认 (acknowledge) : 如果超过一半的acceptor接受, 提议值生效。Proposer发送acknowledge消息通知所有的acceptor提议生效。

Paxos协议需要考虑两个问题: 正确性, 即只有一个提议值生效; 可终止性, 即最后总会有一个提议值生效。Paxos协议中要求每个生效的提议被acceptor中的多数派接受, 并且每个acceptor不会接受两个不同的提议, 因此可以保证正确性。Paxos协议并不能严格保证可终止性, 但是从Paxos协议的执行过程可以看出来, 只要超过一个acceptor接受了提议, proposer很快就会发现, 并重新提议其中序号最大的提议值。因此, 随着协议不断进行, 它会往“某个提议值被多数派接受并生效”这一最终目标靠拢。

Paxos与2PC

Paxos协议和2PC协议在分布式系统中所起的作用并不相同。Paxos协议用于保证同一个数据分片的多个副本之间的数据一致性。当这些副本分布到不同的数据中心时, 这个需求尤其强烈。2PC协议用于保证多个数据分片上的操作的原子性。这些数据分片可能分布在不同的服务器上, 2PC协议保证多台服务器上的操作要么全部成功, 要么全部失败。

常见的做法是，将2PC和Paxos协议结合起来，通过2PC保证多个数据分片上的操作的原子性，通过Paxos协议实现同一个数据分片的多个副本之间的一致性。另外，通过Paxos协议解决2PC协议中协调者宕机问题。当2PC协议中的协调者出现故障，通过Paxos协议选举出新的协调者继续提供服务。

跨机房部署

在分布式系统中，跨机房问题一直都是老大难问题。机房之间的网络延迟较大，且不稳定。跨机房问题主要包含两个方面：数据同步以及服务切换。跨机房部署方案有三个：集群整体切换、单个集群跨机房、Paxos选主副本。下面分别介绍。

1. 集群整体切换

集群整体切换是最为常见的方案。如下图所示，假设某系统部署在两个机房：机房1和机房2。两个机房保持独立，每个机房部署单独的总控节点，且每个总控节点各有一个备份节点。当总控节点出现故障时，能够自动将机房内的备份节点切换为总控节点继续提供服务。另外，两个机房部署了相同的副本数，例如数据分片A在机房1存储的副本为A11和A12，在机房2部署的副本为A21和A22。在某个时刻，机房1为主机房，机房2为备机房。

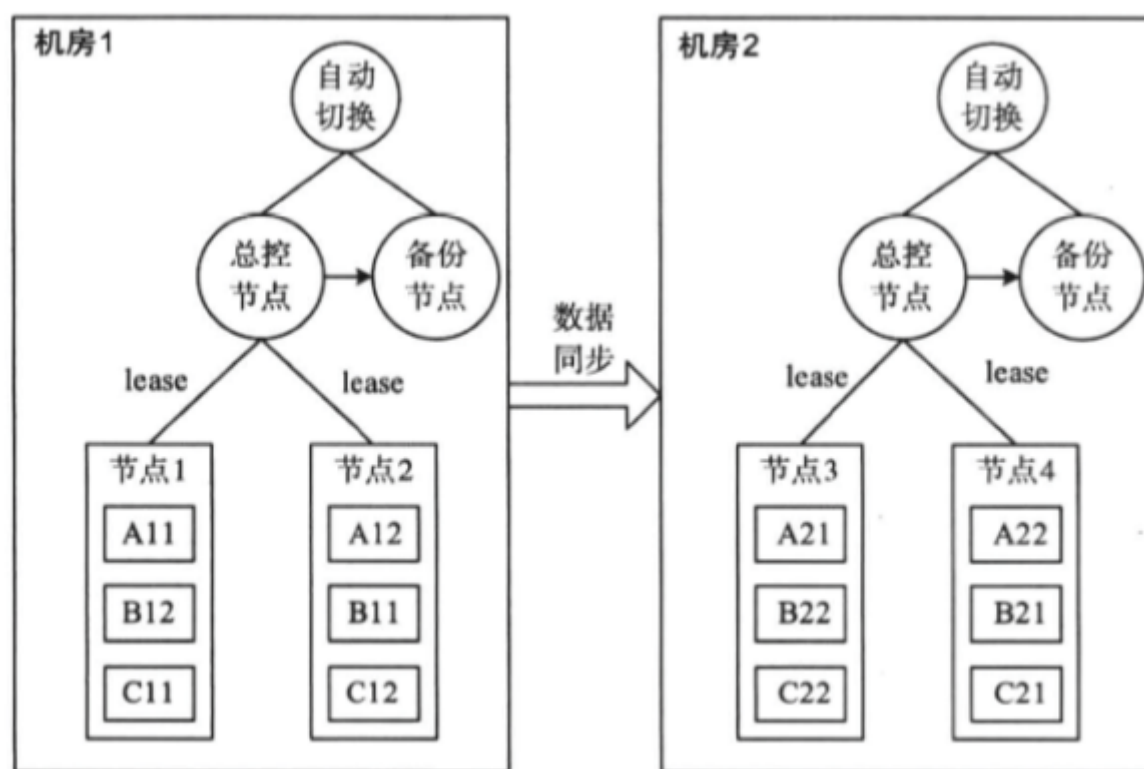


图 3-10 集群整体切换

机房之间的数据同步方式可能为强同步或者异步。

如果采用异步模式，那么备用机房的数据总是落后于主机房。当主机房整体出现故障时，有两种选择：要么将服务切换到备机房，忍受数据丢失的风险；要么停止服务，直到主机房恢复为止。因此，主备切换往往是手工的，因为需要根据业务特点选择“丢失数据”或者“停止服务”。

如果采用强同步模式，那么备机房的数据和主机房保持一致。当主机房出现故障时，可以通过分布式锁服务发现，并自动将备用机房切换为主机房。

2. 单个集群跨机房

上一种方案的所有主副本只能同时存在于一个机房内，另一种方案是将单个集群部署到多个机房，允许不同数据分片的主副本位于不同的机房。如下图所示，每个数据分片在机房1和机房2，总共包含4个副本，其中A1、B1、C1是主副本，A1和B1在机房1，C1在机房2。整个集群只有一个总控节点，它需要同机房1和机房2的所有工作节点保持通信。当总控节点出现故障时，分布式锁服务将检测到，并将机房2的备份节点切换为总控节点。

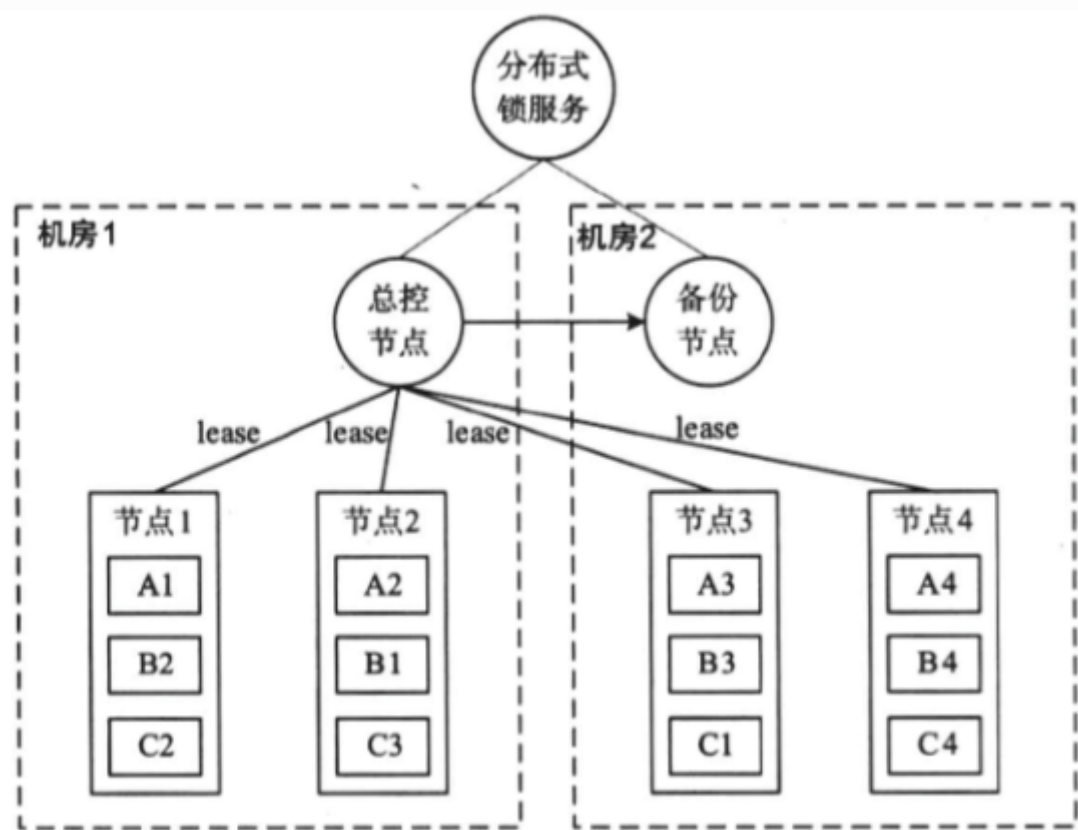


图 3-11 单个集群跨机房

如果采用这种部署方式，总控节点在执行数据分布时，需要考虑机房信息，也就是说，尽量将同一个数据分片的多个副本分布到多个机房，从而防止单个机房出现故障而影响正常服务。

3. Paxos选主副本

在前两种方案中，总控节点需要和工作节点之间保持租约（lease），当工作节点出现故障时，自动将它上面服务的主副本切换到其他工作节点。

如果采用Paxos选主副本，那么，每个数据分片的多个副本构成一个Paxos复制组。如下图所示，B1、B2、B3、B4构成一个复制组，某一时刻B1为复制组的主副本，当B1出现故障时，其他副本将尝试切换为主副本，Paxos协议保证只有一个副本会成功。这样，总控节点和工作节点之间不再需要保持租约，总控节点出现故障也不会对工作节点产生影响。

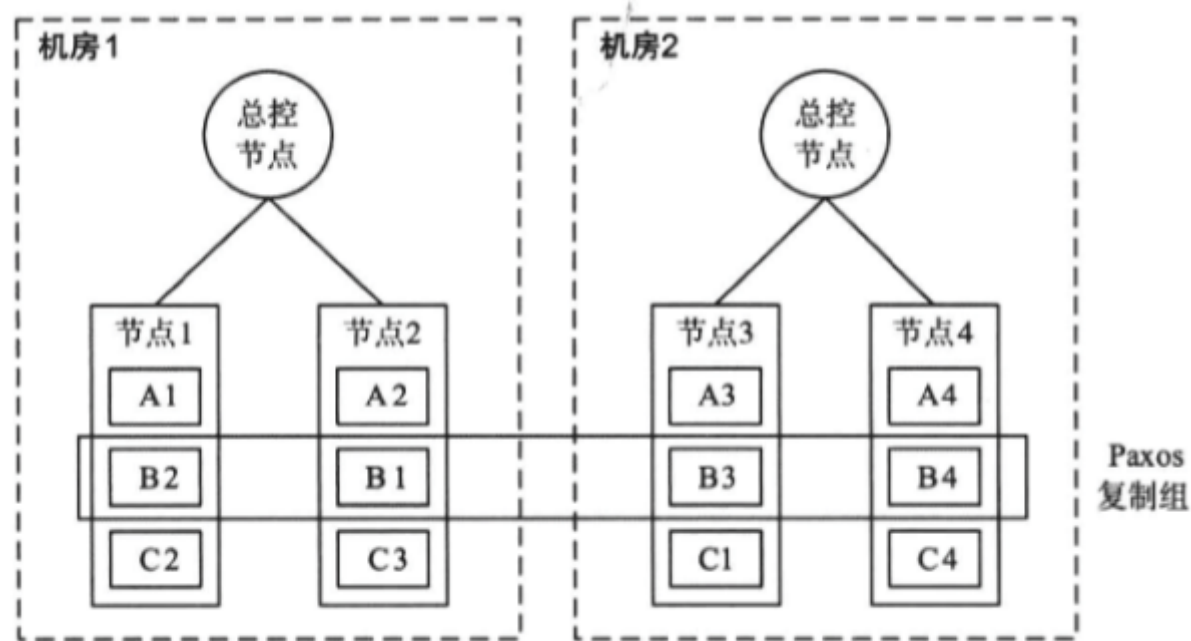


图 3-12 Paxos 选主副本


Google后续开发的系统，包括Google Megastore以及Spanner，都采用了这种方式。它的优点在于能够降低对总控节点的依赖，缺点在于工程复杂度太高，难以在线下模拟所有的异常情况。

新用户152115

我的全0，哈哈。

4月10日 回复

回复一下又不会怀孕...

发布

