分布式协商：建立稳固分布式大数据系统的基石

陈康[1,2,3]，黄剑[1]，刘建楠[4]
1. 清华信息科学与技术国家实验室（筹），清华大学计算机科学与技术系，北京 100084；
2. 深圳清华大学研究院，广东 深圳 518057；3. 浙江清华长三角研究院鄞州创新中心，浙江 宁波 315000；
4. 中国石油天然气股份有限公司庆阳石化分公司，甘肃 庆阳 745002

摘要
分布式协商的目的是在分布式环境下在一组进程之间决定一个共同的值，这是在分布式系统中最基本的问题。分布式协商问题的目标非常简单，但是在面对节点出错、网络出错、网络时延等环境的时候，协议设计以及处理起来十分困难。讨论分布式协商问题的基本形式，在不同的系统假设下的基本结果以及分布式协商在构建稳固的分布式大数据系统中的作用。

关键词
分布式协商；副本状态机；网络错误；安全性；活跃性

中图分类号：TP338.8　　文献标识码：A　　doi: 10.11959/j.issn.2096-0271.2016039

Distributed consensus: fundamental building block for distributed reliable big data system

CHEN Kang[1,2,3], HUANG Jian[1], LIU Jiannan[4]
1. Tsinghua National Laboratory for Information Science and Technology (TNLIST), Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
2. Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
3. Technology Innovation Center at Yinzhou, Yangtze Delta Region Institute of Tsinghua University, Zhejiang, Ningbo 315000, China
4. Petro China Co. Ltd. Qingyang Petrochemical Company, Qingyang 745002, China

Abstract
The goal of distributed consensus is quite simple i.e. how to decide a value among a group of coordinated processes in the distributed environment. However, the problem is turned out to be very difficult while facing the different distributed environment. The even harder problem is that some consensus protocols are hard to be implemented in practical systems. Some results of distributed consensus algorithms under different distributed environment assumptions were reviewed. In addition, some practical systems based on consensus for achieving high reliability were discussed.

Key words
distributed consensus, replicated state machine, network error, safety, liveness

1 引言

随着云计算以及大数据的发展，为了系统的可靠性与性能，将应用系统构建在分布式环境中成为了一个不得不做出的选择。并行处理系统为大数据处理提供大规模并行的处理能力，能够处理更多的数据以及进行更复杂的计算。另外一个方面，分布式环境也带来了系统的可靠性，一个基本的想法是如果系统中有多个计算资源的话，某一小部分计算资源失效不应影响系统的总体运行。这在单机系统中由于计算资源有限的关系不可能实现。在分布式环境下，实现可靠性最常用以及最基本的结构是进行副本复制。在分布式文件系统[1]中，可以通过使用副本的方式将一个逻辑的数据块复制到多个其他物理节点中，这样在某一个物理节点损坏不能工作时，分布式文件系统仍可以继续工作。副本容错的方式不仅仅可以进行数据的容错，也可以进行计算的容错。可以将需要容错的计算分布到不同的物理节点中，这样即使一部分节点失效了，仍然可以保证计算继续进行。

使用副本进行容错最基本的要求是副本之间需要保持一致。在系统的实现上，可以通过名为副本状态机（replicated state machine，RSM）的方式进行副本的复制，这是一种维持多个副本一致的机制。以数据的副本状态机为例，基本的思想是，所有在数据上做的操作，在每一个副本上操作都一样，并且顺序也一样。简单地说，如果数据的初始值是一样的，那么最终获得的数据也是一样的，这就是副本状态机的基本思想。副本状态机不仅可以应用在数据上，还可以应用在计算上。副本状态机顾名思义是与状态机相关的概念，在系统中计算或者数据会被表达为状态机的形式。由于状态机是整个计算机科学的理论基础，因此使用状态机的方式理论上就能够完成所有的计算功能。

图1是副本状态机的基本结构。这里只画出了两个副本的状态变化过程。实际中会使用5个或者更多的副本状态机来达到极高的可靠性。上下两个状态机都是从一个状态S_0开始，即副本在多个服务器上的初始状态是同样的状态。另外，还需要假设所有的状态机都是确定性的状态机。这里“确定性”的含义是给定一个状态，如果给出相同的输入的话，那么都会转换到相同的唯一的一个状态。这样的话，每一次的状态转换都是确定性的，而不是不确定的。如果所有的状态转换的顺序是一样的，那么可以认为最终的状态也是一样的。这一点是很显然的，使用数学归纳法就可以证明。

副本状态机是进行容错的基本结构。例如，在通常提供的可靠性机制中总有一个选项被称为双机热备份。双机热备份提供两个完全一样的运行副本，使得两台机器中的任意一台出现了错误，另外一台可以接管工作，完全掌控剩下的工作。双机热备份的基本思想在很多系统中都有体现。但是，双机热备份实际上并不能真正解决可靠性的问题。最基本的问题是如果双机热备份中的一个节点通过超时的机制判断对方不在线的时候，并不能保证对方

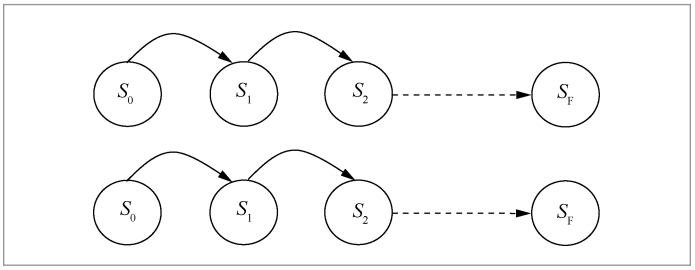


图1 副本状态机的基本结构

节点就一定不在线。因此，如果两个节点都认为对方不在线，那么就会造成两个节点同时服务的情况，很容易打破副本状态机的条件，执行不同的操作，产生状态的分叉，造成不一致。

那么，如何确保系统中有一致的节点各个状态的视图呢？在分布式系统中，通常的做法与投票决定一样。对副本状态机进行状态转换的每一个操作都进行投票，只有大多数都同意的操作才作为下一步的操作。如果有成员不知道当前的投票情况，那么就停止操作，等待将来其知道的时候再把操作补上。这样可以避免出现状态分叉的情况，同时也可以让当系统中的一小部分成员出现错误的时候，分布式系统的工作可以继续。由于在两个成员组成的系统中，大多数成员的情况就完全包含了这两个成员，因此原则上由两个成员组成的系统只有两个成员都正常工作时，才能够保证上层系统继续工作，也就是说失去了容错的能力。因此，一个分布式容错系统起码要包含3个成员（“大多数”为任意两个成员），这样即使有小部分成员出现了失败，大部分成员可以继续通过协商达成一致的结果，推动副本状态机继续执行。

在这样进行容错的系统中，分布式协商（distributed consensus）是一个基本的组成部分，用来让一个副本状态机决定下一步需要做什么样的操作。抽象来说，分布式协商的目的是在一组分布式进程之间协商决定一个值，至于这个值的类型是什么反而不是很重要。分布式协商需要在组成系统的大数据进程都能够正常工作的时候协商出一个值，这样也就有了一定的容错能力。

本文将探讨各种不同的分布式协商协议、分布式协商在副本状态机中的应用以及在实际系统中如何使用分布式协商系统来保证系统的可靠性。本文的目的是揭开分布式协商的神秘面纱，帮助开发者在理解分布式协商的基础上完成可靠的分布式系统的构建。对于希望进一步深入了解分布式协商的研究人员来说，本文也将提供足够的信息来帮助进一步的研究。

2 副本状态机与系统可靠性

在进入分布式协商讨论之前，还需要看一下副本状态机的结构以及在副本状态机上完成系统容错特性需要完成的工作。分布式协商是副本状态机机制重要的一环，但是不是唯一的一环。了解了副本状态机的总体结构之后，就知道分布式协商在副本状态机中所处的位置，进而可以理解其在副本状态机中的作用。副本状态机的结构如图1所示。为了保证副本状态机的正确执行，仍然需要一些条件，下面就对这些条件展开讨论。

在副本状态机的条件中，初始状态相同非常容易保证，这个与具体应用相关。例如，在进行文件系统副本备份的时候，初始状态为一个空的磁盘，之后开始文件系统的操作；或者在进行数据库备份的时候，初始状态为空的数据库；甚至是总体的计算机进行热备份的时候，可以让节点处于同样的状态。这在使用物理节点进行备份的时候不太容易做到，而软件构建的虚拟机很容易做到这一点。本节讨论的内容也是基于虚拟机的方式完成计算的容错。

在副本状态机中的另外一个条件是保证所有的操作是确定性的，不会出现随机的情况，这比保证初始状态一致稍微困难一些，但是使用记录/重放的方式可以达到目的。对于大多数的存储应用，例如文件系统、数据库等，最终的操作会转化为底

层磁盘的读写操作，即读出一个数据块或者写入一个数据块，操作也都是确定性的。对于计算容错来说，最基本的单元是一条指令，指令的执行与具体的节点相关，例如读入当前时钟周期数的指令（x86下是rdtsc，用于读取当前处理器的时钟周期数），很难保证在非常精确的时刻读入两台物理机器相同的时钟周期数。在这个时候需要使用记录/重放技术，在一台物理机中读出真正的数据，在另外一台物理机中只是模拟执行一下，不去真正执行底层的指令。一个更为简单直接的例子是获取随机数，只能是记录一个随机数，然后直接传送给另外一个状态机获得相同的随机数。

最后一个条件就是操作的定序问题。这在文件系统和数据库中最终会表现为所有的磁盘读写操作的顺序。在计算容错方面，最终表现为所有指令的执行顺序。还有一个问题与定序相关，即对于外界请求的定序。对于外部请求进行不同的定序，非常有可能导致最终内部的执行流程是不相同的。多个节点进行定序很困难，因为缺乏全局的时钟（这里所有的操作都要赋予一个自然数的序号）。为解决这个问题，最简单也是最通用的做法是选取副本状态机中的一个状态机所在的节点作为负责定序的节点。所有的请求先在这个节点定出顺序，之后其他的服务器按照相同的顺序操作这些请求。当然，定序节点不应该是固定的，否则如果出现错误的话会导致不能继续定序工作，如何可靠地选择定序服务器也是一个非常困难的问题。

因为是在分布式环境下构建副本状态机，还有一个需要解决的问题是系统成员的变化，被称为配置更新（view change）。这与副本状态机本身没有关系，而是分布式系统特有的问题。一般来说，针对特定的分布式算法需要限定参与的成员。这里不是说成员必须是固定的，而是将情况进行简化，首先考虑成员不变化的情况如何设计分布式算法来达到目的；之后再加上成员变化的情况。成员变化的情况包括增加成员以及减少成员。无论在哪种情况下，原有的分布式算法需要达到的目标在加入或者减少成员之后都应该保持不变。例如，在副本状态机的情况下，原有的状态机都决定在一个文件中写入数据A，那么加入一个新的成员的时候，也需要保持这样一个动作。这种成员变化的情况被称为配置的更新，即配置包括了成员的组成，配置更新代表了成员组成的变化。另外还有一种情况也属于配置更新，即参与成员的角色变化。关于角色的问题实际上是与具体的分布式算法相关的。前面在讨论对于客户端请求的排序上，由于一般通过排序节点完成排序工作，那么这个排序节点就是一个特殊的角色。排序节点的变化也就属于配置更新的情况。总之，配置代表了参与分布式算法中的成员以及成员的角色，其中任意一项发生了变化，都称之为分布式系统的配置更新。分布式算法不仅要在成员固定的时候正确执行，也要在配置更新之前、配置更新进行中以及配置更新完成后都能够正确执行。

综上所述，为了使用副本状态机的方法来构造稳固的分布式系统，需要满足副本状态机的基本条件以及需要对配置更新情况做出正确响应。副本状态机的基本条件包括初始状态相同、所有的状态转换都是确定性的，并且每一步的转换都是相同的。其中，确定每一步转换需要进行的处理动作是通过成员之间协商后确定下来的。这样可以建立一个在大多数成员能够正常工作的条件下推动副本状态机执行的可靠机制。

3 分布式协商

3.1 分布式协商的基本描述与理论结果

维持副本状态机一致的最基本的问题就是让一组状态机（由一组服务器维护）共同决定某个步骤之后的下一个步骤应该执行哪一个操作。因此，核心问题是如何让一组服务器就某一件事情（例如状态机下一步需要做的状态转换）达成一致。这个问题被称为分布式协商。一般意义下，可以说分布式协商是在一组服务器之间协商出一个值。这个值的含义是与具体的应用相关的，本文不再赘述。对于任何应用来说，分布式协商都有一些无意义的平凡协议。这些平凡协议在一组服务器之间决定一件事情或者决定一个值是直截了当的，只需要所有的服务器都预先确定一个值即可。在任何时刻如果要进行协商的话，只需要直接给出这个预定值即可。例如，不管协商的内容是什么，所有的服务器都给出一个固定的值0，显然这也是完成了分布式协商，但并没有什么意义。因此，分布式协商必须要满足有意义这个条件。

实际上，一个分布式协商需要满足的条件大致有以下几点。

- 最后的协商结果只能是一个，协商完成后不能被更改。这是显然的，否则算法就不能被称为分布式协商，这本来就是分布式协商的定义。
- 避免平凡解的条件，即最后决定的值必须是某一个人提出的某一个值（可以称之为提议），不能设置一个默认的值，否则这个算法就没有实际的意义。
- 第三个条件比较隐晦，它涉及如果没有人提出任何值，算法应该怎么办，此时算法不能凭空造出一个值来让成员进行确定；另外算法还没有得出结论的时候，任何一个节点都不能获知这个结论。

因为上述条件的任何一个条件在任何时刻都不能违反，因此这些条件往往被称为安全性（safety）条件。整个算法需要满足一个必须能够结束的条件，即分布式协商最终应当得出结论，选择一个值，而不是由于算法陷入无限循环中。

这个条件是最终必须要得出结果的条件。由于系统网络可能出现数据到达时延、数据丢失等，任何一个算法都不可能在确定的一段时间内完成协商工作，只能保证最终得出结论。这样的条件也常常被称为活跃性（liveness）条件。

分布式协商是否能够达成与网络条件有着密切的关系。有一个著名的结论发表于1985年，被称为FLP不可能性定理。Fischer、Lynch、Patterson 3位科学家指出，在异步通信模式下，即使只有一个参与者发生了失败，也没有任何算法能够保证完成分布式协商，此为结论1。

这虽然是一个令人沮丧的结论，但是实际系统并不是运行在这样一个严酷的环境下。实际系统或多或少是一个同步的系统，例如集群环境下，大部分的数据分组都可以在给定的时间内到达。即使是分布在互联网上的节点，也可以合理假设为到一定超时时间之后，数据分组已经丢失。这样，可以对上述条件进行加强，即在一个更加合理的半异步的模型下，一致性协商是可以达到的，并由明确的算法达到这个协商的目的。半异步模型保证了虽然网络数据分组可以丢弃分组，可以有时延，但是在一段足够长的时间内，大部分节点之间的网络是正常通信，并且在超时之前将数据分组分发给对应的进程。半异步模型是一个更加合理的模型，实际系统中如果需要系统工作，那么在构建系统的时候还是需要底层网络比较可靠的支持，起码需

要保证在大部分的时间内大部分的网络可以正常工作。在这种模型下，有一个著名的Paxos算法[2]能够解决分布式协商的问题。Paxos算法指出，具有f个可能错误节点的半异步通信模式下，总数为$2f+1$个节点是可以达成分布式协商的，此为结论2。可以看到，这里的条件就是让大部分的节点可以正常工作。

最后一个有意思的结论是如果将错误模型进行更改，允许节点“故意”出错，破坏分布式协商的过程，这样的模型被称为拜占庭通信模式[3]。在拜占庭通信模式下，具有f个可能错误的节点，总数为$3f+1$个节点是可以达成分布式协商的，此为结论3。

以上3个结论即为在实际系统中会使用到的结论，其中结论1和结论3主要具有理论意义，能够指出满足假设前提下进行协商的极限。结论2是一个明确的算法，能够直接用来构架可靠的副本状态机。

3.2 实际系统中的分布式协商协议

实际系统往往是前面所说的半异步的通信模型，因此分布式协商是可以在大多数成员之间达成的。当前实际系统中使用的协议包括著名的由图灵奖得主Lamport提出的Paxos协议、Yahoo研究院提出的ZooKeeper协议（ZooKeeper Atomic Broadcast协议，简称ZAB协议）[4]以及Stanford大学研究人员提出的专用于教学的Raft协议[5]。

上述3个协议的核心结构都是相同的。

图2是分布式协商协议中核心的协商结构，上述3个协商协议的核心部分都是基于这样的一个结构来完成的。在这个核心结构中，客户端（client）将请求发送给一个领导者（leader）。领导者是唯一的，用以对并发的客户端请求进行定序，保证输入到副本状态机中的操作是有序的。唯一的领导者也是上述3个协议的活跃性的保证，能够保证最终协商完成。领导者将客户端的请求发送给所有其他副本状态机中的成员，这些成员被称为追随者（follower）。这些追随者都复制领导者的操作，把发过来的操作加入本地的日志队列中。如果大多数的追随者都同意领导者发过来的一条操作（一个值），那么领导者将提交这个值，将其作为分布式协商的最后结论。

在这个核心结构的外围，需要一些辅助的模块，包括两个非常重要的功能，一个是领导者的选择（leader election）；另外一个是如何进行配置更新（成员增加或者减少）。这3个协议在不同的层面上完成了分布式协商，并推动上层的副本状态机的执行。Paxos协议是纯粹的分布式协商协议；ZAB协议考虑了领导者的选择算法，将其集成到协议中；Raft协议则更进一步将配置更新的协议也加入到其中，完善了分布式协商的外围工作。需要注意的是，虽

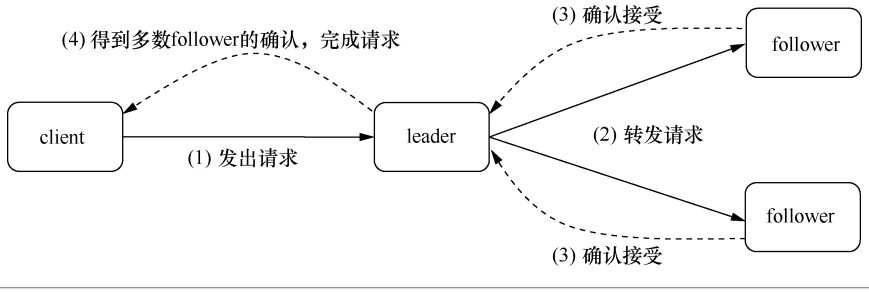


图 2 分布式协商中的核心协商结构

然3个协议共享了类似的协商核心结构，但是具体的协商过程各不相同。

Paxos协议最为简单，是一个纯粹的分布式协商协议。Paxos协议只考虑一次协商，不考虑多次协商，这虽然与副本状态机的要求有差距，但是这还是一个非常基本的问题，能够用来协商副本状态某一次具体的操作。Paxos协议基于投票的思想，一旦某个提议（包含某一个值）被大多数成员接受，那么这个提议的值就作为协商的结果。但是，由于分布式系统的特点，这样一个协商完成的结果不见得被其他成员知道，投票过程可能会继续。分布式协商需要保证的是：如果一个值已经被协商完成，被“固定”在系统中，那么无论如何进一步地投票，最终的投票结果就是已经完成协商的值。核心思想就是在进行值的提议的时候，必须要询问足够的但尽可能少的成员，将可能已经完成协商的值作为值的提议。Paxos协议只解决一次协商问题，但是副本状态机需要多次协商，那么就需要为每一次协商执行一次Paxos算法。Paxos本身不完成定序的问题，多个Paxos协议可以并行执行，通过标记可以相互独立运行。

ZAB协议没有在单个的分布式协商基础上进行讨论。由于使用了副本状态机，ZAB协议考虑多个协商共同进行的情况。ZAB协议包含了一个领导者选举的算法。领导者选举过程中，所有成员都互相交换信息，看看当前自己的内部状态是不是能够跟上最新的副本状态机的状态。包含最新信息的成员自动成为领导者。新领导者被选出之后，为了保证所有操作的有序性，新领导者需要将前面一个领导者（已经失效）遗留的操作都提交一遍。这个过程就是错误恢复的过程，能够保证所有操作的有序性。之后，整个协议将进入上述分布式协商的基本结构中，快速将操作同步到所有的跟随节点。

Raft协议与ZAB协议非常相像，都是尽量让新的领导者帮助完成上一个领导者尚未完成的工作。并且，Raft协议是一个更加完整的协议，因为其加入了配置更新的协议。Raft协议的第一个部分是关于领导者选举的，在一个随机的超时时间范围内进行投票，获得足够多投票的成员自动成为领导者。之后，协议进入上述的分布式协商的基本结构中。领导者会复制自己的整个状态给所有的跟随者，并且在大部分的跟随节点完成复制之后提交对应的日志。这是将上述的分布式协商基本结构与状态恢复的过程结合在一起，在新的领导者完成提交的同时，“顺便”将前一个领导者的协商值的提议一起提交。Raft协议的完整性体现在对于配置更新方面的支持。通过两个阶段转换的方式，Raft协议可以正确完成配置更新，并不影响正常分布式协商的正确性。为了能够维护整个系统的长期稳定运行，配置更新协议也是必不可少的。

3个协议的最大的区别在于对领导者进行转换时处理，即从一个旧的领导者换成新的领导者时需要做什么样的工作。在Paxos协议中，新的领导者将观察自己保存的状态，严格按照协议执行，不破坏已经决定的值。这个过程不涉及多个执行中的Paxos协议，因此领导者在进行转换的时候，没有其他信息帮助提交多个值，只能尽可能去保护现有的可能已经选定的值。ZAB协议和Raft协议不是一个单纯的一次性的分布式协商协议，这两个协议是与副本状态机紧密结合在一起的。在这两个协议中，如果发生了领导者的转换，那么就必须考虑如何处理上一个领导者遗留的尚未提交的协商值。因此，这两个协议都对新的领导者的选择作出了限制，即新的领导者必须知道一些必要的关于当前系统状

态的信息。在ZAB协议中，新的领导者选举出来之后，需要确定在获得的支持成员中有最新的一个请求作为出发点，之后将自己的请求接到最新请求的后面，作为下一个请求。ZAB协议保证必须让新的领导者帮助提交上一个领导者工作时产生的请求，只有全部提交完成之后，新的领导者才能工作，领导者才真正成为领导者。在Raft协议中，也对新的领导者选择做出了限制，为了保证正确性，需要保证在选出新的领导者的时候，必须要从之前的已经完成提交的最后一条请求对应的成员节点中去选择，而不是去任意选择一个成员节点。这样的限制使得在投票选出新的领导者时，每一个成员都需要看一下领导者的候选节点具有的信息是否比自己的信息要更新一点，如果是的话则同意候选节点，否则将拒绝候选节点。通过这种方式，新当选的领导者不必进行帮助前一个领导者提交提议的过程，而是可以直接进入自己提议的过程，并且在这个基础上“顺便”帮助前一个领导者提交遗留在系统中的提议。这也是Raft协议和ZAB协议最大的不同。

上述的3个协议是当前在实际系统中使用最为广泛的分布式协商协议，3个协议各有特点，可以使用在不同的场景下。3个协议要完成的目标是相同的，都是如何可靠地推动副本状态机的执行。3个协议提出的背景各不相同，ZAB协议和Raft协议改进了Paxos协议中的难于理解的困难，特别是Raft协议一开始就是为了教学所提出的。Raft协议已经非常完善地描述了副本状态机需要解决的问题以及相关的解决方案，值得初学者首先选择进行阅读理解。

4 分布式协商协议的应用场景

第3节分析了分布式协商的含义以及在实际系统中的使用的分布式协商协议的情况。由于协议本身的复杂性，这部分内容需要感兴趣的读者花费较长的时间进行理解。但是，在实际系统中，遇到的问题更多的是如何使用分布式协商。分布式协商在实际的系统中具有广泛的应用，对建立可靠的分布式系统起到决定性的作用。下面就以分布式协商系统在BigTable[6]系统中的作用来阐述分布式协商系统如何应用于大规模的大数据系统中来保证系统的可靠性。

将BigTable系统视作一个分布式数据库，并且这个数据库是经过排序的，即按照数据记录的主键进行排序。**图3**是BigTable系统进行数据查找的流程。

从**图3**可以看出，在分布式环境下，BigTable的数据表被组织成类似于分布式环境下的“B+树”的形式。根的数据表和第一级的元数据表用来进行路由工作，之后再依据获得的用户数据表的位置来真正读写用户的数据表。由于表格中的数据太大了，在BigTable中使用了按照行进行分割的方式，被称为数据分表。任何一个客户端，如果需要访问BigTable中的数据分表，首先需要经过元数据表的路由才可以访问到具体的数据分表。整个路由表的根的指针放置在名为Chubby[7]的系统中，这是进行路由启动部分的数据。Chubby基于副本状态机以及Paxos协议[8]建立了一个稳固的分布式协同系统，为其上的应用程序提供基础服务。

在维护系统可靠性方面，两个特别典型的问题需要解决，一个问题是如何得知整个系统中当前能够正常执行的节点的数目；另外一个问题是如何协调正常执行的节点之间的数据访问。

(1) 维护系统正常工作节点的状态信息

关于节点是否正常工作的信息是进行容错的一个基础性问题，只有得知系统中的哪些节点在正常工作，哪些节点已经出

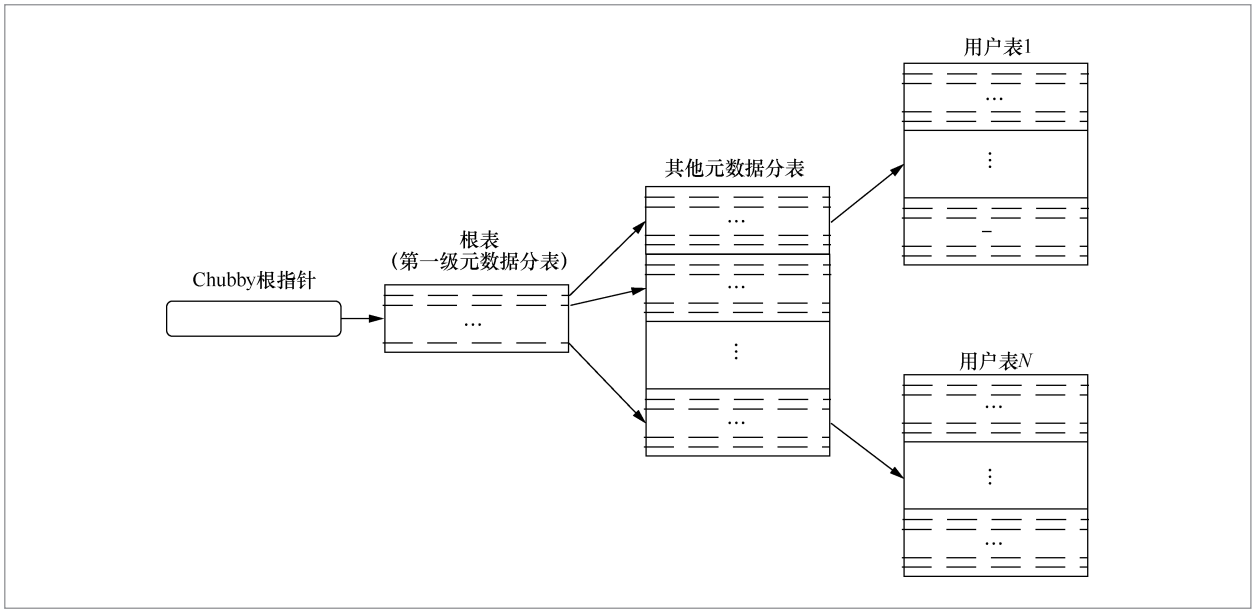


图 3 BigTable 系统中的数据查表的流程

现了错误，才可能进行后续的修复工作。实际上在分布式系统中是非常有必要进行这样的状态维护的，这样不仅可以协调系统的容错恢复功能，也可以给管理员提供建议信息，为自动以及手动维护提供依据。这件事情虽然看起来非常简单，但是如果系统的规模达到了数千台机器，那么进行节点状态维护就变得极为复杂，用人工的办法根本没有办法实现。

维护节点正常工作一般的做法是通过节点之间的心跳。但是，节点之间的心跳会出现信息不准确的问题，即双方都不能探测到对方的存在，但是仍然不能确信对方确实下线，这有信息不一致的问题。这里一个最典型的情况就是引言中讨论的双机热备份的问题。在双机热备份问题中，一台机器为主要的副本，另外一台机器是备份的副本。在这里，关键的问题是主要副本与备份副本之间的网络情况会造成双方都无法确信对方是否真的不在线。

使用分布式协商就不会出现上面的不一致的问题。分布式协商就像一个委员会，对于某一个节点是否在线的问题可以由委员会的大多数成员来决定。这样就可以判断双机热备份的情况下，哪一台是主要副本，哪一台是备份副本。类似地，在BigTable系统中，集群中的成员是否正常工作的信息都保存在一个名为Chubby的系统中。每一个Chubby实例包含了5个成员，这5个成员分布在不同的地理位置，这就保证了在绝大多数的情况下大多数成员可以正常工作。在这5个成员之间组成了文件系统的副本状态机，每一个集群中的服务器的信息都保存在这个文件系统中的某一个文件中。服务器如果在线的话，会按固定的时间间隔更新对应的描述自身文件的信息。其他的服务器包括Chubby中的成员可以检测对应文件的有效性，如果无效的话，可以认为对应的服务器已经下线。当然，即使Chubby中的信息指示的对应的服务器已经下线了，这并不代表着对应服务器确实出现了错误。为了保证正确性，BigTable中某一个数据分表最多由一个服务器负责，否则就会出现数据的不一致。只有Chubby能够通过一致性协商的方式确

定某个服务器是否下线，即使此时对应的服务器还在正常工作，由于其在一定的时间内不能成功更新在Chubby中关于自身的信息，这台服务器依据协议的规定，应当自动解除自己对数据分表的负责工作，等待整个系统安排重新上线。这样，可以保证在任何一个时间点，对于任何一个数据分表来说，最多只有一台服务器负责，避免数据出现不一致的情况。

（2）协调成员节点之间的数据访问

另一个问题是协调正常执行节点之间的数据访问。在单机多线程环境中，有共享数据访问的问题。共享数据访问是内存中设置了共享变量，有两个以及两个以上的线程访问（读/写）这些共享变量，其中至少有一个访问是写入数据。这样的情况会产生数据访问冲突。

在多线程环境中，解决数据访问冲突一般使用保护锁的方法。任何一个线程在访问共享数据之前，首先要获得一个锁，之后再访问数据，最后释放锁。互斥锁的语义使得任何一个时刻只有一个线程可以访问共享的数据，其他的线程只能等待。同样的机制可以扩展到分布式环境中。在分布式环境中，一般建立锁的机制是使用锁服务器。互斥锁在进行访问协调的时候起到了非常重要的作用，如果互斥锁发生了失败，会造成整个分布式系统无法工作。

在分布式环境中，需要建立非常稳固的锁机制，将可能产生的锁失效的概率降到最低。副本状态机此时起到非常重要的作用。在Chubby中，互斥锁被建立到副本状态机中，表现形式为文件系统命名空间中的在文件以及目录这样的有名对象上所实现的锁。在分布式环境中需要对共享资源进行访问的时候，首先要访问锁服务器，只有在成功获得锁的情况下，才可能访问对应的共享资源。但是在分布式环境中的锁服务机制与传统的单机环境的锁服务机制不同。在单机环境中，不需要考虑出错的情况，因为一旦出错，那就是整个节点的错误，不仅仅是锁的问题。因此，单机中没有获得锁的线程原则上只能进行无限的等待。但是，在分布式环境中这样做是不允许的，因为数据分组会在网络中进行时延，也可能进行分组丢弃。访问锁的情况可能会被延迟，获得锁的服务器可能会失效。如果获得互斥锁的服务器发生了失败，那么对应的锁将永远不能释放，这就造成了很大的问题。因此，在分布式环境中，对于锁的访问必须要加上超时机制。基本的方式是在获得锁之后，默认情况下只能拥有这个互斥锁一段时间，超过了这段时间锁将自动被锁服务器释放，便于其他的服务器进行下一步的工作。如果一个服务器获得了锁，并且希望将来继续使用这个锁，那么这个服务器必须要进行续租的操作，延长获得锁的时间。

将上述的原理应用在实际工作的系统中，还需要考虑网络时延以及时间测量的误差，确保在访问共享资源时，最多只有一个节点进行访问，其他节点都被排除在外，这样才算是正确的互斥锁的语义。如果时延的数据分组没有到达，或者没有获得响应，对应的服务器只能认为延长请求失效，退出对于对应共享资源的控制，避免对数据造成不一致的操作。这样锁的协议会比较复杂，也是分布式与单机环境的区别。BigTable中有许多对共享资源的访问，其中就有对于底层的分布式文件系统的文件的访问。另外，也有对于数据分表的再拆分以及数据分表的合并的过程，也有对于共享日志的分配以及恢复操作等。可以说，任何一个分布式系统，对于共享资源的访问是无可避免的操作，因此分布式的互斥锁的机制是无可避免的分布式的系统功能。建立一个稳固的分布式锁的环境是建

立高可靠的分布式系统的关键组成部分。

通过BigTable的具体机制可以看到，通过分布式协商以及副本状态机，Chubby建立了一个非常稳固的、由副本状态机推动的复制的文件系统。虽然文件系统的语义非常基础，不能够提供高层的语义信息，但是Chubby提供了最基本的存储模型。在这个模型的基础之上，Chubby完成了整个系统的完整的活跃信息描述，协调了BigTable对于共享数据分表资源的访问。实际上，Chubby成为了Google(谷歌)内部的分布式大数据系统的基础，为难于实现的可靠性问题提供了基石。在这个基础之上，建立上层应用系统的可靠性难度将大大降低。

5 结束语

本文对现有的保证可靠性的副本状态机进行了详细的讨论，包括副本状态机需要满足的条件、基本的执行流程以及副本状态机在实现可靠分布式系统中的作用。副本状态机需要满足的条件为初始状态相同、状态转换函数必须是确定性的以及转换操作与过程必须使用分布式协商算法。在此基础上，详细分析了现有的分布式协商算法的协议，包括Paxos协议、ZAB协议以及Raft协议。这些协议各有特点，并有不同的分布式系统的实现。Paxos协议是单次的分布式协商，虽然简单，但是需要配合许多其他的组件才能够真正构成副本状态机的工作机制。ZAB协议和Raft协议是完整的分布式副本状态机的协议，能够直接推动副本状态机的执行，其中Raft协议还继续提供了配置更新的协议，完善了在实际系统中的各个细节。在分析完成常用的分布式协商协议的基础之上，还分析了如何通过副本状态机构造可靠的分布式系统。需要提供的主要模块是完成整个系统的节点状态的监测与维护以及协调正常工作节点之间的工作。可以看到，分布式协商技术能够提供可靠性的基本模块，是建立可靠性系统的基础。通过本文的分析，读者可以对分布式协商有一个基本概念，为建立可靠的分布式系统提供基础的模型。

参考文献:

[1] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29-43.

[2] LAMPORT L. Paxos made simple[J]. ACM SIGACT News, 2001, 32(4): 51-58.

[3] LAMPORT L, PEASE M, SHOSTAK R. The byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401.

[4] HUNT P, KONAR M, JUNQUEIRA F P, et al. ZooKeeper: wait-free coordination for internet-scale systems [C]// The 2010 USENIX Annual Technical Conference, June 22-25, 2010, Boston, MA, USA. [S.l.:s.n.], 2010: 1-14.

[5] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C]// The 2014 USENIX Annual Technical Conference, June 19-20, 2014, Philadelphia, PA, USA. [S.l.:s.n.], 2014: 305-319.

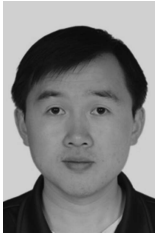
[6] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: a distributed storage system for structured data[J]. ACM Transactions on Computer Systems, 2008, 26(2): 205-218.

[7] BURROWS M. The Chubby lock service for loosely-coupled distributed systems[C]// The 7th Symposium on Operating Systems Design and Implementation, November 6-8, 2006, Seattle, WA, USA. New York: ACM SIGOPS, 2006: 335-350.

[8] CHANDRA T D, GRIESEMER R,

REDSTONE J. Paxos made live: an engineering perspective[C]//The 26th Annual ACM Symposium on Principles of Distributed Computing, August 12-15, 2007, Portland, Oregon, USA. New York: ACM Press, 2007: 398-407.

作者简介



陈康(1976-),男,博士,清华大学计算机科学与技术系、深圳清华大学研究院、浙江清华长三角研究院鄞州创新中心副教授,主要研究方向为分布式系统、存储系统。

黄剑(1993-),男,清华大学计算机科学与技术系硕士生,主要研究方向为文件存储器和分布式系统。

刘建楠(1963-),男,就职于中国石油天然气股份有限公司庆阳石化分公司,主要从事企业经营和信息化管理工作。

收稿日期: 2015-06-20

基金项目: 国家自然科学基金资助项目(No.61433008, No.U1435216, No.61373145, No.61170210);国家高技术研究发展计划("863"计划)基金资助项目(No.2014AA01A302)

Foundation Items: The National Natural Science Foundation of China (No. 61433008, No. U1435216, No.61373145, No.61170210), The National High Technology Research and Development Program of China (863 Program) (No.2014AA01A302)