

硕 士 研 究 生 读 书 报 告



题目 PBFT共识算法详解

作者姓名 赵研

作者学号 22151192

指导教师 李启雷

学科专业 软件工程

所在学院 软件学院

提交日期 二〇二二年一月

**摘要**

PBFT是一种容忍拜占庭容错的分布式一致性算法。支持拜占庭容错的算法越来越重要，因为未来恶意攻击和软件错误将会变得普遍，同时需要可以容忍恶意节点的恶意攻击。之前的类似的一致性算法在实践中效率低下，而PBFT可应用与实践。它可以应用于异步的环境之中，比之前的算法在响应时间上提高了几个数量级。

**关键词**：分布式一致性算法；PBFT；拜占庭容错

**Abstract**

This paper describes a new replication algorithm that is able to tolerate Byzantine faults. We believe that Byzantinefault-tolerant algorithms will be increasingly important in the future because malicious attacks and software errors are increasingly common and can cause faulty nodes to exhibit arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates several important optimizations that improve the response time of previous algorithms by more than an order of magnitude. We implemented a Byzantine-fault-tolerant NFS service using our algorithm and measured its performance. The results show that our service is only 3% slower than a standard unreplicated NFS.

**Keywords:** pbft；Byzantine fault tolerance；distributed consistency algorithm

# 1.简介

因为恶意攻击和软件错误会导致故障节点表现出拜占庭行为，所以拜占庭容错算法变得越来越重要。

算法提出了状态机的概念，算法支持不超过（n-1）/3的恶意节点的系统，算法工作在异步系统之上类似于因特网，它保持系统的一致性并且高效的运行。然而，大多数早期的工作要么关注的是旨在证明理论可行性的技术，但由于效率太低而无法在实践中使用。设计出的接近实际应用的算法，如Rampart和SecureRing，但是他们是建立在同步系统之上的算法，在恶意节点进行攻击时，无法保证有效性。攻击者可以通过延迟非恶意节点之间的交流知道他们被系统判断为成为了恶意节点然后被移除出集群来进行攻击，这种拒绝服务攻击通常比获得对非故障节点的控制更容易。

这篇文献主要做了以下的贡献：

它贡献了第一个在异步网络系统中的支持拜占庭容错的状态机复制协议；它做了很多重要的优化以便在系统中的性能更优；实现了支持拜占庭容错的分布式系统；它提供了实验数据证实了一致性技术的花费。

系统在不超过(n-1)/3的故障节点的情况下可以维持自己的安全性和活性。系统不依赖同步操作来提供安全性

# 2.算法实现

我们的算法是状态机的一种实现。每一个状态机都维护这一个服务状态和改变服务的操作。我们假设有R个数目的节点，并且用{0,1…R-1}来标识这R个节点。为了简化标识，我们令R=3f + 1，其中f是系统允许的最大的故障节点数量。当然系统的节点数量也可以超过3f+1，以提供系统的弹性。

复制机通过views来区分不同的配置代际的信息。在整个系统中，一个节点是primary，其他的都是跟随节点。Primary节点的编号p等于v mod R，v是当前的代际views，primary出现失败的时候会发生views change。整个系统的提供服务的步骤如下：

1. 客户端向primary发送一个请求
2. Primary节点将请求发送给其他的replica节点
3. Replica节点执行这些请求并且将他们返回给客户端
4. 如果客户端收到从f+1个不同节点的相同结果的话，则操作成功完成

系统要求所有的节点必须按照相同的顺序执行客户端的操作才行，以此来保证节点状态机的一致性。

## 2.1客户端

客户端向primary节点发送请求Request<Request,o,t,c>，时间戳Timestamp t被用来保证client发送的请求按顺序被执行，因为后面的时间戳比前面的时间戳更大，从而确定一个顺序性。

每一条由replica返回给client的结果都包含当前的view number，客户端发送给他认为当前是primary的主节点，通过点对点的通信，然后primary将请求发送给所有的其他的节点。

最后replica将结果直接发送给client。结果的形式是<Reply,v,t,c,i,r>，其中v是当前的view number，t是request的时间戳，i是replica的编号，r是返回的结果。客户端等待f+1个不同replica的签名验证成功的一致的结果后，才能确定结果的正确性。

如果客户端不同及时的接收到所有的来自replica的结果，客户端就直接将请求广播到所有的replica上去。如果replica已经处理了这个请求的话，relpica就简单的重新返回处理后的结果，replica会缓存处理的最后一个request的结果，另外，如果replica不是primary的话，他会将请求转发给primary，如果primary没有将请求发送给集群的话，他会被引起怀疑，如果怀疑他的replica的数量足够多的话，会发生view change事件。

在这篇论文中，假设客户端等待一个request返回成功后才会发起下一个request。但实际情况下我们可以允许系统在异步环境中运行，但要保留对它们的排序限制。

## 2.2正常情况下的操作

Replica的状态信息包括服务的状态，具体有消息日志包含replica已经接收的消息，一个表示views的整数。

当primary节点，接收到来自客户端的请求时，开始了三阶段的提交流程。只要客户端提交的信息没有超过给定的信息上限的话，primary会立即执行协议。在这种情况下，primary会缓存请求，同时将请求分片发送给replica。这三阶段提交分别是pre-prepare，prepare和commit。pre-prepare和prepare阶段用于对同一视图中发送的请求进行完全的排序，即使primary节点是异常的。Prepare和commit阶段确保请求最终是按照顺序的。

在pre-prepare阶段，primary分配了一个sequence number给request的message。Message的格式是<<PRE-PREPARE,v,n,d>p,m>，v是message被发送时系统的views，m是系统的具体的消息，d是消息的摘要。

具体的request并不包含在这个message中，目的是缩小容量，因为pre-prepare的目的是为了达成共识，replica接受pre-prepare消息按照下面的格式进行：

1. 在request和pre-prepare的签名正确，通知message的摘要m也是正确的
2. View 是当前的view
3. 目前还没有接受到相同sequence number 和 view 的不同摘要的消息
4. 消息的sequence number在系统允许的范围内

第4条是为了恶意的primary节点发送一个非常大的sequence number来破坏系统的稳定性。

如果replica接收了prepare消息，他会产生消息<PREPARE,v,n,d,i>，并将这个消息传递给其他所有的replica上去，并把消息添加到日志中。如果replica接收到了prepare消息的话，验证签名的正确性，验证view是否是当前view，验证sequence number是不是在h和H之前，所有验证成功才算成功。我们定义prepared<m,v,n,i>正确当且仅当replica i 将这条消息插入到了自己的日志中。Request m，对于m的pre-prepare和接收到来自不同replica的2f个prepare。Pre-prepare和prepare保证了非异常的replica对view下的request达成一致。他保证了：如果prepare（m,v,n,i）是正确的，那么prepare（m’,v,n,j）都是错误的，包括j=i的情况，只要D(m’)!=D(m)。这是正确的结论因为R=3f+1，说明至少有2f+1个非异常节点认同了prepare消息，如果prepare(m’,v,n,i)是正确的只要要有一个节点既发送了m是true又发送了m’是true，这是不可能发生的。

最后replica广播commit<commit,v,n,D(m),i>给其他的replica节点当prepare<m,v,n,i>被证明是正确后，这将开启commit进程。Replica接收commit信息并把他添加到log中只要commit信息的签名正确。我们定义commited（m,v,n）是正确的当且仅当收到f+1个非异常replica的commit，同时commited-local是正确的当prepared是正确的并且replica i已经接收了来自不同的replica的2f+1个commit并且要对应pre-prepare的信息。

Commit确保一个不变的特性，如果committed-local（m,v,n,i）是正确的那么commited(m,v,n,i)一定也是正确的。确保了任何request只要在 非异常replica上本地提交成功，一定会在超过f+1个非异常replica上提交。

每一个replica i 会执行request的操作当commited-local(m,v,n,i)正确的话并且按照sequence number的顺序进行执行。保证了系统的安全性。当执行完操作后，replica会向client返回一个reply，replica会丢弃时间戳小于返回给客户端的时间戳的request，以保证精确的语义。

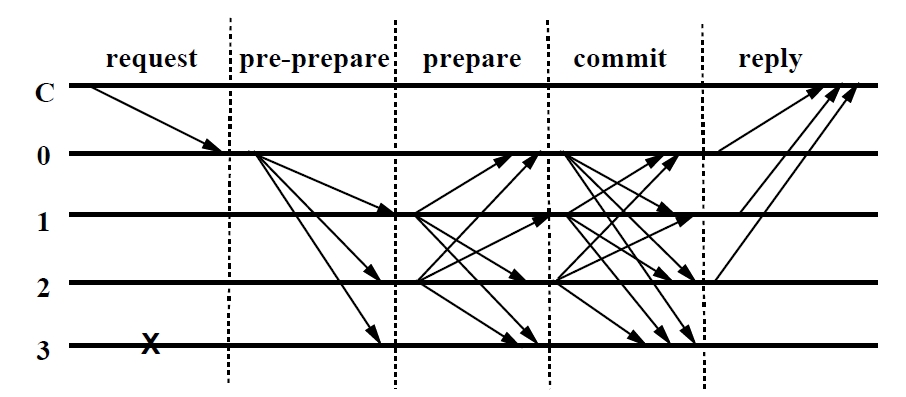


图1 Normal Case Operation

## 2.3 垃圾回收

这节讨论的是log的消息回收机制。当一个sequence number的request被执行完成后，我们将会产生一个checkpoints对这个request，同时当一个checkpoint被证明后，我们称这个checkpoint为stable checkpoint。

为checkpoint生成proof的流程如下：当replica i 生成一个checkpoint，他会向其他replica发送<CHECKPOINT,n,d,i>，n是最后一个请求的sequence number，这个请求已经被执行完毕并且已经改变了replica的状态，d是这个状态的摘要。Replica维持这个checkpoint的log直到超过2f+1个replica已经响应这个checkpointed消息。Checkpointed如果成功的话，replica丢弃掉所有的sequence number小于n的pre-prepare，prepare和commit消息，同时丢地掉所有的前面的checkpoints和checkpoint信息。

通过log计算h和H。h是最后一个stable checkpoint，H是h+k，通常k都比较大，例如每100个request采集一个checkpoint，那么k可能是200。

## 2.4 view changes

View changes协议保证了系统的可用性当primary出现异常的情况下。在超时的情况下会被触发，防止replica一直在等待执行。Replica等待request当且仅当它接收到了一个有效的request并且还没有进行执行。Replica会启动一个计时器当他接收到一个request并且计时器之前没有启动的情况下。它停止计时器当它不在执行request时，当他又开始执行其他的请求时又会重启计时器。

如果在代际为view时repilca发生了超时事件，首先将系统的代际调整为view+1。首先会停止对外服务，不再接收新的消息，同时发送消息<VIEW-CHANGE,v+1,n,C,P,i>消息到其他的replica上去，n是当前的replica知道的replica i 的最后一个stable checkpoint，C是一个拥有2f+1个有效的checkpoint信息来证明s的正确性，P是pm的一个集合对于每一个request m 对于replica i 来说m的sequence numbe要大于n才行。每一个pm包含一个有效的pre-prepare message并且要有2f个匹配，prepare message要有不同的replica的签名，同时拥有相同的view， number，m的摘要等信息。

当当前的primary收到来自不同2f个replica的有效的view-change的v+1的消息时，它广播消息<NEW-VIE sequence W,v+1,V,O>到所有的replica上去，V是一个primary收到的2f个view-change信息的集合和发送的view+1的view-change的集合的并集，也就是2f+1个view-change消息的集合。O是pre-prepare message的集合，O的计算过程如下：

1. primary计算出最小的sequence number是集合V中的最小的stable checkpoint，最大的sequence number 是集合V中prepare message的最大的sequence number
2. primary为min-s到max-s中的每一个sequence number创造一个新的pre-prepare message，有两种情况：
   1. 对于在这个高度上存在的request集合，选取v值最大的request，直接创建<pre-prepare,v+1,n,d>消息
   2. 没有a情况的集合，primary创造一个新的message<PRE-PREPARE,v+1,n,d>，d是pre-prepare消息中的对于V中最高的view的request的摘要，其中d是一种空的request的摘要，与运行其他的协议一样。

接下来primary 将O中的message添加到自己的log中去，如果min-s比最近的stable checkpoint大的话，primary依然将它insert到log中，把sequence number设置为min-s，并且抛弃掉相应的log数据。并且将view设置为view+1。

Replica接收新的new-view信息如果签名正确的话，如果message包含的信息正确的话，同时O正确的话，如果O是正确的话，添加新的信息到log当中，广播他们的prepare信息到其他的replica当中去，将这些prepare添加到log当中，并且将view设置为view+1。

Replica可能错过某些request信息或者某个stable checkpoint。它可以从其他的replica中获取这些丢失掉的信息。

## 2.5 安全性

当所有的非故障节点对同一个sequence number的request达成一致并且本地提交的话，算法可以保证安全性。

首先如果prepare(m,v,n,i)是正确的话，prepare(m’,v,n,i)一定是错误的对于任何的非故障节点来说。这意味着对于同一个sequence number和同一个view来说的话，只能是由两个不同的replica来达成一致并且进行本地提交。

View-change协议确保了非故障节点可以达成一致在不同的replica的不同view中的已经本地提交的消息。Request m本地提交在一个非故障节点上它的sequence number为n，view为v当committed(m,v,n)为正确的时候，这意味着存在一个R1的集合包含至少f+1个非故障节点的prepare(m,v,n,i)是正确的对于每一个replica来说。

非故障节点不会接受view大于当前view的pre-prepare消息，如果他没有接收到new-view消息的话。但是任意的正确的new-view message对于view v’ > v包含了正确的view-change message来自于replica i 在集合R2中有2f+1个replica。因为一个有3f+1个节点，R1和R2一定包含一个重复的节点是非故障的，K的视图更改消息将确保在前一个视图中准备好的事实传播到后续视图，除非new-view消息包含一个视图更改消息，该消息具有一个稳定的检查点，序列号大于m。在第一种情况下，算法将原子组播协议的三个阶段重做为相同的序列号和新的视图。这一点很重要，因为它可以防止任何在前一个视图中被分配了序列号的不同请求被提交。在第二种情况下，没有副本将接受序列号小于的任何消息。在这两种情况下，副本将同意使用序列号提交本地请求。

# 3.实现

复制库的客户端接口由一个过程invoke和一个参数组成，一个输入缓冲区包含调用状态机操作的请求。调用过程使用我们的协议在副本上执行所请求的操作，并从各个副本的应答中选择正确的应答。它返回一个指向包含操作结果的缓冲区的指针。

在服务器端，复制代码对应用程序的服务器部分必须实现的过程进行大量向上调用。有程序执行请求(execute)、维护服务状态的检查点(创建检查点、删除检查点)、获取指定检查点的摘要(get digest)、获取缺失信息(get checkpoint、set checkpoint)。执行过程接收一个包含所请求操作的缓冲区作为输入，执行该操作，并将结果放在输出缓冲区中。其他程序将在第6.3和6.4节中进一步讨论。查看操作结果。

节点间的点对点通信使用UDP实现，副本组播使用UDP通过IP组播实现。每个服务都有一个IP多播组，它包含所有的副本。这些通信协议是不可靠的;他们可能会复制或丢失消息，或者按顺序传递消息。

该算法允许无序交付，拒绝重复交付。视图更改可以用于从丢失的消息中恢复，但这代价很高，因此执行重传非常重要。在正常操作期间，从丢失的消息中恢复是由接收方驱动的:备份在消息过期时向主备份发送否定确认，主备份在长时间超时后重新发送预准备消息。对否定确认的回复可能包括稳定检查点的一部分和丢失的消息。在视图更改期间，副本会重新发送视图更改消息，直到它们收到匹配的新视图消息，或者转移到下一个视图。

复制库目前没有实现视图更改或重传。这并不影响第7节中给出的结果的准确性，因为算法的其余部分已经完全实现(包括对触发视图更改的计时器的操作)，而且我们已经形式化了完整的算法并证明了它的正确性。

我们使用复制库实现了BFS，这是一个拜占庭容错的NFS服务。图2显示了BFS的体系结构。我们选择不修改内核NFS客户机和服务器，因为我们没有数字Unix内核的源代码。

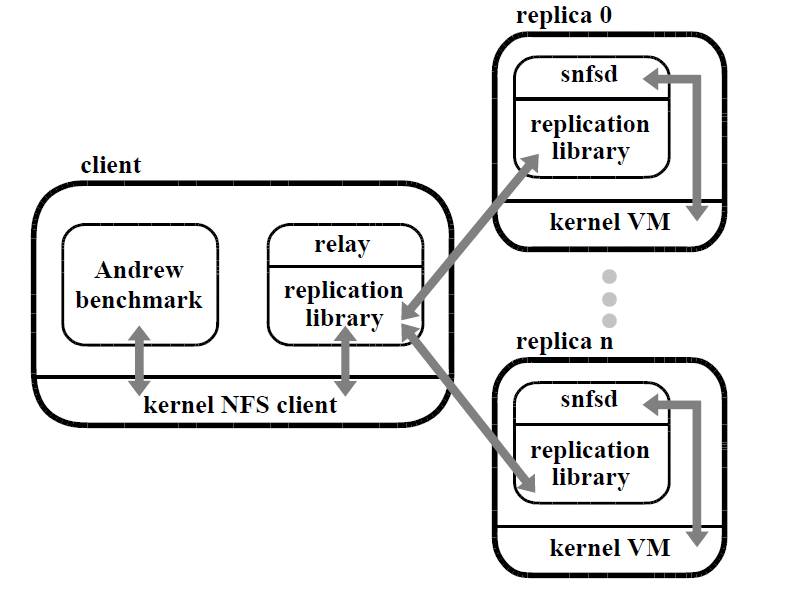


图2 Replicated File System Architecture

由容错NFSservice导出的文件系统像任何常规NFS文件系统一样挂载在客户机上。应用程序进程不加修改地运行，通过内核中的NFS客户端与挂载的文件系统交互。我们依靠用户级别的中继进程来协调标准NFS客户机和副本之间的通信。中继接收NFS协议请求，调用复制库的调用过程，并将结果发送回NFS客户机。

每个副本使用复制库和我们的NFS V2守护进程运行一个用户级进程，我们将其称为snfsd(对于简单的nfsd)。复制库接收来自中继的请求，通过发起向上调用与snfsd交互，并将NFS应答打包为它发送给中继的复制协议应答。

我们使用固定大小的内存映射文件来实现snfsd。所有的文件系统数据结构，例如索引节点、块和它们的空闲列表，都在映射的文件中。我们依靠操作系统来管理内存映射文件页面的缓存，并异步地将修改过的页面写入磁盘。当前的实现使用8KB的块，索引节点包含NFS状态信息和256字节的数据，这些数据用于存储目录中的目录项、文件中的指针块和符号链接中的文本。目录和文件也可以以类似于Unix的方式使用间接块。

我们的实现确保所有状态机副本都以相同的初始状态开始并且是确定性的，这是使用我们的协议实现的服务正确性的必要条件。主节点建议 time-last-modified 和 timelast-accessed 的值，副本选择建议值中的较大值，并且大于为早期请求选择的所有值的最大值。 我们不需要同步写入来实现 NFS V2 协议语义，因为 BFS 通过复制来实现修改数据和元数据的稳定性。

# 4.相关工作

一些协议和共识算法可以容忍异步系统中的拜占庭错误。但是，它们并没有为状态机复制提供完整的解决方案，而且它们中的大多数旨在证明理论上的可行性，并且太慢而无法在实践中使用。 我们在正常情况下运行的算法类似于拜占庭协议算法，但该算法无法在主要故障中幸存下来。

Rampart 和 SecureRing 都必须从组中排除故障副本以取得进展（例如，移除故障主节点并选择新主节点）并执行垃圾收集。 他们依靠故障检测器来确定哪些副本有故障。 但是，故障检测器在异步系统中并不准确，即它们可能会将副本错误分类为故障。 由于正确性要求少于 1 3 的组成员有错误，因此错误分类可能会通过从组中删除非错误副本来损害正确性。 这开辟了一条攻击途径：攻击者获得了对单个副本的控制权，但不会以任何可检测的方式改变其行为； 然后它会减慢正确的副本或它们之间的通信，直到从组中排除足够多的副本。

为了减少错误分类的可能性，可以校准故障检测器以延迟将副本分类为故障。 然而，为了使概率可以忽略，延迟必须非常大，这是不希望的。 例如，如果主节点实际上已经失败，则该组将无法处理客户端请求，直到延迟到期。 我们的算法不容易受到这个问题的影响，因为它永远不需要从组中排除副本。

# 5.结论

本文描述了一种新的状态机复制算法，该算法能够容忍拜占庭错误并且可以在实践中使用：它是第一个在像互联网这样的异步系统中正确工作的算法，并且它将以前算法的性能提高了超过一个数量级。

# 6.参考文献

[1]Miguel Castro,Barbara Liskov. Practical Byzantine fault tolerance[P]. Operating systems design and implementation,1999.