出现在1999年2月在美国新奥尔良举行的第三届操作系统设计和实施研讨会论文集中

实用的拜占庭容错

米格尔卡斯特罗和芭芭拉李斯科夫
麻省理工学院计算机科学实验室，
545 Technology Square, Cambridge, MA 02139
\卡斯特罗，里氏\@ lcs.mit.edu

摘要

本文描述了一种能够容忍拜占庭故障的新复制算法。 我们相信，拜占庭式容错算法在未来将变得越来越重要，因为恶意攻击和软件错误日益普遍，并可能导致错误的节点出现任意行为。 尽管以前的算法假定采用同步系统，或者实际运行速度太慢，但本文描述的算法非常实用：它可以在异步环境中工作，如Internet，并且包含几项重要的优化，可以改善以前算法的响应时间一个数量级。 我们使用我们的算法实施了拜占庭容错NFS服务并测量了其性能。 结果表明，我们的服务比标准的未复制的NFS慢3%。

1 介绍

恶意攻击和软件错误越来越普遍。 行业和政府越来越依赖在线信息服务，使得恶意攻击更具吸引力，并使得成功攻击的后果更为严重。 另外，由于软件的规模和复杂性的增加，软件错误的数量正在增加。由于恶意攻击和软件错误会导致有故障的节点出现拜占庭（即任意）行为，因此拜占庭容错算法变得越来越重要。

本文提出了一种新的实用的状态机复制算法[17,34]，可以容忍拜占庭故障。 该算法提供活力和安全性 $\lfloor \frac{n-1}{3} \rfloor$ 最多提供 n 共有的副本是同时发生故障。 这意味着客户最终会收到他们的请求回复，这些回复根据线性化是正确的[14,4]。 该算法适用于互联网等异步系统，它包含了重要的优化功能，使其能够高效地执行。

协议方面有很多工作要做

这项研究部分由DARPA根据合同DABT63-95-C-005支持，由陆军Fort Huachuca监测，合同F30602-98-1-0237由空军研究实验室监测，部分由NEC监测。 Miguel Castro得到了PRAXIS XXI奖学金的部分支持。

和容忍拜占庭故障的复制技术（从[19]开始）。 然而，大多数较早的工作（例如[3,24,10]）要么涉及旨在证明理论可行性的技术，这些技术太低效而不能用于实践，或假定为同步，即依赖于消息延迟和处理速度的已知界限。 与我们最接近的系统，Rampart [30]和SecureRing [16]，被设计成实用的，但它们依赖于同步假设的正确性，这在存在恶意攻击时是危险的。 攻击者可能通过延迟非故障节点或它们之间的通信来损害服务的安全性，直到它们被标记为有故障并从副本组中排除。 这种拒绝服务攻击通常比获得对无故障节点的控制更容易。

我们的算法不容易受到这种类型的攻击，因为它不依赖于同步的安全性。 此外，它可以将第7部分所述的Rampart和SecureRing的性能提高一个数量级以上。它只使用一个消息往返执行只读操作，而使用两个执行读写操作。 此外，它在正常操作期间使用基于消息认证码的高效认证方案；公钥加密技术被认为是Rampart的主要延迟[29]和吞吐量[22]瓶颈，只有在出现故障时才使用。

为了评估我们的方法，我们实现了一个复制库并用它来实现一个真正的服务：一个拜占庭容错的分布式文件系统，

端口NFS协议。 我们使用安德鲁板凳 -

标记[15]来评估我们系统的性能。 结果显示，在正常情况下，我们的系统比Digital Unix内核中的标准NFS守护程序慢3%。

因此，本文做出如下贡献：

- 它描述了第一个在异步网络中正确生存拜占庭故障的状态机复制协议。
- 它描述了许多重要的优化，可以使算法运行良好，从而可以在实际系统中使用。

- 它描述了拜占庭式容错分布式文件系统的实现。
- 它提供了量化复制技术成本的实验结果。

在本文的其余部分安排如下。 我们从描述我们的系统模型开始，包括我们的失败假设。 第3节描述了算法解决的问题并陈述了正确性条件。 该算法在第4节中描述，第5节中描述了一些重要的优化。第6节描述了我们的复制库以及我们如何使用它来实现拜占庭容错NFS。 第7节介绍了我们的实验结果。第8节讨论相关工作。 我们总结了我们所完成的内容以及对未来研究方向的讨论。

2 系统模型

我们假设一个异步分布式系统，其中节点通过网络连接。 网络可能无法传递消息，延迟它们，复制它们或将它们按顺序传递。

我们使用拜占庭式的失效模式，也就是说，错误的节点可以任意行为，只受下面提到的限制。 我们假设独立的节点故障。 对于这种假设在存在恶意攻击时是正确的，需要采取一些步骤，例如，每个节点应该运行不同的服务代码和操作系统实现，并且应该具有不同的根密码和不同的管理员。 从相同的代码库[28]获得不同的实现是可能的，并且对于低复制度，可以购买来自不同供应商的操作系统。 N版本编程，即不同的程序员团队产生不同的实现，是另一种服务的选择。

我们使用加密技术来防止欺骗和重播，并检测损坏的消息。 我们的消息包含公钥签名[33]，消息认证码[36]以及由抗碰撞哈希函数[32]产生的消息摘要。 我们表示一条消息 由节点标记为\和消息的摘要 通过\。 我们遵循通常的做法，即对消息进行摘要签名，并将其附加到消息的纯文本中，而不是签署完整的消息（\应该以这种方式解释）。 所有副本都知道其他人的公钥来验证签名。 我们允许一个非常强大的攻击者协调故障节点，延迟通信或延迟正确的节点，以便对复制服务造成最大的损害。 我们确实假设对手不能无限期延迟正确的节点。 我们也假设

对手（以及它控制的错误节点）

在计算上是受限制的，因此（很有可能）它不能颠覆上面提到的密码技术。 例如，攻击者不能产生无故障节点的有效签名，通过摘要中的摘要计算摘要信息，或者找到两个具有相同摘要的消息。 我们使用的密码技术被认为具有这些属性[33, 36, 32]。

3 服务属性

我们的算法可以用来实现具有状态和一些操作的任何确定性复制服务。 这些操作不限于简单的读取或写入服务状态的部分； 他们可以使用状态和操作参数执行任意确定性计算。 客户向复制服务发出请求以调用操作并阻止等待答复。 复制服务由副本实现。如果客户和副本遵循第4节中的算法，并且没有攻击者可以伪造他们的签名，那么客户端和副本是没有问题的。

假设不超过\副本错误，算法提供安全性和活跃性。安全意味着复制服务满足线性化[14]（修改为拜占庭故障客户端[4]）：它表现得像一个集中式实现，它一次一个原子地执行操作。 安全性要求限制故障复制品的数量，因为故障复制品可以任意行为，例如可以破坏其状态。 无论有多少有故障的客户端使用该服务（即使他们与错误的副本混合），都可以提供安全性：由无故障的客户端以一致的方式观察由故障客户端执行的所有操作。 特别是，如果服务操作旨在保留服务状态上的一些不变量，则会发生故障客户不能破坏这些不变量。

安全属性不足以防范故障客户端，例如，在文件系统中，故障客户端可以将垃圾数据写入某个共享文件。 但是，我们通过提供访问控制来限制有故障的客户端可以执行的损害的数量：我们对客户端进行身份验证，如果发出请求的客户端无权调用操作，则拒绝访问。 此外，服务可能会提供操作来更改客户端的访问权限。 由于该算法确保所有客户端始终如一地观察到访问撤销操作的影响，因此这为强大的机制从故障客户端的攻击中恢复提供了强大的机制。

该算法不依赖于同步来提供安全性。 因此，它必须依靠同步来提供活力； 否则它可以用于在异步系统中实现共识，这是不可能的[9]。 我们保证活跃，即客户最终收到他们的请求的答复，提供在最 \副本有故障并且延迟 才不是

比无限增长更快t 这里，|是消息首次发送的时刻到目的地接收消息的时刻t（假设发送者不断重发消息直到它被接收）。 （更精确的定义可以在文献[4]中找到。）这是一个相当弱的同步假设，在任何实际系统中，假设网络故障最终都会被修复，这种假设很可能是真实的，但它使我们能够规避[9]。

我们的算法的弹性是最优的：3 1是允许异步系统在最多\副本出现故障时提供安全性和活性属性的最小\副本数（参见[2]证明））。 需要这么多副本，因为\副本与\副本进行通信后必须可以继续进行，因为\副本可能有问题而且没有响应。 但是，没有响应的\副本可能没有错误，因此\响应的副本可能有问题。 即便如此，仍然必须有足够的答复，即来自非错误复制品的错误数量超过错误错误数量的错误数量，即\。 因此\。

该算法没有解决容错隐私问题：错误的副本可能会泄漏信息给攻击者。 在一般情况下提供容错隐私是不可行的，因为服务操作可以使用它们的参数和服务状态执行任意计算； 复制品需要明确这些信息来有效地执行这些操作。 即使存在恶意副本的阈值[13]，对于服务操作不透明的参数和状态部分，也可以使用秘密共享方案[35]来获得隐私。 我们计划在未来调查这些技术。

4 算法

我们的算法是一种状态机复制的形式[17,34]：该服务被建模为一个状态机，该状态机在分布式系统中的不同节点上复制。 每个状态机副本维护服务状态并实现服务操作。 我们通过使用0中的整数表示每个副本的一组副本 1。 为了简单起见，我们假设 $\{ ,...,|\mathcal{R}| - \}$ $|\mathcal{R}| = 3f 1$其中是可能有故障的最大副本数量； 尽管可能有超过31个副本，但额外的副本降低了性能（因为更多和更大的消息正在交换中）而没有提供改进的弹性。

副本将通过一系列称为视图的配置进行移动。 在一个视图中，一个副本是主要的，其他副本是备份。 视图连续编号。 视图的主要部分是副本 这样$p = v \bmod |\mathcal{R}|$，在哪里 是视图编号。 当主显示器出现故障时，将执行查看更改。 已打印的复制[26]和Paxos [18]使用类似的方法来容忍良性缺陷（如第8节所讨论的）

该算法大致如下工作：

1. 客户端向主服务器发送调用服务操作的请求
2. 主要组播请求到备份
3. 副本执行请求并将回复发送给客户端
4. 客户端等待来自不同副本的\1回复，结果相同；这是该操作的结果。

像所有状态机复制技术[34]一样，我们对副本施加两个要求：它们必须是确定性的（即，在给定状态下执行操作并且给定一组参数必须始终产生相同的结果），并且它们必须以相同的状态开始。 考虑到这两个要求，该算法通过保证所有无故障复制品在尽管失败的情况下同意执行请求的总订单来确保安全属性。

本节的其余部分介绍该算法的简化版本。 我们忽略了节点由于空间不足而从故障中恢复的讨论。 我们也忽略了与消息重传相关的细节。 此外，我们假设消息认证是使用数字签名来实现的，而不是基于消息认证码的更有效的方案； 第5节进一步讨论了这个问题。 文献[4]给出了使用I / O自动机模型[21]的算法的详细形式化。

4.1 客户端

客户通过向主节点发送\消息来请求执行状态机操作。时间戳用于确保执行客户端请求的唯一语义。 请求的时间戳是完全有序的，以便后面的请求具有比以前更高的时间戳； 例如，时间戳可以是发出请求时客户端本地时钟的值。

副本发送给客户端的每条消息都包含当前视图编号，允许客户端跟踪视图，从而跟踪当前主要视图。客户端使用点对点消息向其认为是当前主服务器的请求发送请求。 主要使用下一节中描述的协议将请求自动多播到所有备份。

副本将答复直接发送到客户端。 答复的形式为REPLY\，其中是当前视图编号，是相应请求的时间戳，是副本编号，是执行请求的操作的结果。

客户端等待来自不同副本的有效签名的1个回复，以及之前和之后的回复

接受结果。 这可以确保结果是有效的，因为最多\副本可能有问题。

如果客户端没有及时收到回复，它会将请求广播到所有副本。 如果请求已被处理，副本只需重新发送回复； 副本记住他们发送给每个客户端的最后回复消息。 否则，如果副本不是主节点，它会将请求转发给主节点。 如果主服务器不向组播组发送请求，则最终会被足够的副本引起视图更改而怀疑是有问题的。

在本文中，我们假设客户端在发送下一个请求之前等待一个请求完成。 但是，我们可以允许客户端发出异步请求，但仍然保留对它们的排序约束。

4.2 正常情况下的操作

每个副本的状态包括服务的状态，包含副本已接受的消息的消息日志以及表示副本当前视图的整数。 我们在4.3节描述如何截断日志。 当主服务器收到客户端请求时，m，它启动一个三阶段协议以原子方式将请求多播到副本。 主协议立即启动协议，除非协议正在进行的消息数量超过给定的最大值。 在这种情况下，它会缓冲请求。 缓存的请求稍后作为一个组进行多点传送，以在负载较重的情况下减少消息流量和CPU开销；这种优化类似于事务性系统中的组提交[11]。 为了简单起见，我们在下面的描述中忽略了这种优化。

三个阶段是预先准备，准备和提交。 即使提出请求排序的主服务器出现故障，预准备和准备阶段也可用于完全排列以相同视图发送的请求。 准备和提交阶段用于确保提交的请求在视图中完全排序。 在准备阶段，主要分配一个序列号，n，根据请求，多播预备准备消息$\langle\langle PRE\text{-}PREPARE, v, n, d\rangle_{\sigma_p}, m\rangle$搭载所有备份，并将消息追加到其日志中。 该消息具有格式\PRE-PREPARE\，m中v指示消息正在发送的视图，n是个客户的请求消息，和d is m摘要。

预先准备的消息中不包含请求，以减少这些请求。这很重要，因为预先准备好的消息被用作证明请求在视图更改中被分配了序列号。 此外，它将协议解耦以完全从协议中订购请求，以将请求传输到副本；使我们能够针对小型消息使用针对协议消息进行优化的传输，并针对大型请求针对大型消息优化传输。

- 备份接受提供的预先准备消息：请求中的签名和预准备消息是正确的d是的摘要m；
- 它在视野之中v；
- 它没有接受预先准备的信息来查看和序列号n包含不同的摘要；预准备消息中的序列号在低水位
- 标记和高水位标记之间，h H.

最后一种情况是通过选择非常大的一个来防止有故障的小学生耗尽序列号的空间。 我们将在4.3节讨论如何和推进。H h

如果备份接受\PRE-PREPARE\消息，它将通过将\PREPARE\消息多播到所有其他副本并将两个消息添加到其日志中来进入准备阶段。 否则，它什么都不做。 副本（包括主副本）接受准备消息，并将它们添加到其日志中，前提是它们的签名是正确的，它们的视图编号等于副本的当前视图，并且其序列号是之间h和H.

我们将准备好的谓词P定义为true，当且仅当在其日志中插入副本时：请求m，一个预先准备m在视图中v与序列号n，和\从与预先准备相匹配的不同备份准备。 副本通过检查它们是否具有相同的视图，序列号和摘要来验证准备是否与预先准备相匹配。

算法的预先准备和准备阶段保证非故障复制品在视图内的请求的总订单上达成一致。 更确切地说，它们确保以下不变：如果准备好P，则准备好(m', v, n, j)对于任何非故障复制品$i = j$\（包括）以及任何这样的复制品都是错误的$m' \neq m$$D(m') \neq D(m)$。 这是事实，因为准备好了$(m, v, n, i)$和$|\mathcal{R}| = 3f + 1$意味着至少有1个非故障的复制品已经发送了预先准备或者准备鉴于序列号。n因此，对于准备好的P而言，这些副本中至少有一个需要发送两个相互冲突的准备（如果它是主要准备，则需要预先准备），即两个准备具有相同的视图和序列号以及不同的准备消化。 但这是不可能的，因为副本没有错误。 最后，我们关于消息摘要强度的假设确保了\和\的可能性可以忽略不计。

副本多播\COMMIT$\langle\langle COMMIT, v, n, D(m)\rangle_{\sigma_i}, m'\rangle$准备好后再复制到其他副本$(m, v, n, i)$成为现实。 这开始提交阶段。 副本接受提交消息并将它们插入日志中，前提是它们的签名正确，消息中的视图编号等于副本的当前视图，序列号介于h H

我们定义承诺和承诺本地谓词如下：|当且仅当准备好 |在所有 i $f+1$个无故障复制品；并承诺本地 (m,v,n,i) 当且仅当准备好 |为真并已接受\1从与预先准备相匹配的不同复制品提交（可能包括它自己的）时才是真实的； 如果一个提交具有相同的视图，序列号和摘要，则提交与预先准备相匹配。

提交阶段确保以下不变：如果提交本地 (m,v,n,i) 对于一些没有错误然后提交的情况是成立的 i (m,v,n) 是真的。 第4.4节中介绍的这种不变式和视图更改协议确保无故障副本同意本地提交的请求的序列号，即使它们在每个副本的不同视图中提交。 此外，它确保任何在没有错误的副本上进行本地提交的请求都会提交 1个 $f+$ 或多个无故障的复制品最终。

每个副本 i 执行本地提交后所请求的操作 |为 m 真，并且其状态反映序列号较低的所有请求的顺序执行。 i 这可确保所有无故障的副本按照提供安全属性所需的相同顺序执行请求。 执行请求的操作后，副本将向客户端发送回复。 副本放弃时间戳低于发送给客户端的最后一个回复中的时间戳的 请求确保一次语义。

我们不依赖有序的消息传递，因此副本可能无序地提交请求。 这并不重要，因为它保持了预先准备，准备和提交记录的消息，直到相应的请求可以被执行。

图1显示了算法在没有主要故障的正常情况下的操作。 副本0是主节点，副本3有问题，并且是客户端。 C

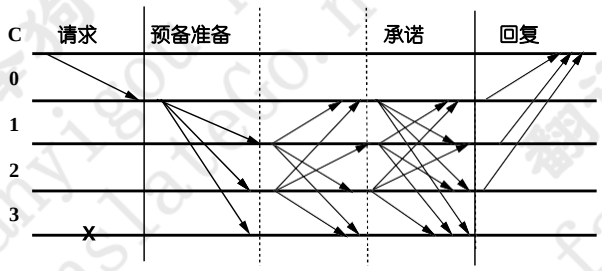

图1：正常情况下的操作

4.3 垃圾收集

本节讨论用于丢弃日志中的消息的机制。 为了保持安全条件，消息必须保存在副本日志中，直到它知道为止他们所关注的请求至少已经由\1个无故障副本执行，并且可以在视图更改中向其他人证明。 另外，如果某些复制品遗漏了所有无故障复制品丢弃的消息，则需要通过传输全部或部分服务状态来更新它。 因此，副本还需要一些证明状态是正确的。

执行每个操作后生成这些证明将是昂贵的。 相反，它们是周期性生成的，当一个序列号可以被某个常数（例如100）整除的请求被执行时。 我们将把执行这些请求所产生的状态称为检查点，并且我们会说具有证明的检查点是一个稳定的检查点。

副本维护着服务状态的多个逻辑副本：最后一个稳定检查点，零个或多个不稳定的检查点以及当前状态。 写入时复制技术可以用来减少空间开销来存储状态的额外副本，如6.3节所述。

如下生成检查点的正确性证明。 当一个副本 i 产生一个检查点，它将一个消息 n \CHECKPOINT\多播到其他副本，其中 是执行被反映在状态中的最后一个请求的序号 是国家的摘要。 d 每个副本在其日志中收集检查点消息，直到它具有\1个序列号 使用由不同副本签备的相同摘要（可能包括其自己的）此类消息）。 这2个 d $2f+1$条消息是检查点正确性的证明。

带有证明的检查点变得稳定，并且副本丢弃所有预先准备，准备和提交序号小于或等于其日志的消息；它也丢弃了所有早期的检查点和检查点消息。 n

计算证明是有效的，因为摘要可以使用第6.3节中讨论的增量密码术[1]来计算，而证明很少生成。 检查点协议用于提高低水位和高水位（限制接受什么信息）。 低水位标志 等于上一个稳定检查点的序列号。 高水位 ，在哪里 足够大以便复制品不会等待检查点变稳定。 例如，如果检查点每隔100分钟进行一次要求， 可能是200。 $H=h+k$ h k

4.4 查看更改

视图改变协议通过允许系统在主要失败时取得进展而提供活跃性。 视图更改由超时触发，以防止备份无限期地等待执行请求。 备份在收到有效请求时正在等待请求

并没有执行它。 备份在接收到请求并且计时器尚未运行时启动计时器。 当它不再等待执行请求时，它停止计时器，但是如果在那时它正在等待执行一些其他请求，则重新启动计时器。

如果备份的计时器 i 在期限内到期 v，备份会启动视图更改以将系统移动到视图 1.它停止接受消息（检查点，视图更改和新视图消息除外）并多播一个\VIEW-CHANGE $v+$ 1 $,n,\mathcal{C},\mathcal{P},i)_{\sigma_i}$ 消息给所有副本。 这里 n 是已知的最后一个稳定检查点的序列号， s i $\mathcal{C}$ 是一组2 $f+$ 1证明正确性的有效检查点消息，并且是包含每个请求的 $\mathcal{P}$ 集合/集合的集合 准备的序列号高于。每个集合/包含一个有效的预先准备消息（没有相应的客户端消息）和\匹配，由不同备份签署的有效准备消息与相同的视图，序列号和摘要 $m.$

当视图1的主视图从其他副本接收到\视图1的有效视图改变消息时，它多播一个\NEW-VIEW 1 $,v+$ $\mathcal{V},\mathcal{O})_{\sigma_p}$ 消息发送给所有其他副本，其中包含由主节点收到的有效查看更改消息的集合，以及发送（或将发送）的主节点的查看更改消息，并且 $\mathcal{O}$ 是一组预先准备好的消息捎带的请求）。 $\mathcal{O}$ 计算如下：

1. 主服务器确定最近一个稳定检查点的序列号min-s和最近一个准备消息中的最高序列号max-s。 $\mathcal{V}$
2. 主要创建一个新的预先准备的消息供查看 $v+$ 1，用于min-s和max-s之间的每个序列号。 有两种情况：（1）中至少有一套 n 组件中的某些视图更改消息 $\mathcal{P}$ 与序号，或（2）没有这样的设置。 在第一种情况下，主要创建一条新消息\PRE-PREPARE $\mathcal{V}$ m \，其中是 $,v+$ d 序列号的预备准备消息中的请求摘要 最高的数字在英寸 . 在第二种情况下，它会创建一个新的预备准备消息\PRE-PREPARE $v+$ \，其中/是特殊空请求的摘要；一个空请求像其他请求一样通过协议，但是它的执行是无操作的。 （Paxos [18] 用类似的技术填补空白。）

接下来，主要将消息附加到其日志中。 如果min-s大于其最新稳定检查点的序列号，则主节点还会在日志中为序号为min-s的检查点插入稳定性证明，并放弃4.3节中讨论的来自日志的信息。 然后进入视图1：此时它能够接受视图1的消息。

如果视图1已正确签名，并且视图更改消息，则备份接受视图1的新视图消息 $v+$ $v+$

$v+$ $\mathcal{O}$ $\mathcal{O}$ 包含对于视图1有效，并且如果该集合是正确的；它通过执行与主要创建的计算相似的计算来验证其正确性。 然后它将新信息添加到它的日志中（如主节点所述），为每封邮件准备好所有其他副本，将这些准备添加到其日志中，然后进入视图1。 $\mathcal{O}$ i v

此后，该协议按4.2节所述进行。 副本重做min-s和max-s之间的消息协议，但它们避免重新执行客户端请求（通过使用它们存储的关于发送给每个客户端的最后一个应答的信息）。

副本可能会丢失一些请求消息 m 或者一个稳定的检查点（因为这些检查点不是以newview消息发送的。）它可以从另一个副本获得缺少的信息。 例如，副本可以从检查点消息验证其正确性的其中一个副本获取缺少的检查点状态 i s $\mathcal{V}$。 由于其中的一个副本是正确的，所以副本将始终获得或稍后获得认证的稳定检查点。 我们可以通过划分状态来避免发送整个检查点，并用修改它的最后一个请求的序号来标记每个分区。 为了使副本保持最新状态，只需要向其发送已过期的分区，而不是整个检查点。

4.5 正确性

本节描绘算法提供安全性和活性的证据；细节可以在[4]中找到。

4.5.1 安全

如前所述，如果所有非故障副本都同意本地提交的请求的序列号，算法将提供安全性。

在4.2节中，我们表明如果准备好 \，准备好了 (m',v,n,j) 对于任何非故障复制品（包括 j $i = j$ $\neq D(m)$ m ）和任何这样的 \。 这意味着两个无故障的副本同意在两个副本的同一视图中本地提交的请求的序列号。

视图更改协议确保非故障副本也同意在不同副本的不同视图中本地提交的请求的序列号。 只有在提交 \为真的情况下，请求才会在无故障的副本中进行本地提交， m 并在视图中显示序列号。 这意味着有一个集合 n v 至少包含 \1个无故障复制品 R，以便为集合中的每个复制品准备好 \。

无故障复制品将不会接受预先准备查看 i 没有收到新的消息 v' m' （因为只有在那时他们才进入视图）。但是任何正确的新观点的消息 包含来自每个副本的正确的视图更改消息 $v' > v$ i

设置R为2 $f+1$个副本。 由于有3个 $f+1$副本，1 R和 R必须至少交叉一个副本 k这不是 错误的。 k 的观点改变消息将确保这一事实 除非 新视图消息包含具有高于序列号的稳定检查点的视 图更改消息，否则将在先前视图中准备的视图传播 到后续视图。 在第一种情况下，该算法重新使用原 子多播协议的三个阶段 n 具有相同 的序列号 m 和新的视图号码。 这很重要，因为它可 以防止分配了序列号的任何不同请求 从以前 的观点来看。 在第二种情况下，新视图中的副本将 不接受序列号低于的任何消息。 无论哪种情况，副 本都会同意提交的请求 n 本地与序列号。 n

4.5.2 活跃度

为了提供活力，如果副本无法执行请求，则必须移动 到新视图。 但重要的是要使至少\1个无故障副本处于 相同视图中的时间最大化，并确保这段时间以指数方 式增加，直到某个请求的操作执行为止。 我们通过三 种方式实现这些目标。

首先，为了避免过早地启动视图更改，为视图1多 播了视图更改消息的副本 $v+$ 等待\1个视图更改消息以供查看 $v+1$ 然后启动它的计时器在一段时间后过期 。 如果定 T 时器在收到有效的新视图消息之前到期 1，或者在 它在新视图中执行之前未执行过的请求之前，它会 开始视图更改 2，但是这次它会在开始视图更改视 图之前等待2 3. $v+$ T

其次，如果一个副本从其他副本接收一组有效视 $v+$ 图更改消息，以查看大于其当前视图的视图，则即 使其定时器未到期，它也会为该集中的最小视图发 送视图更改消息； 这可以防止它太晚开始下一个视 图更改。

第三，错误的副本无法通过强制频繁查看更改来 阻止进度。 错误的副本不能通过 发送视图更改消息来导致视图更改，因为仅当至少 \1副本发送视图更改消息时才会发生视图更改，但 是如果视图更改为主（通过不发送消息或发送错误 消息）。 但是，因为主要的观点 是复制品 这 样mod ，v 主要不能有更多的错误 比一连续的替换。 p

这三种技术可以保证活跃性，除非消息延迟的增 长速度超过无限期的超时时间，这在实际系统中是 不太可能的。

4.6非决定论

状态机副本必须是确定性的，但许多服务涉及某种 形式的非确定性。 例如，通过读取服务器的本地时 钟来设置NFS中的最后修改时间； 如果这是在每个副 本独立完成的，那么非故障副本的状态将会发生分 歧。 因此，需要一些机制来确保所有副本选择相同 的值。 通常，客户端无法选择该值，因为它没有足 够的信息； 例如，它不知道它的请求将如何相对于 其他客户端的并发请求进行排序。 相反，主要需要 独立或基于备份提供的值选择值。

如果主要独立地选择非确定性值，则它将该值与 相关联的请求连接起来，并执行三阶段协议，以确 保无故障复制品就请求和值的序列号达成一致。 这 可以防止发生故障的主节点通过向不同副本发送不 同值来导致副本状态发散。 但是，有缺陷的主节点 可能会向所有副本发送相同的，不正确的值。 因此， 副本必须能够确定性地确定该值是否正确（以及如 果不是这样做的话），仅基于服务状态。

该协议适用于大多数服务（包括NFS），但偶尔副 本必须参与选择值以满足服务规范。 这可以通过向 协议添加一个额外的阶段来完成：主要获得由备份 建议的认证值，将它们连接\1与关联的请求，并启 动连接消息的三阶段协议。 副本通过对\1值及其状 态的确定性计算来选择该值，例如取中值。 常见情 况下，额外阶段可以优化。 例如，如果副本需要一 个与本地时钟“足够接近”的值，那么当它们的时 钟在某个增量内同步时，可以避免额外的相位。

5 优化

本节介绍一些可在正常情况下提高算法性能的优化。 所有的优化都保留了生命力和安全性。

5.1 减少沟通

我们使用三种优化来降低通信成本。 第一个避免发送 最大的回复。 客户端请求指定副本以发送结果； 所有 其他副本都会发送仅包含结果摘要的回复。 摘要允许 客户在减少网络的同时检查结果的正确性

带宽消耗和CPU开销显着为大量答复。 如果客户端没有收到来自指定副本的正确结果，它将像往常一样重新发送请求，请求所有副本发送完整的答复。

第二次优化将操作调用的消息延迟的数量从5减少到4.只要准备好的谓词对请求持有，副本就暂时执行请求，它们的状态反映了所有请求的序列号较低的执行，并且这些请求都是已知的承诺。 执行请求后，副本将暂时的回复发送给客户端。 客户等待2\1条配对暂定答复。 如果它收到了这么多，这个请求最终会保证提交。 否则，客户端重新发送请求并等待1个非临时回复。

如果存在视图更改并且已被空请求替换，那么暂时执行的请求可以中止。 在这种情况下，副本将其状态恢复到新视图消息中的最后一个稳定检查点或其最后一个检查点状态（取决于哪个具有更高的序列号）。

第三个优化改进了不修改服务状态的只读操作的性能。 客户端向所有副本发送只读请求。 在检查请求被正确验证，客户端有权访问并且请求实际上是只读的之后，副本立即以它们的试验性状态执行请求。 只有在暂定状态中反映的所有请求都已发生后，他们才会发送回复；这是防止客户端观察未提交状态所必需的。 客户端等待来自不同副本的2 1个回复，结果相同。 如果并发写入影响结果的数据，客户端可能无法收集2 1个此类回复；在这种情况下，在其重新传输定时器到期之后，它将该请求重新发送为常规读写请求。

5.2 加密

在第4节中，我们描述了一种使用数字签名来验证所有消息的算法。 但是，我们实际上只使用数字签名来处理视图更改和新视图消息（这些消息很少发送），并使用消息认证代码（MAC）对所有其他消息进行身份验证。 这消除了以前系统中的主要性能瓶颈[29,22]。

但是，MAC相对于数字签名具有根本的限制 - 无法证明消息对第三方是真实的。 第4节中的算法和以前的拜占庭容错算法[31,16]用于状态机复制依赖于数字签名的额外功能。 我们修改了我们的算法，以利用它来避开这个问题特定的不变量，例如，两个不同的请求在两个非故障复制品上用相同的视图和序列号准备的不变量。修改的算法在[5]中描述。 这里我们简要介绍一下使用MAC的主要含义。

MAC可以比数字签名快三个数量级来计算。 例如，一个200MHz的Pentium Pro需要43ms来产生一个MD5摘要的1024位模RSA签名和0.6ms来验证签名[37]，而计算一个64字节消息的MAC只需要10.3s我们的实现中使用相同的硬件。 还有其他的公钥加密系统可以更快地生成签名，例如椭圆曲线公钥密码系统，但签名验证速度较慢[37]，在我们的算法中，每个签名都经过多次验证。

每个节点（包括活动客户端）与每个副本共享一个16字节的秘密会话密钥。 我们通过将MD5应用于与密钥串联的消息来计算消息认证码。 我们只使用10个最不重要的字节，而不是使用最终MD5摘要的16个字节。 这种截断具有减小\MAC大小的明显优势，并且还提高了它们对某些攻击的适应能力[27]。 这是秘密后缀方法[36]的变体，只要MD5是抗碰撞[27,8]，它就是安全的。

应答消息中的数字签名被单个MAC所取代，这是足够的，因为这些消息具有单个目标接收者。 所有其他消息中的签名（包括客户端请求，但不包括视图更改）都由我们称为验证者的MAC向量替代。 认证者对发送者以外的每个副本都有条目；每个条目是使用由发送者共享的密钥和对应于该条目的副本计算的MAC。

验证验证器的时间是恒定的，但生成验证器的时间随副本数量线性增长。 这不是问题，因为我们预计不会有大量副本，MAC和数字签名计算之间存在巨大的性能差距。 此外，我们有效地计算验证器：将消息MD5应用于该消息一次，并且通过将MD5应用于相应的会话密钥来将所得到的上下文用于计算每个向量条目。 例如，在一个有37个副本的系统中（即一个系统可以容忍12个同时发生的故障），验证器仍然可以比1024位模数RSA签名计算快两个数量级以上。 认证器的大小随副本的数量呈线性增长，但增长缓慢：等于$30 \times \lfloor \frac{n-1}{3} \rfloor$字节。 一个认证者比一个小RSA签名的1024位模数为13（即可容忍多达4个同时发生的故障的系统），我们预计在大多数配置中都是如此。

6 履行

本节介绍我们的实施。 首先我们讨论复制库，它可以用作任何复制服务的基础。 在第6.2节中，我们描述了如何在复制库之上实现复制的NFS。 然后我们描述我们如何维护检查点并有效地计算检查点摘要。

6.1 复制库

复制库的客户端接口由一个过程组成，它调用一个参数，输入缓冲区包含一个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请求的操作，并从各个副本的答复中选择正确的答复。 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，复制代码会对应用程序的服务器部分必须实现的过程进行一些上调。 有一些程序可以执行请求（执行），维护服务状态的检查点（使检查点，删除检查点），获取指定检查点的摘要（获取摘要）以及获取缺失信息（获取检查点，设置检查点） 。 执行过程接收包含请求操作的缓冲区作为输入，执行操作并将结果放入输出缓冲区。 其他程序将在6.3节和6.4节中进一步讨论。

节点之间的点对点通信使用UDP实现，并且使用基于IP多播的UDP组播实现到组副本[7]。 每个服务都有一个IP多播组，其中包含所有副本。 这些通信协议是不可靠的：他们可能会复制或丢失信息或将它们乱序发送。

该算法容忍无序传递并拒绝重复。 视图更改可用于从丢失的消息中恢复，但这很昂贵，因此执行重新传输很重要。 在正常操作期间，丢失消息的恢复由接收方驱动：当主服务器过期并且主服务器在长时间超时后重新发送预准备消息时，备份会向主服务器发送否定应答。 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。 在查看更改期间，副本将重新发送视图更改消息，直到他们收到匹配的新视图消息或者他们转到稍后的视图。

复制库目前不实现视图更改或重新传输。 这不会影响第7节给出结果的准确性，因为算法的其余部分是完全实现（包括操纵触发视图变化的定时器），并且因为我们已经形式化了完整的算法并证明了它的正确性[4]。

6.2 BFS：拜占庭容错文件系统

我们使用复制库实现了BFS，一种拜占庭容错的NFS服务。 图2显示了BFS的体系结构。 我们选择不修改内核NFS客户端和服务器，因为我们没有Digital Unix内核的源代码。

由容错NFS服务导出的文件系统像任何常规NFS文件系统一样安装在客户机上。 应用程序进程未经修改运行，并通过内核中的NFS客户端与已挂载的文件系统进行交互。 我们依靠用户级中继流程来调解标准NFS客户端和副本之间的通信。 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客户端。

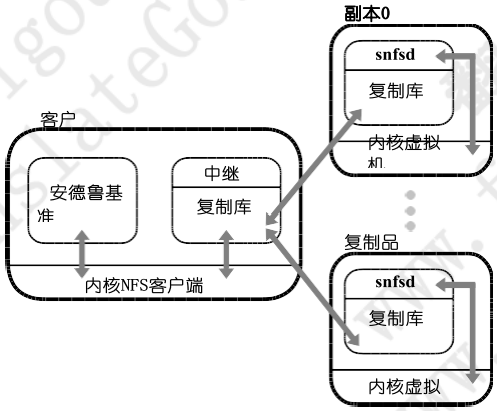

图2：复制文件系统体系结构。

每个副本都使用复制库和我们的NFS V2守护进程运行用户级进程，我们将其称为snfsd（用于简单的nfsd）。 复制库接收来自中继的请求，通过拨打电话与snfsd交互，并将NFS回复打包为发送给中继的复制协议回复。

我们使用固定大小的内存映射文件实现了snfsd。所有文件系统数据结构（例如inode，块及其空闲列表）都位于映射文件中。 我们依靠操作系统来管理内存映射文件页面的缓存，并将修改后的页面异步写入磁盘。当前实现使用8KB块，inode包含NFS状态信息和256字节的数据，用于存储目录中的目录条目，指向文件中的块的指针以及符号链接中的文本。 目录和文件也可能以类似于Unix的方式使用间接块。

我们的实现确保了所有状态机

副本从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的正确性的必要条件。主要建议时间最后修改和时间最后访问的值，副本选择较大的建议值，一个大于为先前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通过复制实现了修改数据和元数据的稳定性[20]。

6.3 维护检查点

本节介绍snfsd如何维护文件系统状态的检查点。回想一下，每个副本都维护着该状态的多个逻辑副本：当前状态，一些尚未稳定的检查点数以及最后一个稳定检查点。

*s*nfsd直接在内存映射文件中执行文件系统操作以保留局部性，并且它使用写时复制来减少与维护检查点相关的空间和时间开销。 snfsd为内存映射文件中的每个512字节块保留一个写时复制位。 复制代码调用make check check upcall时，snfsd设置所有写时复制位并创建一个（易失性）检查点记录，其中包含当前序列号，它作为upcall的参数接收，以及一个块列表。 该列表包含自检查点获取后修改的块的副本，因此它最初是空的。 该记录还包含当前状态的摘要；我们讨论如何在6.4节中计算摘要。

当执行客户请求时修改了内存映射文件的一个块时，snfsd检查该块的写时复制位，如果已设置，则将该块的当前内容及其标识符存储在最后一个检查点记录中检查点。 然后，它用它的新值覆盖该块并重置它的写入时复制位。 snfsd会保留一个检查点记录，直到通过删除检查点upcall来通知检测点记录为止，该检查点由复制代码在稍后的检查点变稳定时生成。

如果复制代码需要检查点发送到另一个副本，它会调用get checkpoint upcall。 要获取块的值，snfsd首先在稳定检查点的检查点记录中搜索块，然后搜索任何稍后检查点的检查点记录。 如果该块不在任何检查点记录中，它将返回当前状态的值。

使用写入时复制技术以及我们至多保留2个检查点的事实确保了保留多个逻辑副本的空间和时间开销较低。 例如，在第7节中介绍的Andrew基准测试中，平均检查点记录大小仅为182个块

最多500个。

6.4 计算检查点摘要

*s*nfsd计算检查点状态的摘要，作为make check check upcall的一部分。 虽然检查点只是偶尔采用，但由于状态可能很大，因此逐步计算状态摘要很重要。 snfsd使用一个称为AdHash [1]的增量式抗碰撞单向散列函数。 该函数将状态划分为固定大小的块，并使用其他一些散列函数（例如MD5）来计算通过连接块索引与每个块的块值获得的字符串的摘要。状态的摘要是块的摘要以某个大整数为模的总和。在我们当前的实现中，我们使用写入时复制技术中的512字节块并使用MD5计算摘要。

为了逐渐计算状态的摘要，snfsd为每个512字节的块维护一个包含散列值的表。 该散列值是通过将MD5应用于与上一个检查点时的块值连接的块索引而获得的。 当调用make checkpoint时，snfsd将获取前一个检查点状态的摘要（来自关联的检查点记录）。 它通过将MD5应用于与当前块值并置的块索引来为其写入时复制位被重置的每个块计算新的散列值。 然后，它添加新的散列值，从中减去旧的散列值，并更新该表以包含新的散列值。 这个过程是有效的，只要修改块的数量很少；如上所述，dAndrew基准平均每个检查点修改182d个块。

7 绩效评估

本节使用两个基准评估我们系统的性能：微基准和Andrew基准[15]。 微基准提供了复制库性能的服务无关评估；它测量调用空操作的延迟，即不执行任何操作的操作。

Andrew基准测试用于比较BFS和其他两种文件系统：一种是Digital Unix中的NFS V2实现，另一种与没有复制的BFS相同。 第一个比较表明，我们的系统通过显示其延迟类似于许多用户每天使用的商业系统的延迟而是实用的。 第二个比较允许我们在实际服务的实现中准确评估算法的开销。

7.1 实验装置

实验测量正常情况下的行为（即没有视图变化），因为这是行为

这决定了系统的性能。 所有的实验都运行一个客户端运行两个中继进程和四个副本。 四个副本可以容忍一个拜占庭故障; 我们预计这个可靠性水平足以满足大多数应用。 副本和客户端在相同的DEC 3000/400 Alpha工作站上运行。 这些工作站有一个133 MHz的Alpha 21064处理器，128 MB的内存，并运行Digital Unix版本4.0。 文件系统由每个副本存储在DEC RZ26磁盘上。 所有工作站都通过10Mbit / s交换式以太网连接，并具有DEC LANCE以太网接口。 该开关是DEC EtherWORKS 8T / TX。 实验在一个孤立的网络上运行。

检查点之间的时间间隔为128个请求，这会导致垃圾收集在任何实验中发生多次。 预备准备消息中副本接受的最大序列号是256加上最后一个稳定检查点的序列号。

7.2 微基准

微基准测量调用空操作的延迟。 它评估一个简单服务的两个实现的性能，没有状态实现带有参数和不同大小结果的空操作。 第一个实现是使用我们的库复制的，第二个是未复制的，并直接使用UDP。 表1报告了在客户端测量的只读和读写操作的响应时间。它们是通过在三次单独运行中对10,000次操作调用进行计时获得的，我们报告了三次运行的中值。 与中位数的最大偏差总是低于报告值的0.3%。 我们用a / b表示每个操作，其中a和b是操作参数的大小，结果以KB为单位。

arg./r	复制		没有复制
es。	读写3.35	只读1.62	0.82
(KB)	(309%)	(98%)	4.62
0/0	14.19 (207%)	6.98 (51%)	4.66
4/0	8.01 (72%)	5.94 (27%)	

表1: 微基准测试结果（以毫秒为单位）; 百分比开销是相对于未复制的情况。

复制库引入的开销是由于额外的计算和通信。 例如，读写0/0操作的计算开销大约为1.06ms，其中包括执行加密操作花费的0.55ms。 剩余的1.47ms的开销是由于额外的通信; 复制库引入额外的消息往返传递，它发送更大的消息，并且它增加了每个节点相对于没有复制的服务接收到的消息的数量。

只读操作的开销显着降低，因为5.1节中讨论的优化减少了计算和通信开销。 例如，只读0/0操作的计算开销大约为0.43ms，其中包括执行加密操作花费的0.23ms，并且通信开销仅为0.37ms，因为执行只读操作的协议使用单个轮次-trip。

表1显示4/0和0/4操作的相对开销较低。 这是因为复制库引入的大部分开销与操作参数和结果的大小无关。 例如，在读写0/4操作中，大消息（回复）只通过网络一次（如第5.1节所述），并且只处理回复消息的加密开销增加。 读写4/0操作的开销较高，因为大消息（请求）两次通过网络并增加了处理请求和预先准备消息的加密开销。

需要注意的是，这个微基准代表了我们算法的最坏情况开销，因为这些操作不会工作，而且未复制的服务器提供了非常弱的保证。 大多数服务需要更强大的保证，例如经过身份验证的连接，并且我们的算法相对于实现这些保证的服务器引入的开销将更低。 例如，复制库相对于使用MAC进行身份验证的未复制服务版本的开销仅为读写0/0操作的243%，以及只读4/0操作的4%。

我们可以估计我们的算法相对于Rampart [30]提供的性能增益的粗略下限。 Reiter报告说，在4个SparcStation 10的10 Mbit / s以太网网络中，Rampart对于一个空消息的多RPC具有45ms的延迟[30]。 多重RPC足以让主服务器调用状态机操作，但是对于任意客户端来调用操作，则需要添加额外的消息延迟和额外的RSA签名和验证来验证客户端; 这会导致至少65ms的延迟（使用[29]中报告的RSA时序）。即使我们将此延迟时间除以1.7，DEC 3000/400和SparcStation 10的SPECint92额定值的比率，我们的算法仍然分别以超过10和20的因子减少了调用读写和只读0/0操作的延迟。请注意，这种缩放比较保守，因为网络占据了Rampart延迟的很大一部分[29]，而Rampart的结果是使用300位模数RSA签名获得的，这些签名在今天被认为是不安全的，除非这些密钥用于

生成它们会非常频繁地刷新。

没有公布的SecureRing性能数据[16]，但它会比Rampart慢，因为它的算法在关键路径中有更多的消息延迟和签名操作。

7.3 安德鲁基准

Andrew基准[15]模拟软件开发工作量。 它有五个阶段：（1）递归地创建子目录；（2）复制源树；（3）检查树中所有文件的状态而不检查其数据；（4）检查所有文件中的每个数据字节；（5）编译和链接文件。

我们使用Andrew基准测试来比较BFS和另外两种文件系统配置：NFS-std（Digital Unix中的NFS V2实现）和BFS-nr，它与BFS相同但没有复制。 BFS-nr在客户端上运行了两个简单的UDP中继，并在服务器上运行了一个薄型单板，该单板与所有检查点管理代码已从中删除的snfsd版本链接。 在回复客户端之前，此配置不会将修改的文件系统状态写入磁盘。 因此，它不会实现NFS V2协议语义，而BFS和NFS-std都可以。

在NFS V2协议中的18个操作中，只有getattr是只读的，因为文件和目录的时间最后访问属性由否则为只读的操作设置，例如读取和查找。 结果是我们对只读操作的优化很少被使用。 为了显示此优化的影响，我们还在第二版BFS上运行了Andrew基准测试，修改了查找操作为只读模式。 这种修改违反了严格的Unix文件系统语义，但不太可能

在实践中的不利影响。

对于所有配置，实际的基准代码运行在使用标准NFS客户端的客户端工作站上在数字Unix内核中执行相同安装选项。 这些选项中最相关的基准是：UDP传输，4096字节读取和写缓冲区，允许异步客户端写入和

允许属性缓存。

我们报告每种配置的10次基准测试的平均值。 运行基准测试总时间的样本标准偏差始终低于报告值的2.6%，但前四个阶段的单独时间高达14%。 NFS-std配置中也存在这种高度差异。 报告的平均估计误差在各个阶段低于4.5%，在总数中低于0.8%。

表2显示了BFS和BFS-nr的结果。 BFS-strict和BFS-nr之间的比较表明，该服务的拜占庭容错开销较低 - BFS-strict只需要多花26%的时间来运行

相	BFS 严格	r / o查找	BFS NR
1	0.55 (57%)	0.47 (34%)	0.35
2	9.24 (82%)	7.91 (56%)	5.08
3	7.24 (18%)	6.45 (6%)	6.11
4	8.77 (18%)	7.87 (6%)	7.41
5	38.68 (20%)	38.38 (19%)	32.12
总	64.48 (26%)	61.07 (20%)	51.07

表2：安德鲁基准：BFS vs BFS-nr。 时间在几秒钟内。

完整的基准。 开销低于微基准的观察值，因为客户端在操作之间花费了相当一部分计算时间，即在接收到操作的回复和发出下一个请求之间，并且服务器上的操作执行了一些操作计算。 但是基准阶段的开销并不统一。 造成这种情况的主要原因是客户花费在计算操作之间的时间。 前两个阶段的相对管理费用较高，因为客户花费大约40%的时间在运营间进行计算，而在最后三个阶段花费大约70%。 该表显示，将只读优化应用于查找可显着提高BFS的性能，并将相对于BFS-nr的开销降低至20%。 这个优化在前四个阶段中具有重要影响，因为等待查找操作在BFS-strict中完成的时间至少为这些阶段所用时间的20%，而这是不到上一阶段所用时间的5%。

相	严格	BFS r / o查找	NFS STD
1	0.55 (-69%)	0.47 (-73%)	1.75
2	9.24 (-2%)	7.91 (-16%)	9.46
3	7.24 (35%)	6.45 (20%)	5.36
4	8.77 (32%)	7.87 (19%)	6.60
5	38.68 (-2%)	38.38 (-2%)	39.35
总	64.48 (3%)	61.07 (-2%)	62.52

表3：安德鲁基准：BFS vs NFS-std。 时间在几秒钟内。

表3显示了BFS和NFS-std的结果。 这些结果表明BFS可以在实践中使用 - BFS严格只需要多花3%的时间来运行完整的基准。 因此，可以通过BFS替代Digital Unix中由许多用户每天使用的NFS V2实现，而不会影响这些用户感知的延迟。 此外，对于查找操作只读优化的BFS实际上比NFS-std快2%。

BFS相对于NFS-std的开销不是

相同的所有阶段。 对于阶段1,2和5，两种版本的BFS都比NFS-std更快，但对于其他阶段则更慢。 这是因为在阶段1,2和5中，客户端发出的操作的很大一部分（在21%和40%之间）是同步的，即需要NFS实现的操作才能确保修改后的文件系统状态在应答之前保持稳定给客户。 NFS-std通过将修改状态写入磁盘实现稳定性，而BFS通过使用复制实现更低延迟的稳定性（如Harp [20]）。 NFS-std在阶段3和4中比BFS（和BFS-nr）快，因为客户端在这些阶段不会发出同步操作。

8 相关工作

以前大多数关于复制技术的工作忽略了拜占庭故障或假设了同步系统模型（例如，[17,26,18,34,6,10]）。 Viewstamped复制[26]和Paxos [18]使用主视图和备份视图来容忍异步系统中的良性故障。 容忍拜占庭故障需要一个更复杂的协议，加密认证，额外的预备阶段，以及不同的技术来触发视图更改和选择初选。 此外，我们的系统仅使用视图更改来选择新的主视图，但决不会选择不同的副本集以形成[26, 18]中的新视图。

一些协议和共识算法容忍异步系统中的拜占庭故障（例如[2,3,24]）。 然而，它们并没有提供一个完整的状态机复制解决方案，而且，它们中的大多数都是为了证明理论上的可行性而设计的，并且实际上太慢而无法使用。 我们的算法在正常情况下的操作类似于[2]中的拜占庭协议算法，但是该算法无法在主要故障中存活。

与我们的工作关系最密切的两个系统是Rampart [29,30,31,22]和SecureRing [16]。 它们实现状态机复制，但比我们的系统慢一个数量级以上，最重要的是，它们依赖于同步假设。

Rampart和SecureRing都必须从组中删除故障副本以取得进展（例如，删除故障主节点并选择新节点）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。 然而，失效检测器在异步系统中不能准确[21]，即它们可能会错误地将副本错误分类。 由于正确性要求组成员少于$\backslash 3$有故障，错误分类可通过从组中删除无故障副本来损害正确性。 这就打开了一道攻击途径：攻击者获得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； 那么它会减缓正确复制品或它们之间的通信，直到足够排除在组之外。

为了减少错误分类的可能性，可以对故障检测器进行校准，以延迟将副本分类为错误。 但是，对于可忽略的概率，延迟必须非常大，这是不希望的。 例如，如果主服务器实际上已经失败，则该组将无法处理客户端请求，直到延迟过期。 我们的算法不容易出现这个问题，因为它从不需要从组中排除副本。 Phalanx [23,25]应用仲裁复制技术[12]来实现异步系统中的拜占庭容错。 这项工作不提供通用的状态机复制；相反，它提供了一个数据仓库，其中包含读取和写入单个变量以及获取锁的操作。 它为读写操作提供的语义比我们的算法提供的语义弱；我们可以实现访问任意数量变量的任意操作，而在Phalanx中，需要获取和释放锁以执行这些操作。 Phalanx没有公布的性能数字，但我们相信我们的算法更快，因为它在关键路径中的消息延迟更少，并且因为我们使用MAC而不是公钥密码术。 Phalanx的方法提供了改进可扩展性的潜力；每个操作仅由副本的一个子集处理。 但是这种可扩展性的方法是昂贵的：它需要 $\backslash 1$容忍$\backslash$故障；每个副本需要一个状态的副本；并且每个副本上的负载随着缓慢下降 （这是$\backslash 1$ ）.

9 结论

本文描述了一种新的状态机复制算法，它能够容忍$n \geq \frac{1}{3}$拜占庭故障，并且可以在实践中使用：它是第一个在异步系统（如Internet）中正确工作的算法，并且它改进了以前算法的性能一个数量级。

该论文还描述了BFS，一种拜占庭式的NFS容错实现。 BFS证明可以使用我们的算法来实现接近未复制服务性能的实际服务--BFS的性能仅比Digital Unix中的标准NFS实现性能差3%。 这种良好的性能归功于许多重要的优化，包括用消息认证代码向量替换公钥签名，减少消息的大小和数量，以及增量检查点管理技术。

为什么拜占庭式容错算法在将来很重要的一个原因是，即使存在软件错误，他们也可以让系统继续正常工作。 并非所有错误都可以生存；我们的方法不能掩盖发生的软件错误

在所有复制品。 但是，它可以掩盖在不同副本中独立出现的错误，包括非确定性软件错误，这是最难以检测的最常见和最持久的错误。 事实上，我们在运行我们的系统时遇到了这样的软件错误，而且尽管如此，我们的算法仍然能够继续正常运行。

改进我们的系统还有很多工作要做。 一个特别感兴趣的问题是减少实现我们的算法所需的资源量。只有当一些完整副本失败时，通过使用\副本作为参与协议的证人才能减少副本的数量。 我们也认为可以将国家的副本数量减少到\1，但细节仍有待解决。

致谢

我们要感谢Atul Adya，Chandrasekhar Boyapati，Nancy Lynch，Sape Mullender，Andrew Myers，Liuba Shrira和匿名裁判对本文的草稿提供的有用评论。

参考

[1] M. Bellare和D. Micciancio。 无碰撞散列的新范式：降低成本的增量性。 密码学进展 - Eurocrypt 97，1997。

[2] G. Bracha和S. Toueg。 异步共识和广播协议。 Journal of the ACM，32（4），1995。

[3] R. Canneti和T. Rabin。 最佳异步拜占庭协议。 技术报告#92-15，希伯来大学计算机科学系，1992年。

[4] M.卡斯特罗和B. Liskov。 一种实用的拜占庭容错复制算法的正确性证明。 技术备忘录MIT / LCS / TM-590，麻省理工学院计算机科学实验室，1999年。

[5] M.卡斯特罗和B. Liskov。 没有公钥密码体制的认证拜占庭容错。 技术备忘录MIT / LCS / TM-589，麻省理工学院计算机科学实验室，1999年。

[6] F. Cristian，H. Aghili，H. Strong和D. Dolev。 原子广播：从简单的信息传播到拜占庭协议。 在1985年的国际容错计算会议上。

[7] S. Deering和D. Cheriton。 数据报互联网络和扩展LAN中的多播路由。 ACM Transactions on Computer Systems，8（2），1990。

[8] H. Dobbertin。 近期攻击后MD5的状态。 RSA Laboratories的CryptoBytes，2（2），1996。

[9] M. Fischer，N. Lynch和M. Paterson。 一个错误过程的分布式共识不可能。 Journal of the ACM，32（2），1985。

[10] J. Garay和Y.摩西。 t + 1轮中n 3t处理器的完全多项式拜占庭协议。 SIAM Journal of Computing，27（1），1998。

[11] D. Gawlick和D. Kinkade。 IMS / VS快速路径中的并发控制。 数据库工程，8（2），1985。

[12] D.吉福德。 复制数据的加权投票。 1979年在操作系统原理研讨会上。

[13] M. Herlihy和J. Tygar。 如何使复制的数据安全。 *密码学进展（LNCS 293），1988。*

[14] M. Herlihy和J. Wing。 并发对象的公理。 在ACM 1987年编程语言原理专题讨论会上。

[15] J. Howard等人。 分布式文件系统中的规模和性能。 *ACM Transactions on Computer Systems，6（1），1988。*

[16] K. Kihlstrom，L. Moser和P. Melliar-Smith。 确保集团通信安全的协议。 在夏威夷国际系统科学会议上，1998年。

[17] L. Lamport。 时间，时钟和分布式系统中事件的顺序。 COMMUN。 ACM，21（7），1978。

[18] L. Lamport。 兼职议会。 技术报告49，DEC系统研究中心，1989。

[19] L. Lamport，R. Shostak和M. Pease。 拜占庭将军问题。 ACM Transactions on Programming Languages and Systems，4（3），1982。

[20] B. Liskov等人 Harp文件系统中的复制。 在ACM的操作系统原理研讨会上，1991。

[21] N. Lynch。 分布式算法。 摩根考夫曼出版社，1996年。

[22] D. Malkhi和M. Reiter。 高吞吐量安全可靠多播协议。 在计算机安全基础研讨会，1996年。

[23] D. Malkhi和M. Reiter。 拜占庭法定系统。 在ACM计算理论研讨会上，1997。

[24] D. Malkhi和M. Reiter。 分布式计算中的不可靠入侵检测。在计算机安全基础研讨会，1997年。

[25] D. Malkhi和M. Reiter。 在Phalanx中进行安全和可扩展的复制。 IEEE可靠分布式系统研讨会，1998。

[26] B. Oki和B. Liskov。 已加密的复制：支持高可用分布式系统的新主复制方法。 在ACM分布式计算原理专题讨论会上，1988年。

[27] B. Preneel和P. Oorschot。 MDx-MAC和通过散列函数构建快速MAC。 在Crypto 95，1995中。

[28] C. Pu，A. Black，C. Cowan和J. Walpole。 提高操作系统多样性的专业化工具包。 1996年在ICMAS免疫系统研讨会上。

[29] *M. Reiter。 安全协议协议。 在ACM计算机和通信安全会议上，1994年。*

[30] M. Reiter。 用于建立高诚信服务的Rampart工具包。 分布式系统理论与实践（LNCS 938），1995。

[31] M. Reiter。 安全组成员协议。 IEEE Transactions on Software Engineering，22（1），1996。

[32] R. Rivest。 MD5消息摘要算法。 互联网RFC-1321,1992。

[33] R. Rivest，A. Shamir和L. Adleman。 一种获取数字签名和公钥密码体制的方法。 ACM的通信，21（2），1978。

[34] F.施耐德。 使用状态机方法实现容错服务：教程。 ACM Computing Surveys，22（4），1990。

[35] A.沙米尔。 如何分享一个秘密。 ACM通讯，22（11），1979。

[36] G. Tsudik。 使用单向散列函数进行消息认证。 ACM Computer Communications Review，22（5），1992。

[37] M. Wiener。 公钥密码体制的性能比较。 RSA Laboratories的CryptoBytes，4（1），1998。