**Achilles：通过回滚弹性恢复实现高效的TEE辅助BFT共识**

**摘要**

利用可信执行环境(TEEs)来提高系统容错能力和性能的BFT共识正在变得越来越流行。然而，现有的工作都面临TEE回滚问题，导致容错能力与性能之间存在权衡。本文提出了Achilles，一种高效的TEE辅助BFT协议，打破了这一权衡。Achilles的核心思想是从提交事务的关键路径上移除TEE昂贵的回滚预防机制。为此，Achilles采用回滚弹性恢复机制，允许节点互相协助恢复各自的状态。此外，Achilles遵循现代链式BFT协议的链接精神，利用定制的链式提交规则实现线性消息复杂度、四个通信步骤的端到端事务延迟，以及对少数拜占庭节点的容错能力。Achilles是首个在这些指标上与CFT协议保持一致的TEE辅助BFT协议。我们基于Intel SGX实现了Achilles的原型，并在局域网和广域网环境中进行评估，展示了其相比几种最新方案的出色性能。

**1 引言**

拜占庭容错(BFT)共识作为分布式计算中的重要原语，由于其在共享数据库、分布式存储和区块链中的应用，近年来重新引起了人们的关注。BFT共识使一组节点能够就相同的不断增长的事务序列达成一致，即使其中一些节点是拜占庭节点(即行为任意)。然而，与处理节点崩溃行为的崩溃容错(CFT)共识相比，这种有前景的容错能力会带来更低的容错阈值和性能。首先，BFT协议能够容忍三分之一的拜占庭节点，而CFT协议能够处理少数节点的崩溃。其次，BFT协议需要更长的延迟(例如，在PBFT中包括两次客户端交互在内的五个通信步骤)或更高的消息复杂度。相比之下，像Raft这样的CFT协议可以实现四个通信步骤的延迟和线性消息复杂度。

为了降低成本，一系列工作利用可信执行环境(TEEs)，如Intel SGX，来设计BFT共识(称为TEE辅助BFT共识)。TEEs使应用程序能够在硬件保护的环境中运行，通过隔离应用程序的代码和数据免受操作系统或虚拟机管理程序的影响，提供完整性和保密性保证。TEEs的完整性可以通过防止二义性或检测二义性的方法防止拜占庭节点发送二义性消息。因此，已验证身份的节点可以在存在少数拜占庭节点(没有二义性)的情况下达成共识，使TEE辅助BFT协议的容错阈值与其CFT对应物保持一致。此外，TEE辅助BFT还可以减少预防领导者发送二义性提案所需的一些通信步骤(例如，PBFT中的预准备阶段)。

尽管有这些令人鼓舞的改进，Gupta等人最近指出了现有TEE辅助BFT协议的三个所谓问题，即限制性响应能力、缺乏并行性和TEE回滚问题。具体来说，回滚问题可以通过现有的预防方法解决，但这些方法对性能影响很大。为了解决这些问题，作者提出了FlexiBFT，将容错阈值从n=2f+1放宽到n=3f+1(n和f分别表示节点总数和故障节点数)。换句话说，FlexiBFT通过实现响应能力、提高并行性和减少对TEE的访问(每次都会产生昂贵的回滚预防)来牺牲容错以换取更好的性能。然而，这种对性能的优先考虑引起了很大争议，因为它与之前追求更高容错能力的工作相冲突。为避免社区内的潜在冲突，一个直接的问题出现了：如何打破容错-性能权衡，设计一种高效的TEE辅助BFT协议？

在本文中，我们旨在通过解决这三个问题来打破性能与容错的权衡。我们发现只有回滚问题仍未解决，因为其他两个问题可以通过稍微修改设计来解决(详见第6.1节)。要解决回滚问题，关键是将相关的预防措施从提交事务的关键路径中移除，从而确保系统性能不再受到影响。在现有的TEE辅助BFT协议中，当节点发送消息(如提案和投票)时，它必须访问在TEE内实现的可信组件(如消息计数器)来认证这些消息。对于每次访问，被调用的可信组件需要增加低性能的持久计数器(例如，TPM计数器或ROTE)以单独防止状态恢复期间的状态回滚(详见第2.2节)。我们的方法不是从本地不可信存储中恢复状态，而是使节点能够互相协助恢复其可信组件。这是因为节点之间交换的消息允许它们推断彼此的最新状态。由于与共识过程中频繁调用可信组件相比，恢复很少发生，这种方法可以显著降低回滚预防的开销。

我们观察到，采用防止二义性方法的可信组件旨在强制一个节点在每个视图中最多发送一次消息。此外，当节点进入新视图时，它会停止发送较低视图的消息。因此，恢复应该使节点的可信组件知道已发送的消息中最高视图的消息。乍看之下，由于网络异步或者拜占庭节点的隐藏，节点的可信组件认证的消息可能无法到达其他节点，因此实现这种精确恢复似乎是不可能的。为解决这种不可能性，一种方法是放宽恢复要求，同时承诺不损害安全性。也就是说，节点在进入更高视图之前不能发送任何消息，因为它不知道是否已在当前视图中发送过消息。

我们提出了Achilles，一种高效的TEE辅助BFT协议，采用回滚弹性恢复来打破性能与容错的权衡。除了回滚问题外，现有的TEE辅助BFT协议在延迟和消息复杂度方面效率不高。例如，FlexiBFT在四个通信步骤中提交事务的消息复杂度为O(n²)。Damysus有线性消息复杂度，因为它使用链结构；然而，它需要六个通信步骤来提交事务。后来，OneShot优化了Damysus，在正常和搭便车执行(即所有先前提议的事务都已提交)情况下仅需四个通信步骤，而在其他情况下仍需六个步骤。与Raft等CFT协议相比，在消息复杂度和延迟方面仍存在差距(见第2.2节)。

为消除这一差距，我们观察到Damysus中的准备阶段可以被移除，因为有防止二义性和链式提交。前者由TEE内的可信组件提供，后者是指后代区块的提交将导致其未提交父区块的提交。利用这些，Achilles采用定制的链式提交规则，实现线性消息复杂度和四个通信步骤的延迟(不包括视图切换阶段)。Achilles是首个在消息复杂度、延迟和容错阈值上与CFT对应物保持一致的TEE辅助BFT共识。

我们基于Intel SGX实现了Achilles的原型。我们在Damysus实现的基础上开发了Achilles。我们在公共云平台上进行了广泛的实验，评估和比较Achilles与三个对应物：Damysus-R(带有回滚预防的Damysus变种)、FlexiBFT和OneShot-R(带有回滚预防的OneShot变种)。我们在局域网和广域网上进行实验，最多有f=30个拜占庭节点。

我们的贡献。本文的贡献如下：

• 我们提出了Achilles，一种高效的TEE辅助BFT共识协议，利用回滚弹性恢复打破容错性能权衡。Achilles从提交事务的关键路径中移除昂贵的回滚预防机制。

• 我们提出了链式提交规则，使Achilles具有线性消息复杂度和四个通信步骤的端到端延迟。Achilles是第一个在容错阈值、消息复杂度和延迟方面与Raft等CFT协议相同的TEE辅助BFT协议。

• 我们还开发了Achilles的概念验证并在公共云平台上评估其性能。我们的评估结果显示，在局域网环境下，f=30个拜占庭节点时，Achilles实现了75.38K TPS的吞吐量和5.12ms的延迟，分别比Damysus-R、FlexiBFT和OneShot-R高17倍、6倍和7倍。

**2 背景和动机**

**2.1 TEEs的回滚问题**

TEEs是CPU扩展，使应用程序能够在安全执行环境(称为enclaves)中运行，通过利用硬件辅助隔离、内存加密和远程认证等技术，提供完整性和保密性保证。有影响力的TEE平台包括Intel SGX和AMD SNP，它们已被用于区块链、可信存储以及认证速率限制等应用中。这些TEE内的应用程序被称为enclave应用程序。具体来说，enclave应用程序必须不断地将其加密状态数据(例如，在SGX中调用seal函数)存储在不可信存储上，以便能够从故障(例如，断电或系统崩溃)中恢复状态。

然而，现有的TEE平台无法保证enclave应用程序重启后检索的状态数据的新鲜度，导致回滚问题。特别是，控制操作系统的攻击者可以提供enclave应用程序存储数据的旧版本，将其状态回滚到先前的状态。状态回滚在许多应用程序中都会产生严重后果，尤其是TEE辅助BFT共识。例如，攻击者可以重置在TEE内实现的虚拟消息计数器，使节点使用相同的计数器值发送二义性消息。请注意，与这些虚拟计数器不同，下面提到的持久计数器不会受到回滚攻击的影响。

回滚预防。现有的回滚预防解决方案依赖于可信持久计数器，其值一旦增加，就不能恢复到先前的值。具体来说，在enclave应用程序更新其状态之前，有两个操作：1)存储操作，它将磁盘上的每个状态数据与计数器值绑定，以及2)增加操作，它将持久计数器增加一。重启后，enclave应用程序可以检查从操作系统检索的状态数据是否与获得的计数器值匹配。有两类：基于硬件和基于软件的持久计数器，如下所示。

1) 基于硬件的持久计数器包括SGX计数器、TPM计数器和TPM NVRAM。所有这些计数器实现都性能不佳，即读/写操作的延迟长，写入周期有限。例如，为状态更新增加TPM计数器大约需要97ms，而读取计数器进行状态检查大约需要35ms。因此，它们对于需要高性能连续状态更新的TEE辅助BFT共识不切实际。

2) 基于软件的持久计数器，如ROTE、Narrator和TIKS，是由TEE的分布式系统实现的。具体来说，这些TEE运行广播协议(至少需要两个通信步骤)来维护一致的内存计数器值。因此，在TEE辅助BFT共识中集成基于软件的计数器将引入多个通信步骤。其他使用可信服务器和客户端检测的预防方法要么依赖于中心化信任，要么不通用，使它们对TEE辅助BFT共识不可行。

**2.2 为什么TEE辅助BFT协议效率低下？**

BFT共识可以利用TEE的完整性防止已验证节点发送二义性消息，从而实现更高的容错阈值和更好的性能(例如，减少消息复杂度，提高系统并行性，或缩短延迟)。我们现在介绍三种最先进的协议：Damysus、FlexiBFT和OneShot，其他协议将在第7节中介绍。

Damysus基于HotStuff，利用两个可信组件，即checker和accumulator，在少数拜占庭节点存在的情况下，通过六个通信步骤提交事务。提交延迟是端到端的，因为它包括接收客户端事务和发送回复的两个步骤。Damysus在正常情况操作和视图切换阶段都具有线性消息复杂度。后来，OneShot通过在正常和搭便车执行中实现四个通信步骤来优化Damysus，而在其他情况下仍然需要六个步骤。与Damysus和OneShot不同，FlexiBFT将容错从2f+1降低到3f+1，以实现更高的并行性、更好的响应性和更少的回滚预防开销。FlexiBFT在四个通信步骤中提交区块，但广播消息的消息复杂度为O(n²)。请注意，为了公平比较，不考虑轮换领导者的视图切换步骤。尽管有这些进步，它们在以下两个方面仍然效率低下。

昂贵的回滚预防阻碍性能。在现有的TEE辅助BFT协议(例如MinBFT)中，要提交事务，领导者(resp.，备份节点)在提议(resp.，为事务投票)时必须访问TEE内的可信组件以防止二义性，如图1所示。每次调用可信组件时，它们都会更新状态并增加可信持久计数器以防止回滚问题。因此，MinBFT中事务的延迟至少包括两次计数器的写入延迟(一次用于领导者，另一次用于备份节点)。更糟糕的是，要在一个视图中提交n个事务，节点必须增加计数器n次，这对于基于硬件的计数器来说是不可行的，因为它们的写入周期有限。

容错-性能权衡。在现有的TEE辅助BFT协议中（例如MinBFT），要提交事务，领导者（分别是，备份节点）在提议（分别是，为事务投票）时必须访问TEE内的可信组件以防止二义性，如图1所示。每次调用可信组件时，它们都会更新状态并增加可信持久计数器以防止回滚问题。因此，MinBFT中事务的延迟至少包括两次计数器的写入延迟（一次用于领导者，另一次用于备份节点）。更糟糕的是，要在一个视图中提交n个事务，节点必须增加计数器n次，这对于基于硬件的计数器来说是不可行的，因为它们的写入周期有限。

为了避免使用昂贵的持久计数器，FlexiBFT接受锁定容错（如图1b所示），在该方案中，备份节点可以回滚其状态。具体来说，FlexiBFT将共识群组从2f+1增加到3f+1。在这种情况下，为了提交变更，FlexiBFT需要来自2f+1个节点的投票，其中至少f+1个节点是正确的（不会发送二义性消息）。然后，BFT共识可以容忍f个节点的二义性行为，而不会损害安全性。但是，这种解决方案会降低系统的容错能力，不再与CFT协议保持一致。

在Damysus-R中，领导者和备份节点都需要使用可信组件两次（有关Damysus-R的详细描述，请参见第5节）。因此，Damysus-R将使用计数器进行回滚预防的开销增加一倍。同样，在OneShot-R中，领导者和备份节点在正常情况下使用可信组件一次，而在其他情况下使用可信组件两次。

然而，现有的基于硬件和软件的持久计数器都有显著的性能限制（见第2.1节）。此外，节点必须在事务提交过程中多次使用这些计数器以缓解回滚攻击。这就是为什么FlexiBFT降低容错能力的原因，备份节点可以回滚其状态以避免使用昂贵的持久计数器。

没有一个协议能够实现与CFT协议一样的线性消息复杂度和四个通信步骤的延迟。像Raft这样的CFT协议可以实现线性消息复杂度和四个通信步骤的延迟。相比之下，现有的TEE辅助BFT协议要么实现线性消息复杂度（例如Damysus），要么实现最优的四个通信步骤（例如FlexiBFT），但不能同时实现两者。

总结。现有的TEE辅助BFT协议在回滚预防和性能（即消息复杂度和延迟）方面效率低下，如表1所总结。这促使我们提出Achilles，一种高效的TEE辅助BFT协议，不使用昂贵的持久计数器进行回滚预防。Achilles还实现了线性消息复杂度、四个通信步骤的端到端延迟、对少数拜占庭节点的容错能力以及回复响应能力（在第6.1节中介绍）。

**3 系统模型和目标**

**3.1 系统模型**

我们遵循现有TEE辅助BFT协议的系统模型。我们考虑一个由n=2f+1个节点{pi}维护的分布式系统，同时被一组客户端访问。我们假设每个节点都配备了TEE，用于运行一些可信组件（在第4.3节中指定）。我们假设节点之间存在公钥基础设施（PKI）来分发验证和消息签名所需的密钥。具体来说，每个节点pi都有一个公钥/私钥对，表示为(pki, ski)，其中私钥只能由节点的可信组件访问。

每对节点之间都通过可靠的通道连接。我们采用Dwork等人提出的部分同步模型，这在BFT共识中常用。存在一个既定的界限Δ和未定义的全局稳定时间（GST）。在GST点之后，保证在Δ限制内在两个诚实节点之间传输的任何消息的传递。也就是说，在GST之后，系统表现为同步。

威胁模型。我们考虑一个攻击者A，它可以在任何时间污染最多f个节点。被污染的节点是拜占庭的，即表现为任意行为，由攻击者控制，但TEE完整性（见下文）和密码学方案（例如，本文中使用的公钥签名和抗碰撞哈希函数）不能被破坏。我们将严格遵循协议且不会崩溃的节点定义为正确节点，其余的是拜占庭节点。

对于被污染的节点，攻击者获得对其操作系统的完全控制，因此可以修改、重排和延迟从/到TEE的网络消息。攻击者可以以任意输入启动、停止和调用本地TEE enclaves，但它不能提取内存内容或操纵enclaves中运行的代码以破坏完整性。攻击者还可以通过提供TEE外部的过时存储数据将TEE的状态回滚到某些先前版本（包括重置状态），这也被称为回滚攻击。此外，分叉攻击（使节点能够访问多个运行相同可信组件的TEE enclave）超出了本文的范围，因为这种攻击可以通过使用TPM PCR或会话密钥机制来缓解。

**3.2 系统目标**

客户端创建一组事务，这些事务被发送到节点。节点可以处理客户端的事务并进一步将它们打包在区块中。每个区块还包括对先前区块（称为父区块）的加密哈希引用。（区块格式和链结构在第4.2节中介绍。）运行协议的每个节点都提交一个链接区块序列，使以下安全属性成立：

• 安全性：如果两个正确的节点在相同的高度提交两个区块b和b′，则b = b′。

• 活性：客户端的事务最终将被包含在正确节点提交的区块中。

**4 Achilles设计**

本节介绍Achilles，一种具有回滚弹性恢复机制的高效TEE辅助BFT协议。Achilles定制了Damysus首次提出的两个可信组件，即checker和accumulator。这种定制允许checker记录来自领导者的最新（未）准备好的区块，accumulator扩展记录在checker中的区块（第4.3节）。借助这些定制组件，Achilles进一步扩展了Damysus的正常情况操作，以移除准备阶段（第4.4节）。更重要的是，Achilles的新回滚弹性恢复允许节点在其他节点的协助下恢复其状态，而不是使用昂贵的持久计数器（第4.5节）。关于Damysus的更好理解，我们建议读者参考附录A47了解更多细节。

**4.1 概述**

Achilles在视图中运行，其中一个被委派的节点（称为领导者）与其他节点（称为备份）协调以提交事务。Achilles使用循环策略为每个视图更换领导者，遵循链式BFT协议（如HotStuff和Damysus）的链接精神。在Achilles中，有两个主要组件：基于链式提交规则的单阶段协议（也称为正常情况操作）来完成事务和回滚弹性恢复，如图2所示。除了这些，Achilles还有一个节拍器组件。接下来，我们提供对这些组件的高层次介绍，详细描述见第4.4节和第4.5节。

正常情况操作。每个视图中的正常情况操作包含三个阶段：1) 新视图阶段，领导者从节点收集至少f+1个包含最新存储区块的视图消息；2) 提交阶段，领导者创建一个区块并从至少f+1个节点收集投票；3) 决策阶段，领导者提交区块，执行它并通知客户端和其他节点。新视图阶段允许新领导者与备份节点同步最新的区块信息。如果相关的领导者接收到在视图v中产生的已提交区块，则可以跳过视图v+1的这个阶段。类似的优化也在链式BFT协议中采用，如Fast-HotStuff、HotStuff-2和OneShot。此外，如果采用稳定领导者策略（即领导者不是每个区块都更换），则可以移除这个阶段。这就是为什么我们在事务延迟（以消息步骤计）中不考虑它，以便与其他协议进行公平比较。有关正常情况操作的更多详细信息，请参见第4.4节。

在上述阶段中，Achilles使用checker和accumulator组件来认证消息。在新视图阶段和提交阶段，节点（包括领导者）访问checker组件，而accumulator组件仅由领导者在提交阶段使用，如图2所示。

回滚弹性恢复。重启后，节点必须在参与正常情况操作、节拍器或回复其他节点的恢复请求之前恢复其可信组件的状态。回滚弹性恢复必须确保无二义性属性，即如果一个节点在一个视图中发送两个相同类型的消息msg和msg′，则msg = msg′。这个属性防止节点发送二义性消息，这对于共识的安全性起着互补作用。

在Achilles中，checker组件维护重要的系统状态，如当前视图、阶段和最后未准备的区块，而accumulator是无状态的。因此，恢复的目标是为checker组件保留状态。要实现这个目标，正在恢复的节点可以向其他节点发送恢复请求来获取其checker组件中的状态，如图2所示。请求中包含一个随机数以防止重放攻击。有关恢复过程的详细描述见第4.5节。

节拍器。Achilles遵循Damysus和HotStuff的节拍器设计，这是一种确保活性的机制，即在GST后使共识取得进展。它有两个目标：1) 使所有正确的节点和一个唯一的领导者在足够长的时间内进入相同的视图，2) 确保领导者扩展一个所有正确节点都会投票的父区块。具体来说，对于第一个目标，一种常见的解决方案是增加视图的超时时间，直到取得进展。对于第二个目标，在Achilles中视图更改后，新领导者必须收集来自f+1个不同节点的新视图消息，以扩展存储在节点checker组件中的最新（未）准备区块（第4.4节）。更多详细信息请参考67。

**4.2 数据结构**

签名和加密哈希。节点和可信组件使用非对称签名方案来验证消息。签名方案提供两个主要功能：SIGN函数，使用私钥sk对消息msg生成签名σ；和VERIFY函数，使用公钥pk验证消息msg上的签名σ。我们使用⟨msg⟩σ（分别是，⟨msg⟩σ⃗）表示使用SIGN函数带有签名σ（分别是，签名列表σ⃗）的已签名消息msg。我们使用σ.id表示签名σ的身份。请注意，签名是由可信组件使用其私钥生成的（见第4.3节）。加密哈希函数H(·)接受任意长度的字符串作为输入，并输出固定长度的字符串。

区块格式。一个区块b是⟨txs, op, hp⟩，其中txs是客户端事务的批处理，op是执行结果，hp是先前区块（也称为父区块）的哈希值。为方便起见，b.x表示区块b的相关参数x。例如，b.txs表示包含的事务。我们假设两个函数：executeTx(txs, hp)函数，为给定以区块哈希hp结尾的链接区块的txs输出执行结果op；和createLeaf(txs, op, hp)函数，创建一个扩展哈希为hp的父区块的新区块。

由于哈希引用，区块可以加密链接形成链结构。因此，b1 ≻ b2表示b1扩展了区块b2，任何人都可以通过哈希引用验证。当b1是对哈希值为h的区块b2的扩展时，我们也写作b1 ≻ h。我们说区块b1与不同的区块b2冲突，如果既不是b1 ≻ b2也不是b2 ≻ b1。特别地，创世区块G是硬编码的。非创世区块的高度是它到创世区块G的距离，因此创世区块G的高度为零。在Achilles中，有效区块b意味着b ≻ G，并且任何满足b ≻ b′的b′都通过执行b′.txs具有有效的执行结果b′.op，并且b.op也是有效的。此外，区块的新鲜度根据它们的视图进行比较。视图最高的区块是最新的。

证书。在Achilles中，节点使用可信组件来认证其消息，防止消息二义性或遵守预定义规则。其他节点可以验证由这些组件签名的证书φ作为证明。我们使用φ⃗表示证书列表，φ⃗n表示该列表长度为n。有五种证书，介绍如下。

• 区块证书。区块证书（表示为φb）是由领导者在提交阶段为其区块创建的。它的形式为⟨prop, h, v⟩σ，其中h是区块的哈希值，v是产生区块的视图号，σ是领导者的签名。为方便起见，φb.view和φb.hash分别表示视图v和哈希h。该证书由算法2中的TEEprepare函数产生，该函数通过设置flag为1保证领导者每个视图只能制作一个区块。

• 存储证书。存储证书（表示为φs）由节点（包括领导者）创建，用于证明在提交阶段收到来自领导者的区块（算法1的第23行）。它的形式为⟨commit, h, v⟩σ，其中h是存储区块的哈希值，v是产生区块的视图，σ是节点的签名。该证书由算法2中的TEEstore函数产生。在调用该函数之前，节点必须验证来自视图v的领导者的区块是有效的。

• 提交证书：提交证书（表示为φc）是领导者在提交阶段产生的f+1个存储证书的组合。f+1个存储证书确保至少有一个正确的节点参与了投票，因此持有相应的区块（正确的节点只对它们收到的区块投票）。它的形式为⟨decide, h, v⟩σ⃗f+1。

• 累加器证书：累加器（表示为acc）在领导者调用TEEaccum函数选择父区块时生成（算法2的第22-25行）。具体来说，该函数输入来自节点的f+1个视图证书。它的形式为⟨acc, h, v, id⃗⟩σ，其中h和v分别是父区块的哈希和视图，id⃗是贡献给累加器的f+1个节点的id向量，即签署了作为TEEaccum参数传递的视图证书，σ是领导者的签名。我们使用acc.hash表示h，acc.v表示v。如果其签名正确，并且id⃗是f+1个唯一id的向量（算法2的第7行），则称acc证书有效。

• 视图证书：视图证书（表示为φv）由节点（包括领导者）在视图成功结束并提交区块时生成，或当超时被触发时生成。它的形式为⟨new-view, h, v, v′⟩σ，其中h是存储区块的哈希值，v是产生区块的视图，v′是当前视图，σ是节点创建的签名。v′防止过时的证书被拜占庭节点重放。该证书由算法2中的TEEview函数产生。

**4.3 可信组件**

Achilles使用在TEE中运行的checker和accumulator组件。它们继承自Damysus20，但被扩展以移除Damysus中的准备阶段。主要的扩展是在Achilles中，checker记录来自领导者的最新区块的信息（即哈希值），这些区块可以是已准备或未准备的。相比之下，在Damysus中，checker只存储已准备区块的信息，即在准备阶段由至少f+1个节点投票认证的区块。相应地，在Achilles中，accumulator允许领导者扩展存储在节点checker组件中的最新（未）准备区块。我们现在介绍这两个组件。

Checker。checker主要提供两项服务：1) 它将每个共识消息（即区块和投票）与每个视图中的唯一标识符绑定，以防止二义性，以及2) 它存储来自领导者的区块，以防止节点隐藏最新区块。因此，节点pi的checker组件的状态有三个组成部分：

• {ski, pk1, ..., pkn}，其中ski是其机密私钥，{pk1, ..., pkn}是节点的公钥；

• (vi, flag)，其中vi是当前视图号，flag表示领导者是否已在视图vi中提出区块。如果领导者产生一个区块，flag从0变为1。如果节点在此视图中存储了一个区块，vi增加一，flag初始化为0；

• (prev, preh)，其中prev和preh分别是最新存储区块的视图号和哈希值。

checker组件提供两个函数：

• TEEprepare(b, h, φ)：此函数输入区块b、区块b的哈希h以及提交或累加器证书。它输出带有领导者签名的区块证书φb。该证书确保b是由φ证明的、扩展b′的唯一区块。

• TEEstore(φb)：此函数输入区块证书φb并输出存储证书φs。任何人都可以通过签名验证此证书。

Accumulator。该组件强制领导者在节点新视图消息中包含的f+1个区块中选择视图最高的存储区块。与checker不同，在一个视图中，只有领导者与其accumulator交互。节点pi的accumulator组件仅维护私钥ski和节点的公钥{ski, pk1, ..., pkn}。该组件提供以下接口：

• TEEaccum(φv, φ⃗n)：该函数接收f+1个视图证书的列表，并通过生成一个累加器acc来断言列表中的第一个元素是具有最高提议视图的区块（即区块被准备的视图）。这个函数为领导者提供证明；它的区块扩展了接收到的f+1个视图证书中最新的区块。

在Achilles中，只有checker组件维护与正常情况操作相关的状态（例如，vi和flag）。节点的私钥和公钥可以从重构信息中获取，这些信息可以存储在磁盘上。因此，恢复节点只需恢复其checker组件的状态。有关更多详细信息，请参见第4.5节。

**4.4 正常情况操作**

算法1显示了正常情况操作的伪代码，它有三个步骤。

❶ 在新视图阶段（算法1的第39-43行），节点首先增加其视图以进入新视图。然后，它向新视图的领导者发送一个视图证书φv，其中包含最新存储区块b的哈希和相关视图。请注意，如果节点在视图中提交了一个区块（算法1的第32-36行），它可以直接增加其视图并将提交证书φc发送给新视图的领导者。（见下面的优化。）

❷ 在提交阶段，领导者使用累加器从f+1个有效的视图证书中选择具有最高视图的存储区块（算法1的第7-10行）。领导者必须确保视图证书中哈希对应的区块是正确的。（注意，在第8行中，φv.view = v′。）然后，领导者提议一个区块来扩展所选区块，并使用TEEprepare来认证它。

当收到来自领导者的区块时，节点首先检查区块是否有效。具体来说，节点接收了该区块及其所有祖先区块。此外，其中所有的执行结果都是有效的。如果通过检查，它将区块存储在preb中（算法1的第22行），并向领导者发送由TEEstore函数生成的存储证书。

❸ 在决策阶段，如果领导者收集到来自节点的f+1个存储证书，它会将它们组合成一个提交证书φc并广播φc。

当收到来自领导者的提交证书φc（即确保f+1个节点存储了一个区块）时，节点在preb中记录该证书（算法1的第34行）作为来自领导者的最新存储区块。此外，节点开始新的视图并通过转发证书和区块（包含执行结果）来回复客户端。客户端在验证一个节点的回复中的证书后可以确认执行结果，从而确保回复响应性（见第6.1节）。

新视图阶段优化。当视图v的领导者接收到视图v-1中区块b的提交证书φc时，它可以直接提议一个区块来扩展区块b，而无需等待f+1个视图证书。领导者必须拥有区块b及其所有祖先区块才能进行扩展。此外，在接收到当前视图的提交证书φc后，节点可以直接增加其视图并向新视图的领导者发送视图证书。

区块同步。当节点从领导者接收到区块⟨b, φb⟩（算法1的第19行）时，它可能没有接收到区块b的祖先区块。因此，它必须从其他节点拉取这些区块。同时，当节点接收到区块b的提交证书φc时，它可以向客户端发送区块b所有未提交祖先区块的回复。这是因为链结构的特性，如果一个区块被提交，其所有祖先区块也被提交。

**4.5 回滚弹性恢复**

当节点重启时，它必须执行恢复协议来获取TEE内可信组件的状态。在完成恢复之前，其状态为恢复中，它不能参与正常情况操作、视图切换协议和回复其他节点的恢复请求。具体来说，节点需要了解节点配置，即获取其密钥对和其他节点的公钥以进行通信。配置信息以加密和认证的方式存储在本地磁盘上，确保攻击者既不能伪造也不能知道它。例如，节点可以使用Intel SGX中的seal函数将配置消息存储在磁盘上，并使用其MRENCLAVE身份的unseal来获取它们。请注意，配置信息可以在不依赖可信第三方的情况下设置。这是因为节点的TEE可以利用互相远程认证来建立PKI，如46中所述。我们假设Achilles中有一组固定的节点，并在第6.2节中讨论动态重构的限制。

如第4.3节所介绍，在Achilles中只有checker组件维护系统状态（即vi、flag、prev和preh），并需要在重启后恢复。算法3显示了恢复过程的伪代码，它包含三个步骤，如下所示：

❶ 正在恢复的节点pj向所有其他节点发送由TEErequest函数创建的恢复请求（形式为⟨req, non⟩）。non是一个随机数，防止拜占庭节点重放恢复回复。特别是，其他节点在其回复中包含相同的non（步骤2），恢复节点检查non是否匹配（步骤3）。

❷ 当接收到来自节点pj的请求时，节点调用TEEreply函数创建恢复回复。首先，该函数检查签名是否由节点pj创建。如果通过检查，该函数创建一个回复φrpy = ⟨RPY, preph, prepv, vi, j, non⟩，其中vi是其视图号，non是请求中的随机数，preph和prepv是最新存储区块的哈希和视图。然后，节点将证书φrpy与相关区块b、其区块证书φb和提交证书φc一起发送给节点pj。

❸ 恢复节点等待接收来自不同节点的至少f+1个恢复回复。接下来，节点从所有回复中选择具有最高视图的领导者回复。它将其视图设置为领导者的视图加二，并使用来自回复的信息更新其状态preb。然后，它以f+1个回复证书和来自领导者的回复为输入调用TEErecover函数。该函数将进行以下检查：

• 所有回复证书都有有效的non和id。non与请求中的匹配，id是其身份。

• 来自领导者的证书在集合中，并且具有最高的视图。

• 有f+1个具有有效签名的回复证书。

通过这些检查后，该函数根据领导者证书中的信息更新状态，即vi、preph、prepv。最后，节点进入新视图并向当前领导者发送视图证书。恢复协议完成。

当当前视图v的领导者进入恢复程序时，它无法从当前视图的领导者获得恢复回复。它必须等待下一个领导者被选出。此外，如果恢复节点在给定时间内无法从领导者那里收集到f+1个恢复回复和最新的回复（即具有最高视图号的恢复回复），它可以向其他节点发送新的恢复请求。

上述步骤中有两个关键点。首先，在f+1个恢复回复中，具有最高视图的回复必须来自该视图的领导者。原因是只有具有最高视图的领导者才有最新的状态信息。如果没有这个规则，就会有安全问题。例如，考虑一个5个节点的简单攻击案例，即{pi}i=1^5。节点p1是视图v的领导者。首先，p1向p2发送一个扩展已提交区块b0的区块b，p2向p1发送存储证书。然后，p2崩溃并从p3、p4和p5恢复其状态，这些节点没有区块b。在对节点p3和p4重复上述过程后，领导者p1可以提交只有它自己存储的区块b。后来，p1与其他节点分离，节点进入视图v+1，领导者为p2。最后，领导者p2从p3和p4获取视图证书，并可以提议扩展区块b0的区块b′。这个冲突的区块b′将被提交，违反了安全性。

其次，当节点从f+1个恢复回复中获得最高视图v′时，它不能为该视图发送任何消息。这是因为节点不知道它在重启前是否已在此视图中发送过消息。特别是，节点必须将其视图设置为v′+2以防止二义性。原因是由于新视图阶段的优化，节点可能因为接收到视图v中区块的提交证书而成为视图v+1的领导者，而大多数节点（包括视图v的领导者）仍停留在视图v。

**4.6 正确性分析**

在本节中，我们提供安全性和活性属性（第3.2节）的证明的高层次视图，并在附录B47中留下详细分析。在证明这两个属性之前，我们首先提供恢复的无二义性属性的证明草图。

定义1（无二义性）。如果一个节点在同一视图中发送两个区块（或存储）证书φ和φ′，则φ = φ′。

回滚弹性恢复的证明草图。对于无二义性属性，我们观察到在没有恢复的理想情况下，由于checker组件，节点只能为一个视图发送一个区块（或存储）证书。在节点在视图v崩溃的情况下，它将在完成算法3中的恢复程序后进入视图v′ ≥ v。因此，领导者（分别是，备份）不能在同一视图中发送二义性区块（分别是，存储）证书。

安全性和活性的证明草图。对于安全性，视图v中的已提交区块必须被视图v′ ≥ v中提议的所有区块扩展。具体来说，在Achilles中，区块要么由收集了f+1个存储证书的领导者直接提交，要么由已提交的子区块b′间接提交。如果区块是直接提交的，它已被至少f+1个节点存储。无二义性确保不存在视图v的其他区块。此外，该区块必须包含在至少一个视图证书中，并被选为父区块。因此，根据TEEaccum和TEEprepare函数，视图v′ ≥ v中的后续区块将扩展它。如果区块是由区块b′直接提交的，所有后续区块将扩展区块b′，也包括它自己。

关于活性，如果所有正确的节点在GST后进入相同的视图，并且有一个正确的节点作为领导者足够长的时间，领导者可以协调备份节点来提交其区块。具体来说，领导者可以从正确的节点收集f+1个视图证书，或者收集前一个视图中区块的提交证书来选择父区块。领导者必须能够获取所选的父区块及其所有祖先区块。然后，领导者可以创建并广播一个包含客户端事务的区块来扩展父区块。然后，该区块将被所有正确的节点存储和投票，从而被提交。

**5 评估**

我们基于Intel SGX实现了Achilles的原型。我们在Damysus实现的基础上开发了Achilles。我们在不同设置（如批处理大小）的局域网和广域网中评估Achilles的性能。我们将Achilles与Damysus、FlexiBFT和OneShot进行比较，以展示性能改进。由于Damysus和OneShot不考虑回滚问题，我们按照FlexiBFT使用持久计数器来提供相关预防。我们将这两个变种分别命名为Damysus-R和OneShot-R。我们还在故障情况下评估Achilles，以说明回滚弹性恢复的开销。我们旨在回答以下问题：

• Q1：与对比方案相比，Achilles在广域网和局域网中随着节点数量变化的性能如何？（第5.2节）

• Q2：Achilles在故障情况下的性能如何？（第5.3节）

• Q3：SGX相关操作引入了多少性能开销？（第5.4节）

**5.1 系统实现与设置**

实现。我们使用Intel SGX提供可信服务，并在Damysus实现的基础上开发Achilles。所有协议都用C++实现。我们定制了Damysus中的可信组件checker和accumulator。我们还为checker组件实现了回滚弹性恢复（第4节）。我们使用OpenSSL库实现带有prime256v1椭圆曲线的ECDSA签名，并使用Salticidae进行节点连接。

基准。我们考虑三个对比方案，Damysus-R、FlexiBFT和OneShot-R，介绍如下。

• Damysus-R。Damysus-R是带有回滚预防的Damysus变种。具体来说，只有checker组件需要回滚预防（第4.3节）。因此，当使用checker组件时，它必须将其状态与磁盘上的持久计数器一起存储，然后将计数器增加一。在我们的实验中，我们运行链式版本的Damysus，因为它使用流水线结构以获得更好的性能。

• FlexiBFT。为了公平比较，我们在相同平台上实现了FlexiBFT。此外，FlexiBFT的实现也像Damysus和Achilles一样使用链式结构来串行提交区块，但采用稳定的领导者模式，即领导者可以连续提议新区块而不触发超时。我们选择FlexiBFT作为对比方案，主要是为了说明即使使用较少，低性能计数器仍然对其性能有显著影响，特别是在局域网中。

• OneShot-R。与Damysus-R类似，OneShot-R是带有回滚预防的OneShot变种，它使用计数器来保护checker组件。

请注意，在我们的实现中，我们遵循FlexiBFT模拟对计数器的访问成本，而不限制计数器类型（如第2.1节中的基于硬件或软件的计数器）。这种模拟方法提供了灵活性，避免了对特定实现的依赖。

实验设置。我们在公共云服务上进行了所有实验。我们租用了多达91个支持SGX的实例，每个节点一个实例，所有进程都在配备8个vCPU和32GB RAM的专用虚拟机上运行，运行Ubuntu Linux 20.04。每个实例都配备了一个带宽为10Gbps的私有网络接口。我们考虑两种部署场景：局域网（LAN）和广域网（WAN）。局域网环境中的节点间RTT为0.1±0.02ms。由于支持SGX的实例的位置限制，我们使用NetEm模拟了一个节点间RTT为40±0.2ms的广域网环境。

参数设置。我们变化故障阈值f∈{1, 2, 4, 10, 20, 30}，区块包含200、400和600个事务（即批处理大小），使用0 B、256 B和512 B的事务负载。Achilles、Damysus-R和OneShot-R的总节点数为2f+1，FlexiBFT为3f+1。除了事务负载外，每个区块还包括8 B的元数据（即客户端和事务ID）。0 B的负载和400个事务数用于评估协议的开销，而其他负载和事务数集合则被选择来观察增加区块大小时的趋势。

我们使用三种指标来评估性能：1）吞吐量：每秒执行的事务数；2）端到端延迟：从客户端发送事务到这些事务被执行和回复的平均延迟；3）提交延迟：从领导者提出事务到这些事务被执行的平均延迟。在大多数实验中，我们遵循Damysus使用提交延迟来评估Achilles及其对比方案，这可以减轻客户端的影响，提供更公平的比较。相反，我们在图4中使用端到端延迟来说明系统的可扩展性。

**5.2 无故障性能**

**5.2.1 广域网性能。我们评估了Achilles和对比方案在广域网环境中的性能。**

Damysus-R、FlexiBFT和OneShot-R使用持久计数器进行回滚预防。为了公平地与其他使用持久计数器进行回滚预防的TEE辅助BFT协议进行比较，我们测量了它们进行状态写入/读取操作的延迟，如附录C.1 47中的表4所示。根据该表，我们将计数器写入操作的延迟设置为20 ms。我们还考虑了{0, 10, 20, 40, 80} ms的写入延迟，并在附录C.2 47中评估了这些协议的性能。

1) 节点数量变化。图3a和3b展示了三种协议在故障阈值f变化时的吞吐量和延迟。所有协议采用每个区块400个事务，每个事务256 B负载。Damysus-R、FlexiBFT和OneShot-R的吞吐量低于Achilles，其中Damysus-R的吞吐量最低，因为每个节点需要访问昂贵的持久计数器两次才能提交一个事务，而FlexiBFT中只有领导者需要访问持久计数器一次。当f增加到20时，FlexiBFT的吞吐量低于Damysus-R，因为FlexiBFT的总节点数是3f+1，而不是Damysus-R中的2f+1，限制了其在故障增加时的可扩展性。类似地，Achilles保持最低的延迟，而当节点数量较少时，Damysus-R有最高的延迟。然而，随着节点数量的增加，FlexiBFT的延迟明显增加，因为总节点数3f+1使得故障增加对FlexiBFT的延迟有更显著的影响。

2) 负载大小变化。图3e和3f展示了三种协议在不同负载大小下的性能结果。负载分别为0 B、256 B和512 B。故障数为10，批处理大小固定为400。结果显示，当负载从0 B增加到512 B时，三种协议的吞吐量大约减少10%，而延迟增加约10%。这表明在广域网环境中，负载大小对三种协议的性能影响相对较小。

3) 批处理大小变化。图3i和3j说明了不同批处理大小对三种协议性能的影响。故障数为10，负载为256 B，批处理大小从200、400变化到600。随着批处理大小从200增加到600，三种协议的吞吐量显著增加约180%。延迟也略有增加，Achilles的延迟增加11.2%，Damysus-R增加3.7%，FlexiBFT增加约6.6%，OneShot-R增加约3.5%。这表明批处理大小的增加显著提高了三种协议的吞吐量，同时也导致延迟略有增加。

**5.2.2 局域网性能。为了最小化网络通信的影响，我们还在局域网中评估了Achilles。**

1) 节点数量变化。图3c和3d分别显示了所有协议在局域网中的吞吐量和延迟。由于在局域网环境中网络通信成本可以忽略不计，访问持久计数器的成本影响变得更加显著。这导致Damysus-R、FlexiBFT和OneShot-R维持相对较低的吞吐量。随着故障数从1增加到30，Damysus-R、FlexiBFT和OneShot-R的吞吐量分别下降19.2%、41.0%和10.0%，而延迟分别增加83.6%、21.1%和62.9%。因为持久计数器的成本成为主导因素，故障数的增加仅略微影响Damysus-R、FlexiBFT和OneShot-R的吞吐量和延迟。相比之下，不使用持久计数器的Achilles表现出显著更高的性能。Achilles的吞吐量约为Damysus-R的18到36倍，FlexiBFT的7到10倍，以及OneShot-R的8到18倍。

2) 负载大小变化。图3g和3h显示了三种协议在不同负载大小下的性能结果，设置与广域网中相同。设置与广域网中相同，用于评估负载大小的影响。类似地，访问持久计数器是Damysus-R、FlexiBFT和OneShot-R性能的主导因素，负载增加对它们的性能影响相对较小。随着负载从0 B增加到512 B，Damysus-R的吞吐量下降13.5%，延迟增加10.0%。对于FlexiBFT，吞吐量下降37.2%，延迟增加约63.4%。对于OneShot-R，吞吐量下降约37.6%，延迟增加约18.1%。相比之下，负载增加对Achilles的吞吐量和延迟造成更显著的变化，吞吐量下降约70%，延迟增加约300%。

3) 批处理大小变化。图3k和3l说明了不同批处理大小对三种协议吞吐量和延迟的影响。节点数为1500，负载为256 B，批处理大小从200、400变化到600。与广域网设置中的结果类似，批处理大小的增加显著提高了三种协议的吞吐量，同时略微增加延迟。此外，在局域网设置中，由于持久计数器成本的显著影响，当批处理大小从200到600时，Achilles的吞吐量和延迟明显优于其他三种协议。

**5.2.3 吞吐量与延迟。图4展示了三种协议在吞吐量增加直至系统饱和时的延迟。故障阈值设为10，负载为256 B，批处理大小为400。结果显示Achilles的最大吞吐量为9.38K。FlexiBFT的吞吐量下降且延迟增加，因为每个视图中的领导者都需要访问持久计数器，并且所需的总节点数为3f+1，导致最大吞吐量为4.95K。OneShot-R的性能比FlexiBFT差，最大吞吐量为4.23K，因为每个视图中的每个节点都必须访问持久计数器。Damysus-R显示最低的吞吐量，最大为2.66K，且延迟最高。这是由于与OneShot-R相比需要额外的通信轮次，以及每个节点都需要访问持久计数器。这表明Achilles由于其最优的2f+1弹性以及不需要访问持久计数器，相比另外两种协议取得了显著更好的性能。**

**5.3 恢复开销**

我们在局域网环境中评估了节点的恢复开销。我们测量了从节点重启TEE中的可信组件到完成恢复并加入正常情况操作所需的时间。恢复过程主要包含两部分。第一部分是初始化过程，其中节点建立连接并重启Intel SGX。第二部分是节点完成算法3中的程序。表2列出了不同节点数下恢复所需的时间。从表中可以看出，随着节点数的增加，初始化和恢复协议所消耗的时间只有轻微增加。这表明Achilles的恢复开销相对较小，使节点能够快速恢复并重新加入系统。

**5.4 开销分析**

为了更好地理解使用SGX的开销，我们实现了Achilles的一个新变种，称为Achilles-C，它在SGX enclave外部运行可信组件。Achilles-C可以被视为CFT协议的链式版本。与Achilles-C的比较突显了使用SGX引入的开销。此外，我们还将Achilles与BRaft（版本：1.1.1）进行比较，BRaft是Raft协议的一个开源实现。与BRaft的比较旨在说明Achilles在最先进的CFT协议基础上提供BFT保证的成本。

表3列出了这些协议在局域网网络中不同f值下的最大吞吐量和延迟。批处理大小为400，每个事务的负载大小为256 B。评估结果还表明，Achilles能够分别达到Achilles-C和BRaft在f=10设置下吞吐量的76.3%和97.3%，同时维持SGX提供的安全优势。

**6 讨论**

**6.1 响应性和并行性问题**

Gupta等人28指出了现有TEE辅助BFT协议的限制性响应能力和缺乏并行性问题。为了解决这些问题，Gupta等人将容错阈值从n=2f+1降低到3f+1（详见第1节）。然而，Bessani等人10表明容错放宽是不必要的，一些简单的修改可以轻松解决这两个问题。接下来，我们介绍这些修改，特别是Achilles中采用的修改。

限制性响应能力。在TEE辅助BFT协议中，客户端通常从共识节点接收f+1个回复以确保事务被提交。然而，如果事务由f+1个节点的法定人数提交，可能只有一个诚实节点回复客户端。因此，客户端无法收集足够的回复，直到所有诚实节点通过检查点进行同步，导致限制性回复响应能力。

Bessani等人10指出，这个问题并非TEE辅助共识特有。这个问题已在PBFT15和BFT-SMaRt11中报告，其中事务读取可以通过收集n-f个回复来避免运行共识9。为解决这个问题，有两个简单的解决方案。首先，当节点提交并回复客户端时，它可以向其他节点广播一个Decision消息来提醒它们[7, 44]。其次，领导者可以执行事务并在区块中包含结果。当区块被提交时，任何节点都可以向客户端发送证书以证明结果。这种方法在以太坊65等区块链中广泛使用。Achilles采用第二种方法，并利用TEE的完整性来避免使用证书。

缺乏并行性。采用二义性检测[39, 68]的TEE辅助BFT协议要求节点串行访问可信组件来认证消息，强制消息序列无间隙10。这种序列化禁止了像PBFT15这样的协议的并行性。Bessani等人10表明，诸如流水线、并行共识实例和面向共识的并行化等技术可以缓解并行性问题。Achilles遵循链式BFT协议的链接精神，实现线性消息复杂度并支持频繁的领导者轮换。链结构，即在区块中链接批处理事务，已经可以实现良好的可扩展性和性能。我们可以通过并行共识实例[27, 40, 55]进一步并行化Achilles，这留给后续工作。

**6.2 动态重构**

Achilles不考虑动态重构，这允许节点随时间动态添加或删除。尽管BFT共识中的重构在先前工作[11, 22]中已被充分探索，但在存在回滚问题的情况下将动态重构与恢复程序集成仍然对Achilles而言具有挑战性。这是因为重启节点依赖配置信息来确定恢复请求应该发送给哪个节点。由于回滚问题，节点可能使用过时的配置信息，可能加入旧的节点组来处理客户端事务。这可能违反安全性属性。此外，如果在恢复期间发生重构，一些先前配置中的节点被移除，重启节点可能无法收集足够的恢复响应，违反活性属性。因此，实现动态重构同时避免这些安全漏洞需要大量努力，留给未来工作。

**6.3 过多故障**

在Achilles中，我们假设不超过f个节点同时重启。如果没有这个假设，系统可能会失去活性，因为没有节点能够从收集f+1个回复中恢复。然而，这种限制并非我们工作特有。无盘CFT协议，如VR38和Paxos[16, 35]的变种，也共享这一约束（不超过f个节点同时崩溃）。

**7 附录**

**A 工件附录**

本附录提供了关于如何复现我们论文中呈现的结果的详细指南。我们基于Damysus框架20构建了Achilles的原型，并遵循了其大部分实验设置。下面，我们描述了我们系统的结构以及复现我们结果所需的步骤。

**A.1 摘要**

Achilles是一种高效的TEE辅助BFT协议，提供状态机复制服务。其主要创新点是回滚弹性恢复和对正常情况操作的优化。我们基于链式Damysus（Damysus的链式版本20）实现了Achilles。我们在代码中为Achilles添加了一个名为ACHILLES的宏，其相应的实现位于#if defined(ACHILLES)下。扩展包括修改可信组件，即Checker和Accumulator（第4.3节），并将Damysus的三阶段正常情况操作减少为两阶段。我们还为重启的TEE添加了回滚弹性恢复。

**A.2 描述与要求**

**A.2.1 如何访问**

用于产生实验结果的代码可在Github仓库和Zenodo上公开获取，其中有5个分支：Achilles、FlexiBFT、Damysus、Oneshot和Achilles-Recovery。最后一个分支用于评估恢复过程的性能。

**A.2.2 硬件依赖**

我们在阿里云ECS机器上进行评估，每个节点使用一个位于香港区域的ecs.g7t.large SGX支持实例。所有进程都在专用虚拟机上运行，配备8个vCPU和32GB RAM，运行Ubuntu Linux 20.04。每台机器都配备了一个带宽为5 Gbps的私有网络接口。由于SGX支持实例的位置限制，我们使用NetEm30模拟了节点间RTT为0.1±0.02ms的局域网环境和节点间RTT为40±0.2ms的广域网环境。更多详情见第5.1节。

**A.2.3 软件依赖**

C++ 14, Python 3.8.10。

**A.2.4 基准测试**

无。

**A.3 设置**

详细的设置说明可在仓库的README文件中找到。部署过程通过位于部署目录中的脚本自动化。这些脚本允许您在任何支持SGX的机器上部署节点和客户端网络，进行实验，分析结果，并生成摘要数据。

要在阿里云上部署实验，您必须设置阿里云账户并注册SSH密钥。仓库包含初始化阿里云ECS实例的脚本，简化了部署过程。首先，您需要启动一个ECS实例作为控制器，负责部署实例、管理实验和处理实验结果。我们提供自动化脚本和Python文件以便于实例部署和实验。有关如何使用控制器部署实验的更多详细信息，请参阅A.4节和仓库的README文件。尽管提供的实验说明是针对阿里云的，但您可以在任何支持SGX的机器上运行实验。

**A.4 评估工作流程**

**A.4.1 主要声明**

\* (C1)：Achilles显著提高了Damysus-R、FlexiBFT和OneShot-R的性能，在广域网实验中故障数为30时，吞吐量分别提高约2倍、7倍和0.2倍，同时延迟分别降低约81%、83%和52%。这些改进在实验E1（稍后介绍）中得到证明，结果如图3所示。

\* (C2)：在局域网实验中故障数为30时，Achilles使Damysus-R的吞吐量提高17倍，FlexiBFT提高6倍，OneShot-R提高7倍，同时延迟分别降低98%、88%和92%。这些改进在实验E2（稍后介绍）中得到证明，结果如图3所示。

\* (C3)：Achilles采用回滚弹性恢复：Achilles的恢复开销相对较小，因为随着节点数量的增加，恢复消耗的时间只显示出轻微增加。这在实验E3（稍后介绍）中得到证明。

**A.4.2 实验**

我们首先概述执行Achilles实验所需的一般步骤。

[准备] 要为实验做准备，请按照以下步骤操作：

\* 阿里云账户和配置：登录阿里云。进入弹性计算服务界面，选择香港区域，并启动一个实例作为控制器。

\* 连接到控制器：连接到控制器，所有实验文件都在/root/Achilles目录下。

[执行] 准备完成后，导航到/root/Achilles目录。如果您想使用默认配置在本地执行，可以直接使用命令python3 run.py --local --p1。如果您想部署分布式实验，请按照以下步骤执行实验：

\* 启动实例：进入部署目录。执行bash cloud\_deploy.sh根据config.json中指定的配置初始化新实例。运行bash cloud\_config.sh为这些实例配置SGX操作环境。使用命令tmux a检查环境配置的执行情况，使用命令exit退出tmux终端。

\* 进行实验：返回/root/damysus\_updated目录。运行python3 run.py --p1 --faults {faults} --batchsize {batchsize} --payload {payload}进行单个实验。此命令使用Achilles协议执行一个实验。要运行一系列实验，请执行/root/Achilles/scripts/目录中的脚本，如表4所示。例如，运行bash scripts/faults\_WAN.sh将生成图3a和图3b的数据。如果操作过程中出现错误或问题，使用python3 close.py停止每个实例上的进程。

\* 关闭实例：使用python3 deployment/delete\_instances.py终止阿里ECS中所有运行的实例。

[结果] 实验完成后，结果将在stats.txt中可用。该目录包含一组全面的详细统计信息，包括吞吐量和延迟。查看这些文件以分析和解释实验结果。例如，"Achilles\_1\_256\_400\_0, 18.1715414, 26.598315"表示Achilles协议在1个故障、每个区块400个事务、每个事务256 B负载的情况下，吞吐量和延迟分别为18.1715414K TPS和26.598315ms。

实验(E1)：[无故障情况下的吞吐量和延迟] [1人时 + 3计算时]：此实验旨在评估系统在广域网环境中的峰值性能，重点关注吞吐量和延迟。该实验涉及运行具有不同配置的各种场景，包括不同数量的故障、负载和批处理大小，并将结果与其他协议进行比较（图3a、图3b、图3e、图3f、图3i和图3j）。相应的执行文件位于scripts/目录中，如表4所示。

实验(E2)：[无故障情况下的吞吐量和延迟] [1人时 + 3计算时]：此实验类似于实验(E1)，评估系统在局域网环境中的峰值性能，重点关注吞吐量和延迟（图3c、图3d、图3g、图3h、图3k和图3l）。执行文件的命名类似，位于scripts/目录中。

实验(E3)：[恢复过程] [1人时 + 1计算时]：此实验评估系统在崩溃故障存在的情况下的鲁棒性（表2）。切换到Achilles-Recover分支，并运行python3 runRecover.py --p4 --faults faults来测试初始时间和恢复时间。

表4. 执行文件和相应的图表

参考文献

|  |  |
| --- | --- |
| **角标** | **文献信息** |
| 1 | Achilles: Efficient TEE-Assisted BFT Consensus via Rollback Resilient Recovery |