软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(6):1346−1360 [doi: 10.3724/SP.J.1001.2013.04395]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

拜占庭系统技术研究综述*

范 捷[1], 易乐天[1], 舒继武[1,2]

[1](清华大学 计算机科学与技术系,北京 100084)

[2](灾备技术国家工程实验室(清华大学),北京 100084)

通讯作者: 舒继武, E-mail: shujw@tsinghua.edu.cn

摘 要: 随着分布式系统规模的增大,设计复杂度也不断提升,系统可靠性所面临的问题也越来越严峻.由于拜占庭协议能够容忍包括人为失误、软件 bug 和安全漏洞等各种形式的错误,其系统技术和实现方法越来越受到研究者们的重视.介绍和总结了目前拜占庭系统技术的研究成果,分析了目前拜占庭系统的研究现状,并探讨了拜占庭系统的发展趋势.通过分析得出:1) 拜占庭系统性能上仍然与已经实用的非拜占庭系统相距较大,占用资源数量仍然较多,需要进一步研究其性能和资源优化技术;2) 通过检测错误或者定期修复来降低系统中的错误,是延长系统可持续运行时间的方法,需要研究新的、高效的全面检测拜占庭服务器、合理定期修复等保障系统可持续运行的方法;3) 实际应用背景和需求及其特定错误类型的处理方法对拜占庭协议和功能等提出了不一样的要求,需要研究拜占庭系统在实际中的应用和可用性.

关键词: 可靠性;容错;拜占庭系统;状态机;Quorum

中图法分类号: TP302 **文献标识码**: A

中文引用格式: 范捷,易乐天,舒继武.拜占庭系统技术研究综述.软件学报,2013,24(6):1346−1360. http://www.jos.org.cn/1000-9825/4395.htm

英文引用格式: Fan J, Yi LT, Shu JW. Research on the technologies of Byzantine system. Ruan Jian Xue Bao/Journal of Software, 2013,24(6):1346−1360 (in Chinese). http://www.jos.org.cn/1000-9825/4395.htm

Research on the Technologies of Byzantine System

FAN Jie[1], YI Le-Tian[1], SHU Ji-Wu[1,2]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](National Engineering Laboratory for Disaster Backup and Recovery (Tsinghua University), Beijing 100084, China)

Corresponding author: SHU Ji-Wu, E-mail: shujw@tsinghua.edu.cn

Abstract: Nowadays, in order to resolve the reliability problem in an enlarging distributed system, Byzantine fault tolerant system has researched popularly for its ability of tolerating arbitrary faults. In this paper, the definitions of Byzantine system and the estimation methods of improving the performance of Byzantine system are introduced. After that, some unresolved problems and some future development trends will be indicated. Finally, after analyzing the status of studies, several conclusions are drawn: 1) The cost of running a Byzantine system is still much higher than non-Byzantine system. Plans to increase the performance and decrease the overhead are need to be explored in further study. 2) While detecting Byzantine faults and proactive recovery can keep Byzantine system from breaking down, they still have some drawbacks. How to eliminate the drawbacks should be studied. 3) Different applications require different aspect of optimization. How to make practical Byzantine systems are needed to be studied.

Key words: reliability; fault tolerance; Byzantine system; state machine; Quorum

在分布式系统中,只有每一个组件的协调合作,整个系统才能正常的运作.如果其中某些组成部分出现问

* 基金项目: 国家自然科学基金(60925006); 国家高技术研究发展计划(863)(2013AA013201)
收稿时间: 2012-04-11; 定稿时间: 2013-03-07

题,那么将威胁系统的正常运作以及所涉及的数据.为了增加计算机的计算能力和存储能力,人们组建分布式系统并且将大量的计算作业和数据布置在分布式系统中,例如亚马逊 ASW[1],Drop Box[2],Windows Azure[3]等等.

随着分布式系统规模的增长,系统软件的程序复杂度不断增加,程序设计的难度也在增大.程序设计人员很难保证整个系统没有软件错误和安全漏洞,甚至连错误和漏洞的种类也不能完全掌握.除此之外,系统在实际部署之后,员工的不当操作也可能导致系统问题.这些隐藏的漏洞和潜在的不当操作[4],严重威胁着计算数据的安全性以及系统的可靠性等.例如:在 2011 年 4 月,Sony 公司旗下网络游戏提供网站 PlayStation Network (PSN)以及云音乐服务 Qriocity 网络遭受黑客入侵,导致 7 700 万用户账号泄露,甚至包括信用卡信息[5].这个事件不仅给用户造成了潜在的损失,也给 Sony 公司带来了极大的信用和财产损失;同样,2011 年 9 月,中国国内大型 B2C 网站——淘宝网,第三方软件“团购宝”发生软件错误,导致大量用户的交易被更改成 1 元,最后淘宝网不得不采取赔偿用户每人 10RMB 的方案挽回社会影响[6].2007 年 7 月,IT Policy Compliance Group 发布的报告显示:“数据丢失会对企业业务造成巨大影响,客户量及相关收入下降 8%,股价下降 8%”[7].

为了防范这些潜在错误,部署一个高效的容错系统来保护重要信息是十分重要的.但是在传统方法中,针对某些特定漏洞的容错方法并不能完全解决分布式系统的容错问题,比如:RAID 技术只能解决系统中的磁盘失效问题,却无法容忍由于网络问题而导致的传输错误;网络重传机制和校验码可以解决网络传输的问题,却无法保证软件在接收到数据之后是否会由于软件设计缺陷而导致数据被篡改.由于在分布式系统中所需要考虑的问题数量巨大,复杂琐碎,通过将传统的容错方法堆叠起来已经不能处理所有可能发生的意外情况,因此,人们需要一种能够容忍任何种类漏洞的容错方案.

拜占庭容错技术(Byzantine fault tolerant)能容忍任何形式的软件错误和安全漏洞,是一种解决分布式系统容错问题的通用方案.最初,拜占庭系统需要指数级的算法才能解决[8],随后,研究者提出了多项式级别的拜占庭协议,极大地降低了拜占庭协议的开销,使其在分布式系统中的使用成为可能[9].近年来,随着大规模云存储系统的部署和兴起,研究者开始关注如何简化拜占庭协议的设计,提高拜占庭系统的性能,用以容忍数据中心的复杂错误.目前,拜占庭协议已经可以嵌入到 HDFS,Zookeeper 等广泛使用的分布式存储系统中[9].

从目前的研究现状来看,拜占庭系统的设计主要分为状态机拜占庭协议和 Quorum 拜占庭协议.两种协议在功能上的主要区别在于:前者需要给所有的请求安排序号,所有请求必须按顺序执行,主要用于对于系统状态敏感的分布式计算系统中;后者不需要为请求安排序号,多个请求可以同时执行,主要被应用于分布式存储系统中.拜占庭容错技术的研究方向主要是降低系统开销,缩小与目前实际应用中的系统之间的差距.虽然从问题提出至今已经有 30 年的研究历史,拜占庭系统仍然没有广泛的在实际中应用:一方面是由于性能上仍然与非拜占庭系统有较大差距;另一方面是,存储资源和计算资源的占用要比非拜占庭系统多.因此,不断深入研究拜占庭系统,探究优化性能、减少资源占用的方法具有重要的意义.

1 背景及模型定义

拜占庭系统来源于拜占庭将军问题[8].在古代,一些拜占庭的将军率领他们的部队要攻占敌人的一个城池,每个将军只能控制他们自己的部队并且通过信使传递消息给其他的将军(这条消息只有参与的两个将军知道,其他的将军可以打听,但是不能验证消息的正确性).如果这些将军中有些是来自敌方的奸细,那么如何使忠诚的将军仍然可以达成一个不差的行动协议而不受奸细的挑拨,就是拜占庭将军问题.

从上面的描述可以发现,拜占庭将军问题和分布式系统是相似的.分布式系统的每一个服务器可以类比成将军,服务器之间的消息传递可以类比成信使,服务器可能会发生错误而产生错误的信息传达给其他服务器.因此,文献[8]给出了拜占庭系统的定义.

拜占庭容错系统是指:在一个拥有 n 台服务器的系统中,整个系统对于每一个请求需满足以下两个条件[8]:

1) 所有非拜占庭服务器使用相同的输入信息,产生一致的结果;

2) 如果输入的信息正确,那么所有非拜占庭服务器必须接受这个信息,并计算相应的结果.

与此同时,在拜占庭系统的实际运行过程中,一般假设整个系统中发生故障的服务器(这类服务器称为拜占

庭服务器)不超过 f 台,并且每个请求还需要满足两个指标[10]:

- 安全性(safety):任何已经完成的请求都不会被更改,它可以被之后请求看到;
- 活性(liveness):可以接受并且执行非拜占庭客户端的请求,不会被任何因素影响而导致非拜占庭客户端的请求不能执行.

拜占庭系统通常被部署在大规模数据中心,目前普遍采用的假设条件包括:1) 拜占庭节点(服务器和客户端)的行为可以是任意的,拜占庭节点之间可以共谋;2) 节点之间的错误是不相关的.为达到这一要求,可以采用为节点安装不同的操作系统,不同节点部署不同版本、不同程序员研发的程序方法;3) 节点之间通过异步网络连接(例如 Internet 网络),网络中的消息可能丢失,乱序到达,延时到达;4) 服务器之间传递的信息,第三方可以知晓,但是不能窜改、伪造信息的内容和验证信息的完整性.这一点通过加密技术实现,在后文描述中用 $\langle m\rangle_{\sigma_i}$ 表示消息 m 由节点 i 签名后发出.

2 状态机拜占庭系统

状态机拜占庭系统的特点是整个系统共同维护一个状态,所有服务器采取一致的行动,一般包括 3 种协议[10−17]:一致性协议(agreement)、检查点协议(checkpoint)和视图更换协议(view change).系统正常运行在一致性协议和检查点协议下,视图更换协议则是只有在主节点出错或者运行缓慢的情况下才会启动,负责维系系统继续执行客户端请求的能力.为了便于理解,本节首先介绍这 3 种协议的原理,然后再介绍基于这 3 种协议的状态机拜占庭系统及其技术.

2.1 状态机拜占庭系统的核心协议

2.1.1 一致性协议

一致性协议的目标是使来自客户端的请求在每个服务器上都按照一个确定的顺序执行.在协议中,一般有一个服务器被称作主节点(primary),负责将客户端的请求排序;其余的服务器称作从节点(backup),按照主节点提供的顺序执行请求.所有的服务器都在相同的配置信息下工作,这个配置信息被称作 view,每更换一次主节点,view 就会随之变化.

一致性协议至少包含 3 个阶段:发送请求(request)、序号分配(pre-prepare)和返回结果(reply).根据协议设计的不同,可能包含相互交互(prepare),序号确认(commit)等阶段.

一致性协议解决一致性的方法主要有:1) 服务器之间两两交互,服务器通过将自己获得的信息传递给其他的服务器;2) 由客户端收集服务器的信息,将收集的信息制作成证明文件再发送给服务器.对于一个包含 $3f+1$ 台服务器的拜占庭系统,需要收集到 $2f+1$ 台服务器发送的一致信息,才能保证达成一致的非拜占庭服务器数量大于拜占庭服务器数量.

2.1.2 检查点协议

拜占庭系统每执行一个请求,服务器需要记录日志.如果日志得不到及时的清理,就会导致系统资源被大量的日志所占用,影响系统性能及可用性.另一方面,由于拜占庭服务器的存在,一致性协议并不能保证每一台服务器都执行了相同的请求,所以,不同服务器状态可能不一致.例如:某些服务器可能由于网络延时导致从某个序号开始,之后的请求都没有执行.因此,拜占庭系统中设置周期性的检查点协议,将系统中的服务器同步到某一个相同的状态.因此,周期性的检查点协议可以定期地处理日志,节约资源,同时及时纠正服务器状态.

处理日志主要解决的问题就是区分哪些日志可以清理,哪些日志仍然需要保留.如果一个请求已经被 $f+1$ 台非拜占庭服务器执行,并且某一服务器 i 能够向其他的服务器证明这一点,那么 i 就可以将关于这个请求的日志删除.目前,协议普遍采用的方式是服务器每执行一定数量的请求,就将自己的状态发送给所有的服务器并且执行一次该协议,如果某台服务器接收到 $2f+1$ 台服务器的状态,那么其中一致的部分就是至少有 $f+1$ 非拜占庭服务器经历过的状态,因此,这部分的日志就可以删除,同时将自己状态更新至较新状态.

2.1.3 视图更换(view change)

在一致性协议里,已经知道主节点在整个系统中拥有序号分配,请求转发等核心能力,支配着这个系统的运行行为.然而一旦主节点自身发生错误,就可能导致从节点接收到具有相同序号的不同请求,或者同一个请求被分配多个序号等问题,这将直接导致请求不能被正确执行.视图更换协议的作用就是在主节点不能继续履行职责时,将其用一个从节点替换掉,并且保证已经被非拜占庭服务器执行的请求不会被篡改.

视图更换协议一般有两种触发方式:1) 只由服务器触发[10,11],这一类触发方式中,判断服务器一致性是否达成的工作是由服务器自身负责,客户端不能从请求的整个执行过程中获得服务器运行状况的信息;2) 客户端触发[12,13],这一类触发方式中,客户端一般负责判断服务器是否达成一致,如果不达成一致,那么就能判断服务器运行出现问题,如果是主节点的问题就会要求服务器更换主节点.

视图更换协议需要解决的问题是如何保证已经被非拜占庭服务器执行的请求不被更改.由于系统达成一致性之后至少有 f+1 台非拜占庭服务器执行了请求,所以目前采用的方法是:由新的主节点收集至少 2f+1 台服务器的状态信息,这些状态信息中一定包含所有执行过的请求;然后,新主节点将这些状态信息发送给所有的服务器,服务器按照相同的原则将在上一个主节点完成的请求同步一遍.同步之后,所有的节点都处于相同的状态,这时就可以开始执行新的请求.

2.2 早期的状态机拜占庭系统

Castro 和 Liskov 在 1999 年提出了 Practical Byzantine Fault Tolerance(PBFT)系统[10],首次将拜占庭协议的复杂度从指数级[8]降低到多项式级别,使拜占庭协议在分布式系统中应用成为可能.PBFT 的一致性协议如图 1 所示,每一个客户端的请求需要 5 个阶段才能完成.其通过采用两次两两交互的方式在服务器达成一致之后再执行客户端的请求.由于客户端不能从服务器端获得任何服务器运行状态的信息,因此,PBFT 协议中主节点是否发生错误只能由服务器监测.如果服务器在一段时间内都不能完成客户端的请求,则会触发视图更换协议.

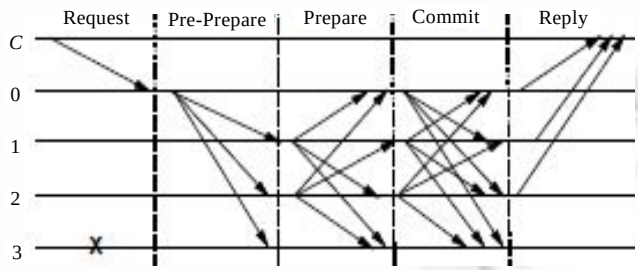


Fig.1 PBFT protocol communication pattern[10]

图 1 PBFT 运行示意图[10]

PBFT 采取的设计思路是将所有的工作都放在服务器端进行,例如达成一致性、监测拜占庭主节点等.因此,它的一致性协议设计较复杂,其中有两个阶段需要服务器之间的两两交互,数据处理量大,计算复杂.

Cowling 等人在 2006 年提出了 HQ[11]来改进 PBFT.图 2 为 HQ 运行示意图,首先,HQ 不需要主节点分配序号,请求的执行序号由 3f+1 台服务器中的 2f+1 台共同决定(write 1 过程),客户端判断服务器是否出于相同状态,从而减少请求执行过程中所需阶段数量;其次,HQ 不需要服务器之间两两交互来保证一致性,客户端收集到的状态信息已经可以保证服务器的一致性.

HQ 没有从根本上改变 PBFT 的结构,主要针对当客户端在发送请求时没有出现竞争的情况进行优化.在这种情况下,HQ 将写入过程分成两个阶段:第 1 阶段,客户端发送请求的同时收集服务器的状态信息.如果有 2f+1 台服务器都处于相同状态,并且同意执行客户端的请求,那么客户端发起第 2 阶段,使服务器执行客户端请求;否则,说明请求遇到竞争,执行 PBFT 的流程,由主节点安排请求执行顺序.

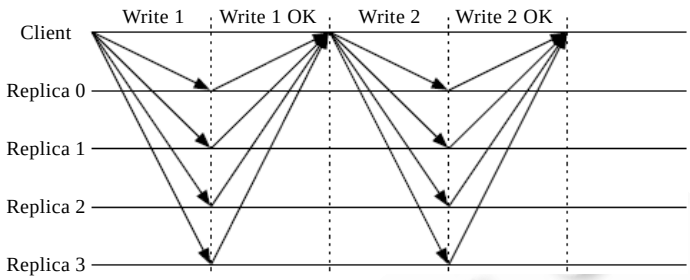


Fig.2 HQ protocol communication pattern[11]

图 2 HQ 运行示意图[11]

2.3 基于Speculation的拜占庭系统

Dahlin 等人在 2007 年提出了 Zyzzyva 和 Zyzzyva5 协议[12],将 speculation 技术引入了拜占庭协议,其主要思想是,服务器绝大部分时间处于正常的状态,因此不用每一个请求都在达成一致性之后再执行,只需要在错误发生之后再达成一致性即可.

与传统的协议相比,speculation 机制的不同主要在于一致性协议部分.图 3 为 speculation 的执行流程,服务器收到客户端的请求之后,并不像传统的协议一样首先进行代价较大的两两交互,而是直接执行了客户端的请求.只要客户端接收到所有服务器反馈的信息(执行结果、服务器状态、执行历史等),并且这些信息是一致的,客户端认为结果正确,并返回给上层应用;否则,客户端在接收到 $2f+1$ 服务器反馈的信息后,就执行一个序号确认的阶段,告诉至少 $f+1$ 台非拜占庭服务器:已经有至少 $f+1$ 台非拜占庭服务器执行了请求.

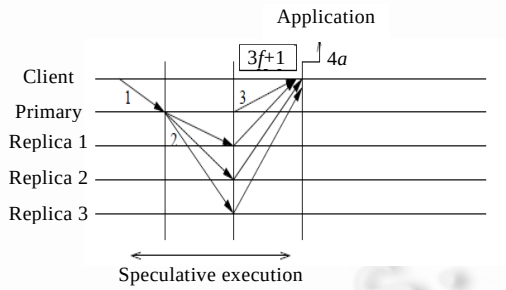


Fig.3 Zyzzyva protocol communication pattern for gracious executuion[12]

图 3 Zyzzyva 不存在拜占庭服务器时的示意图[12]

如果某个请求 m 在序号 n 被执行后,客户端收到 $3f+1$ 个一致的执行结果,那么在视图更换协议中一定有 $f+1$ 台服务器,反馈在序号 n 执行了请求 m.因此说明 speculation 技术是安全的,并不会因为缺少服务器之间的交互而导致系统崩溃.

总的来说,speculation 允许服务器提前执行客户端的请求,而不需要服务器之间为了达成一致的行动而进行十分耗时的两两交互.服务器是否达成一致的问题则交给客户端来判断,如果服务器执行结果是一致的,那么客户端采用这个结果;如果不一致,那么客户端丢弃这个结果,并且反馈给服务器触发视图更换协议.这样虽然可能暂时导致系统不一致,但是并不会影响非拜占庭的客户端请求的执行.这是因为,如果客户端是拜占庭的,那么即使系统结果不一致也没有关系;如果客户端是非拜占庭的,发现系统不一致之后一定会触发视图更换协议将系统重新调整到一致状态.

表 1 对比了传统拜占庭协议与采用 speculation 技术的拜占庭协议之间的不同.从数值上观察,后者完成单一请求所需阶段数较少.理论证明系统要达成一致,至少需要两个阶段(不包括主节点分配序号的阶段),所以

speculation 技术从理论上达到了最优,从而降低了系统响应延时[14,15].除此之外,服务器所需计算量也有所减少,增大了吞吐率.图 4 显示了采用 speculation 技术的拜占庭系统与传统拜占庭系统的性能比较(图中 Q/U 属于 Quorum 系统,将会在后文介绍;Unreplicated 为不采用容错技术的性能),Zyzzyva 和 Zyzzyva5 较大幅度地减少了响应时间,同时较大幅度的增加吞吐率.

Table 1 Comparison between the procotol with speculation and the procotols without speculation

表 1 不采用 speculation 技术的拜占庭协议与采用 speculation 技术的拜占庭协议之间的比较

	PBFT	HQ	Zyzzyva
服务器数量	3*f*+1	3*f*+1	3*f*+1
单一请求所需阶段数	4	4	3
瓶颈服务器计算量(MACs/op)	2+8*f*+1	4+4*f*	2+3*f*

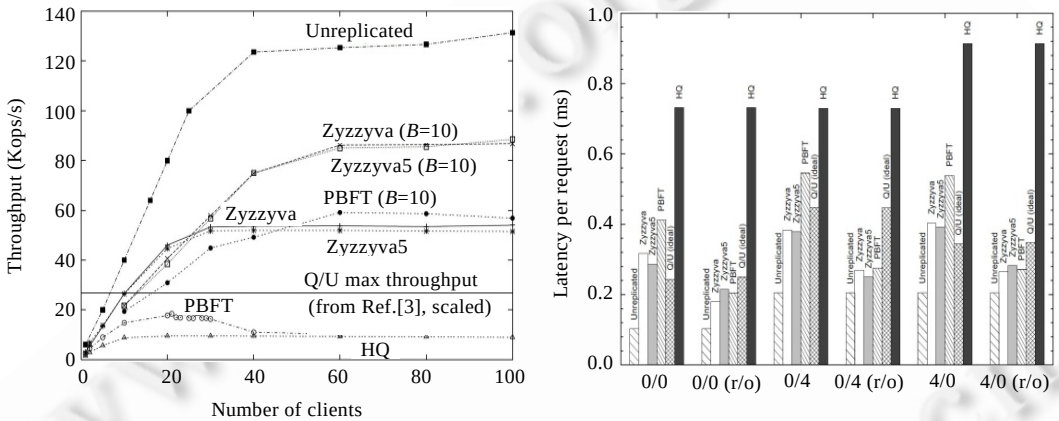


Fig.4 Comparison between different protocols[12]

图 4 状态机协议之间的对比[12]

虽然 speculation 技术已经在很大程度上提升了拜占庭系统的性能,但是在系统中存在拜占庭服务器时候,系统的性能却会大幅度下降.为了解决该问题,Suri 等人提出了 Srcooge 协议[13].Scrooge 协议继承了 Zyzzyva 协议的大部分框架,Scrooge 本质上是增加服务器数量(4*f*)换取系统的性能.通过使用应答子集(replier quorums)和消息历史(message histories)两个技术,前者使得即使有 *f* 台拜占庭服务器存在,只要不在应答子集中,系统还是可以高效运行;即使在应答子集中,也可以通过改变应答子集将相应缓慢的服务器剔除.后者使得整个协议所需要服务器数量为 4*f*.

除了服务器端的 speculation,Chen 等人在 2009 年提出了基于客户端 speculation(client speculation)技术的拜占庭系统[16].其核心思想是,客户端不需要收集到所有服务器的反馈之后再进行下一个操作,而是可以在收到第 1 个服务器的反馈之后就认为结果是正确的并且进行下一步操作.客户端的 speculation 在系统主节点没有出错的情况下可以降低系统的响应时间,但是一旦主节点出现错误,将导致之前没有确认达成一致的请求全部重新执行.

2.4 拜占庭锁技术

拜占庭锁技术[17]是拜占庭协议的扩展,由 Ganger 等人提出并应用在 Zzyzx 协议中.通过利用 IO 绝大多数时间里不会出现竞争的特性[20,21],达到降低服务器响应时间和提高吞吐率与可扩展性的目的.通过拜占庭锁技术,实现将每个请求响应所需阶段数下降到 2,同时增强系统可扩展性.

拜占庭锁技术包括加锁和解锁两个部分:客户端在进行文件操作之前需要进行加锁;如果该文件已经被其他客户端锁住,则需要在加锁前对其进行解锁.加锁和解锁都是通过利用已有的拜占庭协议(例如 Zyzzyva,

PBFT)事先让服务器达成一种特殊一致.这类一致性特殊在于并不是让服务器协商客户端请求的执行顺序,而是确定给予哪一个客户端文件操作的权力.客户端在获得文件操作的权力之后,由于只有一个客户端可以操作文件,所以不再需要主节点分配执行序号,转而直接控制所有服务器的行为.图 5 是使用拜占庭锁后,文件操作示意图.

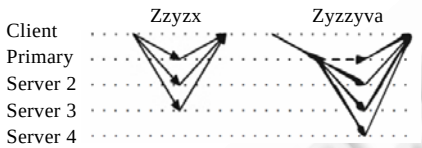


Fig.5 Comparison between Zzyzx and Zyzzyva[17]

图 5 Zzyzx 和 Zyzzyva 运行流程对比[17]

虽然拜占庭锁技术能带来性能的较大提升,但是缺陷也十分明显.由于加锁后某一个文件只能有一个客户端访问,所以当数据共享频繁时,拜占庭锁将会导致不断进行加锁和解锁的过程,来带较大的开销.因此,拜占庭锁一般用于的数据共享不频繁的工作负载中.Zzyzx 实验结果显示,当数据交换不频繁时,拜占庭锁可以使吞吐率提升 2.2~2.9 倍,响应时间下降 39%~43%;当数据交换频繁时,Zzyzx 不再采用拜占庭锁,而是直接使用底层的 Zyzzyva,因而其效率仍然能与 Zyzzyva 持平.

2.5 分离一致性与执行请求

拜占庭系统达成一致需要至少 3*f*+1 台服务器,但是执行客户端请求的服务器却不需要这么多.因此,可以将达成一致与执行请求分开[14],一部分服务器负责一致性协议,一部分服务器负责执行请求.对于执行时间较长的请求,减少执行请求的服务器数量可以较大程度地提升系统吞吐率.

由之前的介绍可以发现,如果客户端请求的执行结果自身可以验证其正确性,或者执行请求的服务器都不是拜占庭节点,只需要安排 *f*+1 台服务器执行请求,就可以保证客户端至少能收到 1 个正确的反馈;如果结果自身不能验证正确性且不能保证执行请求的服务器是非拜占庭节点,那么至多只需要安排 2*f*+1 台服务器执行请求,就可以保证客户端至少能收到 *f*+1 个正确的反馈.

Wood 等人在 2011 年提出 ZZ[15],这是一种广泛适用于当前状态机拜占庭系统的协议,其特点是通过虚拟化技术将目前存在的状态机拜占庭协议执行请求的服务器数量从 2*f*+1 台减少到 *f*+1 台.工作原理是:在没有服务器发生错误时,只有 *f*+1 台服务器执行客户端的请求,而其余的服务器处于休息状态;当执行请求的服务器发生错误时,客户端则通过虚拟化技术,快速启动其他的服务器参与执行请求.由于服务器绝大多数时间是正确的,所以 ZZ 在绝大多数时间下只需要 *f*+1 台服务器执行系统请求,使得系统吞吐率较不采用 ZZ 技术的系统提升约 33%.

Distler 等人提出了 ODRC 协议,也通过分离一致性协议和执行请求的方法,将执行请求服务器数量降低到 *f*+1[22].所不同的是,ODRC 仅仅依靠现有的拜占庭技术并没有采用额外的技术(例如虚拟化技术),也不让多余的服务器处于休息状态,主要目的是合理地分配当前服务器资源,使得 3*f*+1 台服务器能够充分利用.其主要思想是,在服务器达成一致之后以及服务器执行请求之前添加一个选择执行服务器的阶段(selector),选择器根据当前系统所处的状态,选择指定数量的服务器执行请求.

值得注意的是,目前,减少执行请求服务器数量的技术虽然可以提升吞吐率,但是由于每个请求所需要经历的阶段数量没有减少,所以这些技术对于减少服务器响应时间帮助不大.

2.6 遭受攻击

拜占庭系统在受到攻击时一般并不会影响到系统的正确运行(包括状态机拜占庭系统和下文将要介绍的 Quorum 拜占庭系统),但是在遭受攻击时,状态机拜占庭系统的整体执行效率将会大幅下降.

状态机拜占庭系统为了保证系统的活性,需要限制服务器之间交换信息的时限.时限的设定需要考虑服务

器所在的网络环境,如果时限过短,服务器可能不会收集到足够的信息,则可能导致试图更换协议被频繁地触发.频繁地执行试图更换协议将会导致大量的资源被占用,用户的请求则被长期挂起,不能执行;如果时限设定过长,那么一旦主节点被攻击或者被黑客控制,其执行效率会大幅下降.但是由于时限宽松,同时主节点在请求执行的关键路径上,系统的整体效率都会因此大幅下降.

文献[22]中,Amir 等人提出在攻击情况下的拜占庭状态及系统 Prime.Prime 解决主节点遭受攻击的性能下降问题,其将排序过程分成 Perordering Phase 与 Global Ordering Phase 两个阶段进行.通过监测拜占庭主节点来限制拜占庭主节点对整体系统性能的影响.拜占庭主节点必须及时地将 Perordering Phase 的消息返回给其他服务器,才能保证其主节点的地位不被其他服务器替换.Prime 在没有攻击的情况下,性能与其他状态机拜占庭系统相当;当有攻击存在时,其性能将比其他协议高出一个数量级.

2.7 小 结

本节主要对传统的状态机拜占庭系统协议、speculation 技术、拜占庭锁和一致性与执行分开等内容进行了介绍.

可以看出,传统状态机拜占庭系统一般是布置在 $3f+1$ 台服务器上,这些服务器在分别维护各自状态的同时共同维护一个系统状态.系统状态并不由某一个服务器状态决定,而是由其中的任意 $2f+1$ 台服务器决定.speculation 技术是对早期拜占庭系统的改进,该技术重新分配服务器与客户端的职能,将判断系统状态一致性的工作转交给客户端,从而简化了服务器在一致性协议中的工作,减少了计算量和阶段数.一致性与执行分离技术则通过减少参与执行请求的服务器数量来节省系统资源,达到提升系统吞吐率的目的.

分析目前状态机拜占庭系统,我们可以得出如下结论:1) 系统的优化方向基本是减少一致性协议所包含的阶段数量;2) 目前在某些假设条件下(如网络稳定、错误较少等),研究者已经提出了大量高效的一致性协议,但是在许多特定环境中(如数据共享频繁、攻击种类多样的环境中),拜占庭系统性能问题仍然没有有效地解决.因此,如何扩展拜占庭系统的可用范围、提升系统性能仍然是一个需要不断深入研究的问题.

3 Quorum 拜占庭系统(Byzantine quorum system)

状态机拜占庭系统同一时刻只能执行一个请求,因此服务器之间执行请求顺序要求完全一致.而在许多分布式存储系统中,应用通常不要求严格的执行顺序,但对响应时间极为敏感,因此人们提出 Quorum 拜占庭系统.

Quorum 系统[23,24]是指共享数据存储在一组服务器上,通过访问服务器的一个大小恒定的子集(quorum)来提供读/写操作.这类协议都含有一个特性:规定访问的子集大小后,任何一个这样的子集都包含最新的数据,并且一定可以读出来.Quorum 拜占庭系统则是在此基础上保证在服务器出现拜占庭错误时,系统的一致性语义仍然成立.

这一类系统写操作过程是:首先,客户端 c 从某一个大小恒定的服务器子集 Q 中读取时间戳(timestamp) $A=\{(t_u)\}_{u\in Q}$,客户端 c 从可用时间戳 t 中选取大于任何一个 A 中的时间戳并且大于所有 c 使用过的时间戳;然后,客户端 c 发送(v,t)给所有服务器(v 为需更新数据,t 为时间戳),直到收到某一个大小恒定的服务器子集的反馈信息.

这一类系统读操作过程是:客户端 c 获得某一个恒定大小的服务器子集 Q 中所有服务器返回的最新数据信息 $A=\{(v_u,t_u)\}_{u\in Q}$;然后,c 按照具体协议要求从中选取需要的数值.

Quorum 系统根据构造的不同,可以分为许多种,例如 *f-masking*,*grid-masking* 等[23].由于 *f-masking* 所采用的服务器数量最少(*f-masking* 需要 $3f+1$,*grid-masking* 需要$(3f+1)^2$ 台服务器),所以目前拜占庭系统中普遍采用的是 *f-masking* quorum 系统[25−29].下面首先介绍 *f-masking* quorum 系统及其性能优化技术,然后介绍 Quorum 系统中降低空间开销的相关技术.

3.1 *f-masking* quorum系统

f-masking quorum system[23]将拜占庭错误隐藏在系统之中.要实现上述目的,系统中能够提供正确信息的

服务器数量应该永远大于拜占庭服务器的数量.

假设系统服务器用 U 表示,客户端写入数据 x 时使用的子集是 Q_1,在读取数据 x 时使用的子集是 Q_2,系统中拜占庭服务器用 B 表示.那么,$|U|-|Q_1|+|B|$表示客户端在写入时至多有这么多服务器没有写入最新的数据,$|Q_2|-(|U|-|Q_1|+|B|)$则表示客户端在读取时最少有这么多服务器包含了最新的数据.所以,*f-masking* quorum 系统成立就需要有以下条件:

$$|Q_2|-(|U|-|Q_1|+|B|)>|B| \tag{1}$$

一般假设读取和写入的子集大小一样$|Q|=|Q_1|=|Q_2|$,带入化简后可得:

$$|U|<2|Q|-2|B| \tag{2}$$

又因为拜占庭服务器可能不响应客户端的请求,所以不论是在写入过程还是读取过程,客户端一定可以得到服务器响应的数量是

$$|Q|\leqslant|U|-|B| \tag{3}$$

将等式(3)代入等式(2)消去 U,化简后得$|Q|>3|B|$.再代入等式(3)得$|U|>4|B|$.

因此,*f-masking* quorum system 成立需满足:$|Q|>3|B|$且$|U|>4|B|$.但是如果服务器反馈的信息可以证明其自身的有效性,那么公式(1)可以化简成$|Q_2|-(|U|-|Q_1|+|B|)>1$,最后得$|U|>3|B|$,$|Q|>2|B|$.

2005 年,Abd-EI-Malek 等人提出 Q/U[29],它需要 $5f+1$ 台服务器.其主要特点是,当文件共享不频繁时,可以在两个阶段内完成客户端的读/写请求.Q/U 实现快速响应的方法是,在客户端中缓存每一台服务器中文件的时间戳,如果共享不频繁,在写入时就不需要请求服务器当前的时间戳,直接将更新的文件发给所有服务器,然后由服务器执行并返回结果.Q/U 由客户端来判断是否有足够的服务器完成了请求,不需要服务器端的两两交互.如果遇到多个客户端同时写入文件,那么通过一个修复过程,将所有服务器的文件同步到最新版本.

Q/U 虽然通过缓存技术将系统响应时间减少到两个阶段,但是需要消耗较多数量的服务器,同时,每一时刻只能有一个客户端操作文件.因此,其缺点也十分明显:系统使用过多的服务器首先增加了在硬件上的成本,同时增加了搭建和维护系统的花费;其次,系统的可扩展性大幅下降.所以,之后的 Quorum 系统不再采用 Q/U 的结构,而是通过在写操作中添加一个准备阶段来降低服务器数量,将其减少为 $3f+1$;同时,也不再要求同一时刻只能有一个客户端操作文件(write-free)[25,27,28].

这个准备阶段发生在客户端 c 选取时间戳并且发送需更新数据(v,t)给所有服务器之后.服务器 i 在接收到更新数据之后,如果 $t>t_i$ 且第 1 次接收到 t,那么将(v,t)临时保存起来,返回一条 $(PREPARE,t,digest(v))_{\sigma_1}$ (PREPARE 消息)消息给客户端 c.客户端收集到 $2f+1$ 条一致的 PREPARE 消息后,将 $2f+1$ 条 PREPARE 消息制作成证明 C 发送$(COMMIT,C)$(COMMIT 消息)给所有服务器,当服务器接收 COMMIT 消息之后,再正式更新(v,t).

PREPARE 阶段的目的与之前在状态机拜占庭系统中介绍的一致,是为了保证至少有足够的服务器获得了数据(可能是临时存储,也可能是正式更新),从而保证在时间戳 t 不可能再写入其他数值 $v'\neq v$.

3.2 纠删码技术与条带的验证

在大规模存储系统中,如果每一台服务器都存有文件的完整备份,将耗费比较大的存储资源,同时在更新数据时也占用较大的传输带宽.于是,在 Quorum 拜占庭系统中,研究者引入了纠删码技术降低空间开销[25−27],纠删码与存储完整副本相比,磁盘在利用率上有明显优势[30−32].其冗余度低,通过纠删码算法将原始数据进行编码后存储,较大程度上节省了磁盘开销.

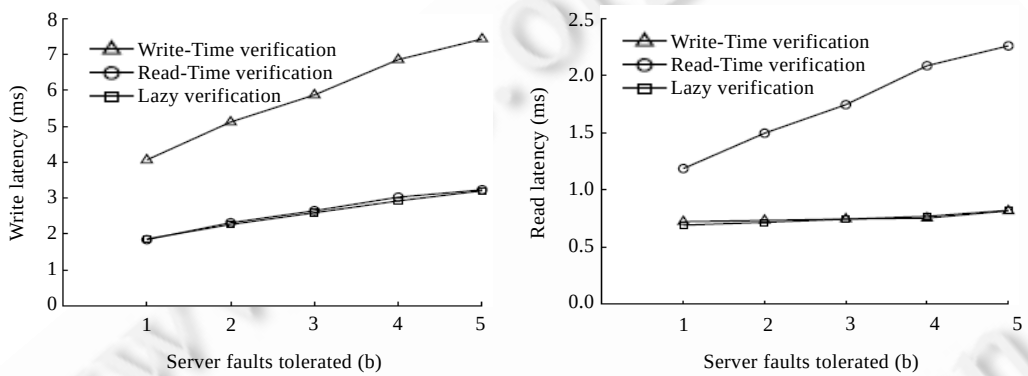
然而,纠删码使得每一个服务器上存储的信息都不完整,仅仅读取一个服务器的数据不能还原出正确的数据.而且拜占庭客户端可以欺骗服务器,同时,拜占庭服务器也可以欺骗读取数据的客户端.因此,如何保证服务器存储信息的完整性并且在读取时可以还原信息,就显得尤其关键.

一种方法是在写操作时将完整的数据传输给服务器,然后再由服务器判断自己所需要存储的纠删码是不是正确(写入时校验,write-time verification)[33];另一种方法是服务器不判断存储的数据是否正确,而是交给客户端在读取时候判断(读取时校验,read-time verification)[26].前一种方法由于需要在每一台服务器上计算纠删码

或者解码还原数据,将会在写入的时候产生大量的计算,这将增大系统的响应延时,降低峰值吞吐率.后一种方法因为不验证写入数据的正确性,因此每一次改动都需要在服务器端另外开辟空间存储,一旦某一个改动有问题,那么还能够读取出这个改动生效之前的版本,直到客户端读取出可以正确还原数据的版本为止.由于客户端可能需要不断读取多次,同时进行多次解码,因此系统读取响应延时较高,客户端计算量大.

对比这两种方法:前一种影响写入性能;后一种则浪费存储空间,同时影响读取性能.Lazy verification 技术[25]将校验纠删码完整性的工作放在后台进行,只有在系统空闲或者客户端要读取数据的时候才会触发校验,属于读取时校验,其在一定程度上提升了系统性能,同时可以及时释放存储空间,系统只需要在较短时间内存储多个版本的信息.

图 6 是系统分别采取 3 种校验方式时在读/写实验中表现的性能.通过实验可以看出,将校验放在后台进行,可以使写入和读取过程中都不进行数据校验,隐藏了校验开销.



(a) Write latency (b) Read latency

Fig.6 Comparison between three verification methods[25]

图 6 3 种校验方式对比[25]

Homomorphic fingerprinting[27,28]技术提供了除 lazy verification 之外的另一种校验方式,该技术通过使用 Homomorphic fingerprinting 编码,可以让服务器只接收自己所需的条块和较少的校验信息,便可以校验接收的条块是否是正确编码,属于写入时校验.Homomorphic fingerprinting 的优势是使得系统不需再存储多个版本,同时减少了读取操作时所需的计算量.

表 2 列出了采用 Homorphic fingerprinting 技术的拜占庭 Quorum 系统在单个写操作时所需时间分布.从表中可以看出:Fingerprinting 占用的时间较小;与读取校验型的拜占庭 Quorum 系统相比,前者花费在加密和 Fingerprinting 的总时间代价更低.由此说明,采用 Homomorphic fingerprinting 不会增加写入响应时间,同时减少了计算,增大吞吐率.

Table 2 Write response time breakdown

表 2 单个写操作所需时间分布情况

	RPC (ms)	Encode (ms)	Hashing (ms)	Fingerprinting (ms)
Homomorphic fingerprinting	2.17	0.79	0.45	0.18
读取校验拜占庭 Quorum 系统	2.80	2.54	0.88	–

3.3 小 结

Quorum 拜占庭系统不同于状态机拜占庭系统,从设计的原理上说,前者不需要在系统中维护一个状态,因此允许多个客户端进行写操作与读操作;后者需要主节点分配序号,在同一时刻则只能有一个客户端更新文件.由此得出,Quorum 拜占庭系统更适合应用于分布式存储领域,而状态机拜占庭系统则更适合于应用在分布式计

算等更广阔的领域.

通过观察针对于 Quorum 拜占庭系统而设计的优化技术,无论是减少服务器数量还是采用纠删码技术,其主要思想都是提高存储效率、减少系统存储冗余.Lazy verification 与 Homomorphic fingerprinting 则是基于纠删码技术的优化.虽然两者出发点和思想不同,但是目标相同,后者与前者相比具有更小的计算量和更快的响应能力.

4 服务器修复

无论是状态机拜占庭系统还是 Quorum 拜占庭系统,在使用过程中,服务器都会不可避免地发生错误.究其原因:首先,电子产品的使用寿命有限,使用过于频繁或者时间过长就可能自然损坏;其次,如果系统里面存储了关键数据,可能会引起黑客的攻击,导致一些服务器被攻破.这些自然损坏或者被黑客攻破的服务器,如果不及时修复就可能导致拜占庭服务器的数量不断增加,直至超出系统可以容忍的范围,导致整个系统崩溃.因此,拜占庭系统在使用的过程中需要有服务器的修复机制.

目前存在的修复机制主要是两种:

- 一种机制是错误检测(failure detection)[34−38].这类方法是通过某一种检测手段定期或者不定期地检查服务器运行状况,如果发现了这类机器,就立即修复.例如:在文献[34]中,负责检错的服务器模拟客户端定期发送请求,如果某台服务器的执行结果与系统正确结果不同,那么就可以认定这台服务器出错;在文献[35]中设定了阈值,一旦在某一个阈值或者条件之内某台服务器不能满足检测机制的要求,那么就认为这台服务器是拜占庭的;
- 另一种机制是定期分组修复服务器(proactive recovery)[10,39,40].这类方法不关心服务器是否有错误发生,而是采取预防的策略,在一段时间内将所有的服务器都修复一遍,从而保证系统长时间可靠运行.例如:在文献[10]中,每一个服务器上有一个计时器,每当执行一定数目的请求或者经过固定的时间,计时器就会控制服务器进行修复过程,每个服务器的修复过程相互独立;在文献[40]中,服务器的修复过程不再相互独立,而是由系统统一安排.独立修复协议设计简单,但是可能导致多个服务器同时处于修复过程中,没有足够的服务器执行客户端的请求.对于非独立修复的协议则设计复杂,一般需要较强的同步网络条件.

虽然服务器的修复问题已经有很多研究成果[10,34−40],但是目前仍然没有被完全解决.对于第 1 种机制,目前的检测手段不能保证 100%的检测准确率,遗漏或者错判都可能发生.主要是因为拜占庭机器具有欺骗性,可以伪装(例如,黑客可以控制他攻破的机器依然按照正常的协议执行,这样有助于他继续攻击其他的机器).同时,拜占庭机器还可以保持沉默,在异步网络下接收不到对方的消息,很难区分一个机器是故意保持沉默还是由于网络问题发生了丢包,阻塞等等情况.对于后一种机制,确定修复时间间隔则十分关键.如果修复间隔太短,频繁的数据同步就会严重影响服务器的执行效率;反之,则有可能使错误不断累加,导致系统崩溃.

除此之外,服务器在修复时需要更新最新的系统信息和数据,因此涉及到数据同步的问题.数据同步过程中需要占用服务器的带宽和计算资源,只有高效的数据同步算法同时避免带宽占用,才能最大限度地减少新服务器对系统正常服务的影响.目前,这个问题也还没有被有效解决,Distler 提出 VM-FIT 结构[41],采用虚拟化技术使服务器在修复之前先创建一个新的虚拟机,待新的虚拟机可以提供正常服务之时,旧的虚拟机再进行修复.VM-FIT 框架假设每台服务器控制虚拟机的结构完全不会出错,适合于分组定期修复服务器的协议.Sousa 则采用增加服务器数量的方式[40],保证服务器在修复的过程中仍然有足够的服务器响应客户端的请求,增加服务器数量导致系统构建成本提升和可扩展性下降.

5 拜占庭系统的应用

5.1 带有可信部件的拜占庭系统

上面介绍的拜占庭系统都是构建在一个服务器完全不可信的环境中.在这样的环境下,任何服务器都可能

说谎话(equivocation),因此至少需要 $3f+1$ 台服务器才能容忍 f 个拜占庭错误,比传统的非拜占庭容错协议多 f 台.但是在某些服务器的设计中,如果加入某些可信部件,则可以极大地降低拜占庭系统的开销.

最初,研究人员使用可信部件主要是负责服务器的检测,控制服务器定期的维护等等与提升系统性能无关的方面.例如:PBFT[10]采用一个检测器(watch dog)定期启动服务器的修复功能;PRRW[40]则是通过构建一个可信的内部同步网络(wormhole)[42]来控制服务器的修复;VM-FIT[39]使用虚拟化技术使修复完成的服务器可以快速同步数据;文献[43]则是引入一块可信的存储空间,通过定期同步正确的数据并且将服务器重置,从而保证系统长期可靠运行.

近年来,研究者开始尝试通过可信计算部件降低系统构建所需要的服务器数量.Chen 等人在 2007 年首先提出在拜占庭服务器中加入不可删改的只增存储器(attested append-only memory)[44]将搭建状态机拜占庭系统所需的服务器数量减少为 $2f+1$.这类存储器需要可信计算平台[45]的支持,做到不可删改,只能增添.因此,使用这类存储器保存服务器,对于每一个请求的操作日志,在服务器进行交互的时候将日志作为证明发送给其他的服务器,就可以起到防止服务器说谎话(equivocation)的作用.

虽然只增存储器实现了将服务器数量减少 f 台的目的,但是需要较大的空间开销,因此,Levin 提出采用一种更为简便的可信部件——可信计数器(trusted incrementer 或 TrInc)降低空间开销[46].可信计数器是可信平台(trusted platform module)[47]的核心部件,常见于当代的计算机中,容易广泛部署.TrInc 使用计数器为每次操作编号,每编号一次,计数器都会加 1.这样,一旦计数器为某一个操作编号为 n,那么其他的操作就不再可能被编号为 n.如果服务器之间交互的信息都是通过计数器编号的,那么服务器也就不能否认其行为,因此就不能说谎话.

目前,通过引入可信计算部件改善拜占庭系统性能的研究刚刚开始几年(A2M[44]在 2007 年被提出),主要解决的问题是降低服务器的数量.而如何通过可信部件优化系统性能,仍然需要不断探索.

5.2 针对服务特点设计拜占庭系统

有一些应用包含的自身特点,拜占庭协议却不能很好地支持.因此,研究者采用扩充拜占庭协议的方法将拜占庭协议融入应用中.例如,Zeno[48]就是针对大型分布式网络服务可能存在网络不稳定的问题而设计的拜占庭系统.在大型网络服务中的主机(数据中心)几乎遍布全球,不同地理位置之间的主机(数据中心)可能由于的网络问题,短时间内不能交换信息.Zeno 在这样的网络之上设计了一套合并算法,即使系统有超过 1/3 的机器暂时不能联络,服务仍然在小范围内保持.待网络修复之后,最初不能通信的节点之间再通过合并操作重新达成一致.

还有某一类分布式应用所面临的问题拜占庭协议不能直接解决,就需要改造拜占庭协议的功能来支持.例如,网络路由、无线网络路由、文件分发、归档存储、合作备份等服务,其系统中的每一个节点处于合作关系,然而整个系统没有统一的管理模式,系统中的每个节点有一定的自由争取自己的利益.对于一般的拜占庭系统而言不能直接应用在这一类网络服务上,因此,Aiyer 等人提出了 BAR[49]——一种针对这类含有理性节点(rational node)分布式服务网络而设计的拜占庭模型.BAR 需要在同步环境下才能保证系统活力(liveness),其采用一种 3 层的架构模式,只有最底层是拜占庭协议层,其余两层的存在都是为了系统的特定功能.

此外,某些特定的应用还能化简拜占庭协议的过程,使拜占庭协议更为简单高效.CIS(critical information infrastructure)[40]是一种分布式防火墙系统,其本身只需要判断客户端传来的请求是否可以通过,因此不需要主节点安排执行序号.在采用了 wormhole 结构的 CIS 中,系统只需要 $2f+1$ 台服务器就可以正常高效地运作.

6 总 结

目前,随着系统种类不断多样化,系统设计与管理越来越复杂,漏洞越来越多,系统可靠性的问题因而也越来越受到重视.随着云存储系统的广泛应用,如何提高系统稳定性、设计完备的容错系统,已经成为当前研究的热点.

传统的容错系统往往只能针对某一个或者多个已知的问题进行容错.拜占庭系统具有可以容忍任何类型错误的特点,于是,人们尝试研究、优化并且使用拜占庭系统来对系统进行防护.拜占庭系统作为通用的容错方案,可以很好地解决系统设计中忽视而未被发现的问题.但是由于其构建系统需要较多的服务器,系统复杂且运

行效率严重低于当前非拜占庭系统等缺陷,导致实际应用的拜占庭系统不多.因此,人们不断研究,尝试提升其性能,降低存储开销.本文综合介绍了拜占庭系统协议目前的研究现状,探讨了拜占庭协议的发展方向.

从目前的研究来看,我们可以得出以下结论:

1) 近年来,拜占庭系统的研究不断深入,成果也不断涌现,但是目前,性能上仍然与已经实用的非拜占庭系统相距较大,占用资源数量仍然较多.因此,想要拜占庭系统得以广泛使用,仍然需要进一步深入地研究其性能和资源的优化技术.

2) 拜占庭系统本身在长时间应用后可能会由于错误的累积导致系统崩溃,目前主要通过检测错误或者定期修复的方法来降低系统中错误的服务器数量.但是目前,如何有效全面检测拜占庭服务器、高效进行数据同步、合理定期修复等等问题都没有有效解决.因此,研究者需要继续不断研究完善当前服务器的修复方法,提出新的思路与方法,从而有效地延长系统可持续运行时间.

3) 拜占庭系统理论上至少需要 $3f+1$ 台服务器才能容忍 f 个错误,但是如果使用一些可信计算部件,在很小的范围内人为保证不出错,那么就有可能将服务器数量减少至 $2f+1$ 或者维持系统长时间可靠运行.同时,在具体的系统环境中,在不同的应用背景下可能发现某一类型的错误发生几率较大,那么可以通过优化针对特定错误类型的处理方法来降低拜占庭系统所需服务器数量.因此,如何让拜占庭系统更好地与实际相结合、如何在实际系统中应用,需要更加广泛的研究.

References:

[1] Amazon. Amazon Web service. http://aws.amazon.com/products/

[2] Houstn D, Ferdows A. Dropbox. https://www.dropbox.com/

[3] Microsoft. Windows azure. http://www.windowsazure.com/en-us/

[4] Schroeder B, Gibson GA. A large-scale study of failures in high-performance computing systems. IEEE Trans. on Dependable and Secure Computing, 2010. 337−350. [doi: 10.1109/TDSC.2009.4]

[5] http://games.qq.com/a/20110503/000013.htm

[6] http://www.chinanews.com/it/2011/09-05/3305704.shtml

[7] http://tech.163.com/07/ 0813/11/3LP86PEM000915BD.html

[8] Lamport L, Shostak R, Pease M. The Byzantine generals problem. ACM Trans. on Programming Languages and Systems, 1982, 4(3):382−401. [doi: 10.1145/357172.357176]

[9] Clement A, Kapritsos M, Lee S, Wang Y, Alvisi L, Dahlin M, Riche T. Upright cluster services. In: Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles. New York: ACM Press, 2009. 277−290. [doi: 10.1145/1629575.1629602]

[10] Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery. ACM Trans. on Computer Systems, 2002,20(4): 398−461. [doi: 10.1145/571637.571640]

[11] Cowling J, Myers D, Liskov B, Rodrigues R, Shrira L. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2006. 177−190.

[12] Kotla R, Alvisi L, Dahlin M, Clement A, Wong E. Zyzzyva: Speculative Byzantine fault tolerance. In: Proc. of the 21st ACM SIGOPS Symp. on Operating Systems Principles. New York: ACM Press, 2007, 45−58. [doi: 10.1145/1294261.1294267]

[13] Serafini M, Nokor P, Dobre D, Majuntke M, Suri M. Scrooge: Reducing the costs of fast Byzantine replication in presence of unresponsive replicas. In: Proc. of the 2010 IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. 2010. 353−362. [doi: 10.1109/DSN.2010.5544295]

[14] Yin J, Martin JP, Venkataramani A, Alvisi L, Dahlin M. Separating agreement from execution for Byzantine fault tolerant services. In: Proc. of the 19th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2003. 253−267. [doi: 10.1145/945445.945470]

[15] Wood T, Singh R, Venkataramani A, Shenoy P, Ceeehet E. ZZ and the art of practical BFT execution. In: Proc. of the 6th Conf. on Computer Systems. New York: ACM Press, 2011. 123−138. [doi: 10.1145/1966445.1966457]

[16] Wester B, Cowling J, Nightingale EB, Chen PM, Flinn J, Liskov B. Tolerating latency in replicated state machines through client speculation. In: Proc. of the 6th USENIX Symp. on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2009. 245−260.

[17] Hendricks J, Sinnamohideen S, Ganger GR, Reiter MK. Zzyzx: Scalable fault tolerance through Byzantine locking. In: Proc. of the 2010 IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. 2010. 363−372. [doi: 10.1109/DSN.2010.5544297]

[18] Pease M, Shostak R, Lamport L. Reaching agreement in the presence of faults. Journal of the ACM, 1980,27(2):228−234. [doi: 10.1145/322186.322188]

[19] Matin JP, Alvisi L. Fast Byzantine consensus. IEEE Trans. on Dependable and Secure Computing, 2006,3(3):202−215. [doi: 10.1109/TDSC.2006.35]

[20] Baker MG, Hartman JH, Kupfer MD, Shirriff KW, Ousterhout JK. Measurements of a distributed file system. In: Proc. of the 13th ACM Symp. on Operating Systems Principles. New York: ACM Press, 1991,25(5):198−212. [doi: 10.1145/121132.121164]

[21] Leung AW, Pasupathy S, Goodson G, Miller EL. Measurement and analysis of large-scale network file system workloads. In: Proc. of the USENIX 2008 Annual Technical Conf. on Annual Technical Conf. Berkeley: USENIX Association, 2008. 213−226.

[22] Amir Y, Coan B, Kirsch J, Lane J. Byzantine replication under attack. In: Proc. of the DSN. 2008.

[23] Malkhi D, Reiter M. Byzantine quorum systems. Jorunal of Distributed Computing, 1988,11(4):203−213.

[24] Martin JP, Alvisi L, Dahlin M. Minimal Byzantine storage. In: Proc. of the 16th Int'l Conf. on Distributed Computing. London: Springer-Verlag, 2002. 311−325. [doi: 10.1007/3-540-36108-1_21]

[25] Abd-EL-Malek M, Ganger GR, Goodson GR, Reiter MK, Wylie JJ. Lazy verification in fault-tolerant distributed storage systems. In: Proc. of the 24th IEEE Symp. on Reliable Distributed Systems. 2005. 179−190. [doi: 10.1109/RELDIS.2005.20]

[26] Goodson GR, Wylie JJ, Ganger GR, Reiter MK. Efficient Byzantine-tolerant erasure-coded storage. In: Proc. of the 2004 Int'l Conf. on Dependable Systems and Networks. 2004. 135−144.

[27] Hendricks J, Ganger GR, Reiter MK. Low-Overhead Byzantine fault-tolerant storage. In: Proc. of the 21st ACM SIGOPS Symp. on Operating Systems Principles. New York: ACM Press, 2007. 73−86. [doi: 10.1145/1294261.1294269]

[28] Hendricks J, Granger GR, Reiter MK. Verifying distributed erasure-coded data. In: Proc. of the 26th annual ACM Symp. on Principles of Distributed Computing. New York: ACM Press, 2007. 139−146. [doi: 10.1145/1281100.1281122]

[29] Abd-El-Malek M, Ganger G, Goodson G, Reiter M. Fault-Scalable Byzantine fault-tolerant services. In: Proc. of the 20th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2005. 59−74. [doi: 10.1145/1095810.1095817]

[30] Luo XH, Shu JW. Summary of research for erasure code in storage system. Journal of Computer Research and Development, 2012, 49(1):1−11 (in Chinese with English abstract).

[31] Li MQ, Shu JW, Zheng WM. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. ACM Trans. on Storage, 2009,4(4):1−22. [doi: 10.1145/1480439.1480444]

[32] Patterson D, Chen P, Gibson G, Katz R. Introduction to redundant arrays of inexpensive disks (RAID). In: Proc. of the Spring COMPCON'89. 1989. 112−117. [doi: 10.1109/CMPCON.1989.301912]

[33] Cachin C, Tessaro S. Asynchronous verifiable information dispersal. In: Proc. of the 24th IEEE Symp. on Reliable Distributed Systems. 2005. 191−201. [doi: 10.1109/RELDIS.2005.9]

[34] Alvisi L, Malkhi D, Pierce E, Reiter MK. Fault detection for Byzantine quorum systems. IEEE Trans. on Parallel and Distributed Systems, 2001,12(9):996−1007. [doi: 10.1109/71.954640]

[35] Chandra TD, Toueg S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 1996,43(2):225−267. [doi: 10.1145/226643.226647]

[36] Lima M, Greve F, Arantes L, Sens P. Byzantine failure detection for dynamic distributed systems. SANTOSDELIMA-2010-584597, RR-7222. Paris, 2010.

[37] Delporte-Gallet C, Fauconnier H, Guerraoui R, Hadzilacos V, Houznetsov P, Toueq S. The weakest failure detectors to solve certain fundamental problems in distributed computing. In: Proc. of the 23rd Annual ACM Symp. on Principles of Distributed Computing. New York: ACM Press, 2004. 338−346. [doi: 10.1145/1011767.1011818]

[38] Liu G, Zhou J, Sun Y, Qin L. A fault detection mechanism in erasure-code Byzantine fault-tolerance quorum. Wuhan University Journal of Natural Sciences, 2006,11(6):1453−1456. [doi: 10.1007/BF02831796]

[39] Reiser HP, Kapitza R. Hypervisor-Based efficient proactive recovery. In: Proc. of the 26th IEEE Int'l Symp. on Reliable Distributed Systems. 2007. 83−92. [doi: 10.1109/SRDS.2007.25]

[40] Sousa P, Bessani AN, Correia M, Neves NF, Verissimo P. Highly available intrusion-tolerant services with proactive-reactive recovery. IEEE Trans. on Parallel and Distributed Systems, 2010,21(4):452−465. [doi: 10.1109/TPDS.2009.83]

[41] Distler T, Kapitza R, Reiser HP. Efficient state transfer for hypervisor-based proactive recovery. In: Proc. of the 2nd Workshop on Recent Advances on Intrusiton-Tolerant Systems. New York: ACM Press, 2008. [doi: 10.1145/1413901.1413905]

[42] Paulo E. Travelling through wormholes: A new look at distributed systems models. Journal of SIGACT News, 2006,37(1):66−81. [doi: 10.1145/1122480.1122497]

[43] Chun BG, Maniatis P, Shenker S, Kubiatowicz J. Tiered fault tolerance for long-term integrity. In: Proc. of the 7th Conf. on File and Storage Technologies. Berkeley: USENIX Association, 2009. 267−282.

[44] Chun BG, Maniatis P, Shenker S, Kubiatowicz J. Attested append-only memory: Making adversaries stick to their word. In: Proc. of the 21st ACM SIGOPS Symp. on Operating Systems Principles. New York: ACM Press, 2007. 189−204. [doi: 10.1145/1294261.1294280]

[45] Felten EW. Understanding trusted computing: Will its benefits outweigh its drawbacks? Journal of IEEE Security Privacy, 2003, 1(3):60−62. [doi: 10.1109/MSECP.2003.1203224]

[46] Levin D, Douceur JR, Lorch JR, Moscibroda T. TrInc: Small trusted hardware for large distributed systems. In: Proc. of the 6th USENIX Symp. on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2009. 1−14.

[47] Trusted Computing Group. Trusted platform module. http://www.trustedcomputinggroup.org/developers/trusted_platform_module

[48] Singh A, Fonseca P, Kuznetsov P, Rodrigues R, Maniatis P. Zeno: Eventually consistent Byzantine-fault tolerance. In: Proc. of the 6th USENIX Symp. on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2009. 169−184.

[49] Aiyer AS, Alvisi L, Clement A, Dahlin M, Martin JP, Porth C. BAR fault tolerance for cooperative services. In: Proc. of the 20th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2005. 45−58. [doi: 10.1145/1095810.1095816]

附中文参考文献:

[30] 罗象宏,舒继武.存储系统中的纠删码研究综述.计算机研究与发展,2012,49(1):1−11.

范捷(1988－),男,广西桂林人,博士生,主要研究领域为网络/云存储,存储安全与可靠性.

E-mail: bevistomato@yahoo.com.cn

易乐天(1983－),男,博士生,主要研究领域为操作系统,分布式系统.

E-mail: lonat.front@gmail.com

舒继武(1968－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络/云存储,存储安全与可靠性,并行处理技术.

E-mail: shujw@tsinghua.edu.cn