TSS-BQS 系统的 Graceful Degradation 机制

王文韬[1),2)] 林璟锵[1)] 荆继武[1)] 罗 勃[3)]

[1)](中国科学院信息工程研究所信息安全国家重点实验室 北京 100195)

[2)](中国科学院研究生院 北京 100049)

[3)](堪萨斯大学电子工程与计算机科学系 堪萨斯州 KS 66045 美国)

摘 要 将门限签名方案分别应用于两种类型的 BQS 系统(Masking BQS 系统和 Dissemination BQS 系统),可以得到两种 TSS-BQS 系统(文中称为 TSS-mBQS 系统和 TSS-dBQS 系统). TSS-mBQS 系统的性能优于 TSS-dBQS 系统. 由此,作者提出了 TSS-BQS 系统的 Graceful Degradation 机制:系统由 $n=3f_d+1$ 台服务器组成,在初始阶段以 TSS-mBQS 状态运行,容忍 $f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$ 台 Byzantine 失效服务器;随着系统运行,可能失效的服务器数量增大,则以降低性能为代价,切换到 TSS-dBQS 状态,容忍 f_d 台 Byzantine 失效服务器. 在不影响容错能力的前提下,Graceful Degradation 机制提高了已有 TSS-BQS 系统的平均性能. 文中完成的 Graceful Degradation 机制能够在不中断存储服务、不影响客户端的前提下完成状态切换,客户端也不需要知道系统的运行状态(处于 TSS-mBQS 或 TSS-dBQS 状态).

关键词 拜占庭容错技术;拜占庭选举系统;优雅降级;门限签名方案

中图法分类号 TP309 **DOI 号**: 10.3724/SP.J.1016.2012.01793

Graceful Degradation in TSS-BQS Systems

WANG Wen-Tao[1),2)] LIN Jing-Qiang[1)] JING Ji-Wu[1)] LUO Bo[3)]

[1)](*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100195)

[2)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

[3)](*Department of Electrical Engineering and Computer Science, the University of Kansas, KS* 66045, *USA*)

Abstract By integrating threshold signature schemes with different types of Byzantine quorum systems (i.e., masking BQS and dissemination BQS in this paper), we can build two TSS-BQS systems, called the TSS-mBQS system and the TSS-dBQS system. The TSS-mBQS system produces better performance than the TSS-dBQS one. Based on this observation, we design graceful degradation in TSS-BQS systems: In the beginning, the system consisting of $n=3f_d+1$ servers, runs in the TSS-mBQS state and tolerates up to $f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$ Byzantine faulty servers; with the increasing number of servers which might be faulty, the system switches to the TSS-dBQS state, and tolerates up to f_d Byzantine faulty servers at the cost of performance degradation. Without harming the system's fault-tolerance, the graceful degradation improves its performance on average. Moreover, this graceful degradation doesn't interrupt the storage services of the system, and is transparent to clients who don't need to know the system state (i.e., the TSS-mBQS or the TSS-dBQS state).

Keywords Byzantine fault-tolerance; Byzantine quorum system; graceful degradation; threshold signature scheme

---

收稿日期:2012-05-21;最终修改稿收到日期:2012-07-20. 本课题得到国家自然科学基金(70890084/G021102,61003273,61003274)、中国科学院知识创新工程项目基金(YYYJ-1013)、中国科学院战略性先导专项基金(XDA06010702)资助.**王文韬**,男,1983 年生,博士研究生,主要研究方向为网络安全. **林璟锵**(通信作者),男,1978 年生,博士,助理研究员,主要研究方向为网络与信息安全. E-mail: linjq@lois.cn. **荆继武**,男,1964 年生,教授,主要研究领域为信息安全. **罗 勃**,男,1978 年生,博士,助理研究员,主要研究方向为网络与信息安全.

1 引 言

Byzantine Quorum System(简称 BQS 系统)[1]是指利用冗余复制技术、容忍服务器 Byzantine 失效[2](即任意失效)、运行于异步网络环境的存储系统. BQS 系统由 n 台独立服务器组成,每次读/写操作都在一定数量的服务器上进行. 写操作时,将数据写入 q_w 台服务器;读操作时,从任意 q_r 台服务器读出数据,再从中选出正确结果. 在 BQS 系统中,通过设定合适的 n、q_w 和 q_r,使读/写操作的服务器群有足够大的交集,从而排除失效服务器的影响,保证读出结果为最近一次写入的数据.

当存储不同类型数据时,BQS 系统由数量不等的服务器组成[1]:为了容忍 f 台 Byzantine 失效服务器,存储普通数据 Generic Data 的 Masking BQS 系统需要 $4f+1$ 台服务器,存储自验证数据 Self-Verifying Data(如带有数字签名的证书[3])的 Dissemination BQS 系统需要 $3f+1$ 台服务器. 所以,当使用 n 台服务器时,Masking BQS 系统只能容忍 $\left\lfloor\frac{n-1}{4}\right\rfloor$ 台失效服务器,Dissemination BQS 系统能容忍 $\left\lfloor\frac{n-1}{3}\right\rfloor$ 台. 但 Dissemination BQS 系统需要额外操作把普通数据转为自验证数据后再写入服务器,其性能不如 Masking BQS 系统.

基于上述分析,我们设计了相应的 Graceful Degradation 机制. Graceful Degradation 是指[4]:当运行环境恶化、超出预先估计时,调整系统状态或参数,保证提供服务、但是服务质量有所降低. 在本文中,BQS 系统的 Graceful Degradation 机制表现为:在初始阶段,系统以 Masking BQS 状态运行;当运行了一定时间、怀疑或检测到失效服务器已达到一定数量时,切换到 Dissemination BQS 状态. 也就是,当失效服务器数量可能将要超出预先估计、使系统无法再提供正常的存储服务时,以降低性能为代价、继续提供存储服务.

在我们的设计和实现中,借鉴了 TSS-BQS 系统[3,5-7](基于门限签名方案 Threshold Signature Scheme 的 BQS 系统)的思想,利用门限签名方案[8-9]来设计协议,使得客户端不需要知道系统的运行状态,同时还保留了 TSS-BQS 系统的优点. 综合以上,本文完成的容错存储服务具备如下特点:

(1) 系统由 $n=3f_d+1$ 台服务器组成,可作为 Masking BQS 系统运行(本文称为 TSS-mBQS 状态)或者 Dissemination BQS 系统运行(本文称为 TSS-dBQS 状态);TSS-mBQS 状态的性能优于 TSS-dBQS 状态.

(2) 在初始阶段,系统处于 TSS-mBQS 状态,容忍 $f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$ 台失效服务器(注意:并不是 $f_m=\left\lfloor\frac{3f_d}{4}\right\rfloor$、不同于普通的 Masking BQS 系统,详见 4.3 节的参数分析). 当估计失效服务器数量可能将要超过 f_m 时,切换到 TSS-dBQS 状态,降低性能、容忍 f_d 台失效服务器,继续提供容错服务.

(3) 系统能在不影响客户端的情况下执行 Graceful Degradation. 不论系统处于哪一种运行状态,客户端都使用完全相同的协议来进行读/写操作.

(4) 系统同时保留了 TSS-BQS 系统支持 Proactive Recovery[3,10-11]、简化客户端密钥管理和客户端通信的特点.

本文第 2 节先简要介绍背景知识;第 3 节描述 TSS-BQS 系统的工作协议;第 4 节给出 TSS-BQS 系统的 Graceful Degradation 机制以及性能分析和实验结果;第 5 节是相关问题讨论;第 6 节介绍相关研究工作;最后是全文总结.

2 背 景

2.1 BQS 系统

BQS 系统[1]由 n 台独立的服务器组成,每台服务器可能是正确的或 Byzantine 失效. 系统存储的每个数据可记为$[x,v,t]$. 其中,x 是变量名,v 和 t 分别是变量的值和时戳. 时戳集合是满足全序(Total Order)性质的集合,例如递增序列号;时戳用来区分值的新旧版本,新版本的值具有更大的时戳. 执行读/写操作的服务器集合(称为读 Quorum 和写 Quorum),分别包含 q_r 和 q_w 台服务器. 考虑到失效服务器的影响,各服务器 S_i 存储的$[x,v_i,t_i]$有可能会不同.

为了容忍 f 台失效服务器,BQS 系统的可用性要求[1]:任何情况下,都能找到由正确服务器组成的读/写 Quorum 来完成操作;有 $q_r\leq n-f$ 和 $q_w\leq n-f$. BQS 系统的一致性要求[1]:从读 Quorum 中获得的结果是最近一次被写入的值和时戳. 对于存储普通数据的 Masking BQS 系统,失效服务器会任意篡改数据,为了排除失效服务器的影响,正确结果

在读 Quorum 中应出现至少 $f+1$ 次，任意读/写 Quorum 的交集不小于 $2f+1$ 台，所以需要有 $q_r+q_w-n\geq 2f+1$；在读出的 q_r 个结果中，去掉出现次数少于 $f+1$ 的结果，剩余结果中时戳最大的就是正确结果. 对于存储自验证数据的 Dissemination BQS 系统，失效服务器无法篡改数据，只能回复旧版本数据，只需保证正确结果在读 Quorum 中出现 1 次，任意读/写 Quorum 的交集不小于 $f+1$ 台，所以需要有 $q_r+q_w-n\geq f+1$；在读出的 q_r 个结果中，去掉验证无效的结果，剩余结果中时戳最大的就是正确结果.

2.2　TSS-BQS 系统

TSS-BQS 系统[3,5-7]基于门限签名方案来实现 BQS 系统：系统有一对公私密钥对(称为 Service Key)，其私钥由 n 台服务器拆分、任意 $f+1$ 台服务器能合作利用 Service Key 私钥来计算数字签名. 每次读/写操作，客户端只需收到带有 Service Key 数字签名的响应消息，即保证完成操作[3,7]：因为 Service Key 门限签名和 TSS-BQS 系统的服务器协议保证了读/写操作已在 q_r/q_w 台服务器上执行、且返回正确结果.

相比普通的 BQS 系统，TSS-BQS 系统的重要优点是支持 Proactive Recovery[3,10-11]. 考虑到 Mobile Adversary 的影响，各服务器应该周期性地恢复到正确状态；否则，攻击者可以逐一地攻占服务器，直至失效服务器数量超过 f、破坏存储服务[10]. TSS-BQS 系统的客户端只配置 Service Key 公钥，而 Service Key 在 Proactive Recovery 过程中不会发生变化、只是重新拆分，所以 Proactive Recovery 不影响客户端. 此外，TSS-BQS 系统还具有简化客户端密钥管理和客户端通信的特点[3].

3　TSS-BQS 系统模型和工作协议

下面我们先给出 TSS-BQS 系统模型和协议，这是 Graceful Degradation 机制的运行基础. 下文描述的系统模型和协议，与已有的 TSS-BQS 系统[3,5-7]基本相同，不同之处在于我们将其从 Dissemination BQS 系统推广到 Masking BQS 系统. 限于篇幅，我们只是简要说明了两种状态的服务器协议；关于 TSS-BQS 系统服务器协议的详细安全性分析，请参考文献[3,5-7]. 此外，因为系统可能处于不同的运行状态(TSS-mBQS 或者 TSS-dBQS 状态)，每一台服务器还需要有独立的 *state* 变量，记录自己的当前运行状态.

3.1　系统模型

系统由 $n=3f_d+1$ 台服务器(记为 S_i，$n\geq i\geq 1$)组成；最多有 f_d 台 Byzantine 失效服务器，失效服务器可以合谋攻击. 系统向不定数量的客户端提供存储服务. 服务器和客户端通过异步 Fair Link 信道[3,5]通信：信道可能丢失消息，攻击者可以篡改、延迟、截获或删除消息；但当发送者无限地发送消息时，接收者能收到消息；信道是异步的，没有最大延迟假设.

系统有一对 Service Key 密钥对，其私钥由 n 台服务器拆分、任意 f_d+1 台能合作利用 Service Key 私钥来进行门限签名. 所有服务器和客户端都知道 Service Key 公钥，客户端只接受带有 Service Key 数字签名的响应消息. 此外，各服务器有自己的公私密钥对(不同于 Service Key)，称为 Server Key，专用于服务器之间的安全通信. 各服务器知道其它服务器的 Server Key 公钥，但是客户端不需要知道 Server Key 的任何信息.

每次读/写操作，客户端 c 周期性地向任意 f_d+1 台服务器发送请求消息，直至收到带有 Service Key 数字签名的响应消息. 由于最多 f_d 台服务器失效，所以至少 1 台服务器(称为此次读/写操作的 Delegate，记为 S_d)会正确地处理客户端的请求，与其它服务器合作执行服务器协议，然后将正确的响应消息发送给客户端. 服务器协议的功能是[3,7]：在 q_r/q_w 台服务器上读取/写入数据，生成响应消息，联合其它服务器对响应消息进行 Service Key 门限签名. 需要指出，Delegate 不是额外的专门服务器，每台服务器都同时具有 Delegate 功能：当正确服务器从客户端收到读/写请求消息时，就自动转为 Delegate、发起服务器协议.

变量的时戳 t 按如下规则产生[3]：时戳由两部分组成，高位是连续递增的整数序号，记为 $seq(t)$，低位由相应的客户端写请求消息 Req_W 确定，有 $t=seq(t)\,|\,Hash(Req_W)$. 其中，$Hash(\,)$ 是单向散列函数. 容易验证，上述方式产生的时戳集合满足全序性质.

在下文的协议描述中，我们使用了如下记号：

$[\cdot]_{SK}$：带有 Service Key 数字签名的消息；

$[\cdot]_c$：由客户端 c 进行数字签名后的消息；

$[\cdot]_d$ 或 $[\cdot]_i$：由服务器 S_d 或 S_i 使用 Server Key 进行数字签名后的消息；

$PS_i(\cdot)$：服务器 S_i 计算的部分签名，f_d+1 台

不同服务器产生的部分签名可合成有效的 Service Key 数字签名.

3.2 客户端协议

客户端 c 使用如下协议来读/写变量 x. 注意:不论系统处于 TSS-mBQS 或者 TSS-dBQS 状态,都使用相同的客户端协议.

READ

A. 客户端 c 发送读请求消息 $Req_R=[\text{ReqRead},c,x,p]_c$ 给任意 f_d+1 台服务器;其中,c 是客户端的唯一标识,x 是变量的唯一标识,p 是防重放攻击的一次性随机数、防止攻击者重放以前的响应消息.

B. 客户端 c 周期性地发送 Req_R 给 f_d+1 台服务器,直至收到读响应消息 $Resp_R=[\text{RespRead},c,x,v_R,t_R,p]_{SK}$;其中 v_R 和 t_R 是读出的值和时戳.

WRITE

A. 客户端 c 首先执行 1 次 READ 操作,得到 $Resp_R$,其中包含变量 x 的当前时戳 t_R.

B. 客户端 c 发送写请求消息 $Req_W=[\text{ReqWrite},c,x,v_W,p,Resp_R]_c$ 给任意 f_d+1 台服务器;其中 v_W 是待写入的值.

C. 客户端 c 周期性地发送 Req_W 给 f_d+1 台服务器,直至收到写响应消息 $Resp_W=[\text{RespWrite},c,x,v_W,t_W,p]_{SK}$;其中 $t_W=(seq(t_R)+1)\mid Hash(Req_W)$.

3.3 TSS-dBQS 状态的服务器协议

TSS-dBQS 状态能容忍 f_d 台失效服务器,读/写 Quorum 包含的服务器数量分别记为 q_{dr} 和 q_{dw}. 当 S_d 的 *state* 变量表明当前是 TSS-dBQS 状态时,使用如下协议. 在协议描述中,我们先假定各服务器的 *state* 变量相同;然后,在 4.2 节讨论了当各服务器 *state* 变量不同的处理方法.

READ

A. 当 S_d 收到 $Req_R=[\text{ReqRead},c,x,p]_c$ 时,验证有效后,发送 $[\text{Read},state,d,Req_R]_d$ 给所有服务器. 其中,*state* 是 S_d 的当前运行状态,设定为 TSS-dBQS;d 是 S_d 的唯一标识.

B. 当 S_i 从 S_d 收到 Read 消息时,验证其中的 Req_R 有效后,回复 $[\text{Replica},state,i,[x_i],Req_R]_i$. 其中,$i$ 是 S_i 的唯一标识;$[x_i]$ 是 S_i 存储的数据,可能是自验证数据形式 $[x,v_i,t_i]_{SK}$ 或者是普通数据形式 $[x,v_i,t_i]$. 如果系统在切换到 TSS-dBQS 状态之后,还尚未执行过写操作,则 S_i 存储 $[x,v_i,t_i]$;否则就存储 $[x,v_i,t_i]_{SK}$.

C. S_d 周期性地重复执行步骤 A,直至收到 q_{dr} 台服务器(包括 S_d)的 Replica 消息. 然后 S_d 从中选出正确结果,记为 $[x,v_R,t_R]$. 选出正确结果的方法,详见 4.3 节的说明.

D. S_d 向所有服务器发送门限签名请求 $[\text{SignRead},state,d,M_R,\Sigma_R,Req_R]_d$,其中,$M_R=[\text{RespRead},c,x,v_R,t_R,p]$,$\Sigma_R$ 是在步骤 C 从 q_{dr} 台服务器收到的 Replica 消息.

E. 当 S_i 从 S_d 收到 SignRead 消息时,检查:(a)Σ_R 是否是 q_{dr} 台服务器针对 Req_R 的 Replica 消息;(b)M_R 是否包含 Σ_R 中的正确结果. 检查通过后,S_i 计算部分签名 $PS_i(M_R)$,回复 $[\text{PSRead},state,i,PS_i(M_R),Req_R]_i$.

F. S_d 周期性地重复执行步骤 D,直至收到 f_d+1 台服务器(包括 S_d)的 PSRead 消息. 然后 S_d 将 f_d+1 个部分签名合成为 $[\text{RespRead},c,x,v_R,t_R,p]_{SK}$,发送给客户端 c.

WRITE

A. 当 S_d 收到 $Req_W=[\text{ReqWrite},c,x,v_W,p,Resp_R]_c$ 时,验证有效后、发送 $[\text{SignReplica},state,d,Req_W]_d$ 给所有服务器.

B. 当 S_i 从 S_d 收到 SignReplica 消息时,验证其中的 Req_W 有效后,计算部分签名 $PS_i(x,v_W,t_W)$,回复 $[\text{PSReplica},state,i,PS_i(x,v_W,t_W),Req_W]_i$. 其中,$t_W=(seq(t_R)+1)\mid Hash(Req_W)$,$t_R$ 是包含在 $Resp_R$ 中的原有时戳.

C. S_d 周期性地重复执行步骤 A,直至收到 f_d+1 台服务器(包括 S_d)的 PSReplica 消息. 然后 S_d 将 f_d+1 个部分签名合成为 $[x,v_W,t_W]_{SK}$.

D. S_d 发送 $[\text{Write},state,d,[x,v_W,t_W]_{SK},Req_W]_d$ 给所有服务器.

E. 当 S_i 从 S_d 收到 Write 消息时,验证 $[x,v_W,t_W]_{SK}$ 是有效的自验证数据后,回复 $[\text{Ack},state,i,[x,v_W,t_W]_{SK},Req_W]_i$. 然后 S_i 比较 $[x,v_W,t_W]_{SK}$ 和自己存储的 $[x,v_i,t_i]_{SK}$ 或 $[x,v_i,t_i]$,如果 $t_W>t_i$,就更新自己的数据.

F. S_d 周期性地重复执行步骤 D,直至收到 q_{dw} 台服务器(包括 S_d)的 Ack 消息.

G. S_d 向所有服务器发送门限签名请求 $[\text{SignWrite},state,d,M_W,\Sigma_W,Req_W]_d$,其中,$M_W=[\text{RespWrite},c,x,v_W,t_W,p]$;$\Sigma_W$ 是在步骤 F 从 q_{dw} 台服务器收到的 Ack 消息.

H. 当 S_i 从 S_d 收到 SignWrite 消息时,检查:(a)Σ_W 是否是 q_{dw} 台服务器针对 Req_W 的 Ack 消息;(b)M_W 是否是 Req_W 对应的写响应消息. 检查通过

后，S_i 计算部分签名 $PS_i(M_W)$，回复 $[\text{PSWrite}, state, i, PS_i(M_W), Req_W]_i$.

I. S_d 周期性地重复执行步骤 G，直至收到 f_d+1 台服务器（包括 S_d）的 PSWrite 消息. 然后 S_d 将 f_d+1 个部分签名合成 $[\text{RespWrite}, c, x, v_W, t_W, p]_{SK}$，发送给客户端 c.

3.4 TSS-mBQS 状态的服务器协议

TSS-mBQS 状态能容忍 f_m 台失效服务器，读/写 Quorum 包含的服务器数量分别记为 q_{mr} 和 q_{mw}. 当 S_d 的 *state* 变量表明当前是 TSS-mBQS 状态时，使用如下服务器协议. 相比 TSS-dBQS 状态，TSS-mBQS 状态的服务器协议有如下变化：

(1) 各消息中的 *state* 设定为 TSS-mBQS；相应地，读/写 Quorum 的服务器数量变为 q_{mr} 和 q_{mw}.

(2) 因为 TSS-mBQS 状态存储普通数据，WRITE 操作不需要步骤 B、C 和 D 对数据进行门限签名. 在步骤 A，S_d 直接发送 $[\text{Write}, state, d, [x, v_W, t_W], Req_W]_d$ 给所有服务器. 所以，在步骤 E，当 S_i 从 S_d 收到 Write 消息时，验证其中的 Req_W 有效、$[x, v_W, t_W]$ 是对应的普通数据后，回复 Ack 消息.

4 TSS-BQS 系统的 Graceful Degradation 机制

3.3 节和 3.4 节描述了系统的两种运行状态. TSS-mBQS 状态容忍的失效服务器数量小于 TSS-dBQS 状态（见 4.3 节的分析），但是读/写操作性能更高（见 4.6 节的详细性能分析和实验结果）. 基于上述观察和分析，我们设计了 TSS-BQS 系统的 Graceful Degradation 机制：在初始阶段，系统以 TSS-mBQS 状态运行，容忍 $f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$ 台失效服务器；当运行一定时间或者出现安全事件后（关于状态切换的发起，详见 5.2 节讨论），切换到 TSS-dBQS 状态，降低性能、容忍 f_d 台失效服务器. 需要指出，在状态切换期间，最多只能有 f_m 台失效服务器；只有状态切换完成后，TSS-BQS 系统才能容忍 f_d 台失效服务器.

相比现有的 TSS-BQS 系统[3,5-7]（由 $3f_d+1$ 台服务器组成、容忍 f_d 台失效服务器），在不影响容错能力、容忍 f_d 台失效服务器的前提下，我们的系统在部分时间里运行在性能更高的 TSS-mBQS 状态，所以 Graceful Degradation 机制能带来更高的平均性能. 在我们的系统中，Graceful Degradation 机制表现为由服务器合作执行的状态切换协议，状态切换协议使各服务器的 *state* 变量从 TSS-mBQS 变为 TSS-dBQS.

下面，我们先列出切换协议应满足的条件：

(1) 容错性. 状态切换应由多台服务器共同发起或者经过多台服务器同意才能完成；否则，失效服务器就可以在不必要的时候（例如，系统刚开始运行时），恶意地切换到 TSS-dBQS 状态，就相当于系统始终运行在 TSS-dBQS 状态，不能提高性能.

(2) 完整性. 在 f_m 台失效服务器的情况下，切换协议能够顺利地执行. 而且，在状态切换协议结束之后，所有读/写操作都必须在 TSS-dBQS 状态下进行，不可能再执行 TSS-mBQS 状态的读/写操作.

(3) 兼容性. 切换协议能与读/写操作协议并行执行，可在不中断存储服务、不影响客户端的情况下完成.

(4) 一致性. 切换协议与客户端/服务器的读/写操作协议使用一致的系统模型.

(5) 高效性. 切换协议应能够快速地执行；否则，如果切换协议耗时太长，甚至不如恢复失效服务器的速度，状态切换就没有意义.

4.1 状态切换协议

在下面的状态切换协议描述中，我们先假定至少会有 1 台正确服务器来发起切换协议（也称为 Delegate、记为 S_d）. 在 5.2 节的讨论中，我们将说明如何保证有正确服务器作为 Delegate 发起切换协议. 此外，需要指出，状态切换操作、读/写操作的 Delegate 相互之间没有联系，任意服务器都可以作为 Delegate、各操作的 Delegate 可以不同.

A. S_d 发送 $[\text{SignTicket}, d, M_G, reason]_d$ 给所有服务器. 其中，$M_G=[\text{Ticket}, I_g]$，*reason* 是发起本次切换协议的原因（详见 5.2 节的说明），$I_g=Hash(reason)$ 是本次状态切换的唯一标识.

B. 当 S_i 收到 SignTicket 消息时，验证其中的 *reason* 有效后，计算部分签名 $PS_i(M_G)$，回复 $[\text{PSTicket}, i, PS_i(M_G), I_g]_i$.

C. S_d 周期性地重复执行步骤 A，直至收到 f_d+1 台服务器（包括 S_d）的 PSTicket 消息. 然后 S_d 将 f_d+1 个部分签名合成为 $[\text{Ticket}, I_g]_{SK}$.

D. S_d 发送 $[\text{Degrade}, d, [\text{Ticket}, I_g]_{SK}]_d$ 给所有服务器.

E. 当 S_i 收到 Degrade 消息时，验证其中的 $[\text{Ticket}, I_g]_{SK}$ 的 Service key 数字签名有效后，存储

$[\text{Ticket}, I_g]_{SK}$、将自己的 *state* 变量设定为 TSS-dBQS，回复$[\text{Echo}, i, I_g]_i$.

F. S_d 周期性地重复执行步骤 D，直至收到 $n-f_{\text{m}}$ 台服务器(包括 S_d)的 Echo 消息.

4.2 并发执行情况的处理

在切换协议执行过程中，如果同时有并发执行的读/写操作，则参加读/写操作的服务器的 *state* 变量可能不同：因为在异步网络环境中，有些服务器已经切换到 TSS-dBQS 状态、有些仍然处于 TSS-mBQS 状态.

我们采取如下方式处理：

(1) S_i 只处理与自己 *state* 变量相同的读/写操作消息. 当 S_i 处于 TSS-mBQS 状态，只处理 *state* 是 TSS-mBQS 的消息；当 S_i 处于 TSS-dBQS 状态，只处理 *state* 是 TSS-dBQS 的消息.

(2) 当 S_i 处于 TSS-dBQS 状态，如果收到 TSS-mBQS 状态的读/写操作消息，则直接回复自己存储的$[\text{Ticket}, I_g]_{SK}$，要求 S_d 切换到 TSS-dBQS 状态，然后重新执行读/写操作的服务器协议. 因为 Ticket 消息带有 Service Key 数字签名，任何服务器都可以验证.

(3) 当 S_i 处于 TSS-mBQS 状态，如果收到 TSS-dBQS 状态的读/写操作消息，则 S_i 请求 S_d 先发送$[\text{Ticket}, I_g]_{SK}$. 在 S_i 验证$[\text{Ticket}, I_g]_{SK}$有效、切换到 TSS-mBQS 状态后，继续执行读/写操作的服务器协议.

(4) 当读/写操作的 S_d 处于 TSS-dBQS 状态，如果收到其它服务器的请求，则回复$[\text{Ticket}, I_g]_{SK}$.

(5) 当读/写操作的 S_d 处于 TSS-mBQS 状态，如果收到有效的$[\text{Ticket}, I_g]_{SK}$(无论是在切换协议的 Degrade 消息中、或者是读/写操作的服务器 S_i 回复消息中)，就先切换到 TSS-dBQS 状态，然后重新开始执行读/写操作.

4.3 参数分析

首先，系统参数需要满足 BQS 系统[1]要求，才能够保证读/写操作的服务器协议的顺利执行和正确性：

(1) Dissemination BQS 系统要求，$q_{\text{dr}} \leq n-f_{\text{d}}$，$q_{\text{dw}} \leq n-f_{\text{d}}$，$q_{\text{dr}}+q_{\text{dw}}-n \geq f_{\text{d}}+1$. 当 $n=3f_{\text{d}}+1$ 时，有 $q_{\text{dr}}=q_{\text{dw}}=2f_{\text{d}}+1$.

(2) Masking BQS 系统要求，$q_{\text{mr}} \leq n-f_{\text{m}}$，$q_{\text{mw}} \leq n-f_{\text{m}}$，$q_{\text{mr}}+q_{\text{mw}}-n \geq 2f_{\text{m}}+1$.

其次，因为引入了状态切换，我们还需要考虑状态切换时的特殊情况. 在切换到 TSS-dBQS 状态之后，如果尚未执行过 TSS-dBQS 状态的写操作，则各服务器仍存储普通数据$[x, v_i, t_i]$. 所以，此时执行 TSS-dBQS 状态的读操作，不但要考虑自验证数据，还需要考虑普通数据的情况：

(1) 因为状态切换已完成、系统处于 TSS-dBQS 状态，可能已有 f_{d} 台失效服务器. 所以，在最差情况下，会有 $f_{\text{d}}+n-q_{\text{mw}}$ 台服务器向 S_d 回复错误的普通数据(其中，有 f_{d} 台失效服务器，又有 $n-q_{\text{mw}}$ 台没有参加上一次 TSS-mBQS 状态的写操作). 所以，只要 S_d 得到 $f_{\text{d}}+n-q_{\text{mw}}+1$ 个相同数据，就可以确定是正确结果.

(2) 为了保证在各种失效情况下，S_d 都能够得到：$f_{\text{d}}+n-q_{\text{mw}}+1$ 个相同数据，则 TSS-mBQS 状态的写操作至少写入 $f_{\text{d}}+(f_{\text{d}}+n-q_{\text{mw}}+1)$ 台服务器，才能容忍 f_{d} 台服务器在写入数据后失效，不参加 TSS-dBQS 状态的读操作；所以，有 $q_{\text{mw}} \geq f_{\text{d}}+f_{\text{d}}+n-q_{\text{mw}}+1 \Rightarrow q_{\text{mw}} \geq (n+2f_{\text{d}}+1)/2$.

(3) Masking BQS 系统要求 $q_{\text{mw}} \leq n-f_{\text{m}}$，可得到：$n-f_{\text{m}} \geq (n+2f_{\text{d}}+1)/2 \Rightarrow f_{\text{m}} \leq f_{\text{d}}/2$. 仅在 $q_{\text{mw}}=n-f_{\text{m}}$ 时，不等式中的等号才成立，此时 f_{m} 取最大值$\left\lfloor \frac{f_{\text{d}}}{2} \right\rfloor$，使系统尽可能长时间处于 TSS-mBQS 状态以提高平均性能.

(4) 由 $q_{\text{mw}}=n-f_{\text{m}}$，容易推导得到：当 $n-f_{\text{m}} \geq q_{\text{mr}} \geq 3f_{\text{m}}+1$ 时，满足 Masking BQS 系统的可用性和一致性要求.

基于上述分析，在 TSS-dBQS 状态，从 $q_{\text{dr}}=2f_{\text{d}}+1$ 个数据中选出正确结果的方法：先尝试寻找出现次数不少于 $f_{\text{d}}+f_{\text{m}}+1$ 的结果，作为正确结果；否则，将验证有效的、时戳最大的自验证数据作为正确结果. 在 TSS-mBQS 状态，从 $q_{\text{mr}}=3f_{\text{m}}+1$ 个数据中选出正确结果的方法：去掉出现次数少于 $f_{\text{m}}+1$ 的结果，剩余结果中时戳最大的就是正确结果. 注意：在 TSS-dBQS 状态，应该优先尝试寻找出现次数不少于 $f_{\text{d}}+f_{\text{m}}+1$ 的结果，以避免失效服务器回复上一个 Proactive Recovery 周期的、旧的自验证数据.

4.4 安全性分析

下面，我们分析状态切换协议是否满足上文提出的各项要求：

(1) 容错性. Ticket 消息需要 $f_{\text{d}}+1$ 台服务器合作进行 Service Key 门限签名，其中必定有正确服务器参加. 所以，即使 f_{d} 台失效服务器合谋，也不能在不必要的时候切换状态.

(2)完整性. 首先，状态切换协议需要 f_d+1 台服务器进行 Ticket 消息的 Service Key 门限签名、$n-f_m$ 台服务器回复 Echo 消息，因为在状态切换过程中最多有 f_m 台失效服务器，所以肯定能顺利执行结束.

其次，在状态切换完成后，至少有 $n-f_m$ 台服务器已收到 Ticket 消息(其中可能包含有 f_d 台失效服务器). 在此之后，如果要成功执行 TSS-mBQS 状态的写操作，则至少要有 q_{mw} 台服务器参加，易验证有 $q_{mw}+(n-f_m)-n>f_d$ 成立；也就是说，在 TSS-mBQS 状态的写操作过程中，必然有已收到 Ticket 消息、处于 TSS-dBQS 状态的正确服务器参加，该服务器会使 S_d 切换到 TSS-dBQS 状态.

最后，对于 TSS-mBQS 状态的读操作，则略有不同. 根据上一节分析，要求 $n-f_m\geq q_{mr}\geq 3f_m+1$，同时还要满足 $q_{mr}+(n-f_m)-n>f_d$，才会使得在此之后的 TSS-mBQS 状态的读操作不能执行结束，保证有正确服务器会使 S_d 切换到 TSS-dBQS 状态，并重新进行 TSS-dBQS 状态的读操作. 容易计算，满足上述条件的 q_{mr} 最小值是 f_m+f_d+1，使得在切换协议结束之后，所有读/写操作都在 TSS-dBQS 状态下进行.

综合以上分析，表 1 列出了各系统参数的表达式.

表 1　系统参数示例

f_d	签名门限	$f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$	$n=3f_d+1$	$q_{dw}=q_{dr}=2f_d+1$	$q_{mw}=n-f_m$	$q_{mr}=f_m+f_d+1$
2	3	1	7	5	6	4
3	4	1	10	7	9	5
4	5	2	13	9	11	7
5	6	2	16	11	14	8

(3)兼容性. 首先，客户端发出的读/写请求消息中不包含系统的状态信息，在 TSS-mBQS 状态或者 TSS-dBQS 状态下，读/写操作的 S_d 都可以用来发起服务器协议. 其次，根据 4.2 节的并行执行情况处理，在状态切换过程中开始的读/写操作，也会转为 TSS-dBQS 状态的读/写操作并顺利执行. 因为切换协议中使用了带有 Service Key 数字签名的 Ticket 消息，各服务器都可以独立地验证是否有效、分别切换状态，所以 S_d 可以直接要求其它服务器切换状态，配合执行服务器协议. 第三，根据 4.3 节的参数分析，在切换到 TSS-dBQS 状态后，虽然各服务器存储普通数据，读操作仍可以顺利执行并得到正确结果；状态切换后 TSS-dBQS 状态的写操作，也可以顺利执行.

(4)一致性. 上述的安全性分析，使用了与客户端/服务器协议完全一致的系统模型.

4.5　性能分析和实验结果

本节先从通信量和计算量分析了两种状态下的读/写操作、状态切换的性能；然后给出了原型系统的实验结果. 理论分析和实验结果都表明：TSS-mBQS 状态的性能优于 TSS-dBQS 状态，状态切换的资源消耗与 1 次读/写操作相当、能够快速完成.

4.5.1　性能分析

图 1 给出了两种状态下的读/写操作、状态切换的通信和计算情况. 其中，在分析写操作的性能时，不包括发送写请求消息 Req_W 之前的读操作(用来获得当前时戳)；对于 TSS-dBQS 状态的读操作，只考虑自验证数据的情况，因为在大部分情况下，TSS-dBQS 状态的读操作都是得到自验证数据.

在图 1 中，箭头表示通信过程、方框表示计算过程. 并行执行的通信过程，在图中表示为 1 次通信. 例如，S_d 同时向多台服务器发送 Read 消息、多台服务器向 S_d 同时回复 Replica 消息，所需通信时间与服务器数量基本无关、取决于其中耗时最长的某一次通信. 可以看出，TSS-mBQS 状态的读/写操作需要 3 轮通信；TSS-dBQS 状态的读操作需要 3 轮通信、写操作需要 4 次通信；状态切换需要 2 轮通信.

COCA[3] 和 CODEX[5] 的实验测试结果表明：在 TSS-BQS 系统中，主要计算消耗是公钥密码计算(Service Key 门限签名和验证、Server Key 签名和验证)，其它的计算(如 Hash 计算等)可忽略不计. 图 1 中，T_s 和 T_p 分别表示计算 1 次 Server Key 数字签名和 1 次 Service Key 部分签名的时间，T_v 表示 1 次数字签名验证的时间，T_c 表示合成 Service Key 签名的时间. 同样地，不同服务器并行执行的计算过程，计为 1 次计算；例如，多台服务器同时执行 Service Key 门限签名、多台服务器分别用自己的 Server Key 签名消息，所需计算时间与服务器数量基本无关、取决于其中计算最慢的某 1 台服务器. 但是，由同一台服务器执行的连续多次计算，则计为多次；例如，S_d 验证 q_{mr} 个 Replica 消息上的 Server Key 数字签名. 表 2 列出了各操作的计算时间公式以及当 $f_d=2$ 时的值(在原型系统中，$f_d=2$).

4.5.2　原型系统和实验结果

按照上文描述的协议，我们使用 C 语言实现了原型系统. 在原型系统中，$f_d=2$、签名门限 $f_d+1=3$，其它参数按照表 1 来设定. Service Key 和 Server Key 都是 1024-bit RSA 密钥对，使用 Shoup 门限签名方案[9] 来拆分 Service Key 私钥. 每一个变量有

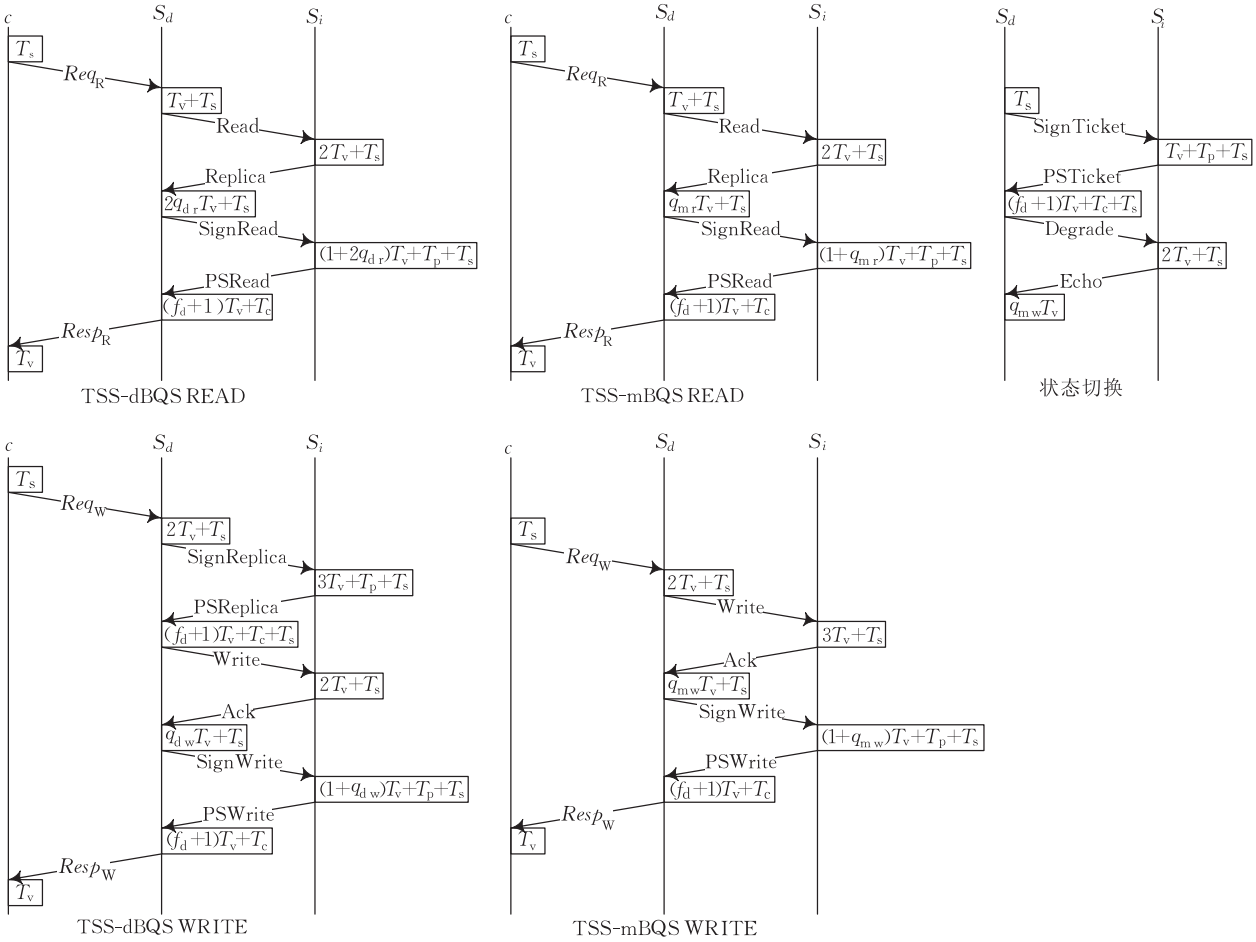


图 1 TSS-BQS 系统的通信量和计算量

表 2 TSS-BQS 系统的计算量

操作	状态	计算时间公式	计算时间($f_d=2$)
读操作	TSS-dBQS 状态	$T_p+5T_s+T_c+(6+4q_{dr}+f_d)T_v$	$T_p+5T_s+T_c+28T_v$
	TSS-mBQS 状态	$T_p+5T_s+T_c+(6+2q_{mr}+f_d)T_v$	$T_p+5T_s+T_c+16T_v$
写操作	TSS-dBQS 状态	$2T_p+7T_s+2T_c+(11+2q_{dw}+2f_d)T_v$	$2T_p+7T_s+2T_c+25T_v$
	TSS-mBQS 状态	$T_p+5T_s+T_c+(8+2q_{mw}+f_d)T_v$	$T_p+5T_s+T_c+22T_v$
状态切换操作	—	$T_p+4T_s+T_c+(4+q_{mw}+f_d)T_v$	$T_p+4T_s+T_c+12T_v$

32-bit 唯一标识和 512-bit 值,时戳是 192-bit(包括 32-bit 的整数序号和 160-bit 的 SHA-1 散列值).系统由 7 台服务器和 1 台客户端组成,它们之间用 100 M 以太网连接、UDP 通信,配置都是 Intel Pentium 4 (2.4 GHz) CPU、256M RAM、Windows 2000 Server (SP2)操作系统.

表 3 列出了原型系统的实验结果.其中,读操作的测量时间是从客户端生成 Req_R 到验证 $Resp_R$,写操作的测量时间是从客户端生成 Req_W 到验证 $Resp_W$、不包括发送 Req_W 之前的读操作,协议切换操作的测量时间是从 S_d 生成 SignTicket 消息到验证 q_{mw} 个 Echo 消息.

表 3 实验结果 (单位:ms)

	TSS-mBQS 状态时间	TSS-dBQS 状态时间
读操作	146.25	159.15
写操作	151.15	286.95
协议切换操作	94.70	

5 讨 论

5.1 与 Proactive Recovery 的结合

TSS-BQS 系统的重要优点是支持 Proactive Recovery.对于 TSS-BQS 系统的存储服务,每一次的 Proactive Recovery 都需要如下处理[3,12-13]:

(1)每一台服务器从可信的只读介质重新启

动，恢复到运行代码无错误的状态.

(2) 对于每一个变量，每一台服务器要重新获取正确的值和时戳，恢复到数据无错误的状态.

(3) 每一台服务器更新 Server Key 并向其它服务器分发 Server Key 公钥、重新拆分 Service Key 私钥，保证攻击者不能获得超过 f 台服务器的 Server Key 私钥或者恢复出 Service Key 私钥.

完成上述 Proactive Recovery 需要消耗大量的服务器资源；尤其当存储大量数据时，需要繁重的通信过程才能保证恢复到数据无错误的状态. 所以，在 TSS-BQS 系统中，应尽量减少 Proactive Recovery 次数. 另一方面，对于给定的容错系统和运行环境，执行 Proactive Recovery 的周期与系统容忍的失效服务器数量成正比：容忍的失效服务器数量越少，需要越频繁地定期执行 Proactive Recovery[10].

结合本文设计的 Graceful Degradation 机制，TSS-BQS 系统能以如下方式来执行 Proactive Recovery：

(1) 在系统初始启动或者每一次 Proactive Recovery 之后，各服务器以 TSS-mBQS 状态运行.

(2) 以 TSS-mBQS 状态运行 $\frac{f_m}{f_d}T_{PR}$，然后切换到 TSS-dBQS 状态. 其中，T_{PR} 是根据运行环境的特性，估计不超过 f_d 台服务器失效的最长时间；也就是，已有 TSS-BQS 系统[3,5-7] 的 Proactive Recovery 周期.

(3) 以 TSS-dBQS 状态运行 $\left(1-\frac{f_m}{f_d}\right)T_{PR}$，然后执行 Proactive Recovery.

本文设计的 Graceful Degradation 机制，具有如下比较优势和灵活性：

(1) 如果 n 台服务器组成纯粹的 TSS-mBQS 系统(始终以 TSS-mBQS 状态运行)，只能容忍 $\left\lfloor\frac{n-1}{4}\right\rfloor$ 台失效服务器. 相比而言，我们的系统能容忍 $f_d=\left\lfloor\frac{n-1}{3}\right\rfloor$ 台失效服务器，Proactive Recovery 周期延长了约 1/3，减少了 Proactive Recovery 次数；代价则是只有约一半时间具备 TSS-mBQS 系统的性能、其余时间的性能只相当于 TSS-dBQS 系统.

(2) 相比纯粹的 TSS-dBQS 系统，我们的 Proactive Recovery 周期相同，但是有约一半时间以 TSS-mBQS 状态运行、性能更高，提高了系统的平均性能.

5.2　状态切换协议的发起

状态切换协议可以在如下情况下发起：

(1) 结合 Proactive Recovery，在预定时间定期发起. 任何正确服务器，在预定时间到达时，自动转为 Delegate，发起切换协议；或者，也可以采取 PBFT[12] 所使用的轮换方式：各服务器按顺序轮流担任 Delegate，如果发现当前 Delegate 未完成状态切换协议，则认为该 Delegate 已失效；由其余服务器合作确定下一台服务器来担任 Delegate. 切换协议中的 $reason$=[Schedule, $time$]，其中 Schedule 表示是按预定时间执行，$time$ 则是预定的时间. S_i 验证 $reason$ 是否有效：$time$ 是否与 S_i 设定的时间一致、$time$ 与 S_i 本地时钟的误差在允许范围内.

(2) 发生某些预料之外的重大安全事件，对运行环境的原有估计不足，则需要在预定时间之前就发起切换协议. 例如，爆发全新的严重病毒和木马、可能会影响服务器状态，或者发现了服务器的系统漏洞，但相应补丁尚未发布等. 此时，发现安全事件的正确服务器会转为 Delegate，发起切换协议，相应的 $reason$=[Event, $description$]，其中 Event 表示是因为安全事件而切换状态，$description$ 给出安全事件的描述(例如，发布病毒、木马或系统漏洞的可信网站链接). S_i 验证 $reason$ 需要人工干预，由管理员阅读 $description$，判断是否有必要进行状态切换.

6　相关研究

文献[4]最早提出了 Graceful Degradation 的概念：运行环境恶化、超出预先估计时，系统调整运行状态，继续提供服务但是服务质量有所降低. 此后的研究工作将 Graceful Degradation 概念应用于集群服务[14]、存储阵列[15] 等领域，设计了以降低性能[14,16]、减少可访问文件数量[15]、降低响应结果全面性[16] 和准确性[17] 等为代价的 Graceful Degradation 机制，保证失效情况下的服务可用性. 针对状态机复制 State Machine Replication 系统，文献[18-19]分别讨论了当失效服务器数量超过预先估计时，通过限制失效服务器的恶意操作[18] 和支持部分操作命令[19] 来实现 Graceful Degradation. 但是，上述研究的 Graceful Degradation 机制都不是针对 BQS 系统的也无法直接应用于 BQS 系统.

COCA[3] 首次结合了门限签名方案和 Dissemination BQS 系统，完成了 TSS-BQS 系统(也相当于本文的 TSS-dBQS 状态). 使用与 COCA 相同的服务器协议，CODEX[5] 实现了容错的机密性数据存储

服务. SP-II 设计了一种新的 TSS-BQS 系统服务器协议[6]. 文献[7]证明:TSS-BQS 系统只存在两类有效的服务器协议,且两类协议的代表分别就是 COCA[3]和 SP-II[6]. 类似地,PBFT-BC[20]利用门限签名方案来扩展了 BFT-BC 协议[21],完成了容忍失效客户端的 BQS 系统,同时支持 Proactive Recovery.

在设计思想上,TSS-BQS 系统的 Proactive Recovery[3,10-11]和 Dynamic BQS 系统[22],与本文的 Graceful Degradation 机制有一定的相似之处:在失效服务器数量有可能要超过预先估计时,提前采取措施,使系统能继续提供容错服务. Proactive Recovery是定期性地将所有服务器恢复到初始的正确状态,避免失效服务器数量超过预先估计;而本文 Graceful Degradation 机制是改变存储数据的类型(也就是改变 BQS 系统的类型),使得能够容忍更多的失效服务器.此外,需要注意,本文提出的 Graceful Degradation 机制完全可以和 Proactive Recovery 同时结合使用.

Dynamic BQS 系统[22]是由 $4f+1$ 台服务器组成的 Masking BQS 系统,最多能容忍 $f(f>1)$台失效服务器:在初始阶段,设定适当的 q_r/q_w,只容忍 1 台失效服务器;随着系统运行,不断地动态改变 q_r/q_w,降低性能、容忍 2、3、…、f 台失效服务器. 本文的 Graceful Degradation 机制与 Dynamic BQS 系统的区别主要在于:(1) Dynamic BQS 系统的容错参数变化针对同一类型的 BQS 系统,而 Graceful Degradation 机制是在两种不同类型的 BQS 系统之间切换;(2) Dynamic BQS 系统只是变化了每次读/写操作的服务器数量,通信轮数和计算时间没有明显变化,性能提高有限,而我们的状态切换改变了通信轮数和计算时间,性能提高更为明显;(3) Dynamic BQS 系统的客户端需要自己处理容错参数,而我们的机制基于 TSS-BQS 系统,客户端不需要知道系统的运行状态.

7 总 结

本文设计了用于 TSS-BQS 系统的 Graceful Degradation 机制:系统由 $n=3f_d+1$ 台服务器组成,在初始阶段以 TSS-mBQS 状态运行,能容忍 $f_m=\left\lfloor\frac{f_d}{2}\right\rfloor$台 Byzantine 失效服务器;随着系统运行,失效的服务器数量增大、有可能将要超过 f_m,以降低性能为代价,切换到 TSS-dBQS 状态,能容忍 f_d 台 Byzantine 失效服务器. 在我们的 Graceful Degradation 机制中,客户端并不需要知道系统的运行状态(TSS-mBQS 或 TSS-dBQS 状态),系统可以在不影响客户端、不中断存储服务的前提下完成状态切换.

本文设计的状态切换协议,使得相比纯粹的 TSS-dBQS 系统[3,5-7](由 $n=3f_d+1$ 台服务器组成、容忍 f_d台 Byzantine 失效服务器),能以更高的平均性能运行;在失效服务器数量达到 f_m、丧失容错能力时,通过快速的状态切换,继续提供容错的存储服务,相比纯粹的 TSS-mBQS 系统,减少了 Proactive Recovery 次数. 在不影响 TSS-BQS 系统容错效果的前提下,Graceful Degradation 机制带来了更高的平均性能,是已有 TSS-BQS 系统 Proactive Recovery 的重要补充.

参 考 文 献

[1] Malkhi D, Reiter M. Byzantine quorum systems. Distributed Computing, 1998, 11(4): 203-213

[2] Lamport L, Shostak R, Pease M. The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401

[3] Zhou L, Schneider F, Renesse R. COCA: A secure on-line certification authority. ACM Transactions on Computer Systems, 2002, 20(4): 329-368

[4] Herlihy H, Wing J. Specifying graceful degradation. IEEE Transactions on Parallel and Distributed Systems, 1991, 2(1): 93-104

[5] Marsh M, Schneider F. CODEX: A robust and secure secret distribution system. IEEE Transactions on Dependable and Secure Computing, 2004, 1(1): 34-47

[6] Jing Ji-Wu, Wang Jing, Lin Jing-Qiang, Xie Yong-Quan, Gu Qing. Server protocols of Byzantine quorum systems implemented utilizing threshold signature schemes. Journal of Software, 2010, 21(10): 2631-2641(in Chinese)
(荆继武,王晶,林璟锵,谢永泉,顾青. 基于门限签名方案的 BQS 系统的服务器协议. 软件学报, 2010, 21(10): 2631-2641)

[7] Lin Jing-Qiang, Liu Peng, Jing Ji-Wu, Wang Qiong-Xiao. Impossibility of finding any third family of server protocols integrating Byzantine quorum systems with threshold signature schemes//Proceedings of the 6th International ICST Conference on Security and Privacy in Communication Networks (SecureComm). Singapore, 2010: 307-325

[8] Desmedt Y. Society and group oriented cryptography: A new concept//Proceedings of the Advances in Cryptology-Crypto'87. Santa Barbara, USA, 1987: 120-127

[9] Shoup V. Practical threshold signatures//Proceedings of the Advances in Cryptology-EuroCrypt'2000. Bruges, Belgium, 2000: 207-220

[10] Ostrovsky R, Yung M. How to withstand mobile virus attacks//Proceedings of the 10th ACM Symposium on Principles of Distributed Computing (PODC). Montreal, Canada, 1991: 51-59

[11] Canetti R, Herzberg A. Maintaining security in the presence of transient faults//Proceedings of the Advances in Cryptology-Crypto'94. Santa Barbara, USA, 1994: 424-438

[12] Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 2002, 20(4): 398-461

[13] Castro M, Liskov B. Proactive recovery in a Byzantine-fault-tolerant system//Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation (OSDI). San Diego, USA, 2000: 273-287

[14] Fox A, Gribble S, Chawathe Y, Brewer E, Gauthier P. Cluster-based scalable network services//Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP). St. Malo, France, 1997: 78-91

[15] Sivathanu M, Prabhakaran V, Arpaci-Dusseau A, Arpaci-Dusseau R. Improving storage system availability with D-GRAID. ACM Transactions on Storage, 2005, 1(2): 133-170

[16] Brewer E. Lessons from giant-scale services. IEEE Internet Computing, 2001, 5(4): 46-55

[17] Fox A, Brewer E Harvest. Yield and scalable tolerant systems//Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS). Rio Rico, USA, 1999: 174-178

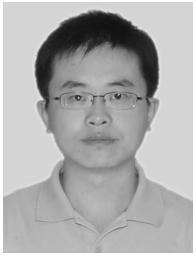
[18] Li J, Mazieres D. Beyond one-third faulty replicas in Byzantine fault tolerant systems//Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI). Cambridge, USA, 2007: 131-144

[19] Zhou L, Prabhakaran V, Ramasubramanian V, Levin R, Thekkath C. Graceful degradation via versions: Specifications and implementations//Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC). Portland, USA, 2007: 264-273

[20] Alchieri E, Bessani A, Pereira F, Fraga J. Proactive Byzantine quorum systems//On the Move to Meaningful Internet Systems Workshops (OTM), Vilamoura, Portugal, 2009: 708-725

[21] Liskov B, Rodrigues R. Tolerating Byzantine faulty clients in a quorum system//Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS). Lisboa, Portugal, 2006: 105-114

[22] Alvisi L, Malkhi D, Pierce E, Reiter M, Wright R. Dynamic Byzantine quorum systems//Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). New York, USA, 2000: 283-292



WANG Wen-Tao, born in 1983, Ph. D. candidate. His current research interests focus on network security.

LIN Jing-Qiang, born in 1978, Ph. D., assistant professor. His research interests include network and information security.

JING Ji-Wu, born in 1964, professor. His research interests include information security.

LUO Bo, born in 1978, Ph. D., assistant professor. His research interests include network and information security.

Background

Lots of distributed storage systems and cloud storage services are built on Byzantine quorum systems (BQSs). Graceful degradation has been proposed for more than 20 years, and our work designed the first graceful degradation mechanism for TSS-BQS systems, a special variation of BQSs. Our work is an important part of the Strategy Pilot Project of Chinese Academy of Sciences "Information Security Essential Technologies for Sea-Cloud Computing". The graceful degradation design of TSS-BQS systems, helps to implement a more efficient and fault-tolerant cloud storage system. Besides the graceful degradation design, we had designed a new server protocol of TSS-BQS systems and proved that there are only two efficient server protocols of TSS-BQS systems.