ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.12, December 2010, pp.3199−3210
doi: 10.3724/SP.J.1001.2010.03718

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

大规模集群中一种自适应可扩展的RPC超时机制*

钱迎进+, 肖 侬, 金士尧

(国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

Adaptive Scalable RPC Timeout Mechanism for Large Scale Clusters

QIAN Ying-Jin+, XIAO Nong, JIN Shi-Yao

(National Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: coolqyj@163.com

Qian YJ, Xiao N, Jin SY. Adaptive scalable RPC timeout mechanism for large scale clusters. *Journal of Software*, 2010,21(12):3199−3210. http://www.jos.org.cn/1000-9825/3718.htm

Abstract: Timeouts are usually used for failure detection in RPC (remote produce call) based systems, which are typically reported on a per-call basis. During pressure testing, on a very large cluster system, it has been found that the traditional fixed timeout mechanism leads lots of unnecessary timeouts, especially when the server loading is involved. This paper proposes an Adaptive Scalable RPC Timeout (AST for short) mechanism that considers network conditions, server load, scalability, and performance. Under this control, the timeout value, set by clients, can be adapted and adjusted in a dynamic fashion, according to congestion of the network and the server. Moreover, the server can notify the client to modify the timeout value of the RPC. Via a series of simulations, it has been proved that the AST mechanism is a more suitable failure detection mechanism for RPC models with timeouts, and it enhances the system responsibility, reliability, and stability without negative impact on performance, even for large-scaled cluster systems.

Key words: RPC (remote produce call); failure detection; timeout; large scale; scalability; responsibility; reliability

摘 要: 在基于 RPC(remote produce call)构建的分布式系统中,超时是一种通用的失效检测手段.在超大规模 Lustre 存储集群的压力测试中,发现传统的固定超时机制会导致很多不必要的超时而存在缺陷.提出了一种综合考虑了网络条件、服务器负载、扩展性和性能等因素的自适应可扩展的 RPC 超时机制(Adaptive Scalable RPC Timeout mechanism,简称 AST).在其控制下,客户端超时值可以根据网络和服务器的拥塞情况动态地调整设置,而且服务器可以通过额外消息传递通知客户端修改原超时值.经过一系列的模拟和验证,其结果表明,AST 是一种更适合的 RPC 失效检测模型,增强了系统的响应性、可靠性和稳定性,而且对系统的性能没有过大的负面影响.

关键词: 远程过程调用;失效检测;超时;大规模;扩展性;响应性;可靠性

中图法分类号: TP316 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60736013 (国家自然科学基金)
Received 2009-04-28; Revised 2009-06-09; Accepted 2009-08-12

据统计,在全球计算能力最强大的 500 台计算机中,集群所占的比重已经超过了 70%[1].集群系统已经成为构建高性能计算机系统的主流体系结构之一,并有向超大规模发展的趋势.这些集群系统一般采用了客户端服务器模型,节点间以远程过程调用(remote produce call,简称RPC)的方式进行通信.在基于RPC构建的分布式系统中,消息丢失、网络失败和节点崩溃等失效情况会导致系统的可靠性问题.如何准确及时的发现系统中的失效是实现高可靠的重要环节.如果失效未能及时检测出来,将严重影响系统的响应性和可用性;相反,如果经常发生虚警,则会导致错误的恢复行动,降低系统效率.因此,在RPC协议设计中,尤其是那些面向大规模系统应用的RPC协议,如何检测失效是一个需要密切关注的问题.超时是一种常用的失效检测手段,它通常与一个远程过程调用绑定在一起[2].大多数RPC协议都使用超时来检测失效[3−7].这种失效检测机制对于底层传输协议不可靠的RPC协议尤为有用.因而,RPC超时机制直接影响到基于RPC构建的分布式系统的很多方面,特别是响应性、可靠性、稳定性等.

关于分布式系统中的RPC超时失效检测的研究较少.传统的RPC协议通常使用简单的固定超时机制. DCE RPC[8,9]和ONC RPC[4,10]是两个被广泛使用的RPC协议.DCE RPC调用库为应用程序提供一些设置客户端服务器间超时值的API.超时值在客户端与服务器绑定通信的句柄时设置,并在整个通信会话期间保持不变.根据ONC RPC标准,客户端在等待RPC应答消息时会简单为其设置一个固定的超时值.如果应答消息没有在指定的超时期限内到达,RPC底层实现就会通知发生失效.此时,客户端就会执行应用程序指定的失效修正操作,如重传RPC请求、终止客户端应用程序或者直接向用户终端报错.在专利[8]中提出了一种自适应超时设置方法,考虑了客户端和服务器间的通信时间.客户端跟踪RPC的响应时间,并记录在一个RPC响应时间数组中.在设置超时值时,客户端根据记录的响应时间计算出 1 个最佳的超时值.据我们所知,针对超大规模集群系统,本文提出的AST机制是第 1 个同时考虑网络条件、服务器负载、可扩展和性能等因素的RPC失效检测机制.

本文提出的AST机制主要有两点贡献:首先,在超大规模分布式集群系统中,由于网络和服务器负载情况随时间动态变化的原因,RPC的往返时间(round trip time,简称RTT)是随时间动态变化的,一般很难用一个合理的时间窗口来预测.有时,RPC RTT甚至会达到几百秒,因而很难安全地评估RPC超时值的上限.针对这个问题,提出了自适应超时(adaptive timeout,简称AT)策略,客户端的超时值可以根据网络和服务器的拥塞情况动态地调整设置,从而避免不必要的超时发生;本文的另一个贡献是提出了及早回复(early reply,简称EP)策略.在其控制下,客户端超时值是可变的,服务器可以通过额外消息传递通知客户端修改原超时值,进一步减少了不必要的超时的发生.同时,将它和其他的RPC失效检测机制进行了比较.例如在Cedar RPC[11]中,客户端周期的向服务器发送一个探测报文,请求从服务器获得确认.通过这种策略,一旦服务器崩溃或者发生严重的通信失效,客户端能够检测到并通知用户程序发生了异常;NCA/RPC[7,12]提供了一个专门的例程供客户端发送“ping”报文来查询挂起的RPC请求,而且可以发送“quit”报文通知服务器终止一个远程调用的执行.Spirte RPC[6]模型虽然使用了超时机制,但它还使用了其他机制来进行失效检测.当远程调用很长时间还未处理完,导致客户端RPC超时,客户端会重新发送请求消息给服务器,请求获得一个显式的应答.服务器回应一个显式的应答并指示远程调用还在处理中.上述机制都是通过客户端驱动的额外消息传递来实现失效检测.分析得出,这类机制会造成可观的额外网络流量,特别是在承受重负载的超大规模集群中,对系统的性能有较大的影响.相对地,我们的AST机制更加适用.它的额外消息传递是由服务器驱动的.它产生更少的额外消息传递,对系统性能没有过大的负面影响.

1 观测与分析

我们以广泛应用于HPC的分布式集群文件系统Lustre[13]对超大规模集群系统服务器端RPC进行了观测.根据 2008 年度统计,全球十大超级计算机中有 6 个以上以及 40%的TOP100 超级计算机都在使用它[1].其最大配置规模已达 25 000 客户端节点.Lustre实现了自己的RPC协议,支持多种底层网络协议实现,包括TCP, InfiniBand,OFED,RapidArray,Quadrics Elan,Myrinet等.图 1 显示了服务器端挂起RPC请求的数目随时间的动态变化.该测试结果来源于ORNL的大规模Jaguar[14]集群系统,8 000个客户端产生的读写I/O负载分布到72个存储服务器,存储服务器后端磁盘的峰值带宽为400MB/sec,节点间使用了4Gbps的光纤互联.图中X轴为时间, Y轴为

服务器各个时刻挂起的RPC请求的数目,它包括不同的RPC类型,如READ,WRITE,CREATE,DESTORY,CONNECT,DISCONNECT,SET_INFO,ENQUEUE,PING等.从图中可以看出,服务器端挂起的DISCONNECT RPC请求数目最多达到 5 500 个,READ RPC请求数目最多达到 800 个.在对大规模Lustre集群文件系统进行极端压力测试的过程中,观测到了很多超时的发生.通过调查研究我们发现,服务器端挂起的RPC请求数目最多达到近 60 000 个(其中,每个RPC请求在执行的过程中传递读/写 1M的I/O数据);如此之多的RPC请求竞争共享磁盘带宽,最大的RPC服务时间达到可观的 100 多秒;客户端预先设置的固定超时值过小而不能适应大规模RPC工作量的变化,从而导致了很多无效的超时的发生.我们观测到在重载拥塞的集群中,RPC请求的服务时间非常可观.随着集群规模的扩大,我们发现了很多因固定超时机制导致的扩展性问题.

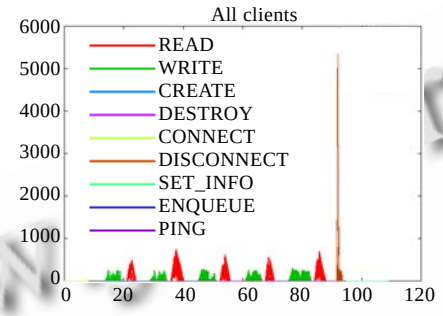


Fig.1 Traces of the number of queued RPCs over time on the server of the Lustre cluster

图 1 大规模 Lustre 集群服务器端 RPC 请求观测

在基于RPC的分布式系统应用中[7],每个RPC请求都有一个超时值,该值在发送到目标节点前确定,并在整个与服务器通信会话期间使用.静态固定超时值机制一般为所有或者同一类RPC请求设置特定的超时值,没有足够考虑网络条件和服务器负载的动态变化而存在缺点.在客户端/服务器应用环境中,由于不断增加的工作负载和网络拥塞,服务器的响应延迟会显著增加,客户端原来设置的固定超时值可能太短而不能适应环境的变化.当超时发生时,可能会导致客户端应用过早地非正常退出.有些客户端应用在发生超时后会试图重新连接服务器,重传出错失败的请求.这些操作反而会加重网络负荷,加剧网络拥塞和服务器工作负载,甚至导致客户端其他操作的超时;另一方面,并不是客户端应用将超时值设置得越大越好,当服务器负载或网络拥塞减轻时,大的超时值反而会减弱超时机制的响应性.换句话说,当服务器由于某种原因(如断电、网络失效等)不能对客户端请求做出响应时,客户端在检测到失效时可能会多等一段不必要的时间,从而损害了性能.

根据以上分析,对于新型的超大规模集群系统,特别是有密集共享资源访问的分布式应用系统,固定超时机制不再适用.在下面两节将介绍我们的自适应可扩展超时机制(AST),其主要目的是减少或避免无效超时的发生和提高系统响应性.本文的很多研究工作都是基于大规模存储集群 Lustre 这种特定的应用环境展开,而且该论文给出的算法已经整合到 Lustre v1.6.5 版本中.虽然如此,本文给出的有些算法也具有通用性,可以应用到一般的基于 RPC 构建的分布式系统中.

2 自适应超时策略

为弥补现有研究的不足,我们提出了自适应超时策略.其主要原则为,客户端设置的 RPC 的超时值可以根据网络条件和服务器负载以一种动态的方式进行调整.当网络或服务器拥塞、RPC RTT 增大时,客户端的超时值必须随之相应地增加.类似地,当服务器负载或网络流量减少、RPC RTT 减少时,客户端的超时值能够随之降低.

根据 RPC 处理流程,我们将 RPC RTT 用如下公式表示:

$$RTT_{rpc}=T_{net}+T_{service},$$

其中,RTT_{rpc}表示RPC RTT;T_{net}代表RPC的网络延迟;$T_{service}$表示RPC的服务时间,它包含请求在服务器上的等待的时间和执行的时间两部分.

一般而言,超大规模集群系统的工作负载虽然随时间动态变化,但在时间上却表现出一定程度上的规律性和持续性.例如在面向科学计算的集群存储系统中,来自各个客户端的数据访问吞吐率就呈现这种特性[15].我们的自适应超时算法的基本思想为:客户端和服务器跟踪记录RPC的发送时刻(T_{send}),到达服务器时刻($T_{arrival}$)以及服务时间($T_{service}$)等;服务器根据RPC的$T_{service}$的历史记录、当前负载状况等信息估测RPC的服务时间EST(estimated service time),并将EST捎带在RPC回复消息中反馈给客户端;客户端根据客户端服务器间过去一段时间RPC的T_{net}的历史记录,估测出当前的网络延迟ENT(estimated network latency time);客户端根据ENT和反馈的EST(feedback estimated service time,简称FEST)来设置当前准备发送的RPC请求的超时值.

2.1 滑动时间窗口算法

自适应超时算法的核心是滑动窗口(sliding time window,简称STW)算法.每个STW记录了最近时间长度为H的历史记录.整个时间窗口被分为N等分,每部分被称为子时间窗口(sub sliding time windows,简称SSTW),其时间长度为H/N.每个SSTW都有一个记录.它按照具体算法应用的需求记录在它自身所管辖的时间窗口内加入的记录值.整个时间窗口以时间步长H/N的整数倍向前移动.如果一次向前滑动多个步长,则重置没有加入任何记录的SSTW.我们使用STW来跟踪RPC的$T_{service}$和T_{net}的变化历史记录.根据STW中的记录值,估测ENT和EST.

首先,我们提出简单的 STW 算法,称为 MAX 算法.该算法中,保存于每个 SSTW 中的记录值为在它所管辖的时间窗口内加入记录的最大值,估测值为 N 个 SSTW 记录值的最大值.为便于理解,将 MAX 算法的 STW 定义为一个三元组 $STW=\langle H,T,v[N]\rangle$.其中:$H$ 表示整个时间窗口长度;T 表示时间窗口的起始时刻;N 表示 SSTW 的个数;v 是一个有 N 个元素的一维数组,其中每个元素用于保存对应 SSTW 的记录值.MAX STW 算法中计算估测值的公式可以表示为

$$Estimate(v)=\mathrm{MAX}_{i=0}^{N-1}(v[i]).$$

图 2 展示了应用MAX STW算法的一个简单的例子,其中,H=50s,N=5,各个时间窗口记录值的初始值为 0.在t_0时刻,加入值为 30s的记录,$T=t_0$,$v[0]$=30;在t_0+25s时刻,加入值为 40s的记录,向前移动两个时间窗口,$T=t_0+20$,$v[0]$=40,$v[2]$=30;在t_0+28s时刻,接收到值为 50s的记录,与前一记录位于同一个时间窗口,但新记录值大于旧记录值,故$T=t_0+20$,$v[0]$=50,$v[2]$=30.

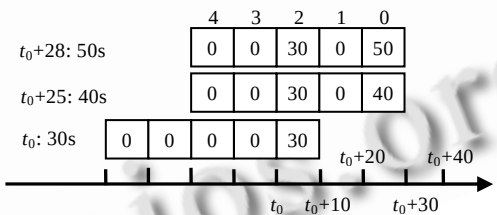


Fig.2 Sliding time window algorithm

图 2 滑动时间窗口算法

2.2 基于STW的自适应超时算法

上面提到过,AT策略使用了STW来估测EST和ENT.每个服务器都有一个EST STW来跟踪记录自身RPC的$T_{service}$随时间的动态变化;在客户端维护有每个客户端服务器对(client server pair,简称CSP)间的ENT STW和FEST STW,分别用来跟踪记录每个CSP随时间的动态变化的网络延迟以及服务器反馈的EST.我们可以对每个服务器的EST进行跟踪,甚至可以跟踪服务器提供的每个RPC服务类型的EST.每个RPC跟踪记录自身的发送时刻(T_{send})、达到时刻($T_{arrival}$)、服务时间($T_{service}$).每当服务器处理完一个RPC请求时,计算出它的$T_{service}$并加入到EST滑动时间窗口;当客户端接收到回复消息时,计算出RPC的T_{net}加入对应的ENT滑动时间窗口中,并将反馈的EST加入到对应的CSP的FEST滑动时间窗口中.客户端根据ENT和FEST STW中的记录值来设置发送RPC请求的超时值,其计算公式为

$$T_{timeout}=Estimate(ENT)+\lambda\times Estimate(FEST),$$

其中:$T_{timeout}$是设置的超时值;*Estimate*是使用STW中的N个历史记录作为输入的一个估测函数,第 2.1 节中的MAX STW算法公式就是一个估测函数;$Estimate(ENT)$表示ENT STW估测出的网络延迟值;$Estimate(FEST)$表示FEST STW的估测出的服务器服务时间值;λ表示该值的放大系数,由于FEST与服务器当前时刻的EST存在偏差,故$\lambda\geqslant 1$.同时,为了避免客户端设置的超时值过大或者过小,我们定义了它的上下限$AtMin\leqslant T_{timeout}\leqslant AtMax$.

根据一般RPC协议的处理流程,自适应超时算法涉及到客户端发送RPC请求时设置超时值、接收回复消息时增加网络延迟和FEST记录到对应的STWs、服务器端接收RPC请求的处理以及处理完后发送RPC回复消息等 4 个过程.为了方便算法描述,我们将RPC表示为以下的五元组:$RPC=\langle T_{send},T_{arrival},deadline,T_{service},FEST\rangle$,其中,$T_{send}$表示RPC发送时刻,$T_{arrival}$表示RPC到达服务器时刻,*deadline*表示客户端设置的RPC超时期限时刻,$T_{service}$表示RPC的服务时间,*FEST*表示服务器反馈给客户端的当前EST值.客户端服务器对表示为$CSP=\langle stwNet,stwFEST\rangle$,其中,*stwNet*表示记录客户端服务器对间网络延迟的滑动时间窗口,*stwFEST*表示记录服务器反馈的EST记录的滑动时间窗口.服务器端用来跟踪RPC服务时间的STW被表示为*stwTSE*.基于MAX STW算法的自适应超时算法的伪代码描述见算法 1:

算法 1. 基于 STW 的自适应超时算法.

1: **Procedure** *SendRPCRequest*(*rpc*)
2: $rpc.T_{send}=now$; //@now: current time
3: $rpc.deadline=rpc.T_{send}+Estimate(CSP.stwNet)+\lambda\times Estimate(CSP.stwFEST)$;
4: Send the *RPC* request message to the server;
5: **end procedure**

6: **Procedure** *ReceiveRPCRequest*(*rpc*)
7: $rpc.T_{arrival}=now$;
8: Enqueue the new *RPC* request, waiting for service;
9: **end procedure**

10: **Procedure** *SendRPCReply*(*rpc*)
11: $rpc.T_{service}=now-rpc.T_{arrival}$;
12: $AddRecord(stwEST,rpc.T_{service})$; //add the service time value to the *stwEST* on the server
13: $rpc.FEST=Estimate(stwEST)$;
14: Send the *RPC* reply message with FEST to the client;
15: **end procedure**

16: **Procedure** *ReceiveRPCReply*(*rpc*)
17: $netlatency=now-rpc.T_{send}-rpc.T_{service}$;
18: $AddRecord(CSP.stwNet,netlatency)$;
19: $AddRecord(CSP.stwFEST,rpc.FEST)$;
20: Process the *RPC* reply message;
21: **end Procedure**

2.3 基于STW的服务时间估测算法

在实现中,客户端的ENT和FEST STW都使用了简单的MAX STW算法.由于使用高速网络互连,在超大规模HPC集群系统中,网络通常不是系统的瓶颈;而相对的,I/O瓶颈一直是达到高性能的主要阻碍.从重负载的Lustre集群的RPC日志中我们观测到,服务器挂起的RPC请求数目在短时间内可以达到上万个.而且RPC的T_{net}

只占RPC RTT很小的一部分,通常小于 1ms;而RPC的$T_{service}$通常占用很大一部分,有时甚至会高达几百秒;其中,RPC的执行时间只有几十毫秒,在服务器RPC队列中等待被服务的时间通常占用非常大一部分.因此在这种环境中,过重的服务器负载是造成无效超时发生的主要原因,RPC的服务时间的估测显得尤为重要.在本节,我们主要介绍基于STW的服务时间估测算法.

在上述环境下,由于T_{net}很小,反馈给客户端的最佳的EST应该是当前时刻到达RPC请求的$T_{service}$.对于持续稳定的RPC工作流,RPC的$T_{service}$通常是一个常量或者变化不大,MAX STW算法足以正确地评估服务器的EST,但是MAX STW算法不适用于RPC工作流变化比较剧烈的环境.在MAX STW算法中,EST是过去一段时间在RPC完成时刻加入到STW的最大的$T_{service}$.我们可以将其视为根据RPC完成时刻($T_{arrival}$+$T_{service}$)估测的最佳EST,而不是当前新到达请求的最佳EST.在STW中具有最大$T_{service}$的RPC在等待被服务的这段时间内,可能会有很多的新RPC请求达到服务器,因此,MAX STW算法没有考虑在这段时间内服务器负载的动态变化而导致服务时间的变化.我们提出了一种新的RPC的$T_{service}$估测算法,称为最小二乘法曲线拟合服务时间估测算法(LCF算法).它扩展了加入SSTW中的记录值,根据N个SSTW记录对EST进行预测.

在LCF算法中,记录值是一个时间值对的形式$\langle t_i,v_i\rangle$,(i=0,1,…,N–1).其中,t_i是RPC的到达服务器时刻$T_{arrival}$,v_i是对应的RPC的实际服务时间$T_{service}$,每个SSTW记录了加入它管辖的时间窗口内的具有最大RPC的$T_{service}$的记录值.根据N个离散的时间值对记录我们使用最小二乘法曲线拟合方法确定一条曲线拟合方程,然后根据公式计算出当前时刻t的估测值.那么当前时刻的EST v的估测方程可以表示为

$$Estimate(v)=f(t)=a_0+a_1\times t,$$

$$a_1=\left[\sum_{i=0}^{N-1}tv_i-\left(\sum_{i=0}^{N-1}t_i\sum_{i=0}^{N-1}v_i\right)\Big/N\right]\Big/\left[\sum_{i=0}^{N-1}t_i^2-\left(\sum_{i=0}^{N-1}t_i\right)\Big/N\right],$$

$$a_0=\left(\sum_{i=0}^{N-1}v_i\right)\Big/N-a_1\times\left(\sum_{i=0}^{N-1}t_i\right)\Big/N.$$

图 3 显示了使用LCF算法估测RPC的服务时间的例子.它共有 8 个SSTW,通过记录于STW的 8 个离散的时间值对拟合出曲线方程,估测出当前时刻 30s新到达的RPC请求的服务时间为 35s.LCF算法根据过去一段时间RPC的$T_{service}$的变化趋势来预测当前EST,它比MAX算法更加适合于工作流变化剧烈的环境.

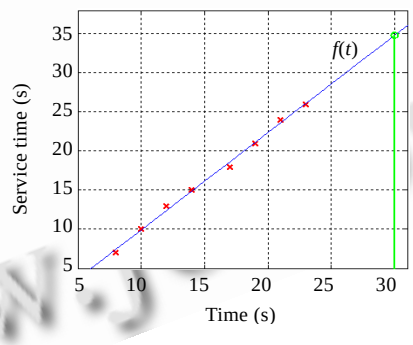


Fig.3 LCF service time estimation algorithm

图 3 LCF 服务时间估测算法

3 及早回复策略

在大规模集群系统中,网络或节点失效发生比较频繁,它们通常是通过超时来检测.当前,大规模集群要求系统具有高可靠性,节点断电重启、网络短暂失效后系统能够自动恢复.在服务器恢复的过程中,成千上万的客户端要尝试与服务器重新建立连接,重传未提交的 RPC 请求.所有这些操作都会产生突发 RPC 工作流,加剧网络拥塞和服务器工作负载,可能会导致其他操作或客户端超时,处理不当甚至会造成整个系统崩溃.显然,恢复的代价是巨大的.尽管 AT 策略可以动态调整超时值以适应环境的变化,但是由于 EST 反馈延迟以及 AT 策略对突发工作流很难进行迅速的估测,故它不能完全避免超时的发生.在大规模集群中,服务器工作负载过重会造成

成千上万的RPC请求缓冲于服务器,导致处理RPC请求的阻塞,增加RPC请求的平均响应延迟,有时会导致超时的发生.此时,很难判断造成超时的原因是服务器负载过重还是服务器或网络发生故障,客户端会错误地认为网络或服务器发生失效.这种无效的超时会引发错误的恢复操作,明显会降低系统的性能.

在分布式系统中,服务器可能成为其他服务器的客户端,允许嵌入式RPC调用.有些分布式操作会涉及到多个节点,出现叠加超时问题.如图4所示的嵌入式RPC调用链:它涉及到*A*,*B*,*C*这3个节点,其中,*A*是客户端节点,*C*是服务器节点,*B*兼具客户端服务器两种功能.首先,*A*发送RPC请求r_1给节点*B*,*B*在执行r_1的过程发送RPC请求r_2给节点*C*.在该调用链中,任何一个交互过程的超时都会导致分布式操作失败,而且r_1的超时值必须大于r_2的超时值,否则节点*A*可能错误地认为*B*发生了失效.上述的嵌入式RPC调用过程是一个典型的叠加超时例子,但在一般分布式系统中很难保证r_1的超时值大于r_2的超时值.

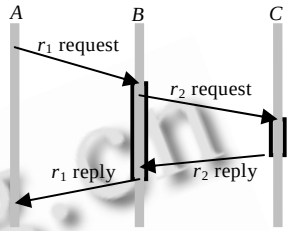


Fig.4 Nested timeout

图4 嵌入式超时

为了能够区分服务器因负载过重而拥塞和服务器宕机,以及解决叠加超时问题,我们提出了一种及早回复策略,其基本原理为:当服务器知道它不能在客户端期待的响应时间内应答RPC请求时,它将提前发送一个轻量级的及早回复消息给该客户端并告知一个估测的额外的服务时间.我们的及早回复策略的实现方法为服务器有一个按照*RPC.deadline*排序的及早回复计时链表(early reply timed list,简称ERTL)以及一个相应的计时器.当服务器接收到一个RPC请求时,先将它加入到ERTL,同时调整计时器到期时刻值为(*RPC.deadline*–*epReserve*)和当前计时器到期时刻两者的最大值,其中*epReserve*为及早回复消息传递预留的时间;当服务器完成RPC请求处理时,将其从及早回复链表中删除;当计时器过期时,检查ERTL,对即将超时的RPC请求,发送一个轻量级及早回复消息给客户端,并告之服务器估计的额外需要的服务时间,然后修正服务器RPC存根的*deadline*,重新加入到ERTL;客户端接收到及早回复消息后,重新调整客户端RPC存根的超时值,然后额外等待一段时间,期待获得正常的RPC回复消息.及早回复预留时间一般根据应用环境设置,它要满足以下条件,即及早回复消息能够在该时间内到达从服务器发送并达到客户修改超时值.服务器可以根据自身的负载状况按照一定的策略智能地反馈额外的服务时间.为了简化实现,在第4节的评估实验中,我们简单地将额外的服务时间固定设置为30s.

在相关专利[10]中给出了一种基于客户端轮询的RPC超时处理的机制:当客户端发送RPC请求后,周期性地发送一个次RPC请求消息给服务器判断服务器是否正在处理主RPC请求.如果次RPC请求被成功处理,并且回复消息指示主请求仍在处理中,那么客户端将会继续等待直到主请求处理完毕或者一段时间间隔后发送下一个次RPC请求进行轮询.这种策略有点类似ping机制,通过这种方法虽然也可以解决上述的两个问题,但每次轮询至少需要两次消息传递,而且轮询的时间间隔很难确定,过大会降低系统的响应性,过小会造成很多额外的轮询消息,增加网络流量,不适用于超大规模集群系统.我们的及早回复策略是服务器驱动的,服务器根据客户端设置的RPC超时值来发送及早回复消息,并根据自身的负载状况智能地反馈一个RPC额外需要的服务时间值,客户端根据此值调整超时期限.对于即将超时的RPC请求,一般情况下,服务器只需要发送一次及早回复消息;相对于基于客户端轮询的RPC超时处理机制,额外的消息传递更少.

在后面的模拟实验中,我们将*epReserve*固定设置为5s.在该时间内,及早回复消息一般都能在对应的RPC超时之前到达客户端,修改超时值;如果没有接收到此类消息而发生超时,客户端可以判定网络或者服务器发生了故障.通过及早回复策略,对于图4的嵌入式RPC调用,当r_1的超时值设置过小时,节点*B*会发送及早回复消息给客户端*A*调整r_1的超时值,避免发生超时,它解决了分布式系统嵌入式RPC调用的叠加超时问题.在大规模HPC集群中,一般设置大超时值对于客户端到服务器的RPC请求是可以接受的;但对于服务器发送给客户端的RPC请求,一般都需得到快速的响应,故设置得相对较小.此时,及早回复策略可以通过及早回复消息提高响应速度,同时又避免因超时值设置过小而导致的不必要的超时的发生.而且额外的及早回复消息是一个轻量级的消息,一般不会加剧网络拥塞,不会对整个系统的性能造成很大的负面影响.

4 评 估

本文以广泛应用于HPC的开源分布式集群文件系统Lustre为实验平台,通过使用Lustre模拟器[16]对我们的AST机制进行了模拟评估,比较并验证了其有效性.比较的指标主要有RPC超时率、算法的自适应性、扩展性和性能等.

Lustre 模拟器是 Sun 开发的一个专门针对 Lustre 文件系统的模拟器,用来研究 Lustre 文件系统在不同集群规模下的扩展性问题、I/O 行为以及设计各种算法.它模拟实现了磁盘、Linux I/O 调度器、文件系统、网络以及 Lustre 文件系统的客户端、元数据服务器和数据存储对象服务器等模型.它可以模拟 100 000 客户端的并发操作.在该模拟器上,我们模拟并评估了 AST 机制.

在模拟实验中,服务器使用 10Gbps NIC,服务器后端的磁盘带宽约为 300MB/s.为了适应超大规模集群存储环境的模拟,设定固定超时机制的超时值为 50s,自适应超时机制的公用参数为 AtMax=600s,AtMin=30s,λ=1.25,滑动时间窗口时间长度为 40s,子窗口个数为 8.

我们通过对 Lustre 存储集群中来自各个客户端的并发写操作的模拟来评估 AST 机制.写 RPC 处理过程为:客户端发送写 RPC 请求给服务器;服务器接收请求后,按照 FCFS 的顺序对请求进行管理,等待服务;当服务器有空闲服务线程资源时,选择一个 I/O 请求在该线程环境中执行,执行过程中会通过网络传递 I/O 数据,并将数据写入到磁盘,完成后发送 RPC 回复消息给客户端.在实际的 Lustre 集群文件系统中,文件数据被条带化到多个数据存储服务器中以达到并行的聚合 I/O 性能,而且每个 CSP 间最大可并行处理的 RPC 请求个数是一个可调的参数,默认值为 8.为了简化实验,我们将其设置为 1,并且着重考察了所有客户端将 I/O 集中到一个存储服务器节点的极端情况.

4.1 超大规模系统RPC RTT模拟评估

我们对不同规模 Lustre 系统的 RPC 最大服务器时间进行了模拟评估.实验模拟评测了一个数据存储服务器并发处理来自 1 000~64 000 不同规模客户端数目的并发写操作.每个客户端持续发送 4 个 1MB 的 I/O 请求给存储服务器,各个客户端启动 I/O 的时间偏差为 10s.图 5 显示了模拟结果,可以看出,服务器端挂起的 RPC 请求以及最大 RPC 服务时间随客户端规模的增大而增加.当客户端数目为 1 000 时,最大挂起 RPC 请求数目为 501,最大 RPC 服务时间为 2s;当客户端规模达到 64 000 时,最大挂起 RPC 请求数目为 55 440 个,最大 RPC 服务时间达到近 200s.

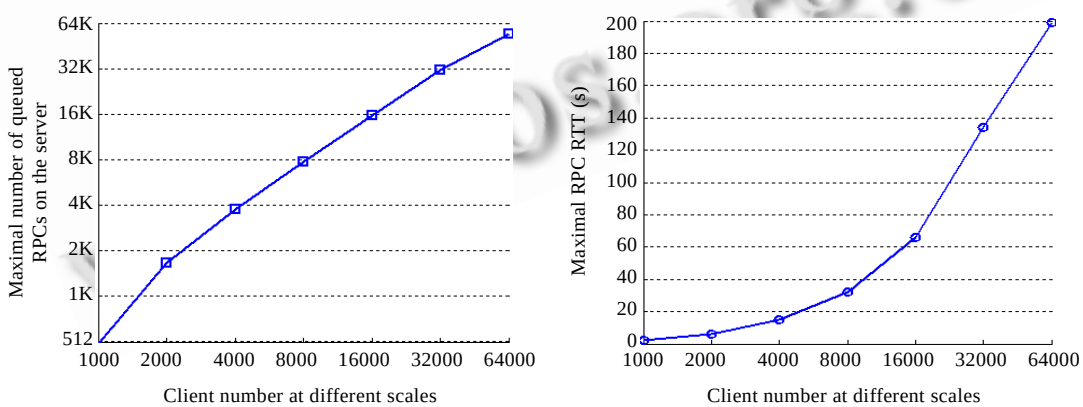


Fig.5 Max RPC RTT on the clusters at different scales

图 5 不同集群规模下最大的 RPC RTT

在模拟实验中,通过对 NIC 模块的监测我们发现,RPC 请求和回复消息占用的网络带宽很小,网络延迟也低

于 1s.在实际的大规模 Lustre 集群系统中,由于每个 CSP 间默认可以同时并行发送处理 8 个 RPC 请求,更小集群规模就可以产生成千上万的 RPC 请求挂起于服务器,导致上百秒的 RPC RTT.下面我们重点考察拥塞集群中 RPC 的服务时间的估测算法,并对我们的 AT 策略进行了模拟评估.

4.2 自适应超时策略评估

为了评估 AT 算法,对一个服务器端挂起 RPC 请求呈波峰状的实例进行了模拟,实验设计如下:

采用了 32 000 个客户端和一个数据存储服务器,为了体现 RPC 数据流的持续性,每个客户端持续发送 4 个 1M 的 I/O RPC 请求给存储服务器,将客户端分为 16 个集合,每个集合 2 000 个客户端,各个集合启动发送 I/O 给存储服务器的时间间隔为 5s.实验的主要目的是对各种服务时间估测算法进行了模拟评估,分别对比模拟测试了固定超时算法(简称为 FIX 算法)、基于 MAX STW AT 算法(简称 MAX 算法)和基于 LCF STW AT 算法(简称 LCF 算法).

图 6 显示了AT策略下各种RPC的跟踪信息.图 6(a)显示了服务器端挂起的RPC请求随时间的动态变化,可以看出,服务器端挂起RPC数目最多达到 30 000 个.在我们的实验中,它可以视为衡量服务器负载的一个度量标准.图 6(b)显示了系统每秒发生超时次数随时间的动态变化,它以时间为X轴,各种超时机制的每秒RPC超时次数为Y轴绘制而成.通过RPC日志记录,我们统计了每个RPC的$T_{service}$以及两种自适应超时算法的EST随时间的动态变化,如图 6(c)所示.曲线“ST”和“AST”都是以各个RPC请求的$T_{service}$为纵坐标,分别以RPC的完成时刻($T_{arrival}+T_{service}$)和到达时刻$T_{arrival}$为横坐标绘制而成;曲线“AST”可以被认为是最佳EST曲线;服务器RPC的最大服务时间值达到 120 秒.曲线“MAX”和“LCF”分别显示了MAX和LCF算法的EST随时间的动态变化.从图 6(b)可以看出,超时值恒定为 50s的固定超时算法从 90s开始发生超时,直到 460s结束超时.对于MAX算法,图 6(c)显示,曲线“MAX”和曲线“ST”几乎重合,而且在 260s之前它的EST一直小于曲线“AST”显示的最优值,图 6(b)显示超时主要发生在该时间段.从图 6(b)可以看出,对于MAX算法,当服务器端挂起的RPC请求数目呈上升或下降趋势时,EST增长或降低都过慢;尽管如此,当挂起的RPC请求数目恒定于某个值,服务器负载稳定时,它的EST也几乎呈现出了相似的稳定趋势.LCF算法根据服务时间的变化特性来预测EST.在 90s前,图 6(c)显示曲线“LCF”和最佳EST曲线“AST”几乎重合.在大约 150s,图 6(a)显示服务器端挂起的RPC请求开始下降,服务器负载随之下降;但是,LCF算法的EST不能迅速地减少,直到服务器完成处理RPC请求、RPC的$T_{service}$加入到对应的STW后,它才能检测到RPC服务时间开始减小的变化;尽管如此,实验结果显示,它不会导致新的超时,而且一旦检测到因工作负载变化而导致的服务时间的变化,它的EST能够相应地迅速降低.从图 6(c)的“MAX”和“LCF”曲线中我们也可以看出,相对于MAX算法而言,LCF算法的EST能够相对更智能地跟随服务器负载的变化.最终实验结果为,固定超时算法的超时率为 78%,使用MAX算法,超时率降低到 41%,而使用LCF算法,降低到 8%.

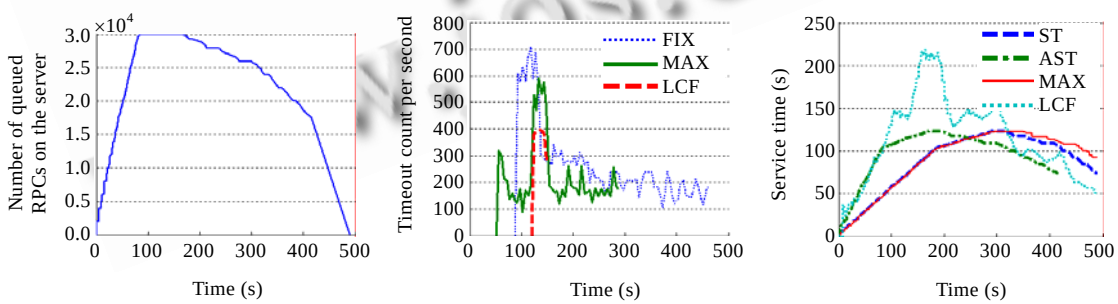


Fig.6 Traces of the RPCs using the AT strategy

图 6 AT 策略下各种 RPC 信息跟踪

在 Lustre 应用环境中,时间窗口 H 是一个可调的参数,其值一般设置为几十秒到几分钟.我们对 LCF 算法进行了如下测试:固定 H 为 40s,测试了子时间窗口分别为 2,4,8 这 3 种情况,得到的超时率分别为 14%,12%和 8%.

实验表明,时间窗口 H 选择越大,子窗口个数 N 设置越多,包含的历史记录信息就越多,从而可以对记录值按不同算法进行更加准确的预测,达到更好的超时控制,但它同时也增加了算法的计算量.

4.3 及早回复策略评估

在第 4.2 节模拟实验的基础上,我们打开了及早回复开关,模拟评估了及早回复策略.其中,每个及早回复消息大小为 4K,及早回复预留时间 *epReserve* 为 5s,及早回复消息反馈给客户端的额外的服务时间固定为 30s.实验结果显示,所有的及早回复消息都在预留时间内到达客户端并修正了客户端 RPC 存根的超时值,即使一些极端的 RPC 请求也可以通过多次及早回复消息对 RPC 超时值进行修正.最终结果显示,使用及早回复策略后,超时率为 0%.图 7 统计了不同超时机制下服务器 NIC 的输出网络流量,图中曲线“Normal”表示的正常的 RPC 回复消息产生的网络流量,其占用的网络带宽约为 0.5MB/s;曲线“*EP*(*FIX*)”表示的固定超时机制下的 RPC 应答消息和及早回复消息的网络流量,所占用的带宽不到 3MB/s.当结合自适应策略 MAX 和 LCF 算法时(分别如图中曲线“*EP*(MAX)”和“*EP*(*LCF*)”所示),它们产生的及早回复消息更少,占用的网络带宽更小.

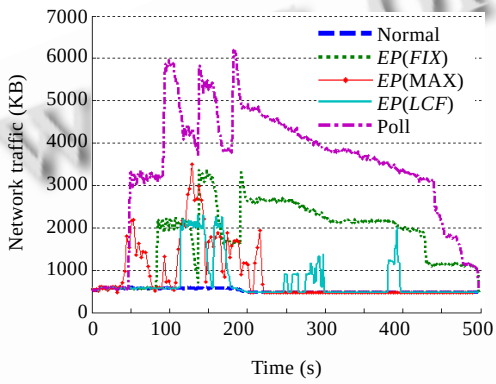
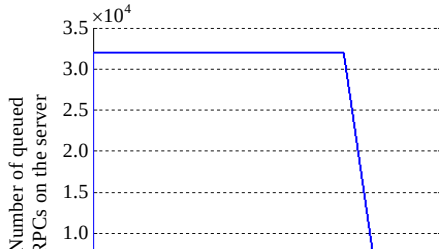


Fig.7 Traces of the network traffic for EP strategy

图 7 及早回复策略网络流量统计

我们对文献[10]中基于客户端 RPC 轮询的超时处理进行了模拟评估,并将两者进行了比较.模拟实验中,我们设置发送轮训次 RPC 请求的时间间隔为 25s.同样,在第 4.2 节模拟实验的基础上,我们对 Poll 机制进行了模拟评估.图 7 曲线“Poll”显示其所占用的服务器输出网络流量,占用网络带宽最多高达 5MB/s.计算输入网络流量的话,总共占用的额外网络带宽最多高达 10MB/s,明显高于其他几种策略.然后我们做了一个特别的实验,在相对稳定的工作流情况下,对两种机制产生的额外网络负载进行了对比.实验设计如下:总共 32 000 个客户端,每个客户端持续发送 8 个 1M 的 I/O 请求给服务器.图 8(a)显示了服务器端挂起的 RPC 请求数目的动态变化,其最大值达到 32 000 个,而且在整个实验过程中,几乎一直恒定于该值.图 8(b)分别显示了轮询策略和 EP 策略产生的网络流量.我们可以看出,在大部分时间里,客户端驱动的轮询机制产生的额外消息占用的网络带宽大约为 5MB/s×2=10MB/s;而使用 EP 策略,从 300s 后,它几乎不产生任何及早回复消息.根据以上的模拟结果,在拥有上百个服务器的大规模集群中,如果每个服务器都承受如此的工作负载,整个系统会有数目非常庞大的 RPC 请求被并行处理,额外的轮询消息将会造成非常可观的网络流量,占用的网络带宽甚至会达到几个 GB/s.这明显会伤害到性能.相对地,使用及早回复策略,对于持续稳定的 RPC 工作流,结合自适应超时策略,RPC 的超时值的上限可以被正确地评估,因此它很少产生额外的及早回复消息传递.



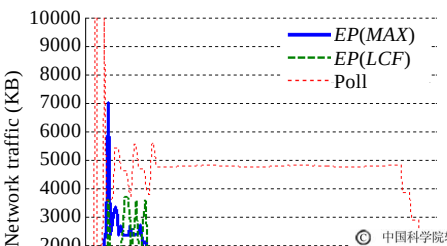


Fig.8 Comparison between the EP strategy and polling mechanism

图 8 及早回复策略与轮询机制比较

5 总 结

本文对基于 RPC 构建的分布式系统常用的失效检测方法超时机制进行了研究.为了解决出现在超大规模集群中的固定超时的问题,我们提出了 AST 机制,它同时考虑了网络条件、服务器负载、可扩展性和性能等因素.它包括两个策略:基于 STW 的自适应超时策略和及早回复策略.在自适应超时策略中,客户端设置的超时值可以根据客户端服务器间的网络情况以及服务器的工作负载动态地进行调整,以适应集群环境的变化,从而避免不必要的超时造成整个系统性能的降低;同时,为了区分服务器因负载过重拥塞和网络/节点失效,以及为了解决嵌入式超时问题,提出了一种及早回复策略:当服务器知道它不能在客户端期待的响应时间内回复 RPC 请求时,那么它将提前发送一个轻量级的及早回复消息给客户端,并指示一个估测的额外需要的服务时间.该策略进一步减少了超时的发生,提高了系统的响应速度.然后,我们在 Lustre 模拟器上做了一系列的模拟评估.结果表明:与固定超时机制相比,使用 AT 策略 RPC 超时率从 78%降低到 8%,结合 EP 策略,超时率甚至降低到 0%;在基于 RPC 的超大规模存储集群中,其他的一些 RPC 失效检测机制,如客户端驱动的轮询或探测机制,会产生大量的不必要的网络流量,存在扩展性问题.而我们的及早回复策略是一个更吸引人的策略,它通常只产生少量的网络流量.模拟评估结果表明,我们的 AST 是一种更适合的基于超时的 RPC 失效检测模型,它增强了系统的响应性、可靠性和稳定性,但对系统性能没有过大的负面影响.

致谢 该工作部分是由 SUN Lustre 开发组支持,在此,特别感谢他们的帮助和对本文给予的评论.感谢清华大学高性能研究所的舒继武教授和易乐天同学对本文的建议.

References:

[1] TOP 500 Supercomputers home page. http://www.top500.org

[2] Birman KP, Glade BB. Consistent failure reporting in reliable communication systems. Technical Report, TR93-1349, Ithaca: Cornell University, 1993.

[3] Panzieri F, Shrivastava SK. Rajdoot: A remote procedure call mechanism supporting orphan detection and killing. IEEE Trans. on Software Engineering, 1988,14(1):30−37. [doi: 10.1109/32.4620]

[4] Muller G, Volanschi EN, Marlet R. Scaling up partial evaluation for optimizing the Sun commercial RPC protocol. ACM SIGPLAN Notices, 1997,32(12):116−126. [doi: 10.1145/258994.259010]

[5] Bouteiller A, Desprez F. Fault tolerance management for a hierarchical GridRPC middleware. In: Proc. of the 8th IEEE Int'l Symp. on Cluster Computing and Grid (CCGRID 2008). Lyon: IEEE Press, 2008. 484−491. http://icl.cs.utk.edu/news_pub/ submissions/ bouteiller-FTgridRPC.pdf

[6] Welch BB. The sprite remote procedure call system. Technical Report, CSD-87-302, Berkeley: University of California at Berkeley, 1986.

[7] Tay BH, Ananda AL. A survey of remote procedure calls. ACM SIGOPS Operating Systems Review, 1990,24(3):68−79.

[8] Frances C, Kao IL, Lin CL. Adaptive timeout value setting for distributed computing environment (DCE) applications. United States Patent 6526433, 2003-02-25. http://www.freepatentsonline.com/6526433.html

[9] Khandker AM, Honeyman P, Teorey TJ. Performance of DCE RPC. In: Proc. of the 2nd Int'l Workshop on Services in Distributed and Networked Environments. Whistler: IEEE Computer Society, 1995.

[10] Delaney WP, Copas KW, Jantz RM, Lewis CW. Polling-Based mechanism for improved RPC timeout handling. United States Pattent 7146427, 2002-04-23. http://www.freepatentsonline.com/7146427.html

[11] Birrell AD, Nelson BJ. Implementing remote procedure calls. ACM Trans. on Computer Systems, 1984,2(1):39−59. [doi: 10.1145/2080.357392]

[12] Dineen TH, Leach PJ, Mishkin NW, Pato JN, Wyant GL. The network computing architecture and system: An environment for developing distributed applications. In: Proc. of the 33rd Int'l Conf. on IEEE Computer Society. 1988. 296−299.

[13] Schwan P. Lustre: Building a file system for 1 000-node clusters. In: Proc. of the Linux Symp. 2003. 380−386. http://www.kernel.org/doc/ols/2003/ols2003-pages-380-386.pdf

[14] Fahey M, Larkin J, Adams J. I/O performance on a massively parallel Cray XT3/XT4. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS 2008). Miami: IEEE Computer Society, 2008. 1−12.

[15] Nieuwejaar N, Kotz D, Purakayastha A, Ellis CS, Best ML. File-Access characteristics of parallel scientific workloads. IEEE Trans. on Parallel and Distributed Systems, 2005,7(10):1075−1089. [doi: 10.1109/71.539739]

[16] Lustre simulator. 2009. https://bugzilla.lustre.org/show_bug.cgi?id=13634

钱迎进(1981－),男,湖北大悟人,博士生,主要研究领域为计算机网络,存储系统.

金士尧(1937－),男,教授,博士生导师,主要研究领域为图形图像,系统仿真.

肖侬(1969－),男,博士,教授,博士生导师,CFF 高级会员,主要研究领域为网格计算和数据网格,高性能并行分布处理技术.