·云计算专题· 文章编号:1000-3428(2016)05-0001-07 文献标识码:A 中图分类号:TP302.8

志愿云环境下的拜占庭容错研究

雷长剑,林亚平,李晋国,赵江华

(湖南大学信息科学与工程学院,长沙 410082)

摘 要: 志愿云环境下的节点具有动态性高、可靠度低的特点,系统容易出现拜占庭错误。拜占庭一致性算法可使系统在出现 f 个恶意节点时保证一致性,但现有算法冗余度较高。针对该问题,提出一种基于 Gossip 协议的拜占庭容错算法,使系统冗余度降低到 $2f+1$。该算法无需设计主节点,所有计算节点处于对等地位,可避免主从模式冗余系统发生单点故障。理论分析和实验结果表明,所提算法能达到拜占庭容错要求,有效减小系统冗余度,与 BFTCloud 和 Zyzzyva 算法相比,提升了系统吞吐量。

关键词: 志愿云;云计算;拜占庭错误;一致性;容错;Gossip 协议

中文引用格式: 雷长剑,林亚平,李晋国,等. 志愿云环境下的拜占庭容错研究[J]. 计算机工程,2016,42(5):1-7.

英文引用格式: Lei Changjian, Lin Yaping, Li Jinguo, et al. Research on Byzantine Fault Tolerance Under Volunteer Cloud Environment[J]. Computer Engineering, 2016, 42(5):1-7.

Research on Byzantine Fault Tolerance Under Volunteer Cloud Environment

LEI Changjian, LIN Yaping, LI Jinguo, ZHAO Jianghua

(College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China)

【Abstract】 Since the nodes of the volunteer cloud have the characteristics of high dynamic and low reliability, the volunteer cloud is prone to Byzantine fault. Byzantine consensus algorithm can make the system keep consistent when f malicious nodes are present. However, the existing algorithms have high redundancy degree. Aiming at this problem, this paper proposes a Byzantine fault tolerance algorithm based on Gossip protocol, which reduces the system redundancy degree to $2f+1$. It does not need to set the master node and all the computing nodes in system are set to be in peer status, so as to avoid single point failure in master-slave redundancy system. Theoretical analysis and experimental results show that the proposed algorithm can not only satisfy the Byzantine tolerance requirement, but also reduce the system redundancy degree. Compared with BFTCloud and Zyzzyva algorithm, it improves the system throughput.

【Key words】 volunteer cloud; cloud computing; Byzantine fault; consistency; fault tolerance; Gossip protocol

DOI: 10.3969/j.issn.1000-3428.2016.05.001

1 概述

云计算[1]现已被广泛应用于 IT 行业,目前云服务一般按 2 种类型部署:(1)具有良好管理、可靠性较高的商业云[2](如 Amazon, IBM, Google, Microsoft 等提供的商业云);(2)由许多分布在不同地理位置上的用户志愿提供的计算资源,即志愿云[3]。志愿云结合了志愿计算[4]和云计算 2 个概念,是利用闲置计算资源组成的具有计算能力的云,又被称为分布式云[3]或非专用云[5]。

志愿云具有以下优点:(1)计算资源都是志愿奉献或只收取少量成本,在使用成本上有更大的优势[3];(2)商业云需在其数据中心设置大量的专用资源,并且需要消耗大量的电力资源[6],管理成本和资源消耗较大,而志愿云的节点都是闲置的非专用资源,可以高效利用资源,减少能耗开销;(3)分布在世界各地的计算资源能提供强大的计算能力。上述优点都显示出志愿云广阔的应用前景。然而,志愿云环境下的节点具有较高动态性和不可靠性,由于多种原因(如操作失误、软件错误、系统崩溃、恶意攻击等)均会引起系统错误,因此这些错误被称为拜占庭错误[7]。

基金项目: 国家自然科学基金资助项目"面向云存储的多元数据安全查询机制和算法研究"(61472125)。

作者简介: 雷长剑(1986-),男,硕士研究生,主研方向为云计算、拜占庭系统;林亚平,教授;李晋国,博士;赵江华,硕士研究生。

收稿日期: 2015-05-07 **修回日期:** 2015-06-09 **E-mail:** cj12@hnu.edu.cn

随着云计算的发展,其安全性和可靠性备受学者的关注[8],尤其是在不可靠云环境中可能出现的拜占庭错误。关于拜占庭问题可参考文献[7]的原型描述,本文略去说明。拜占庭节点可以合谋欺骗正确节点从而破坏系统一致性,因此,需要设计一种志愿云环境下的拜占庭一致性算法以保证系统的正常运行。基于此,本文提出一种志愿资源云环境下基于 Gossip 协议的拜占庭一致性协同算法(Gossip protocol-based Byzantine consistency Agreement algortihm,GBA)。

2 相关研究

现有多数拜占庭算法都是基于冗余的思想设计的,在这些算法中,容忍 f 个节点错误至少需要 $n \geqslant 3f+1$ 个节点。传统拜占庭系统一般包含 3 个协议:一致性协议,检查点协议以及视图更换协议。系统正常时运行一致性协议和检查点协议,系统出错时运行视图更换协议。文献[9]提出拜占庭一致性问题,并使用递归口信(Oral Message,OM)算法解决该问题。OM 算法采用多轮通信,总消息数为指数级:$O(n^{f+2})$。文献[10]使用早停技术和错误掩盖技术将指数级算法改善到多项式级,总消息数为 $O(n^5)$。文献[11]提出了 PBFT 算法,系统中每个客户端请求需要 5 个阶段(Request, Pre-prepare, Prepare,Commit,Reply)才能完成,主节点出错更换视图开销较大。PBFT 需要 $3f+1$ 个节点容忍 f 个副本的错误。文献[12]提出了 Zyzzyva 算法,该算法采用投机机制,由于服务器大部分时间处于正常状态,因此不用每个请求都在达成一致性之后再执行,只需要在错误发生后再达成一致即可,相比 PBFT 开销有所改进,但还是存在冗余度较大的问题。文献[13]在志愿资源云里提出拜占庭错误容忍的解决方案 BFTCloud,该方案也是基于主节点和副本模式,在客户端和计算资源之间搭建了一个中间件模块,负责将请求序列发送给主节点并接收主节点返回的结果。在请求执行阶段结束后,如果中间件模块裁定结果不一致,则会触发主节点更新和副本更新。BFTCloud 的思想类似于 Zyzzyva,即先执行再检验一致性。但两者需求的容错节点数均为 $3f+1$,系统冗余程度大,且系统均需配置一个主节点,一旦主节点出错,会引发视图更换和全局成员节点的视图变化,很大程度上影响系统性能,即给整个系统造成了单点故障[14]。

在拜占庭系统应用方面也有相关研究。文献[15]提出了一种基于拜占庭协议的入侵容忍模型,该模型能保证系统在黑客攻击下仍能正确运行。文献[16]在无线 Mesh 网络中(Wireless Mesh Network,WMN)引入拜占庭单元的概念,提出了一种拜占庭算法以改进现有 WMN 路由协议,使得协议能对异常节点信息进行容错处理。

鉴于志愿云广阔应用前景和现有工作的不足,本文提出一种志愿资源云环境下基于 Gossip 协议的拜占庭一致性协同算法(GBA)。本文的工作总结如下:

(1)减少了系统冗余程度。当节点满足 $n \geqslant 2f+1$ 时能满足拜占庭错误容忍条件。

(2)由于 Gossip 协议中的节点都是对等关系,这与之前状态机拜占庭系统需设置一个主节点相比,不会产生因为主节点故障带来的性能瓶颈。

3 系统模型和问题描述

如图 1 所示,一个云应用通常按照一定的秩序分成若干小任务,每个小任务由一组分布于不同空间位置上的志愿节点完成,节点之间通过链路连接。拜占庭错误可能出现于任意节点,错误节点能够通过合谋等方式欺骗正确节点,造成系统的不一致。为了预防消息被伪造,在节点将消息发送前,用基于对称密钥的消息认证码(Message Authentication Code,MAC)对节点间的通信信息认证。利用 MAC 认证技术,保证了消息的不可欺骗性,即恶意节点不能伪造或篡改正确节点发送的消息。

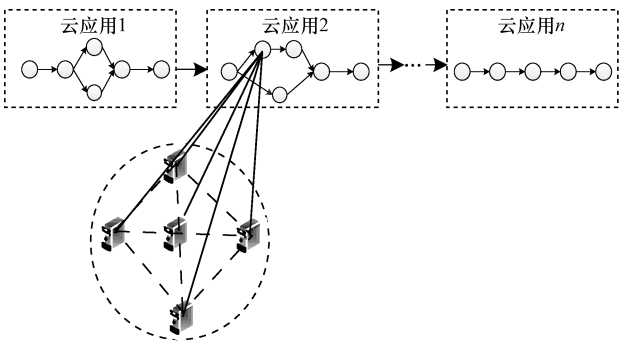

图 1 云应用模型

图 2 描述了志愿云系统的架构,客户通过一个服务终端接口提交请求,节点选择算法从资源池里选取一些节点组成虚拟群组,为客户提供服务,所选择的节点之间通过底层链路连接。由于无法避免组成虚拟群组的计算节点存在拜占庭节点,而拜占庭节点会发送错误信息,从而破坏系统一致性,因此本文就此问题设计了 GBA 算法。

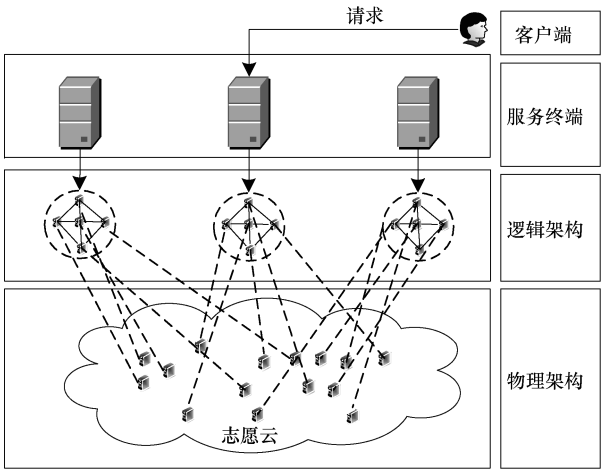

图 2　志愿云系统架构

4　算法说明

4.1　节点选择算法

在志愿云环境下,计算节点呈现多样化,节点的加入和离开呈现高动态性,节点的可靠性低,且无有效办法阻止恶意提供方提供计算机资源,在这样的环境下拜占庭错误可能非常普遍。因此,选择最可靠的节点参与运算能更好地提高系统性能。为了选出综合性能较好的节点,本文按如下算法选择节点:

节点选择的依据是其服务质量(Quality of Service,QoS)指标(如计算性能、网络延时、在线时间等。采取何种指标,依据实际需求来综合制定),通过 QoS 指标的综合排名,选择最优的节点参与运算。设定 $N=\{n_1,n_2,\cdots,n_i\}$ 为可分配使用节点的集合,$\boldsymbol{Q}$ 为各个节点对应 QoS 值的二维向量,如式(1)所示。

$$\boldsymbol{Q}=\begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & \vdots & & \vdots \\ q_{i1} & q_{i2} & \cdots & q_{im} \end{bmatrix} \tag{1}$$

$\boldsymbol{Q}$ 中第 i 行向量$[q_{i1},q_{i2},\cdots,q_{im}]$表示节点 n_i 对应的 m 个 QoS 指标的取值,其取值根据节点的实际性能取整数(如一个闭区间内的实数$[-10,10]$)。为了计算方便,将 QoS 值通过式(2)映射到区间$[0,1]$上。其中,x 表示节点对应标准的 QoS 取值;$q_{\min}$ 和 $q_{\max}$ 分别代表该标准的最小值和最大值。

$$q_{ik}=(x-q_{\min})/(q_{\max}-q_{\min}) \tag{2}$$

设定一个权值集合 $\boldsymbol{w}=\{w_1,w_2,\cdots,w_m\}$($w_j\in\mathbb{R}^+$),该向量分别对应 QoS 的 m 个指标的权值,且 $\sum_{k=1}^{m}w_j=1$。依据以上描述,对于第 i 个节点的加权评级值 r_i 可以表述为:

$$r_i=\sum_{j=1}^{m}(q_{ij}\times w_j) \tag{3}$$

显然,r_i 的取值范围也是$[0,1]$。据式(3)计算得出各个节点的评级值 r_i,将其从大到小排序,按需选取靠前的 k 个节点,用向量 $\boldsymbol{Top}[k-1]$表示。选举前 k 个节点的算法步骤如下:

算法 1　加权评级排名前 k 个节点选择算法

步骤 1　对 N 中的每个节点 n_i 分别按式(3)计算其加权评级值。

步骤 2　将步骤 1 中得到的各节点的评级值按从大到小排序,得到节点加权评级值的排序数组$\boldsymbol{R}[i-1]$。

步骤 3　将 $\boldsymbol{R}[i-1]$中前 k 个元素赋值给数组 $\boldsymbol{Top}[k-1]$,返回 $\boldsymbol{Top}[k-1]$数组,便得到前 k 个节点。

4.2　基于 Gossip 协议的拜占庭一致性算法

Gossip 协议是模仿社交网络流言传播得到的一种通信方法,协议思想大致为:节点 P 的数据更新后,它会将此更新推送给它的邻居节点,如果节点 P 在一段时间内没有数据更新,则会收到其他节点推送的更新信息,从而被动更新,经过必要的交换轮次后,最终使得所有节点维持信息一致性。该协议主要用于解决分布式系统的一致性问题。在协议中,每个节点具有一个保存少数成员信息的视图,称作 localview。在每个通信周期里,节点从其 localview 中选择 *fan*(fanout 参数)个通信目标,采取 3 种通信模式之一与目标节点通信:

(1)push 模式:节点 A 选择节点 B 作为通信目标,然后将本地数据推送给节点 B,节点 B 根据收到的信息更新本地数据。

(2)pull 模式:节点 A 选择节点 B 作为通信目标,B 将其数据发送给 A,A 根据收到的信息更新本地数据。

(3)push-pull 模式:此模式结合了 push 与 pull 模式。

文献[17]证明了 Gossip 协议能保证消息高概率的传播,该协议的正确性证明可参考文献[18-19]。最简单的一种 Gossip 协议工作模型是 SI 模型(Susceptible,Infected),其逻辑增长函数如式(4)所示,其中,c 代表通信轮次;i_0 是 $c=0$ 时 $I(c)$的取值;*fan* 代表每个周期选择的目标个数。

$$I(c)=\frac{i_0 e^{fanc}}{1-i_0+i_0 e^{fanc}} \tag{4}$$

GBA 算法中设定了 3 种数据结构:*LD*,***DV*** 和 *FD*。*LD* 是各节点计算得到的值;***DV*** 是包含 n 个节点 LD 值的一维向量;*FD* 是各节点根据其 ***DV*** 向量,按照评估算法评估出来的大多数值(即正确值)。在上述 3 种通信模式中,无论采用哪一种模式通信,参与通信双方的节点,有一方是主动更新数据,称为主

动进程;另外一方是被动更新数据,称为被动进程。主动方和被动方的主函数算法分别如算法 2 和算法 3所示。

算法 2 主动方主函数算法

步骤 1 节点计算出自身的 *LD* 值,并将其赋值给 ***DV*** 数组中对应元素。

步骤 2 从 localview 中选出 1 个目标节点,将步骤 1 中得到的 ***DV*** 推送给目标节点(若节点工作在 push 模式下)。

步骤 3 从目标节点处收到其 ***DV***,然后调用 updateInfo 函数更新自身的 ***DV***(若节点工作在 pull 模式下)。

步骤 4 重复步骤 2 或步骤 3 共 *fan* 次。

步骤 5 调用 checkConsensus 函数。

算法 3 被动方主函数算法

步骤 1 节点计算出自身的 *LD* 值,并将其赋值给 ***DV*** 数组中对应元素。

步骤 2 节点监听自身是否被选为目标节点。一旦自身被别的节点选中,则将收到对方推送过来的 ***DV***,然后调用 updateInfo 函数更新自身的 ***DV***(若节点工作在 push 模式下)。

步骤 3 节点将自身的 ***DV*** 发送给对方(若节点工作在 pull 模式下)。

步骤 4 调用 checkConsensus 函数。

步骤 5 节点持续监听自身是否被其他节点选为目标节点。

updateInfo 函数和 checkConsensus 函数算法分别如算法 4 和算法 5 所示。

算法 4 updateInfo 函数算法

步骤 1 若 $a[i]\neq b[i]$ && $b[i]\neq$ null,置 $a[i]=b[i]$,从而 a 中元素得到更新。

步骤 2 若 $a[i]\neq b[i]$ && $a[i]\neq$ null,置 $b[i]=a[i]$,从而 b 中元素得到更新。

步骤 3 否则不做任何操作。

算法 5 chechConsensus 函数算法

步骤 1 节点判断出其 ***DV*** 向量中超过半数的数组元素值,则该值即为 *FD* 值。

步骤 2 若没有超过半数的元素一致,则 *FD* 值为空,说明系统不满足 $n\geqslant 2f+1$ 的条件,无法判断正确值和识别错误节点。

4.3 算法实例

为了说明本文算法的工作原理,列举一个实例对算法进行说明补充,其中系统设置如表 1 设置:系统中有 5 个节点(A,B,C,D,E),每个节点的 localview 包含 2 个邻居节点,A,C,D 为正确节点,B,E 为拜占庭节点,正确节点对应的 ***DV*** 为 a,节点 B 对应的 ***DV*** 值为 b,节点 E 对应的 ***DV*** 值为 c。节点关系符合 $n\geqslant 2f+1$,节点通信使用 push-pull 模式,算法共执行 2 轮,设定 $fan=1$,算法执行结果如表 2 所示。

表 1 算法实例系统设置

节点	localview	*LD*	是否为拜占庭节点
A	{nodeC,nodeD}	a	否
B	{nodeA,nodeE}	b	是
C	{nodeB,nodeE}	a	否
D	{nodeB,nodeC}	a	否
E	{nodeA,nodeD}	c	是

表 2 算法实例执行结果

行号	通信节点	初始的 ***DV***	协商后的 ***DV***
1	A	[a,null,null,null,null]	[a,null,a,null,null]
	C	[null,null,a,null,null]	
2	B	[null,c,null,null,null]	[null,c,null,null,d]
	E	[null,null,null,null,d]	
3	C	[a,null,a,null,null]	[a,e,a,null,d]
	B	[null,e,null,null,d]	
4	D	[null,null,null,a,null]	[a,e,a,a,d]
	C	[a,e,a,null,d]	
5	E	[null,c,null,null,f]	[a,c,a,null,f]
	A	[a,null,a,null,null]	
6	A	[a,c,a,null,f]	[a,c & e,a,a,d&f]
	D	[a,e,a,a,d]	
7	B	[a,g,a,null,d]	[a,h,a,a,d&f]
	A	[a,c&e,a,a,d&f]	[a,c&e&g,a,a,d&f]
8	C	[a,e,a,a,d]	[a,c&e,a,a,d&h]
	E	[a,c,a,null,h]	[a,c&e,a,a,i]
9	D	[a,c&e,a,a,d&f]	[a,c&e&j,a,a,d&f]
	B	[a,j,a,a,d&f]	[a,k,a,a,d&f]
10	E	[a,c&e,a,a,l]	[a,c&e&j,a,a,m]
	D	[a,c&e&j,a,a,d&f]	[a,c&e&j,a,a,d&f&l]

在表 2 中,第 1 行 ~ 第 5 行为第 1 轮通信,第 6 行 ~ 第 10 行为第 2 轮通信。第 1 行表示节点 A($\boldsymbol{DV}_{\mathrm{A}}$=[$a$,null,null,null,null])从其 localview 中选取节点 C($\boldsymbol{DV}_{\mathrm{C}}$=[null,null,$a$,null,null])为目标节点,2 个节点互相向对方推送自身的 ***DV***。然后更新自身的 ***DV*** 信息,得到一致的结果,均为[a,null,a,null,null]。注意到第 2 行中节点 B 发送的 ***DV*** 向量中,对应节点 B 得到的 *LD* 值为 c,这是因为节点 B 是恶意节点,它发送的值可能是任意的。同时,由于各节点发送的信息均由 MAC 认证,信息不可被伪造或篡改,因此错误节点不能伪造 ***DV*** 中其他节点对应的值。

第 2 行~第 8 行的执行分析类似。观察结果可知,在第 4 行表示的操作结束时,节点 D 和节点 C 可以判定节点 B 和节点 E 可能出错。在第 6 行表示的操作结束时,节点 A 和节点 D 可以判定节点 B 和节点 E 可能出错。在第 7 行表示的操作结束时,节点 A 可以判定节点 B 和节点 E 可能出错。在第 8 行表示的操作结束时,节点 C 可以判定节点 B 和节点 E 可能出错。在第 9 行和第 10 行表示的操作结束时,节点 D 均可判定节点 B 和节点 E 可能出错。该实例表明,在经过 2 轮通信后,所有节点均能达成一致,可对正确值做出判断,并识别出错误节点,从而实现拜占庭容错的目的。

5　算法分析和证明

要证明算法 2 和算法 3 能保证所有正确的节点都能得到相同的决定值,在本文所述的条件下,这个证明可以等价于 $n \geq 2f+1$ 情况下的交互一致问题[9],因此,需要满足如下条件:

(1)有效性:如果节点 n_i 是正确的,那么所有正确节点的 $\boldsymbol{DV}[i-1]$ 向量中对应 n_i 节点位置的元素都是节点 n_i 所得到的 LD 值。

(2)有穷性:每个正确的节点最终都能根据其 $\boldsymbol{DV}$ 得到一个最终值,即 FD。

(3)一致性:所有正确节点的 FD 是一致的。

定理 1　GBA 算法满足有效性要求。

证明:利用反证法证明定理的正确性。假设此定理错误,在一个共有 k 个正确节点的系统中,设 n_i 和 n_j 是正确节点,n_i 选择 n_j 为通信目标。n_i 经过计算得到 LD_{n_i},将其赋给 $\boldsymbol{DV}_{n_i}$ 中对应的元素,且 LD_{n_i} 中加入了签名信息,其他节点无法篡改或伪造。又因为 $\boldsymbol{DV}_{n_j}$ 是根据 $\boldsymbol{DV}_{n_i}$ 更新的,所以 $\boldsymbol{DV}_{n_j}$ 中包含了 n_i 的 LD 值,且这个值必定是 n_i 计算所得,n_j 不可篡改。如上文所述,在 Gossip 协议中消息能高概率传播,因此,LD_{n_i} 将很快传播到系统中的其他节点,并且所有正确的节点都将更新其 $\boldsymbol{DV}$ 向量中节点 n_i 对应的元素,这与假设矛盾,定理得证。

定理 2　GBA 算法满足有穷性要求。

证明:

(1)当系统只有 1 个正确节点时,该节点将其得出的 LD 值赋值给其 $\boldsymbol{DV}$ 向量(此时 $\boldsymbol{DV}$ 中元素个数为 1),显然该节点能得出 FD 值,算法满足有穷性。

(2)当系统中有 k 个正确的节点,且系统满足 $n \geq 2f+1$ 时,设 n_i 和 n_j 是正确节点,根据定理 1,如果 n_i 计算出了其 LD,那么其他正确节点的 $\boldsymbol{DV}[i-1]$ 都是节点 n_i 所得到的 LD 值。同理,其他正确节点的 $\boldsymbol{DV}[j-1]$ 都是节点 N_j 所得到的 LD 值。最终,k 个正确节点都能确定各自 $\boldsymbol{DV}$ 中 k 个元素的值。因为 $k \geq f+1$,所以 k 个正确节点都将通过评估算法得到其各自的 FD 值。因此,当系统中有 k 个正确节点时,算法满足有穷性。

(3)当系统中有 $k+1$ 个正确的节点,且系统满足 $n \geq 2f+1$ 时,设 n_{k+1} 是系统中第 $k+1$ 个正确节点,且它选择了节点 n_i 作为通信目标。依照通信模式分 2 种情况讨论:1)若节点 n_{k+1} 工作在 push 模式下,根据有穷性,其余 k 个正确节点均已确定了各自 $\boldsymbol{DV}$ 的 k 个元素,则节点 n_i 能够确定其 $\boldsymbol{DV}$ 中 $k+1$ 个元素的值,并且 n_i 能够根据评估算法计算出其 FD。随着 Gossip 通信轮次的增加,这含有 $k+1$ 个正确元素的 $\boldsymbol{DV}$ 向量将被传送到其他节点处。2)若节点 n_{k+1} 工作在 pull 模式下,则节点 n_{k+1} 能决定自身 $\boldsymbol{DV}$ 中的 $k+1$ 个元素,并能够根据评估算法计算出其 FD。同样,随着 Gossip 通信轮次的增加,含有 $k+1$ 个确定元素的 $\boldsymbol{DV}$ 向量将传送到其他节点处。那么所有正确节点都能根据其 $\boldsymbol{DV}$ 得到 FD 值,因此,当系统中有 n_{k+1} 个正确节点时,算法满足有穷性。

定理 3　GBA 算法满足一致性要求。

证明:采取反证法证明。假设所有正确节点的 FD 是不一致的。设系统共有 k 个正确节点,n_i 和 n_j 为正确节点。对于 n_i 而言,它得到 LD 后,赋值给 $\boldsymbol{DV}$,然后与其他节点进行数据更新同步,当达到需要的交换轮次后,就可以通过评估函数计算出 $\boldsymbol{DV}$ 的大多数值。又因为系统最少有超过一半的节点是正确节点(即 $k \geq n/2+1$),所以节点 n_i 计算出来的大多数值即是系统的正确值。对于 n_j 节点,同理也会得到一个 $\boldsymbol{DV}$ 的大多数值,该值也是系统的正确值。因此,n_i 和 n_j 的 FD 是一致的。同理,对所有 k 个正确节点,在遵守算法 2 和算法 3 的前提下得到的 FD 也将会是一致的。

定理 4　GBA 算法允许系统中最多存在 $\lfloor n/2 \rfloor$ 个 $(n\%2=1)$ 拜占庭节点或 $\lfloor n/2 \rfloor - 1$ 个 $(n\%2=0)$ 拜占庭节点。

证明:本文算法要求系统中节点数满足 $n \geq 2f+1$,其中,n 表示所有节点数总和,f 是拜占庭节点。此定理表示当正确节点的个数超过错误节点个数时,本文算法能保证系统正确运行。根据算法 2 描述,每个正确的节点都会定期与其他节点交换信息,并得出一致的元素个数(假设为 k 个)。如果 k 超过节点数目的一半即可得出正确有效的 $\boldsymbol{DV}$ 向量。由于 Gossip 协议能保证消息高概率的交换,在达到必须的 c 次交换后,所有正确节点最终都能根据 $\boldsymbol{DV}$ 向量决定其 FD 值。因此可以认为,系统中允许存在最大的拜占庭节点个数取决于系统的总节点数 n。进一步看来,因为要保证正确节点数大于拜占庭节点的个数,即当 n 为奇数时,最大的拜占庭节点数

是⌊n/2⌋;当 n 为偶数时,最大的拜占庭节点数是⌊n/2⌋-1。

6 实验与结果分析

6.1 实验平台与参数配置

本文实验原型系统运行在 4 台服务器上,其中 3 台运行服务器程序,其配置 CPU 为 8 核 Intel i7-3770 3.4 GHz,内存 16 GB。另外 1 台运行客户端程序,其配置 CPU 为 4 核 2.50 GHz Intel i5-2520M,内存 16 GB。操作系统内核均为 Linux 2.6.18,算法是基于 Java 语言实现。Gossip 协议的性能与 fanout 和 localview 的参数配置有直接关系,为了突出评测的重点,在实验中统一设置为 2 和 5。笔者选取 3 个方面进行实验评估:(1)比较不同工作模式对本文算法性能的影响;(2)比较优化节点选择方案后对本文算法性能的影响;(3)在同等的实验条件下,将本文算法与经典的 Zyzzyva 算法和 BFTCloud 算法进行吞吐量比较。

6.2 实验结果

图 3 显示了节点运行在不同的通信模式时系统吞吐量的比较结果,从中可以看出,push 模式下的吞吐量优于 pull 模式,而 push-pull 模式下的吞吐量是 3 种模式中最好的,其原因分析如下:在 push 模式下,节点会主动与其邻居节点交换信息,而在 pull 模式下,节点会等待一个设定的时间,在没有收到更新时才会与其邻居节点通信以更新信息,因此,pull 模式下的收敛速度小于 push 模式。而 push-pull 模式结合了其余 2 种模式的优点,所有节点均会主动寻找通信目标,其收敛速度是 3 种模式中最快的。

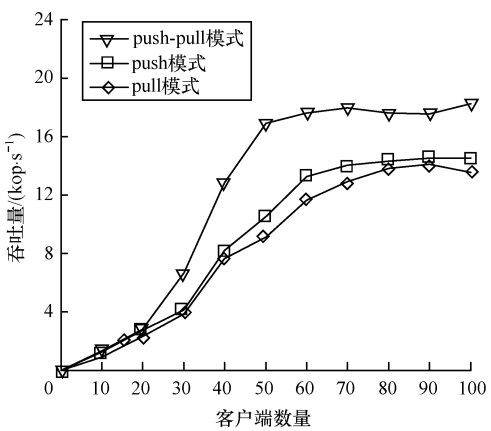

图 3 不同通信模式下的系统吞吐量比较

图 4 显示了本文算法工作在 push-pull 通信模式下,采取不同策略选择计算节点对算法性能的影响,其中 GBA-I 表示使用节点选择算法选择节点时的系统吞吐量曲线,GBA-II 表示随机选择节点时的系统吞吐量曲线。实验结果显示 I 型明显优于II 型,且 II 型曲线呈现随机的波动,其原因分析如下:(1)在高动态的志愿云环境下,节点的可靠度是影响系统性能的关键因素,使用正确的节点选择算法能使系统保持较低的故障率,从而提高吞吐量。(2)未使用节点选择算法时,新增节点连续出错会引起频繁的节点更换,带来较大的开销,影响系统性能,因此,系统吞吐量呈现随机波动。

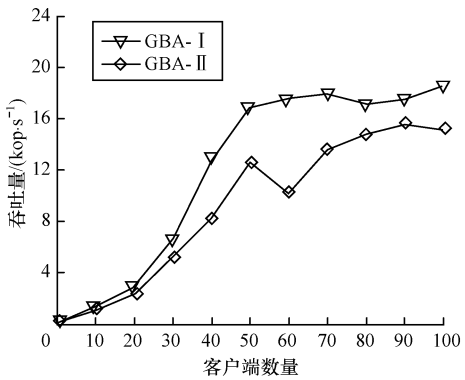

图 4 节点选择优化对系统吞吐量的影响

图 5 显示了本文算法工作在 push-pull 模式下,且采用优化节点选择方案时,其与 BFTCloud 和 Zyzzyva 算法的比较结果,从中可以看出,本文算法的吞吐量明显优于 BFTCloud 和 Zyzzyva 算法,其原因分析如下:(1)本文算法基于 Gossip 协议,从协议的工作模式和算法描述可知算法能保证高效率的消息交付[17-19],体现了算法的先进性;(2)因为 Zyzzyva 和 BFTCloud 算法均工作在一个主节点和若干个从节点的模式下,主节点一旦出错,引起的节点更换开销较大。而本文算法所有节点均处于平等位置,不存在主节点带来的单点故障现象,副本更换开销较小;(3)本文算法在选择节点时综合考虑了节点的工作性能,综合优先选择最优化节点,相比另外 2 个算法,能保证系统相对较低的出错概率。

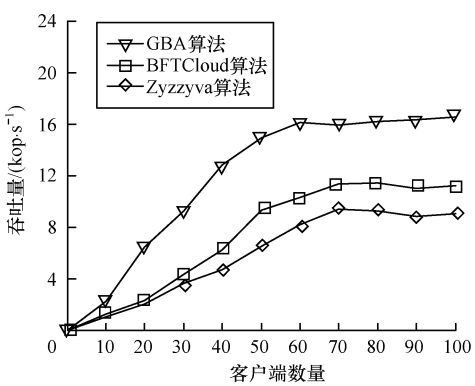

图 5 不同算法的系统吞吐量比较

7　结束语

志愿云相比商业云具有独特的优势,发展前景广阔。随着志愿云计算的广泛应用,系统规模和设计复杂度不断增加,系统可靠性问题日益严重。为减少志愿云环境下的拜占庭错误,本文提出一种基于 Gossip 协议的拜占庭一致性算法,其优势主要体现在以下 3 个方面:(1)系统冗余度为 $n \geq 2f + 1$;(2)所有节点之间为对等关系,不会引起单点故障问题;(3)与 BFTCloud 和 Zyzzyva 算法相比,系统吞吐量更高。

在本文算法设计中,fanout 和 localview 参数都是固定的,参数设定直接影响算法性能,因此,如何调整这两个参数达到更优化的性能是下一步的研究方向。此外,平等节点带来的控制开销比主从模式的开销大,如何在主从节点模式下达成志愿云的拜占庭容错也需要做进一步研究。

参考文献

[1] Armbrust M, Fox A, Griffith R, et al. A View of Cloud Computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[2] Tang Longji, Dong Jing, Zhao Yajing, et al. Enterprise Cloud Service Architecture[C]//Proceedings of the 3rd IEEE International Conference on Cloud Computing. Washington D. C., USA: IEEE Press, 2010: 27-34.

[3] Chandra A, Weissman J. Nebulas: Using Distributed Voluntary Resources to Build Clouds[C]//Proceedings of Conference on Hot Topics in Cloud Computing. [S. l.]: USENIX Association, 2009.

[4] Anderson D P, Fedak G. The Computational and Storage Potential of Volunteer Computing[C]//Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid. Washington D. C., USA: IEEE Press, 2006: 73-80.

[5] Andrzejak A, Kondo D, Anderson D P. Exploiting Non-dedicated Resources for Cloud Computing[C]//Proceedings of 2010 IEEE Network Operations and Management Symposium. Washington D. C., USA: IEEE Press, 2010: 341-348.

[6] Gupta A, Awasthi L K. Peer Enterprises: A Viable Alternative to Cloud Computing[C]//Proceedings of 2009 IEEE International Conference on Internet Multimedia Services Architecture and Applications. Washington D. C., USA: IEEE Press, 2009: 1-6.

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

[8] 冯登国,张　敏,张　妍,等. 云计算安全研究[J]. 软件学报,2011,22(1):71-83.

[9] Pease M, Shostak R, Lamport L. Reaching Agreement in the Presence of Faults[J]. Journal of the Association for Computing Machinery, 1980, 27(2): 228-234.

[10] Berman P, Garay J A, Perry K J. Towards Optimal Distributed Consensus[C]//Proceedings of the 30th Annual Symposium on Foundations of Computer Science. Washington D. C., USA: IEEE Press, 1989: 410-415.

[11] Castro M, Liskov B. Practical Byzantine Fault-tolerance and Proactive Recovery[J]. ACM Transactions on Computer Systems, 2002, 20(4): 398-461.

[12] Kotla R, Alvisi L, Dahlin M, et al. Zyzzyva: Speculative Byzantine Fault Tolerance[C]//Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles. New York, USA: ACM Press, 2007: 45-58.

[13] Zhang Yilei, Zheng Zibin, Lyu M R. BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-resource Cloud Computing[C]//Proceedings of IEEE International Conference on Cloud Computing. Washington D. C., USA: IEEE Press, 2011: 444-451.

[14] Amir Y, Coan B, Kirsch J. Prime: Byzantine Replication Under Attack[J]. IEEE Transactions on Dependable and Secure Computing, 2011, 8(4): 564-577.

[15] 邹立新,丁建立. 基于拜占庭协议的入侵容忍系统模型设计[J]. 计算机工程,2006,32(7):88-90.

[16] 王吉喆,赵蕴龙,吴　静. WMN 中拜占庭容错网络结构及算法[J]. 计算机工程,2011,37(20):83-86.

[17] Ganesh A J, Kermarrec A M, Massoulie L. Peer-to-peer Membership Management for Gossip-based Protocols[J]. IEEE Transactions on Computers, 2003, 52(2): 139-149.

[18] Allavena A, Demers A, Hopcroft J E. Correctness of a Gossip Based Membership Protocol[C]//Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing. New York, USA: ACM Press, 2005: 292-301.

[19] Gurevich M, Keidar I. Correctness of Gossip-based Membership Under Message Loss[J]. SIAM Journal on Computing, 2009, 39(8): 3830-3859.

编辑　金胡考