ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.2114−2126
DOI: 10.3724/SP.J.1001.2008.02114

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

一种基于关键属性的优化数据一致性维护方法*

周 婧+, 王意洁, 李思昆

(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)

An Optimistic Data Consistency Maintenance Method Based on Key-Attributes

ZHOU Jing+, WANG Yi-Jie, LI Si-Kun

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

+ Corresponding author: E-mail: jingle77@126.com

Zhou J, Wang YJ, Li SK. An optimistic data consistency maintenance method based on key-attributes. *Journal of Software*, 2008,19(8):2114−2126. http://www.jos.org.cn/1000-9825/19/2114.htm

Abstract: The features of simple description, small updates item and weak dependence are the main characteristics of updates of key-attributes in P2P systems. Accordingly, an optimistic data consistency maintenance method based on key-attributes is proposed. In the method, the update of key-attributes is separated from user update requests. Key-Updates are propagated by latency-overlay update propagation model, that is, updates are always propagated to the nodes having maximum or minimum latency, and assured and uncertain propagation paths of updates are all taken into account. Based on classifying key-update conflicts, a double-level reconciling mechanism including buffer preprocessing and update-log processing is applied to detect and reconcile conflicts, and then conflicts are solved by policies as last-writer-win and divide-and-rule. Lastly, update-log management method and maintenance method brought by node failure and network partitioning are discussed for the above is deployed based on the information storied in update-log. Delaying key-attributes updates cannot occur by the optimistic disposal method, and then it cannot depress efficiency of resource location based on key-attributes, which adapts well to P2P systems for Internet. The simulation results show that it is an effective optimistic data consistency maintenance method, achieving good consistency overhead, resource location and resource access overhead, and having strong robustness.

Key words: peer-to-peer distributed storage system; data replication; data consistency; resource location

摘 要: 针对关键属性更新的易描述、更新项较小和弱相关性三个特点,提出一种基于关键属性的优化数据一致性维护方法.在该方法中,首先分离出用户提交的更新请求中关于关键属性的更新;然后采用基于延迟-覆盖的更新

* Supported by the Supported by the National Natural Science Foundation of China under Grant Nos.60503042, 69903011 (国家自然科学基金); the Science Foundation for the Innovation Research Colony of the National Natural Science Foundation of China under Grant No.60621003 (国家自然科学基金创新研究群体科学基金); the National Basic Research Program of China under Grant No.2002CB312105 (国家重点基础研究发展计划(973)); the Foundation for the Author of National Excellent Doctoral Dissertation of China No.200141 (高等学校全国优秀博士学位论文作者专项资金项目)

Received 2006-05-26; Accepted 2007-04-03

传播模型进行更新传播,即基于副本间的网络延迟选择具有最大和最小网络延迟的结点转发更新,并在传播过程中记录和综合考虑更新的确定传播和不确定传播两条路径;在对关键更新冲突分类的基础上,采用更新缓冲区和更新日志两层更新协商机制并结合最新写胜出和分而治之规则,优化关键更新冲突的发现和解决;更新日志信息是方法中各种策略开展的基础,讨论了更新日志中信息管理方法以及结点失效和网络划分发生时信息的维护方法.关键属性更新的优化处理使得不会产生因为关键属性更新的延迟而降低系统基于关键属性的资源定位效率,满足面向Internet 的 P2P 系统的要求.模拟测试结果表明,该方法在一致性维护开销、资源定位开销与资源访问开销以及鲁棒性方面均具有较好的性能.

关键词: P2P 分布存储系统;数据复制;数据一致性;资源定位

中图法分类号: TP311 **文献标识码:** A

利用 P2P 计算技术构建大规模分布存储系统,是当前 P2P 计算研究和应用的热点[1−4].P2P 分布存储系统采用新的 P2P 结构,通过分布在 Internet 上的大量结点的协作,使得分布存储系统的可扩展性和自组织性大为增强;适应 Internet 的动态环境,利用分布在 Internet 上巨大的存储资源为用户提供数据共享和存储服务,支持海量用户和海量数据的存储需求.

数据复制[5−7]是改善分布式系统性能的重要技术.为了提高系统的可用性和系统性能,P2P 分布存储系统在多个结点上复制数据对象.虽然数据复制技术可以避免服务器单点失效、减少访问响应时间、减少通信开销和 I/O 开销以及均衡负载,但同时不可避免地面临副本一致性维护的问题.在弱一致性系统中,同一数据对象的多个更新可能在不同副本上同时发布,然后按照任意次序传播到其他副本上,复制算法必须确保每个副本都可以接收到所有副本发布的全部更新,并依据一定的规则对这些更新进行排序和应用,从而实现副本的最终一致.

关于 P2P 领域中数据一致性维护的研究不再拘泥于传统的文件更新的一致性维护,其研究领域涉及到 P2P 系统的多个方面.部分 P2P 系统对数据一致性的研究重点放在对传统的数据一致性维护方法进行改进[8−10],使之适用于 P2P 环境的特点上;部分 P2P 系统将其研究重心放在数据定位信息的一致性维护[11,12],从而更加突出 P2P 环境数据资源有效共享的特性;有的 P2P 系统对底层的网络拓扑结构进行一致性维护[13];针对 P2P 系统的动态特性,部分系统侧重于提高更新传播速度[8,14]或在一定副本组织方法的基础上如何进行更新传播[14,15]等.

由于 P2P 系统通常具有规模巨大、分布性强、动态性强等特点,可能导致部分更新被长时间地延迟.如果被延迟的更新是关于数据对象关键性描述信息的修改,而这些描述信息在用户检索数据时通常作为搜索条件被使用,则更新的延迟将降低资源搜索的性能.先前的 P2P 系统中的数据定位方法所考虑的数据定位信息一般是静态的,我们针对 P2P 分布存储系统中的数据定位信息(关键性描述信息)动态变化的情况,提出一种基于关键属性的优化数据一致性维护方法(optimistic data consistency maintenance method based on key-attributes for P2P distributed storage systems,简称 OCKP).该方法将重点放在快速的数据定位信息的一致性维护上,从而数据定位的性能不会因为数据定位信息的更新而降低.

在 OCKP 中,从用户提交的更新请求中分离出关于关键属性的更新,然后基于副本间的网络延迟选择具有最大和最小网络延迟的结点进行更新传播,记录并综合考虑确定和不确定两条更新传播路径.对关键更新冲突分类,针对每类冲突的特点,通过更新缓冲区预处理和更新日志协商机制,并采用最新写胜出(last-writer-win)和分而治之(divide and rule)规则优化关键更新冲突的发现和解决.

1 关键更新和更新日志

1.1 相关概念

P2P 分布存储系统往往包含多种以具体应用为背景的数据资源,每类资源具有各自的特点,并根据一定的标准描述、管理和定位.面向 Internet 的 P2P 分布存储系统通常支持基于关键属性的资源搜索,结点保存用于资源定位的资源对象的描述信息.P2P 系统将原始的资源搜索请求转发到各个结点上,搜索条件的具体匹配操作

通常是由本地结点来完成.

定义 1(关键属性集(key-attribute set)). 关键属性是描述 P2P 分布存储系统中资源的属性、对资源定位和管理、有助于数据检索的数据.在 P2P 分布存储系统中,同类资源的所有关键属性组成的集合为该类资源的关键属性集.资源 Re_s 的关键属性集记为 $KS(Re_s)$.

元数据、搜索引擎中常用的搜索条件等,都可以作为数据对象的关键属性.P2P 分布存储系统中对关键属性的更新具有以下特点:

- 易描述(simple description)——关键属性一般分属于资源的某个属性,因此,在描述对关键属性的更新时较为简单,不会产生歧义;并且属性名称不会随着数据对象内容的改变而变化.
- 更新项较小(little updates item)——由上述关键属性的描述可知,关键属性本身一般都不大,因此可以推断对关键属性的更新也较小;小的更新在传输时不会占用过多的通信资源,并且允许存在一定的冗余传输.
- 弱相关性(weak dependence)——多个关键属性之间一般相互独立,不存在一定的关联性,因此,在更新应用时不必考虑更新之间的因果关系.

用户提交的对数据对象的更新(或者系统信息的自我维护更新)往往包含多个对数据对象中不同数据内容的修改操作,将每项修改操作分别用语义描述,则更新为修改操作语义描述的集合.OCKP 中用语义表示更新,即对更新中的每项修改分别进行描述,记为 $Update_{i,j}(DE)$.其中,DE 为数据对象标识,i 为更新发布结点 ID,j 为更新序号,并且假设以各个结点上的数据对象为单位进行分配.

$Update_{i,j}(DE)$是二元组$\langle I_{i,j,k}(DE),U_{i,j,k}(DE)\rangle$的集合,其中,$I_{i,j,k}(DE)$是数据对象 DE 被更新的数据信息的描述,$U_{i,j,k}(DE)$是数据对象被修改之后相应的信息,k 为修改项目的编号.我们根据上述的更新语义描述方法,给出关键更新的定义.

定义 2(关键更新(key update)). 如果更新 $Update_{i,j}(DE)$满足

$$\{I_{i,j,k}(DE)|(DE\in Re_s)\wedge(\langle I_{i,j,k}(DE),U_{i,j,k}(DE)\rangle\in Update_{i,j}(DE))\}\subseteq KS(Re_s),$$

则认为更新 $Update_{i,j}(DE)$是对数据对象 DE 的关键更新.

用户提交的更新可能包含对关键属性的更新,同时也可能包含对非关键信息的更新.本文只对关键更新的传播和冲突进行研究,而非关键更新的相关内容不在讨论范围之内.因此,虽然用户提交的更新可能不是单纯的关键更新,但是用语义描述更新之后,很容易从中提取关键更新.

另外,本文假设关键属性的变化都不是由于数据内容本身的更新而引起的,因此,文中不存在关键属性的更新传播和数据本身的更新传播不同步的问题.

1.2 关键更新冲突

传统意义上的更新相互冲突是指结点 N_1 和 N_2 对同一数据对象 O 分别发布了更新 u_1 和 u_2,更新 u_1 的发布时间早于更新 u_2,且结点 N_2 在发布更新 u_2 时没有看到更新 u_1,则更新 u_1 和 u_2 相互冲突.根据弱一致性维护方法,更新 u_2 必须等到更新 u_1 被确认之后才可以确认.

假设同一数据对象的更新 u_1 和 u_2 修改的关键属性分别为$\{k_1,k_2,k_3\}$和$\{k_1,k_2,k_4,k_5\}$,更新发布时间分别为 t_1 和 t_2,且 $t_2>t_1$.假设更新 u_1 和 u_2 之间没有因果关系,副本 A 先接收到更新 u_2,按照传统的更新冲突解决策略,更新 u_2 必须等到副本 A 接收到更新 u_1 之后才能生效.但值得注意的是:

① 伪冲突更新——更新 u_2 对关键属性$\{k_4,k_5\}$的修改与更新 u_1 是不冲突的(伪冲突),单纯地对更新进行排序会阻止“伪冲突”更新的传播;

② 无意义更新——如果按照先 u_1 后 u_2 的顺序应用更新,则最终用户所看到的关键属性$\{k_1,k_2,k_3,k_4,k_5\}$的内容为$\{k_1(u_2),k_2(u_2),k_3(u_1),k_4(u_2),k_5(u_2)\}$,$u_1$ 对关键属性$\{k_1,k_2\}$的修改被 u_2 覆盖,而其是否可见,只对 u_1 的发布者有一定的意义;

③ 丢失更新——假设不受更新排序的约束而直接应用接收到的更新,以数据对象为最小更新考察单位,不考虑更新的具体内容,就会拒绝后续接收到的早期更新,带来的问题是关键属性 k_3 没有被修改.

OCKP 基于关键属性更新的特点,针对以上分析,定义关键更新冲突为

定义 3(关键更新冲突(key-update conflict). 同一数据对象 DE 上的关键更新 $Update_{i,j}(DE)$和 $Update_{l,m}(DE)$冲突当且仅当 $KS_{i,j}(DE)\cap KS_{l,m}(DE)\neq\varnothing$.其中,

$$KS_{i,j}(DE)=\{I_{i,j,k}(DE)|(\langle I_{i,j,k}(DE),U_{i,j,k}(DE)\rangle\in Update_{i,j}(DE))\},$$

$$KS_{l,m}(DE)=\{I_{l,m,n}(DE)|(\langle I_{l,m,n}(DE),U_{l,m,n}(DE)\rangle\in Update_{l,m}(DE))\}.$$

为了进一步明确关键更新冲突的冲突原因,OCKP 中将关键更新冲突分为 3 种冲突类型.冲突的关键更新 $Update_{i,j}(DE)$和 $Update_{l,m}(DE)$之间的冲突类型为

- 全等型冲突(congruence):$KS_{i,j}(DE)=KS_{l,m}(DE)$.
- 覆盖型冲突(coverage):$(KS_{i,j}(DE)\subset KS_{l,m}(DE))\vee(KS_{i,j}(DE)\supset KS_{l,m}(DE))$.
- 相交型冲突(intersection):

$$(KS_{i,j}(DE)-KS_{l,m}(DE)\neq\varnothing)\wedge(KS_{l,m}(DE)-KS_{i,j}(DE)\neq\varnothing)\wedge(KS_{l,m}(DE)\cap KS_{i,j}(DE)\neq\varnothing).$$

OCKP 在解决更新冲突时,首先判断更新之间的冲突类型,然后根据每种冲突类型的特点分别加以解决,见第 2.2 节.

1.3 更新日志中的数据结构

OCKP 方法在数据一致性的维护过程中跟踪副本的状态,记录更新的传播轨迹,并组织成更新日志.更新日志除了用于正常的更新传播中的信息交流以外,当出现结点失效和网络划分时,还可以根据更新日志中记录的信息,实现最终的数据一致.OCKP 的更新日志中主要包含副本索引表和关键更新表两个数据结构.

- 结点 p 的副本索引表 $Rlist(p)$

系统增加或撤销副本时,会以一定的机制通知其他副本,但由于网络连接或结点失效的存在以及网络延迟的差异,每个副本上维护的副本信息不同,结点 p 用副本索引表 $Rlist(p)$记录本地数据对象的副本信息.$Rlist(p)$是二元组$\langle DEE,S(p)\rangle$的集合,其中,$S(p)$为结点 p 所知的数据对象 DE 的副本结点 ID 集合.

- 结点 p 对资源 Re_s 的关键更新表 $Kupda(p,Re_s)$

结点 p 提取用户提交的更新的语义描述中的关键更新信息,并记录在 $Kupda(p,Re_s)$中,同时将从其他副本接收的关键更新信息根据第 2.2 节的更新冲突解决算法写入 $Kupda(p,Re_s)$, $Kupda(p,Re_s)$ 是六元组$\langle DE,Update_{i,j}(DE),\ T_{i,j},P_{i,j},UP_{i,j},Tag_{i,j}\rangle$的集合.其中:$T_{i,j}$ 为更新发布时钟;$P_{i,j}$ 是更新传播路径结点 ID 的集合,$P_{i,j}[0]$默认为更新发布结点;$UP_{i,j}$ 是不确定的更新传播结点 ID 的集合;$Tag_{i,j}$ 为更新标记,默认值为 0.

在不会造成误解的前提下,为了书写简洁,下面的叙述中所涉及的各种数据结构省略数据对象信息,比如 $Update_{i,j}(DE)$简写为 $Update_{i,j}$,$KS_{i,j}(DE)$简写为 $KS_{i,j}$ 等.

2 基于关键属性的优化数据一致性维护

基于关键属性的优化数据一致性维护方法 OCKP 的提出主要是为了满足面向 Internet 的 P2P 分布存储系统对关键属性信息快速更新的要求.该方法基于关键更新的特点,采用基于延迟-覆盖的更新传播模型进行更新的快速广域传播;基于关键更新冲突的分类,利用基于双层机制的更新冲突发现和解决过程实现关键更新的一致性;在更新传播和更新冲突协商过程中,通过更新日志记录更新并对其进行管理和维护.

OCKP 中关键更新一致性维护过程如下(如图 1 所示):

① 更新发布结点接收到用户提交的关于数据对象的更新.

② 提取更新中的关键更新信息,在同一数据对象的本地范围内对该更新依次分配编号.

③ 将更新写入相应的更新日志中的关键更新表(图 1 中标注为表 A).

④ 依据本地结点存储的更新日志中的副本索引表(图 1 中标注为表 B),获取结点间的网络延迟,并选择更新传播(主动推(push))对象.

⑤ 基于延迟-覆盖模型进行更新传播,详见第 2.1 节.

⑥ 副本将同时接收到的多个其他副本结点发送的关键更新消息放入更新缓冲(图 1 中标注为 Buffer),并对

该缓冲进行预处理.

⑦ 经过预处理的关键更新消息与关键更新表中的更新进行冲突检测,删除过时更新,增加并应用新的更新;并按照第④~⑦步继续传播更新.

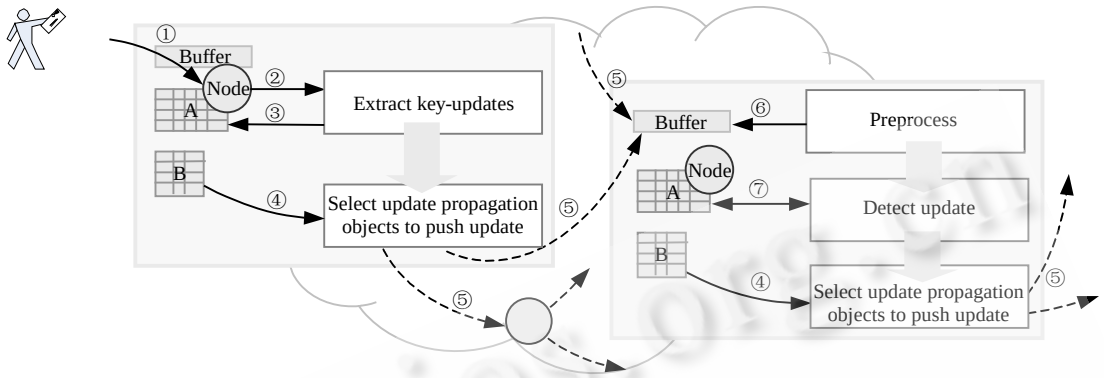

Fig.1 An illustration of update consistency maintenance process

图 1 更新一致性维护过程示意图

2.1 基于延迟-覆盖的更新传播

研究表明,对于网络连接稳定且不存在结点失效的分布式环境,每个副本结点上维护的副本生存信息都是最新且完整的,然后辅以多播[16],可以进一步加速更新的传播.但是,P2P 系统具有很强的动态性,结点的动态加入与退出是系统的一种常态.由于网络连接或结点失效的存在以及网络延迟,导致系统中各结点维护的副本信息不完全并存在差异.在这种环境中,依靠多播传播更新是不可靠的,往往是接收到更新消息的副本继续以多播的形式传播更新,导致系统中存在大量冗余消息.

更新传播效率是数据一致性维护的基础.在设计更新传播策略时,需要尽量避免传播的盲目性和冗余性.OCKP 方法采取基于延迟-覆盖的更新传播模型,具体算法描述见算法 1.

算法 1. OCKP 的关键更新传播算法.

Procedure *PropUpdate*(Peer *p*,Peer *q*,*UpdateMessage M*(*DE*))
  if $q=\varnothing$ **then** *UserUpdate*(*p*,*M*(*DE*)) **else** *PeerUpdate*(*p*,*M*(*DE*));
  for each Re_s **while** $Buffer\neq\varnothing$ **then** *PreocessBuffer*(*p*,Re_s);
  for each $Update_{i,j}\in Kupda(p,Re_s)$
    $List\leftarrow GetReplicaList(Rlist(p),Re_s,DE)$; //得到 *DE* 的副本列表
    if $(Tag_{i,j}=1)\wedge Size(List-P_{i,j}-UP_{i,j})>0$
      then $SendUpdate(p,List-P_{i,j}-UP_{i,j},Update_{i,j}(DE))$;
    if $(Tag_{i,j}=1)\wedge(Size(List-P_{i,j}-UP_{i,j})=0)\wedge(UP_{i,j}\neq\varnothing)$
      then $SendUpdate(p,UP_{i,j},Update_{i,j}(DE))$;
Procedure *UserUpdate*(Peer *p*,*UpdateMessage M*(*DE*)) //结点 *p* 接收到用户对数据对象 *DE* 的更新请求
  $j=AssignNumber(Re_s,DE,M)$;
  $Update_{p,j}(DE)\leftarrow CreateKeyUpdate(M(DE))$; //分配编号,提取关键更新信息
  $AddKupda(Kupda(p,Re_s),DE,Update_{p,j}(DE),T_{now},\{p\})$; //写入更新信息表, T_{now} 为更新发布时刻的时钟
  $List\leftarrow GetReplicaList(Rlist(p),Re_s,DE)$; //从副本索引表获取副本信息
  $SendUpdate(p,List-\{p\},Update_{i,j}(DE))$; **return**;
Procedure *PeerUpdate*(Peer *p*,*UpdateMessage M*(*DE*)) //结点 *p* 接收到结点 *q* 发送的消息
  $AddBuffer(Buffer(p),M(DE))$; **return**; //将 *M* 放入关键更新缓冲 Buffer 中
Procedure *SendUpdate*(Peer *p*,List *list*,*Update* $U_{i,j}(DE)$)
  $O[0,\ldots,h-1]\leftarrow OrderbyDelay(List,p)$; //按照 *List* 中各结点与结点 *p* 之间的网络延迟对其排序
  $m\leftarrow CreateMess(DE,Update_{i,j},T_{i,j},P_{i,j},UP_{i,j}\cup O[h-1])$; //创建更新消息

$m'\leftarrow CreateMess(DE,Update_{i,j},T_{i,j},P_{i,j},UP_{i,j}\cup O[0])$;
$PropaUpdate(O[0],m)$;$PropaUpdate(O[h-1],m')$; //传播更新消息
$RecoverTag(Re_s,DE,Update_{i,j})$; **return**; //标记位复位,赋值为 0

- 双向延迟传播(double-delay-biased):更新接收副本结点,获取与其他副本结点之间的网络延迟,将更新转发给网络通信延迟最小(min_*node*)以及最大(max_*node*)的副本结点.选取网络通信延迟最小的副本可以尽量不增加网络负载,确保更新传播到其他结点;选取远程副本结点的目的是提供更新传播的广域扩散,加快更新的传播,并避免更新传播在系统中的局部聚集.

  网络延迟测量会产生一定的数据流量,尤其在广域网络环境中,每个数据对象的副本结点数量很大,频繁的网络延迟测量产生的数据流量会对系统性能造成一定影响.为了避免该问题,OCKP 方法中按照周期测量网络延迟,并且可以根据更新的发生频率适当地调整测量周期的大小.例如,当更新的发生非常少时,可以增大网络延迟的测量周期,避免冗余的网络测量;当更新的发生很频繁时,可以缩短网络延迟的测量周期,尽可能地体现测量结果的"实时性",避免网络测量的误差对算法性能的影响.
- 双路径覆盖传播(double-path-overlay):更新信息中记录确定传播到的结点集合($P_{i,j}$)和不确定是否传播到的结点集合($UP_{i,j}$),并在更新传播时进行综合考虑.例如,对于上面的 min_*node* 和 max_*node*,不能确保向这两个结点同时传播的更新一定被接收到,因此,每个结点都被记录在对方更新信息的 $UP_{i,j}$ 中.引入 $P_{i,j}$ 可以避免信息的冗余发送,而 $UP_{i,j}$ 可以解决信息传播的不可靠问题,双路径机制提高了更新传播效率,并加速了更新传播的聚合.

2.2 基于双层机制的更新冲突解决

2.2.1 更新冲突解决规则

OCKP 方法采用"最新写胜出"和"分而治之"的规则解决关键更新冲突.

- "最新写胜出"规则解决"更新对象"相同的更新冲突(这里所指的"更新对象"不是数据对象,而是数据对象中具体的关键属性信息).该规则针对全等型冲突,主要面向"无意义更新"问题,即对于同一(或同组)关键属性上的更新,发布时间晚的更新胜出.

  假设同一(或同组)关键属性上的更新μ_1和μ_2,且更新发布时间 $T(\mu_1)<T(\mu_2)$.如果结点 N 按照$\mu_1\mu_2$的顺序接收更新,则更新μ_1的更新结果被更新μ_2所覆盖;如果结点 N 按照$\mu_2\mu_1$的顺序接收到更新,则更新μ_1的更新结果被忽略.
- "分而治之"规则针对覆盖型冲突和相交型冲突,主要面向"伪冲突更新"和"丢失更新"问题.该规则在保证"最新写胜出"规则有效的前提下,不丢失关键属性信息的更新,并可以避免因为"伪冲突"导致的关键更新传播延迟.

我们仍以第 1.2 节中的更新 u_1 和 u_2 为例,则直观上更新将被分解为 $u_1'\{k_1,k_2\}$,$u_2'\{k_3\}$ 和 $u_3'\{k_4,k_5\}$ 这 3 个更新,对每个分解后的更新分别应用"最新写胜出"规则以及分别进行更新传播处理,详见下一节的更新冲突发现与解决算法.

2.2.2 更新冲突发现与解决过程

在 OCKP 方法中,通过更新缓冲预处理和检测关键更新表两层机制识别和解决更新冲突.

- 更新缓冲记录副本从其他副本同时接收到的多个关键更新信息,设立更新缓冲并进行预处理带来的好处是:① 合并复制更新的传播路径,删除冗余的更新,复制的更新主要是由于更新在系统中沿着多条路径并行传播产生的;② 提早发现、解决更新缓冲中的全等型冲突,减少关键更新表中更新冲突解决时的计算开销.算法 2 是更新缓冲中等待处理的更新信息的更新冲突预处理算法.

算法 2. OCKP 的预处理算法.

Procedure *ProcessBuffer*(Peer p,Resource Re_s)
$DS\leftarrow GetDataList(p,Re_s,Buffer(p))$; //获取 $Buffer(p)$中的数据对象集合
for each $DE\in DS$

```
    if (Size(Buffer(p),DE)>1
      then if ∃Update_{i,j},Update_{l,m}∈Buffer(p)(P_{i,j}[0]=P_{l,m}[0]∧T_{i,j}=T_{l,m})   //重复发送的关键更新
             then P_{i,j}←P_{i,j}∪P_{l,m};
                  UP_{i,j}←(UP_{i,j}∪UP_{l,m})−P_{i,j}−P_{l,m};
                  DeleteBuffer(p,Re_s,Update_{l,m});                                       //删除重复的项目
           if ∃Update_{i,j},Update_{l,m}∈Buffer(p)(KS_{i,j}=KS_{l,m})                      //全等型冲突
             then 比较 T_{i,j} 与 T_{l,m},并假设 T_{i,j}≥T_{l,m};
                  DeleteBuffer(p,Re_s,Update_{l,m});                                       //删除较早的项目
    L(DE)←GetBuffer(p,Re_s,DE);                                                            //提取缓冲中的关键更新信息
    ProcessKupda(p,Re_s,L(DE));                                                            //处理更新冲突
    return;
```

- 针对关键更新表的更新冲突检测,主要是指经过预处理之后的更新缓冲中的关键更新信息与关键更新表中的更新进行比对,具体分析更新冲突的各种类型,分辨新接收到的更新与复制的更新、冲突与“伪冲突”、有意义更新与无意义更新,并分别给出具体的解决方案.算法 3 给出了相应的算法描述.

算法 3. OCKP 的更新冲突检测与解决算法.

```
Procedure ProcessKupda(Peer p,Resource Re_s,UpdateList L(DE))
  for each Update_{l,m}∈L(DE)
    tag=0;E_{ks}←∅;N_{ks}←∅;G_{ks}←∅;
    for each Update_{i,j}∈Kupda(p,Re_s)
      if (KS_{l,m}∩KS_{i,j}≠∅) then tag=1;
      if KS_{i,j}=KS_{l,m}                                                  //全等型冲突
        then if T_{l,m}<T_{i,j} then break;
             if (T_{l,m}<T_{i,j})∧(P_{l,m}[0]=P_{i,j}[0])                   //复制的更新
               then P_{i,j}←P_{i,j}∪P_{l,m};UP_{i,j}←(UP_{i,j}∪UP_{l,m})−P_{i,j}−P_{l,m};   //合并传播路径
                    break;
               else DeleteKupda(p,Re_s,Update_{i,j});                       //Update_{i,j} 为较早更新,从关键更新表中删除
                    AddKupda(p,Re_s,DE,Update_{l,m},T_{l,m},P_{l,m}∪{p},UP_{l,m},1);   //向关键更新表中添加更新
                    break;
      if KS_{i,j}⊃KS_{l,m}                                                 //覆盖型冲突
        then if T_{l,m}<T_{i,j} then break;
             else ShrinkKupda(p,Re_s,DE,Update_{i,j},KS_{i,j}−KS_{l,m});   //更新项目分裂
                  AddKupda(p,Re_s,DE,Update_{l,m},T_{l,m},P_{l,m}∪{p},UP_{l,m},1);
                  break;
      if KS_{i,j}⊂KS_{l,m}                                                 //覆盖型冲突
        then if T_{l,m}<T_{i,j} then E_{ks}←E_{ks}∪{KS_{i,j}};
             else N_{ks}←N_{ks}∪{KS_{i,j}};
                  DeleteKupda(p,Re_s,Update_{i,j});
                  AddKupda(p,Re_s,DE,Update_{l,m}(KS_{i,j}),T_{l,m},P_{l,m}∪{p},UP_{l,m},1);
      if (KS_{i,j}−KS_{l,m}≠∅)∧(KS_{l,m}−KS_{i,j}≠∅)∧(KS_{l,m}∩KS_{i,j}≠∅)   //相交型冲突
        then if T_{l,m}<T_{i,j}                                             //G_{ks} 记录新分裂出的更新所包含的关键字
               then G_{ks}←KS_{l,m}−E_{ks}−N_{ks}−(KS_{i,j}∩KS_{l,m});
                    E_{ks}←E_{ks}∪(KS_{i,j}∩KS_{l,m});
               else G_{ks}←KS_{l,m}−E_{ks}−N_{ks};
                    ShrinkKupda(p,Re_s,DE,Update_{i,j},KS_{i,j}−KS_{l,m});
      if G_{ks}≠∅ then AddKupda(p,Re_s,DE,Update_{l,m}(G_{ks}),T_{l,m},P_{l,m}∪{p},UP_{l,m},1);
      if tag=0                                                              //与 Kupda 中的项目完全不冲突的更新
        then AddKupda(p,Re_s,D_{s,d},Update_{l,m},T_{l,m},P_{l,m}∪{p},UP_{l,m},1);
    if ∃Update_{i,j},Update_{l,m}∈Kupda(T_{i,j}=T_{l,m}∧P_{i,j}[0]=P_{l,m}[0])   //合并 Kupda 中同一发布结点上同时发布的项目
```

then *MergeKupda*(p,Re_s,DE, $Update_{i,j}$,$Update_{l,m}$);

return;

对于算法 3,有以下几点需要说明:

- 算法中,函数 *ShrinkKupda*(p,Re_s,DE,$Update_{i,j}$,$KS_{i,j}$–$KS_{l,m}$)的解释是:在更新信息表 *Kupda*(p,Re_s)中,更新 $Update_{i,j}$ 所属的对数据对象 DE 的更新项目中其他信息保持不变,$Update_{i,j}$ 修改为

$$Update'_{i,j}=\{\langle I_{i,j,k},U_{i,j,k}\rangle \mid \langle I_{i,j,k},U_{i,j,k}\rangle \in Update_{i,j} \wedge I_{i,j,k} \in KS_{i,j} \wedge I_{i,j,k} \notin KS_{l,m}\}.$$

- 更新分裂再合并:更新冲突解决算法对关键更新表中因为中间操作而造成的更新的分裂进行合并.算法过程中,分裂由同一结点同时发布的更新内容是为了简化更新缓冲中其他更新的比对操作;算法结束前,合并存在的分裂更新(满足 $\exists_{Update_{i,j},Update_{l,m}\in Kupda}(T_{i,j}=T_{l,m} \wedge P_{i,j}[0]=P_{l,m}[0])$ 的更新),可以减少传播的更新消息数量,并进一步减小后续接收到更新的结点的计算开销.因此,算法结束后,关键更新表中的更新项目满足:$\neg(\exists_{Update_{i,j}(D_{s,d}),Update_{l,m}(D_{s,d})\in Kupda(p,Re_s)}(T_{i,j}=T_{l,m} \wedge P_{i,j}[0]=P_{l,m}[0]))$.
- 算法结束后,关键更新表内任意两个项目中的更新所包含的更新属性完全不相同:

$$\forall_{Update_{i,j}(D_{s,d}),Update_{l,m}(D_{s,d})\in Kupda(p,Re_s)}(KS_{i,j}(D_{s,d}) \cap KS_{l,m}(D_{s,d})=\varnothing).$$

假设资源 Re_s 的关键属性集大小为 k,则该类资源的任一数据对象在关键更新表 *Kupda*(p,Re_s)中最多具有 k 个项目.

- 不同副本结点在同一时间对同一类资源的关键属性集的相同子集发布更新,这种状况的出现是小概率事件,因此,假设系统中不存在此类冲突,并在算法中不予考虑.

2.3 更新日志维护

更新的转发基于更新日志中副本索引表记录的副本信息,更新冲突的发现和解决基于更新日志中关键更新表记录的更新信息,下面讨论更新日志中数据结构的维护方法.

2.3.1 副本索引表的维护

副本索引表的维护主要体现在增加和撤销副本时副本信息的一致性维护,副本索引表的一致可以在一定程度上避免冗余信息的传输,并加速更新传播的聚合.

OCKP 方法将增加副本和撤销副本作为对副本索引表的更新,由于副本的增加和撤销需要传播的消息很小,因此,为了加快副本索引表的一致性维护过程,由增加的副本(或撤销的副本)以多播的形式发布.首次接收到该消息的副本对本地的副本索引表进行相应的增加(或删除)操作,并继续以多播的形式转发给除了发送方之外的其他副本;如果副本接收到复制的消息,则不进行任何操作.我们在第 3.1 节中对采用多播通知后,副本的增加对系统性能的影响进行了比较测试.

在副本索引表中,同一副本的增加和撤销之间具有因果关系,在更新过程中必须维护“先增加后撤销”的更新排序.如果副本 r 接收到撤销副本 r'的消息,但是副本 r 本地存储的副本索引表中并没有关于副本 r'的记录,由于增加或撤销副本是系统根据一定的复制策略全局决策的,因此,不可能出现“增加立即撤销”,故此推断副本 r 在前一段时间内可能处于网络划分状态,与部分副本失去了联系.这时,副本 r 需要启动网络划分恢复时的一致性维护,见第 2.3.2 节.

2.3.2 关键更新表的维护

关键更新表记录本地副本结点接收到的最新更新.副本接收到比关键更新表中的项目较新的或是从未看到的更新时,将其记录在本地的关键更新表中,并按照传播策略将其继续传播到其他副本.

如果数据对象 DE 在副本索引表中记录的副本(*List*)全部包含于更新 $U_{i,j}(DE)$的传播路径($P_{i,j}$)和不确定传播结点集合($UP_{i,j}$)的并集,为了避免多播的不可靠性带来的更新传播不完全,可以选择 $UP_{i,j}$ 中的结点继续传播,直到 $UP_{i,j}$ 为空,更新 $U_{i,j}(DE)$的传播“暂时”停止.“暂时”停止主要来自两方面的原因:① 如果不能保证副本索引表强一致的前提,就不能确定 *List* 包含了全部副本,即不能确保所有副本都接收到了该更新;② 从失效或网络划分中恢复的结点需要从其他副本获取未曾看到的更新,从而再次启动了更新传播.

当副本 r 从失效中恢复时,对于结点 r 上的每个数据对象:结点 r 获取与本地副本索引表中该数据对象各个副本的通信延迟,与具有最小通信延迟的副本 r' 比较最新的关键更新的发布时钟;如果副本 r 的最新关键更新的发布时钟小于副本 r' 的相应值,则副本 r 复制副本 r' 上该数据对象对应的关键更新表中的信息.副本 r 还需要从副本 r' 复制并覆盖本地的副本索引表中的相应信息.

当副本 r 从网络划分中恢复时,其一致性维护过程如下:

① 获取一致性协商的会话副本 r'.副本 r 获取结点集合 S,S 为本地所有数据对象最新的且发布时钟距离当前时钟不超过阈值 T_0 的关键更新的确定传播路径与不确定传播结点集合的并集,则认为集合 S 中的结点与副本 r 之前处于同一划分中.对于副本 r 上的每个数据对象 DE,副本 r 获取集合 $S'=(S(r)-S)|\langle DE,S(r)\in Rlist(r)\rangle$,选取与 S' 中通信延迟最小的副本 r' 进行一致性协商.

② 与副本 r' 进行一致性协商.副本 r 读取副本 r' 中的关键更新信息放入本地缓冲;根据第 2.2 节中"OCKP 的更新冲突发现与解决算法"进行处理;副本 r' 复制处理结束后副本 r 的关键更新表中的信息,从而副本 r 与 r' 的关键更新表达成一致.另外,副本 r 与副本 r' 比对并统一副本索引表中的信息.

由于每类资源的关键属性集是有限集合,并且关键更新表中只存储最新的更新,因此,关键更新表不会无限地占用存储空间.另外,系统通过定期对关键更新表进行维护,清空表中的所有内容,可以避免长时间没有接收到更新的数据对象占用存储空间.

3 模拟测试

在已有的一致性维护方法中,最常用的是反熵方法(anti-entropy)[17].反熵是一种一致性协调机制,每个结点阶段性地与选取的另一结点进行一致性协调,直到两个结点上的数据对象一致为止.根据选择会话结点的方式,反熵方法被分为随机方法、确定性方法和基于拓扑结构方法,我们选择其中比较有代表性的均匀方法(uniform)、基于延迟的方法(delay-biased)和基于树形结构的方法(tree)[18]与 OCKP 方法进行性能比对.在均匀方法中,每次会话对象的选择对于所有副本来讲是等概率的;在基于延迟的方法中,每次选择通信开销最小的副本会话;基于树形结构的方法将所有副本组织成树形结构,结点只能选择与其父结点或子女结点会话.

OptorSim[19]是一个用于测试复制算法的模拟器.OptorSim 模拟的分布存储系统由一个资源代理器和多个存储结点构成.资源代理器负责接收外部数据请求,并将数据请求发送到各个存储结点.每个存储结点包括计算单元、存储单元和副本管理器,具有一定的计算能力和存储能力.副本管理器负责副本的选择和动态调整.我们将基于该模拟器进行性能测试.

在我们的性能测试中模拟了 1 000 个结点的 P2P 分布存储系统,系统中共有 10 类数据资源、300 个关键属性;每类数据资源从 300 个关键属性中随机选取 50 个组成该类资源的关键属性集;每类数据资源有 100 个数据对象,每个数据对象为 1GB;每个数据对象在产生时刻对其关键属性赋予随机初值,结点记录本地存储的数据对象的关键属性信息.

我们的性能测试中模拟了 6 类更新,每类更新随机地从本类关键属性集中选取任意 k 个关键属性组成,见表 1.

Table 1 Update types

表 1 更新类型

Type identifier	The number of key-attributes	Probability (%)
1	1~5	25
2	5~10	40
3	10~20	25
4	20~30	6
5	30~40	3
6	40~50	1

3.1 一致性的消息开销

假设系统中每个数据对象有 *Num_Replica* 个副本.如果结点 P 上发布了关于数据对象 *Date* 的关键更新 u,

则理想状况下,只需要发送 *Num_Replica*−1 个消息,就可以保证数据对象 *Date* 的所有副本都接收到更新 *u*,实现一致性维护.

但是,大规模分布式网络环境中存在消息丢失问题,即不能保证发送的更新消息可以被目标结点所接收,OCKP 方法针对该问题,引入不确定更新传播路径($UP_{i,j}$)来避免更新传播的不完全.

假设系统中消息的丢失率为 *failure*(0≤*failure*<1).根据图 2 中的基于延迟-覆盖的更新传播模型,为了确保更新 *u* 被所有副本接收到,消息的传输量 *Num_Info* 表示为

$$Num\_Info=\lim_{x\to\infty}\sum_{i=0}^{x}(Num\_Replica-1)\times failure^{i}.$$

与理想状态下的消息开销相比,消息的增加比例 *IncreRatio* 为

$$\begin{aligned}IncreRatio&=\frac{Num\_Info-(Num\_Replica-1)}{(Num\_Replica-1)}\\&=\frac{\lim\limits_{x\to\infty}\sum\limits_{i=0}^{x}(Num\_Replica-1)\times failure^{i}-(Num\_Replica-1)}{(Num\_Replica-1)}\\&=\frac{(Num\_Replica-1)\times\left(\lim\limits_{x\to\infty}\sum\limits_{i=0}^{x}failure^{i}-1\right)}{(Num\_Replica-1)}\\&=\lim_{x\to\infty}\sum_{i=1}^{x}failure^{i}.\end{aligned}$$

假设 *Num_Replica*=1001, *failure*=1/100,则对任一关键更新的一致性维护,OCKP 方法所带来的消息开销为

$$Num\_Info=\lim_{x\to\infty}\sum_{i=0}^{x}(1001-1)\times(1/100)^{i}\approx1000\times(1+1/100+(1/100)^{2})=1010.$$

与理想状态下的消息开销相比,消息的增加比例为

$$IncreRatio=\lim_{x\to\infty}\sum_{i=1}^{x}(1/100)^{i}\approx1/100.$$

OCKP 方法在进行关键更新一致性维护时,为了避免更新传播的不完全性,需要重复发送更新.由上述分析可知,这种重复发送的更新消息的数量较少,并且与理想状态下发送的更新消息的数量相比,消息的增加比例也是非常小的.另外,由于关键更新消息本身也非常小,因此增加的消息传输不会造成网络拥塞.

3.2 一致性的时间开销

我们模拟测试在副本数量固定的情况下,OCKP 方法与其他 3 种方法在一致性维护方面的开销(如图 2 所示).

初始状态时,每个数据对象产生固定数量的副本,并将这些副本分散到随机选择的结点上.我们对每种方法在不同的副本数量下分别进行一组模拟实验,每组模拟实验都对 P2P 系统提交 5 000 个更新,模拟运行结束条件为完成 5 000 个更新的传播,所有副本最终一致,比较更新的平均传播时间.从实验结果可以看出:树形结构方法在传播更新时受限于先前建立的结点间的层次关系,因此开销最大;均匀方法在副本数量较小时性能最佳;OCKP 方法由于每次都选择通信开销最小和最大的结点进行更新传播,因此,其性能要优于基于延迟的方法.

我们模拟在副本数量动态变化的情况下,OCKP 方法和具有可比性的基于延迟的方法在一致性维护方面的开销;同时比较了增加副本时采用多播通知其他副本和不采用多播通知其他副本对性能的影响(如图 3 所示).

为了模拟副本数量动态变化,我们采用根据访问频率增加和撤销副本的复制策略:如果某个结点对某个数据对象的访问频度超过设定的阈值,就需要在该结点增加一个该数据对象的副本;如果结点有空闲的存储空间,则不撤销任何副本;否则,撤销访问频率最低的副本.

在每组模拟的初始时刻,每个数据对象产生固定数量的副本,并将这些副本分散到随机选择的结点上.在实验进行过程中,除了对 P2P 系统提交 5 000 个更新之外,还另外提交 10 000 次随机访问请求.从实验结果可以看

出:采用多播通知增加副本对 OCKP 方法带来了性能上的提高,因为尽快地获取新增加副本的存在,有利于更新的广域传播;对于基于延迟的方法,由于复制与一致性维护总是通过“最近”的副本来完成,因此,远程副本对新增副本的存在并不关心,采用多播对性能几乎没有影响.

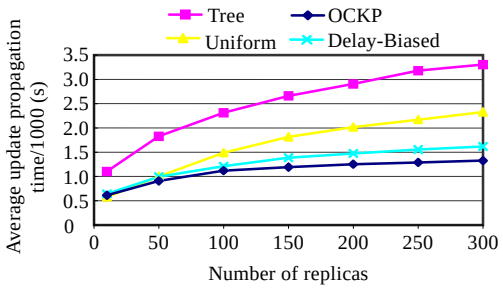

Fig.2 Comparisons of consistency overhead in fixed number of replicas

图 2 副本数量固定情况下的一致性开销对比

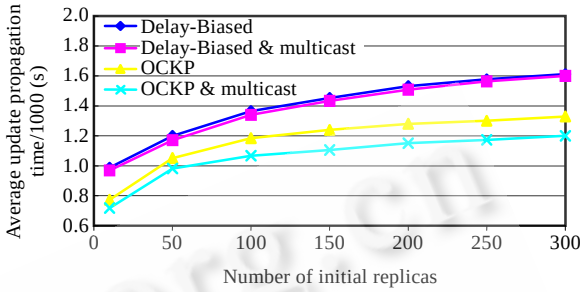

Fig.3 Comparisons of consistency overhead in dynamic alteration of number of replicas

图 3 副本数量动态变化情况下的一致性开销对比

3.3 资源定位与资源访问的时间开销

资源定位(resource location)与资源访问(resource access)的区别是:资源定位是指在分布系统中根据一定的条件搜索到数据资源所在的位置;资源访问在此基础上将定位到的数据资源从远程结点读取到本地.

我们采用二路随机转发方法(two-way random forwarding)[20]作为资源定位方法,模拟比较 OCKP 方法与其他几种方法资源定位的时间(如图 4 所示).二路随机转发方法指结点每次随机选择两个邻居结点转发搜索消息.

模拟初始化时,每个结点上存储了随机选择的 100 个数据对象.每组模拟实验都对 P2P 系统提交 5 000 个更新并同时进行 10 000 次搜索,每隔 1 000 次搜索统计一次在当时的系统中资源搜索的平均搜索延迟(取附近 200 次搜索的平均值).在模拟实验中,每次搜索请求都由系统中随机选择的一个结点上发出,并随机选择某类资源中的 10 个关键属性进行搜索.从实验结果可以看出,OCKP 方法在解决关键属性的更新冲突时根据其特性进行了特别设计,因此,其关键属性的更新传播速度较快,从而缩短了资源定位时间.

我们在上面的资源定位实验的基础上模拟测试了资源访问开销(如图 5 所示).

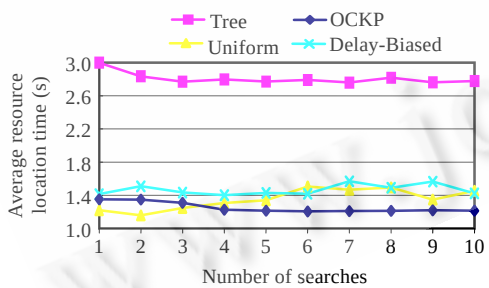

Fig.4 Comparisons of resource location overhead

图 4 资源定位开销对比

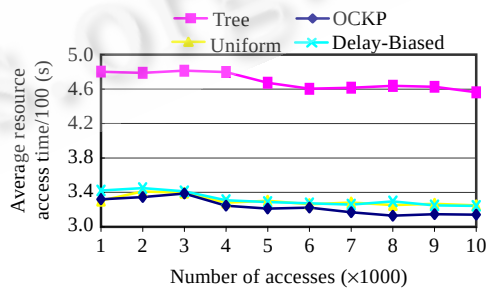

Fig.5 Comparisons of resource access overhead

图 5 资源访问开销对比

从实验结果可以看出,OCKP 方法由于其更新传播速度的提高,使得定位到的数据对象副本距离访问发起结点较近,从而避免了远程的数据传输.另外,随着访问次数的增加,系统中新创建了数据副本,副本数量的增加也使得每种方法的访问开销均有小幅度的降低.

3.4 鲁棒性

我们模拟测试 OCKP 方法与均匀方法和基于延迟的方法在不同的结点失效率下资源定位的性能(如图 6

所示).

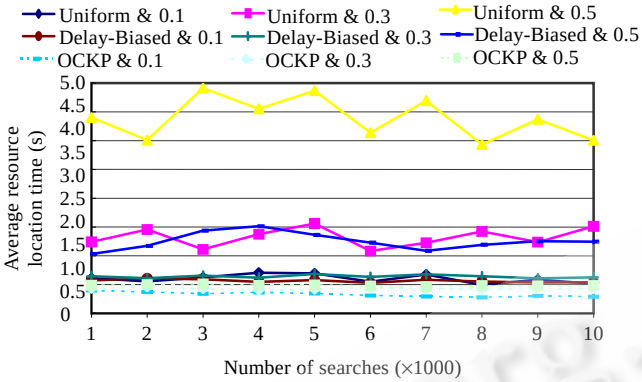

Fig.6 Comparisons of robustness

图 6 鲁棒性对比

在模拟初始化时,每个结点上存储了随机选择的 100 个数据对象.每组模拟实验中,随机地从系统中选择数量为(1000×结点失效率)个结点,使其处于失效状态,处于失效状态的结点在随机的时间段后恢复,并按照各自方法的设计选择与系统中其他副本达成一致;对 P2P 系统提交 5 000 个更新并同时进行 10 000 次搜索,每隔 1 000 次搜索统计一次在当时的系统中资源搜索的平均搜索延迟(取附近 200 次搜索的平均值).从模拟结果可以看出:均匀方法受结点失效的影响最大,因为该方法在传播更新时,盲目地将更新发送给远程副本结点,对结点的状态并不关心;OCKP 方法根据通信延迟选择结点进行更新广域传播,更新总是可以传送到未处于失效状态的结点,因此具有较强的鲁棒性.

4 结 论

针对传统的复制算法不能满足面向 Internet 的 P2P 分布存储系统对关键信息快速更新的要求问题,优化数据一致性方法 OCKP 分离出用户提交的更新请求中的关键属性更新;采用基于延迟-覆盖的更新传播机制进行更新的快速广域传播;对关键更新冲突分类,利用最新写胜出和分而治之的冲突解决规则,结合基于双层机制的更新冲突发现和解决过程实现关键更新的一致性;并对更新日志中主要数据结构信息的更新以及结点失效和网络划分发生时信息的维护进行了讨论.

OCKP 实现了 4 个目标:① 解决更新传播的不可靠、盲目性和冗余性问题,提高更新传播效率;② 关键更新冲突的分类明确冲突产生的原因,利于具体问题具体分析;③ 关键属性更新的优化处理适用于面向 Internet 的 P2P 系统的要求,不会因为关键属性更新的延迟而降低系统基于关键属性的资源定位效率;④ 在结点失效和网络划分的情况下,确保副本关于关键更新的一致.模拟测试结果也表明,该方法具有较小的一致性维护开销、基于关键属性的资源定位开销和资源访问开销,并具有较强的鲁棒性,适用于具有很强动态性的 P2P 系统.

References:

[1] Rhea S, Wells C, Eaton P, Geels D, Zhao B, Weatherspoon H, Kubiatowicz J. Maintenance-Free global data storage. IEEE Internet Computing, 2001,5(5):40−49.

[2] Druschel P, Rowstron A. PAST: A large-scale, persistent peer-to-peer storage utility. In: Proc. of the HotOS VIII. Washington: IEEE Computer Society, 2001. 75−80.

[3] van Renesse R. Efficient reliable Internet storage. In: Proc. of the Workshop on Dependable Distributed Data Management. Washington: IEEE Computer Society, 2004.

[4] Zhang Z, Lin SD, Lian Q, Jin C. RepStore: A self-managing and self-tuning storage backend with smart bricks. In: Proc. of the 1st IEEE Int'l Conf. on Autonomic Computing. Washington: IEEE Computer Society, 2004. 122−129.

[5] Dahlin M, Gao L, Nayate A, Venkataramani A, Yalagandula P, Zheng JD. PRACTI replication for large-scale systems. Technical Report, TR-04-28, Austin: University of Texas at Austin, 2004.

[6] Kang BBH. S2D2: A framework for scalable and secure optimistic replication [Ph.D. Thesis]. Berkeley: University of California, 2004.

[7] van Renesse R, Schneider FB. Chain replication for supporting high throughout and availability. In: Proc. of the 6th Symp. on Operating Systems Design & Implementation (OSDI 2004). Berkeley: USENIX Association, 2004. 91−104.

[8] Leontiadis E, Dimakopoulos VV, Pitoura E. Creating and maintaining replicas in unstructured peer-to-peer systems. In: Proc. of the 12th Int'l Euro-Par Conf. on Parallel Processing (EURO-PAR). LNCS 4128, Berlin, Heidelberg: Springer-Verlag, 2006. 1015−1025.

[9] Yu HF, Vahdat A. Consistent and automatic replica regeneration. ACM Trans. on Storage, 2005,1(1):3−37.

[10] del Vecchio D, Son SH. Flexible update management in peer-to-peer database systems. In: Proc. of the 9th Int'l Database Engineering & Application Symp. (IDEAS 2005). Washington: IEEE Computer Society, 2005. 435−444.

[11] Roussopoulos M, Baker M. CUP: Controlled update propagation in peer-to-peer networks. In: Proc. of the 2003 USENIX Annual Technical Conf., General Track. San Antonio: USENIX Association, 2003. 167−180.

[12] Yin LZ, Cao GH. DUP: Dynamic-tree based update propagation in peer-to-peer networks. In: Proc. of the 21st IEEE Int'l Conf. on Data Engineering (ICDE 2005). Washington: IEEE Computer Society, 2005. 258−259.

[13] Aberer K, Cudré-Mauroux P, Datta A, Despotovic Z, Hauswirth M, Punceva M, Schmidt R. P-Grid: Dynamics of self-organizing processes in structured P2P systems. In: Proc. of the Peer-to-Peer Systems and Applications. LNCS 3485, Berlin, Heidelberg: Springer-Verlag, 2005. 137−153.

[14] Chen X, Ren SS, Wang HN, Zhang XD. SCOPE: Scalable consistency maintenance in structured P2P systems. In: Proc. of the 24th Conf. of the IEEE Communications Society (INFOCOM 2005). Washington: IEEE Computer Society, 2005. 1502−1513.

[15] Wang ZJ, Das SK, Kumar M, Shen HP. Update propagation through replica chain in decentralized and unstructured P2P systems. In: Proc. of the 4th IEEE Int'l Conf. on Peer-to-Peer Computing (P2P 2004). Washington: IEEE Computer Society, 2004. 64−71.

[16] Golding RA, Long DDE. Design choices for weak-consistency group communication. Technical Report, UCSC-CRL-92-45, Santa Cruz: University of California, 1992.

[17] Petersen K, Spreitzer MJ, Terry DB. Flexible update propagation for weakly consistent replication. In: Proc. of the 16th ACM Symp. on Operating Systems Principles. New York: ACM Press, 1997. 288−301.

[18] Golding RA, Long DDE. Modeling replica divergence in a weak-consistency protocol for global-scale distributed data bases. Technical Report, UCSC-CRL-93-09, Santa Cruz: University of California, 1993.

[19] Bell WH, Cameron DG, Capozza L, Millar AP, Stockinger K, Zini F. Optorsim: A grid simulator for studying dynamic data replication strategies. Int'l Journal of High Performance Computing Applications, 2003,17(4):403−416.

[20] Lü Q, Cao P, Cohen E, Li K, Shenker S. Search and replication in unstructured peer-to-peer networks. In: Proc. of the 16th ACM Int'l Conf. on Supercomputing (ICS 2002). New York: ACM Press, 2002.

周婧(1977－),女,江苏徐州人,博士,讲师,主要研究领域为网络计算,虚拟现实.

李思昆(1941－),男,教授,博士生导师,CCF 高级会员,主要研究领域为虚拟现实与可视化,嵌入式系统,SoC 设计方法.

王意洁(1971－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络计算,数据库技术,移动计算.