文章编号:1001-2486(2007)04-0042-05

一种面向关键属性更新的优化数据一致性算法*

周 婧[1],王意洁[1],李思昆[1],王元志[2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073; 2. 工程兵指挥学院,江苏 徐州 221004)

摘 要:规模巨大且分布性强的P2P系统可能导致部分数据副本发布的更新被长时间地延迟,从而降低Internet上资源定位的效率。针对关键属性更新的特点,提出一种解决关键属性更新冲突的优化数据一致性算法。算法中分离出用户提交的更新请求中关于关键属性的更新,在对关键更新冲突分类的基础上,采用更新缓冲预处理和关键更新表两层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。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算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
中图分类号:TP393 **文献标识码**:A

An Optimistic Data Consistency Algorithm for Update of Key-attributes

ZHOU Jing[1], WANG Yi-jie[1], LI Si-kun[1], WANG Yuan-zhi[2]
(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Command Institute of Engineer Corps, Xuzhou 221004, China)

Abstract: In generally large-scale and strong distributed P2P systems, updates issued by replicas may be delayed, and then resource location performance on the Internet may be depressed. According to the characteristics of updates about key-attributes in P2P systems, an optimistic data consistency maintenance algorithm is proposed. The update about key-attributes was separated from user update request, and based on classifying key-update conflicts, a double-level mechanism including preprocessing buffer and key-update table were applied to detect and reconcile conflicts, and then conflicts were solved by policies as last-writer-win and divide-and-rule. 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 algorithm, achieving good consistency overhead, resource location and resource access overhead, and having strong robustness.

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

数据复制是改善分布式系统性能的重要技术,但不可避免地面临副本一致性维护的问题[1-3]。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。本文提出一种面向P2P分布存储系统的解决关键属性更新冲突的优化算法(solving update conflict of key-attributes optimistic algorithm,UCKA)。

1 关键更新和更新日志

定义1 关键属性集: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。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_i 的关键属性集记为 $KS(Re_i)$。

* 收稿日期:2006-10-12
基金项目:国家自然科学基金重大项目;高等学校全国优秀博士学位论文作者专项资金项目(200141);国家部委资助项目
作者简介:周婧(1977—),女,博士生。

UCKA中用语义表示更新,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 为修改项目编号。

定义2 关键更新:如果更新 $Update_{i,j}(DE)$满足

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

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

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。但是,(1)伪冲突更新,更新 u_2 对$\{k_4, k_5\}$的修改与更新 u_1 是不冲突的("伪冲突");(2)无意义更新,即使按照先 u_1 后 u_2 的顺序应用更新,最终用户所看到的$\{k_1, k_2, k_3, k_4, k_5\}$的内容为$\{k_1(u_2), k_2(u_2), k_3(u_1), k_4(u_2), k_5(u_2)\}$;(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,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。

定义3 关键更新冲突: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 $KS_{i,j}(DE) \wedge KS_{l,m}(DE) \neq \varnothing$。其中:

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

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

为了进一步明确关键更新冲突的冲突原因,UCKA中将关键更新冲突分为三种冲突类型。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:$KS_{i,j}(DE) = KS_{l,m}(DE)$
- 覆盖型冲突:$(KS_{i,j}(DE) \subset KS_{l,m}(DE)) \vee (KS_{i,j}(DE) \supset KS_{l,m}(DE))$
- 相交型冲突:

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

UCKA的更新日志中主要包含两个数据结构:(1) 副本索引表 $Rlist(p)$。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$\langle DE, S(p)\rangle$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的集合。(2) 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,并记录在该表中,同时将从其他副本接收的关键更新信息根据更新冲突解决算法写入该表,$Kupda(p, Re_s)$是$\langle DE, Update_{i,j}(DE), T_{i,j}, P_{i,j}, UP_{i,j}, Tag_{i,j}\rangle$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。

2 优化数据一致性算法——UCKA

UCKA采用"最新写胜出"(LWW)和"分而治之"(DAR)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(组)关键属性上的更新 μ_1 和 μ_2,其更新发布时间 $T(\mu_1) < T(\mu_2)$。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 μ_2 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。以上节中的更新 u_2 和 u_1 为例,则直观上更新将被分解为 $u'_1\{k_1, k_2\}$、$u'_2\{k_3\}$和

$u'_3\{k_4,k_5\}$三个更新，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。

UCKA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。

• 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。图1给出了相应的算法。

```
Procedure ProcessKupda(Peer p, Resource Re_s, UpdateList L(DE))
  for each Update_{l,m} ∈ L(DE)                          //从"缓冲预处理算法 ProcessBuffer"中提取的缓冲中的关键更新信息
    tag = 0; E_ks ← ∅; N_ks ← ∅; G_ks ← ∅;
    for each Update_{i,j} ∈ Kupda(p, Re_s)               // 将缓冲中的更新与关键更新表中的更新逐一进行冲突检测
      if (KS_{l,m} ∩ KS_{i,j} ≠ ∅) then tag = 1;         // 关键更新冲突，接下来判断冲突类型
      if KS_{i,j} = KS_{l,m}                             // 全等型冲突
        then if T_{l,m} < T_{i,j} then break;            // 根据 LWW 规则，如果 buffer 中更新的发布时间更早，则 buffer 中的更新可以被忽略
          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}; break;   // 合并传播路径
          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); break;   //向关键更新表中增加较新的更新
      if KS_{i,j} ⊃ KS_{l,m}                             // 覆盖型冲突
        then if T_{l,m} < T_{i,j} then break;            //根据 LWW 规则，buffer 中的更新可以被忽略
          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}|;   // E_ks 记录关键更新表中胜出的更新所包含的关键属性
          else N_ks ← N_ks ∪ |KS_{i,j}|;                 // N_ks 记录 buffer 中胜出的更新所包含的关键属性
            DeleteKupda(p, Re_s, Update_{i,j});          // 关键更新表中的更新较早，从关键更新表中删除
              // 增加较新的更新，其中 Update_{l,m}(KS_{i,j}) = {(I_{l,m,n}, U_{l,m,n}) | (I_{l,m,n}, U_{l,m,n}) ∈ Update_{l,m} ∧ I_{l,m,n} ∈ KS_{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属性，即这些属性不包含于关键更新表中的任一更新
          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; N_ks ← N_ks ∪ |KS_{i,j} ∩ KS_{l,m}|;
            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 then AddKupda(p, Re_s, D_{s,d}, Update_{l,m}, T_{l,m}, P_{l,m} ∪ |p|, UP_{l,m}, 1);   // 与关键更新表中的项目完全不冲突的更新
    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})
```

图1 UCKA算法的关键更新冲突与解决

Fig.1 Detecting and reconciling key-updates conflicts in UCKA algorithm

3 模拟测试

已有的一致性维护方法中，最常用的是反熵方法(anti-entropy)[4]。根据会话结点的选择方式，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等，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)[5]与UCKA进行性能对比。

基于OptorSim[6]模拟器进行性能测试。模拟了1000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。在性能测试中模拟了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，如表1所示。

表 1 外部更新请求

Tab.1 The schedule of updates

更新种类	关键属性数量	概率
1	1~5	25%
2	5~10	40%
3	10~20	25%
4	20~30	6%
5	30~40	3%
6	40~50	1%

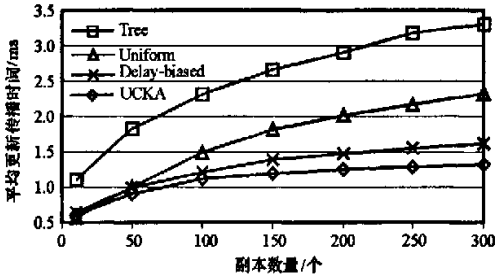

图 2 副本数量固定情况的一致性开销对比

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

(1) 副本数量固定情况下的一致性开销

模拟该情况下 UCKA 和其他三种方法的一致性维护开销(图 2)。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。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 5000 个更新,模拟运行结束条件为完成 5000 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。实验结果表明,树形结构方法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;UCKA 由于每次都选择通讯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。

(2) 副本数量动态变化情况下的一致性开销

模拟在该情况下 UCKA 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;同时比较增加或撤消副本时采用多播和不采用多播通知其他副本对性能的影响(图 3)。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消副本的复制策略。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 5000 个更新之外,还另外提交 10 000 次随机访问请求。实验结果表明,多播通知副本存在状态的变化对 UCKA 带来了性能上的提高,因而尽快地获取其他副本的存在信息,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,采用多播对性能只有微弱的影响。

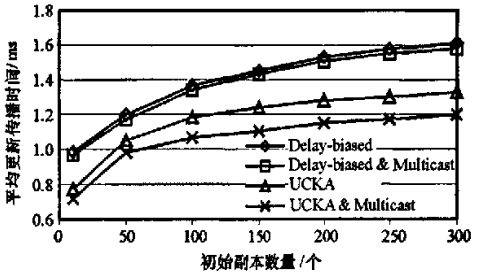

图 3 副本数量动态变化情况下的一致性开销对比

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

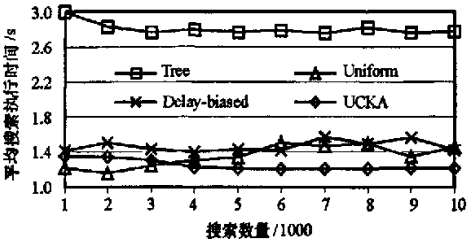

图 4 搜索延迟对比

Fig.4 Comparisons of resource location overhead

(3) 资源定位开销

采用二路随机转发方法(two-way random forwarding)[7]作为资源定位方法,模拟比较 UCKA 与其他几种方法资源定位的时间开销(图 4)。二路随机转发方法中,初始搜索结点产生两个搜索消息,发送给随机选择的两个邻居,然后每个消息在系统中独立地进行随机转发,并周期性地同初始结点联系以决定是否继续转发。一般说来,与泛洪方法相比,随机搜索方法产生的消息开销较少。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 5000 个更新并同时进行 10 000 次搜索,每隔 1000 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)。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发出,并随机选择某类资源中的

10个关键属性进行搜索。实验结果表明,UCKA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。

(4) 资源访问开销

在上面实验(3)的基础上,模拟测试资源访问开销(图5)。实验结果表明,UCKA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,系统新创建了数据副本,副本数量的增加也使得访问开销均有小幅度的降低。

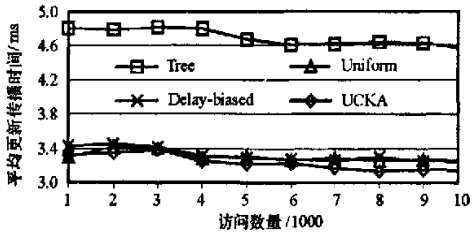

图5 资源访问开销对比

Fig.5 Comparisons of resource access overhead

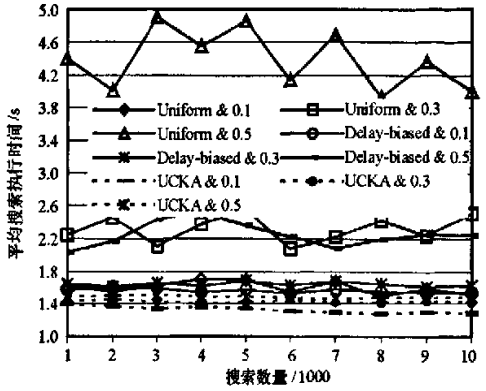

图6 鲁棒性对比

Fig.6 Comparisons of robustness

(5) 鲁棒性

模拟测试UCKA与均匀方法和基于延迟的方法在不同的结点失效率下的资源定位性能(图6)。模拟初始化时,每个结点存储随机选择的100个数据对象。在每组模拟实验中,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;数据获取方法同(3)。实验结果表明,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,对结点状态并不关心;UCKA根据通讯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

优化数据一致性算法UCKA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。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,并且关键属性更新的优化处理适用面向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。

参考文献:

[1] Leontiadis E, Dimakopoulos V V, Pitoura E. Creating and Maintaining Replicas in Unstructured Peer-to-peer Systems[C]//Proceedings of the 12th International Euro-Par Conference on Parallel Processing (EURO-PAR), Dresden, Germany, August 2006: 1015 - 1025.

[2] Yu H, Vahdat A. Consistent and Automatic Replica Regeneration[J]. ACM Transactions on Storage, February 2005, 1(1): 3 - 37.

[3] Vecchio D D, Son S H. Flexible Update Management in Peer-to-peer Database Systems[C]//Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS 05), Washington: IEEE Computer Society, 2005: 435 - 444.

[4] Petersen K, Spreitzer M J, Terry D B. Flexible Update Propagation for Weakly Consistent Replication[C]//16th ACM Symposium on Operating Systems Principles, New York: ACM Press, October 1997: 288 - 301.

[5] Golding R A. Modeling Replica Divergence in a Weak-consistency Protocol for Global-scale Distributed Data Bases[R]. University of California at Santa Cruz, Technical Report: UCSC-CRL-93-09, February 1993.

[6] Bell W H, Cameron D G, Capozza L, et al. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies[J]. International Journal of High Performance Computing Applications, 2003, 17(4): 403 - 416.

[7] Lv Q, Cao P, Cohen E, et al. Search and Replication in Unstructured Peer-to-peer Networks[C]//Proc. of 16th ACM International Conference on Supercomputing (ICS'02), New York, USA, June 2002.

一种面向关键属性更新的优化数据一致性算法

作者： 周婧， 王意洁， 李思昆， 王元志， ZHOU Jing， WANG Yi-jie， LI Si-kun， WANG Yuan-zhi

作者单位： 周婧,王意洁,李思昆,ZHOU Jing,WANG Yi-jie,LI Si-kun(国防科技大学,计算机学院,湖南,长沙,410073)， 王元志,WANG Yuan-zhi(工程兵指挥学院,江苏,徐州,221004)

刊名： 国防科技大学学报 ISTIC EI PKU

英文刊名： JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY

年，卷(期)： 2007,29(4)

参考文献(7条)

1. Leontiadis E;Dimakopoulos V V;Pitoura E Creating and Maintaining Replicas in Unstructured Peer-to-peer Systems[外文会议] 2006

2. Yu H;Vahdat A Consistent and Automatic Replica Regeneration 2005(01)

3. Vecchio D D;Son S H Flexible Update Management in Peer-to-peer Database Systems[外文会议] 2005

4. Petersen K;Spreitzer M J;Terry D B Flexible Update Propagation for Weakly Consistent Replication 1997

5. Golding R A Modeling Replica Divergence in a Weak-consistency Protocol for Global-scale Distributed Data Bases[Technical Report:UCSC-CRL-93-09] 1993

6. Bell W H;Cameron D G;Capozza L Optorsim:A Grid Simulator for Studying Dynamic Data Replication Strategies 2003(04)

7. Lv Q;Cao P;Cohen E Search and Replication in Unstructured Peer-to-peer Networks 2002

本文读者也读过(10条)