文章编号: 1673- 9469(2010) 04- 0086- 06

一种多粒度集群数据库并发控制新算法

王大海[1], 贾 玉珍[2], 靳 冰[2]
(1. 新乡职业技术学院, 河南 新乡 453000; 2. 南阳理工学院 软件学院, 河南 南阳 473004)

摘要: 为保证集群系统中全局事务的并发正确执行, 采用基于谓词级的多级粒度冲突检测机制, 并通过检测谓词冲突图中是否存在环的方法来避免冲突的全局事务可能会产生的全局死锁, 不仅减小了死锁检测粒度, 而且还提高了全局事务处理的并发度, 同时也保持了集群中局部数据库的自治性。此外, 还改进了一种以事务提交图为中心的并发事务调度算法来保证全局事务的可串行化提交, 实验结果表明, 该算法有效地提高了全局事务执行的并发度, 增加了事务吞吐率和减少了响应时间。

关键词: 集群系统; 多级粒度冲突检测; 死锁
中图分类号: TP391 文献标识码: A

A new cluster algorithm for DB parallel control

WANG Da-hai[1], JIA Yu-zhen[2], JIN Bing[2]
(1. XinXiang Vocational and Technical Collage, He' nan Xinxiang 453000, China; 2. Software School, Nanyang Institute of Technology, He' nan Nanyang 473004, China)

Abstract: In order to realize the large scale and high parallel performance of database cluster system, the techniques of cluster are applied to the database system, and a common middleware system with high parallelism is proposed based on share- nothing database cluster, which provides the architecture of single system image for the clients and realizes the collaboration and parallel execution in the database cluster by using the techniques of the meta data management, the multi- thread mechanism and the parallel transaction pre- processing, and it is well suitable for the high performance requirement of the OLTP commercial application and has an ideal price/performance ratio. The database cluster system not only keeps the autonomy of the local database sites but also improves the parallel performance of the database cluster system and solves the performance bottleneck of large database system.

Key words: database cluster system; share- nothing database cluster; deadlock

数据库集群系统以集群技术与数据库系统相结合, 是一组完整的、自治的计算处理单元, 每个节点均有独自的 CPU、内存以及磁盘等硬件资源, 运行独立的操作系统和自治的数据库系统, 通过高速专用网络或商业通用网络互连, 彼此协同计算, 作为统一的数据库系统提供并行事务处理服务[1]。

近年来, 集群系统中的负载平衡问题受到人们的关注。负载平衡包含许多因素, 例如系统结构[2]、算法[3]、资源管理[4]和数据分布[5]以及负载应用类型[3]等都会影响系统的性能平衡。

为了保证全局事务执行的正确性和一致性, 本文在研究数据库集群中的事务并发控制方法的基础上, 提出了一种多粒度的冲突判断和死锁检测方法, 使事务并发控制的粒度达到谓词级, 并讨论各种谓词的提取方法, 既提高全局事务处理效率, 又不需要对局部数据库做任何限制。本文还改进了一种以事务提交图为中心的并发事务调度算法来保证集群系统中全局事务的可串行化提交, 以增加吞吐率和减少响应时间。

收稿日期: 2010- 09- 13
作者简介: 王大海(1981-), 男, 河南濮阳人, 助教, 硕士, 从事信息管理与信息安全、图像处理与模式识别的研究。

1 多粒度集群数据库并发控制算法

1.1 集群系统中的事物模型

数据库集群系统中的事务分为两种: 全局事务和局部事务, 我们把只在一个站点上执行的事务称为局部事务或本地事务[6]。

定义 1: 一个提交到某个数据库站点 j 上的执行的事务Li 是局部事务, 当且仅当 $O_i = D^j$, $O_i = R_i \cup W_i$, $R = \{R(x) \mid R(x) \in L_i\}$, $W_i = \{W(x) \mid W(x) \in L_i\}$, $D^j \in \{D^1, D^2, \cdots, D^n\}$。其中 D^j 表示局部站点j 上的所有数据集合, O_i 表示事务L_i 的所有读、写操作的数据对象的集合, 而 $R(x)$ 和 $W(x)$ 则分别表示对数据项 x 的读操作和写操作。

全局事务是需要在多个站点上执行的事务。

定义 2: 一个事务是全局事务 G_i, 当且仅当 (任意 $D^j \in \{D^1, D^2, \cdots, D^n\})(O_i \in D^j)$, $O_i = R_i \cup W_i$, $R_i = \{R(x) \mid R(x) \in G_i\}$, $W_i = \{W(x) \mid W(x) \in G_i\}$。该定义表示全局事务操作的读写集所需访问的数据不仅仅只包含于一个站点而是跨多个站点的。

由以上定义可知, 局部事务仅在集群中一个自治的数据库站点上执行, 而全局事务需要在集群系统中被分解为多个子事务然后发送到多个数据库站点上执行。本文把全局事务划分为对应站点上执行的子事务称为全局子事务, 每个子事务仅对应一个数据库站点操作, 因此全局子事务也可以看作是仅在一个数据库站点上执行的局部事务。

由于集群中各数据库站点的自治性和局部性, 在每个局部站点的数据库系统上既可以执行由全局事务划分的全局子事务又可以执行不需划分的局部事务, 虽然局部事务的一致性可以在单站点数据库上得到保证, 而作为全局事务所划分的子事务在对应的多个数据库站点上执行时, 其全局事务的原子性和一致性难以得到保证, 因此这里将主要讨论集群系统中跨多个站点执行的并发全局事务间的一致性和正确性。

1.2 多粒度的并发控制

数据库集群系统中并发控制的核心部件是事务管理器, 如何提高事务管理器中事务执行的并发度, 提高处理效率成为研究的主要问题。集群系统中的事务管理器在调度全局事务的子事务在集群系统中的多个站点上执行时, 由于资源竞争会产生冲突, 这些全局事务间的冲突主要可分为直接冲突和间接冲突。全局事务间的冲突归根结底是由两个或两个以上的事务在相同站点上同时访问相同的数据对象而引起的, 因而这些事务间的冲突类型主要有读- 写、写- 写和写- 读冲突三种。

首先, 两个全局事务间的直接冲突[7] 定义如下:

定义 3: 当且仅当下列条件成立时称两操作 p 和 q 是直接冲突的, 记为 pCT q:

1) $p \in T_i, q \in T_i, T_i \neq T_j$。

2) $(\exists x \in S)(p(x) = W(x) \vee q(x) = W(x))$。

3) $\neg \exists (p \in T_i, q \in T_j)((p < q) \vee (q < p))$。其中$<$ 表示先序关系。

由以上定义可知, 两个或多个数据操作产生冲突的条件是: (1) 两个操作属于不同的全局事务; (2) 两个操作访问相同站点上的同一数据项且其中至少有一个为写操作, 否则就不会产生冲突; (3) 两个操作都同时访问数据对象, 亦即两全局事务在时间上同时发生才可能产生冲突。

为了保持局部站点数据库的自治性, 并发全局事务间的冲突检测必须由集群系统中的事务管理器来执行, 而且全局事务间的冲突检测粒度会直接影响事务执行的并发度, 在全局事务的并发控制中如果采用传统的元组级封锁, 虽然可以实现细粒度的并发控制, 但其缺点是开销代价太高。本文在事务管理器中采用谓词技术而不需封锁元组来检测全局事务间的直接或间接冲突, 以实现多级粒度的冲突检测机制, 同时也进一步减小了死锁检测粒度, 提高了全局事务处理的并发度。

谓词的概念及提取技术一: 为了进行全局事务间冲突的判断, 首先必须要分析出全局事务所要操作的数据粒度, 而采用谓词方法就是一种表示全局事务多级粒度操作对象的有效手段, 谓词是指全局事务的 SQL 语句中 DML 四种操作所带的 where 分词条件, 这种分词条件一般可分为谓词常项(表示具体性质和关系的词) 和谓词变项(表示抽象的或泛指的谓词)[3- 4]。

根据谓词提取及转化的定义和规则, 具体的 where 谓词条件分解过程如分解算法 1 所示。

分解算法 1:

1) if 谓词 pred 条件的右边是常量: (1) 获得谓词 pred 左边的属性定义; (2) if 属性定义中存在一个选择子句, 找到选择子句中所涉及的关系, 并将

它加入条件谓词列表中; (3) 调用算法 2, 分解属性定义中的查询子句; (4) else/* 属性定义是一个简单引用* /, 找到简单引用所涉及的关系, 并将它加入条件谓词列表中; end if。

2) else/* 谓词 pred 的右边不是常量* /: (1) 分别得到谓词左右两边的属性定义; (2) if 左右两边的属性定义都是查询, for 右边属性定义的所有选项 r, for 左边属性定义的所有选项 l, 找到 r 和 l 所涉及的关系, if 右边 r 和左边 l 的关系相同加入到谓词列表中, else 发送继续执行, end for;/* 左边属性定义的选择项* /, 调用算法 2, 分解左边属性定义中的查询子句。end for;/* 右边属性定义的选择项* /, 调用算法 2, 分解右边属性定义中的查询子句; (3) else if; 左边的属性定义是简单引用, 右边的属性定义是查询找到左边属性定义中对应的关系, for 右边属性定义的所有选项 r, 找到 r 所对应的关系。if 左右两边关系相同, 加入条件谓词列表中。end for;/* 右边属性定义的选择项* /, 调用算法 2, 分解右边属性定义中的查询子句; (4) else if; 左边的属性定义是查询, 右边的属性定义是简单引用。找到右边属性定义中对应的关系, for 左边属性定义的所有选择项 l, 找到左边 l 所对应的关系。if 左右两边的关系相同加入条件谓词列表中。end for;/* 左边属性定义的选择项* /, 调用算法 2, 分解左边属性定义中的查询子句; (5) else if 左右两边的属性定义都是简单引用。如果关系左右两边属性定义中各自对应的关系相同, 则加入条件谓词列表中; (6) end if;/* 判断左右两边的属性定义是简单引用还是查询* /。

3) end if;/* 判断谓词 pred 的右边是不是常量* /。下面是分解属性定义中的查询子句算法 2。

分解算法 2:

for q 的所有查询条件 cond, 找到 cond 左右两边对应的关系; 如果两边关系相同, 则加入到条件谓词列表中。end for; 在算法 1 的谓词分解流程中, 当谓词条件在左右两边的属性定义都是查询时最复杂, 假设当左右边各有 n 个属性定义时, 算法 1 的时间复杂度为 $O(n^2)$。

谓词的概念及提取技术二: 集群系统中的全局事务在以SQL 语句形式执行时只有对应的 DML 操作可能会产生读- 写、写- 写和写- 读三种冲突, 而 DML 操作主要有 4 种: select、update、insert 和 delete, 如果将 select 等效于读操作, 而 insert、update 和 delete 操作等效于写操作来替代, 冲突矩阵如下表 1。

表 1 SQL 语句冲突矩阵

Tab. 1 SQL statements conflict matrix

	Select	Update	Insert	Delete
Select	Y	N	-	N
Update	N	N	-	N
Insert	-	-	Y	-
Delete	N	N	-	N

在表 1 中, “Y” 表示操作在谓词级上冲突是相容的; “N” 表示操作在谓词级上可能是冲突相容也可能是不相容的, 需要进一步判断是否有谓词交集; “-” 表示插入操作与其他操作在谓词级上无法判断是否冲突 (因为插入操作一般无条件谓词, 只能加表级锁) 进而退化到表级是冲突不相容的, 另外如果允许关系中存在重复元组时, insert 与 insert 操作应该在表级上是冲突相容的。以上直接冲突判断的矩阵防止了数据库中出现的幻象干扰。

全局事务间基于谓词的多粒度直接冲突判断算法流程如图 1 所示, 该过程不需对具体的冲突对象进行繁琐的上锁操作从而减少了系统开销, 因此可以提高事务处理的并发度, 使全局事务间直接冲突判断的粒度能精确到谓词级。此外, 为了方便处理, 当 SQL 语句中没有分词条件时, 例如 insert 语句或不带条件的 delete、select 和 update 语句亦即没有 where 条件和 having 条件时, 谓词的提取粒度就直接限定在表级; 如果谓词级条件不在相同属性上并且也没有定义不同属性列间的相关匹配情况, 就无法判断元组是否有交集, 因此也不能缩小检测粒度, 只能在表级判断是否冲突。

两个全局事务间的直接冲突虽然通过以上流程实现了多粒度的检测, 但对两个以上的并发全局事务间的间接冲突是由两两全局事务间的直接冲突构成的, 因此, 全局事务间的间接冲突检测比较复杂, 首先需要在谓词冲突图中建立全局事务两两之间的直接冲突关系, 才能进一步检测多个并发的全局事务间是否存在间接冲突, 若谓词冲突图中存在一条从一个全局事务 G_i 到另一个全局事务 G_j 间的路径, G_i 和 G_j 存在间接冲突。此外, 谓词冲突图还可检测多个并发全局事务间产生的死锁。

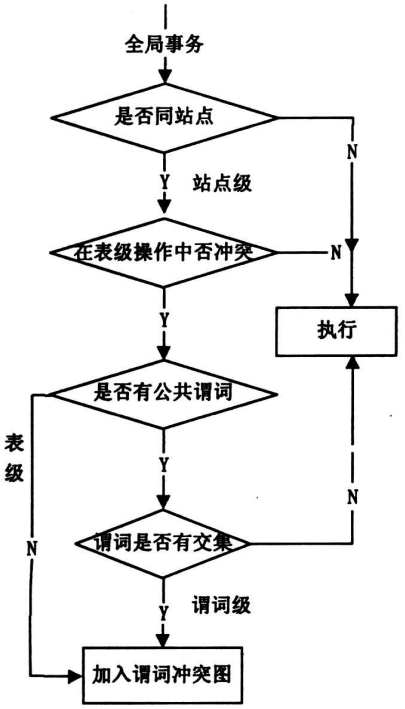

图1 多粒度的直接冲突判断流程

Fig. 1 The multi-granularity direct conflict judgment process

谓词冲突中的死锁检测: 传统的解决死锁问题的方法有死锁预防和死锁检测。死锁预防要求用户进程事先申报所需的资源或按严格的规程申请资源, 而死锁检测原则上应允许死锁发生, 在适当的时机检查, 若发生死锁, 则设法排除之。预防死锁与检测死锁相比, 前者过于保守, 导致全局事务的并发度不高。

多个并发的全局事务可能因为直接冲突或间接冲突而形成死锁, 借助谓词冲突图中全局事务间的冲突依赖关系需要进一步检测全局事务是否形成死锁。

定理 1: 全局事务间形成死锁, 当且仅当谓词冲突图中存在环。

证明: (必要性) 如果谓词冲突图中存在环, 则全局事务会产生死锁。谓词冲突图是一个有向图, 图中从 i 指向j 的边即$i \rightarrow j$, 表明j 被阻塞了并且等待 i 释放冲突站点上的某些资源。当存在环时表示存在一条有向路径并且第一个节点和最后一个节点是重合的, 亦即 $i \rightarrow j \ldots k \rightarrow i$, 则表明全局事务 i 与自己产生了间接冲突形成了一个资源等待环, 因此就产生死锁。

(充分性) 当全局事务间产生死锁, 则谓词冲突图中至少存在一个有向环。当全局事务发生死锁时, 至少存在两个或以上的事务间彼此等待对方所持有的某些资源, 分两种情况讨论: 当两个全局事务 i 和j 间产生死锁时, 在谓词冲突图中既存在 $i \rightarrow j$ 的有向边也存在$j \rightarrow i$ 的有向边表示, 因此在图中形成了有向环; 当两个以上的全局事务间产生死锁时, 例如全局事务 i 与k 之间产生间接冲突时, 表明 k 间接等待i 释放某些资源, 则在谓词冲突图中存在 $i \rightarrow j \ldots \rightarrow k$ 的有向边, 另由于 i 也在等待k 所持有的某些资源, 则在图中存在 $k \rightarrow i$, 因此综合可得 $i \rightarrow j \ldots \rightarrow k \rightarrow i$, 图中存在一条有向环。证毕。

全局事务在谓词冲突图中形成的环可分为两类: 两个全局事务由直接冲突形成的环和两个以上全局事务由间接冲突形成的环。对于一个全局事务只在一个站点上而另一全局事务却是跨站点执行情况, 即如图 2 所示。

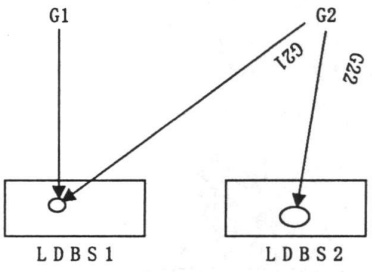

图2 跨站点的全局事务冲突

Fig. 2 Cross-site global transaction conflicts

在图 2 所示的情况下两个全局事务不具备成环条件, 它们在站点 1 上产生的冲突可由该站点数据库处理。

两个全局事务由直接冲突产生死锁条件: (1) 至少存在两对或以上的直接冲突。(2) 必须存在两个不同时序关系的冲突对, 这样才可能在谓词冲突图中形成环。此外, 两个全局事务间产生的冲突环可分为在同一站点和不在同一站点上两种情况, 如图 3 所示。

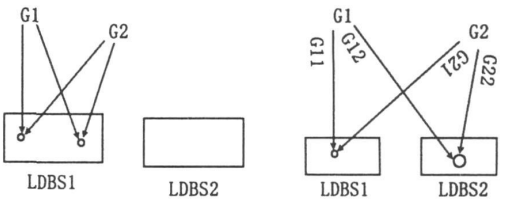

图3 两个全局事务产生环的可能情况

Fig. 3 Two global affairs may produce ring

对于全局事务在同一站点上可能形成环的冲突图, 如图4左分图不必由集群系统中的事务管理器处理, 因为全局事务的子事务都是局部化到一个站点上执行的, 而本地站点上的DBMS有相应的并发控制机制。由于本地站点的自治性而无法知道其他站点全局事务间的关系, 所以事务管理器需要处理的是在跨站点的全局事务产生的环, 而这种环可能是两个全局事务由直接冲突产生的如图4右分图, 也可能是由下面将要提到的间接冲突产生的。对于两个以上的全局事务由间接冲突产生的环, 需要检测谓词冲突图中的每个全局事务 G 是否通过间接冲突与自己冲突即 $G \xrightarrow{} \circ \xrightarrow{} G$, 那么在两个以上的全局事务间存在间接死锁, 如图5所示。图5对应的间接谓词冲突如图6。

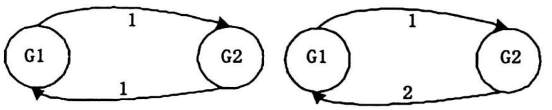

图4 两个全局事务由直接冲突产生的谓词冲突图

Fig. 4 Two global transactions generated by the direct conflict predicate conflict graph

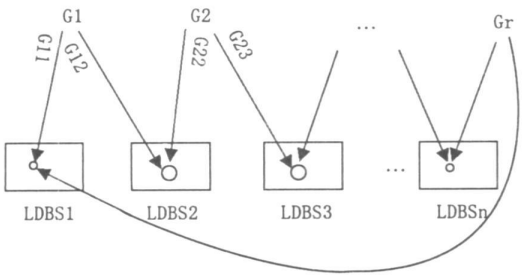

图5 全局事务间的间接冲突产生的环

Fig. 5 Indirect conflict between global transactions generated ring

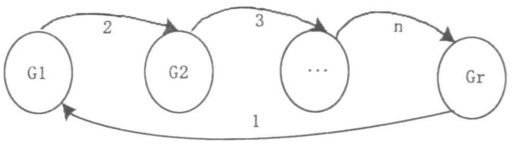

图6 全局事务间的间接谓词冲突图

Fig. 6 Global transactions generated by the indirect conflict predicate conflict grap

集群系统中的事务管理器只处理跨站点的全局事务形成的死锁, 当在谓词冲突图中检测到冲突的全局事务形成环(包括本身形成的间接环如 $G \xrightarrow{} \ldots \xrightarrow{} G$ 和两个全局事务形成的直接环如 $G1 \xrightarrow{} G2$ 和 $G2 \xrightarrow{} G1$)时, 就需要立即解除死锁的发生, 这里事务管理器采取回滚(rollback)最近加入环的一个全局事务, 保证资源利用的最大化。如果事务管理器调度的冲突全局事务在谓词冲突图中没有形成环或无冲突时, 就可以继续发送到底层集群数据库站点上执行。

2 试验分析

集群系统中的事务管理器所采用的基于谓词的多粒度冲突检测机制不需要对底层数据库上的数据进行上锁操作, 保证了局部数据库站点的自治性, 从而减少了系统开销, 提高了全局事务处理的并行度。谓词冲突图也有效防止了全局事务形成的死锁, 其控制粒度也比事务等待图更精确, 更灵活。另外, 事务等待图只适合单站点数据库的死锁检测, 而谓词冲突图在基于数据库集群的多站点间事务全局死锁检测中性能表现优良, 两者之间的性能比较如图7所示。

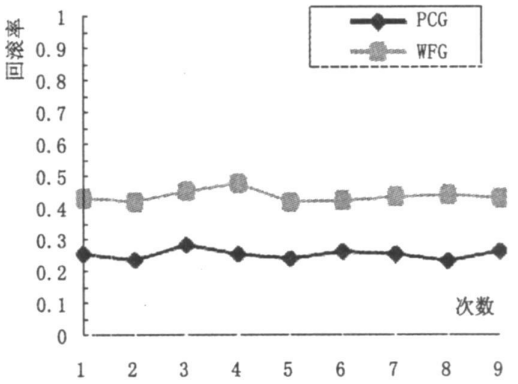

图7 PCG与WFG的回滚率对比

Fig. 7 PCG's rollback rate compared with the WFG

图7中的模拟结果表明, 当采用两种不同的方式检测到死锁时, 都采用回滚最近加入图中的事务方法来解除死锁, 因此死锁的效率体现在事务的回滚率上, 事务间发生的死锁越多, 回滚率也就越高, 事务间的并发度越低。由于两种方式检测死锁的粒度不同从而导致事务的回滚率也不同, 从图中可以看出事务等待图的执行效率比谓词冲突图低, 多次试验表明事务等待图的事务回滚率都在40%以上, 而谓词冲突图的回滚率较低, 由此导致运行相同时间内, 采用谓词冲突图成功执行和提交的事务数量较多。此外, 本文在集群系统的事务管理器中实现了三级粒度的并发控制: 站点级 $\xrightarrow{}$ 表级 $\xrightarrow{}$ 谓词级, 并对比了只到表级具

有两级粒度的并发控制性能。试验结果对比可知, 表级并发控制的粒度相对于谓词级的大, 体现在事务的回滚率上表级粒度平均在 44. 11%, 而谓词级则平均为 26. 01%。由此可见, 在运行时间相同的条件下, 后者(谓词级粒度) 成功执行和提交的事务数量较多, 所以谓词级多粒度控制的事务并发控制效率比表级有了较大的提高。因此, 基于谓词级的多粒度事务并发控制方法有效提高了全局事务执行的并行度。

3 结束语

集群中各局部站点数据库的自治性和局部性只能对本地站点上的事务并发控制, 而无法保证并发全局事务执行的一致性和正确性, 无法防止全局事务间的冲突和死锁发生, 因此在集群系统的事务管理器中实现了全局事务间冲突检测的多级粒度依次为: 站点级→表级→谓词级。另外, 还通过检测谓词冲突图中是否存在环的方法来避免冲突的全局事务可能会产生的全局死锁。该并发控制方法不需具体的上锁操作使并发控制粒度达到谓词级, 不仅减小了死锁检测粒度而且还提高了全局事务处理的并发度, 同时也不需要对局部数据库做任何限制。

(上接第 73 页)

[2] 何成连, 王正伟, 丘 华. 水轮机尾水管内部压力脉动试验研究[J]. 机械工业学报, 2002, 38(11): 62- 65.

[3] 毛汉领, 熊焕庭, 沈炜良. 偏相干分析在水电站振动传递路径识别的应用[J]. 广西大学学报(自然科学版), 1998, 23(1): 6- 9.

[4] HUI W, QINGYU P, ZHANG ZHI C, et al. Vibration analysis of elastic plate submerged in incompressible viscous fluid by copuling finite element method [J]. ACTA Mechanical Solida Sinica, 1998(11): 1- 12.

[5] 刘玉民, 孙开朗, 张 帆, 等. 水轮机发电机组动力稳定性研究[J]. 哈尔滨工业大学学报, 1998, 30(增刊): 82- 84.

[6] 赵林明. 人工神经网络的虚拟输入方法及其在水轮机建模中的应用[J]. 华北水利水电学报, 1995, 16(2): 62- 66.

[7] 陈予恕. 非线性振动[M]. 北京: 高等教育出版社, 2002.

[8] 赵林明, 吕为亮, 张贵棉. 考虑水力振动特性求取转桨式水轮机的协联关系[J]. 河北工程大学学报(自然科学版), 2007, 24(4): 74- 75.

参考文献:

[1] THAKKAR S S, SWEIGER M. Performance of an OLTP application on symmetry multiprocessor system[C]. American: IEEE Computer Society, 1990, 228- 238.

[2] NISHIKAWA H, STEENKISE P. A general architecture for load balancing in a distributed- memory environment[C]. American: IEEE Computer Society, 1993, 47- 54.

[3] DU X, ZHANG X. Coordinating parallel processes on networks of workstations [J]. Journal of Parallel and Distributed Computing, 1997, 46(2): 125- 135.

[4] LEE J L, SCHEAUERMANN P, VINGRALEK R. File assignment in parallel I/O systems with minimal variance of service time [J]. IEEE Transactions on Computers, 2000, 49(2): 127- 140.

[5] LITWIN W, NEIMAT M A, SCHNEIDER D A. LH* - a scalable, distributed data structure[J]. T ODS, 1996, 21(4): 480- 525.

[6] KEN BARKER, TAMER M OZSU. Concurrent transaction execution in multidatabase systems [C]. American: IEEE Computer Society, 1990, 282- 288.

[7] ZHANG A, ELMAGARMID A K. On global transaction scheduling criteria in multidatabase systems[C]. American: IEEE Compurter Society, 1993, 117- 124.

(责任编辑　刘存英)

[9] 杨江天, 陈家骥. 时间序列关联维数在非线性系统运动性态识别中的应用[J]. 航天学报, 2003, 24(1): 28- 30.

[10] ANGELO C. Parallel computation of the correlation dimension from a time series [J]. Parallel Computing, 1999, 25(6): 639- 666.

[11] 潘罗平. 水轮机压力脉动试验方法的研究[J]. 水力发电学报, 2003(3): 107- 113.

[12] 何成连, 王正伟, 丘 华. 水轮机尾水管内部压力脉动试验研究[J]. 机械工业学报, 2002, 38(11): 62- 65.

[13] WOLF A, SWIFT J B, SWINNEY H L, et al. Determining lyapunov exponents from a time series [J]. Physica D: Nonlinear Phenomena, 1985, 16(3): 285- 317.

[14] 张安兵, 张俊芳, 李喜盼, 等. 煤矿井筒变形混沌特征分析[J]. 河北工程大学学报(自然科学版), 2009, 26(3): 85- 88.

[15] 万书亭, 李和明, 李永刚. 发电机绕组故障时振动的关联维数分析及诊断[J]. 振动、测试与诊断, 2005, 25(3): 210- 213.

(责任编辑　马立)