ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.1, January 2006, pp.31−38
DOI: 10.1360/jos170031

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

一种任务类型敏感的J2EE事务调度算法*

丁晓宁[1,2+], 张 昕[1,2], 金蓓弘[1], 黄 涛[1]

[1](中国科学院 软件研究所 软件工程技术研究中心,北京 100080)

[2](中国科学院 研究生院,北京 100049)

A Task-Type Aware Transaction Scheduling Algorithm in J2EE

DING Xiao-Ning[1,2+], ZHANG Xin[1,2], JIN Bei-Hong[1], HUANG Tao[1]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-82680329, Fax: +86-10-62562580, E-mail: dxn@otcaix.iscas.ac.cn

Ding XN, Zhang X, Jin BH, Huang T. A task-type aware transaction scheduling algorithm in J2EE. *Journal of Software*, 2006,17(1):31−38. http://www.jos.org.cn/1000-9825/17/31.htm

Abstract: J2EE is the primary middleware platform for distributed enterprise applications. Current J2EE (Java 2 platform enterprise edition) transaction scheduling to entity beans is still a simple First-Come First-Served (FCFS) policy. Due to this simple policy, transaction manager is unable to distinguish the key tasks from the trivial ones and the performance of the key tasks is reduced when the server's load is heavy. Since some characteristics of J2EE middleware, the traditional scheduling algorithms can not be applied to J2EE directly. This paper puts forward a scheduling algorithm named TMPBP (threaded multi-queue priority-based protocol). The algorithm includes a heuristic priority assignment algorithm named HRS, which can recognize the key tasks dynamically. TMPBP improves the Quality of Service (QoS) of the key tasks effectively under heavy load. It is safe and has no starvation or priority inversion. The performance experiments indicate that a considerable improvement on QoS of key tasks has been achieved under the proper parameters of the algorithm.

Key words: J2EE (Java 2 platform enterprise edition); transaction scheduling; priority; heuristic

摘 要: J2EE(Java 2 platform enterprise edition)是构建分布式企业应用的基础中间件平台,当前的 J2EE 事务对资源的访问调度仍然是简单的先来先服务策略,导致服务器负载很重时,次要任务和关键任务争夺有限的资源,降低了关键任务的性能与成功率.为此,有必要识别任务类型,在资源不足时优先保证关键任务事务的执行.但提交给 J2EE 的事务基于交互方式执行,且缺乏必要的调度信息,因而不能简单地沿用已有的优先级驱动调度算法.提出一种新的事务调度算法 TMPBP(threaded multi-queue priority-based protocol),该算法能够安全、有效地提高系统重负载情况下关键事务的服务质量,防止饥饿和优先级倒置.TMPBP 包含了一种新的启发式优先级分派算法 HRS(heuristic resource statistics),可以在调度信息缺乏的 J2EE 环境下动态识别关键事务.结果表明,通过合理地选择参数,TMPBP 算法能够显著地提高关键事务的服务质量.

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2001AA113010, 2002AA413610, 2003AA413010, 2003AA115440 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 (国家重点基础研究发展规划(973))

Received 2005-06-27; Accepted 2005-08-03

关键词: J2EE;事务调度;优先级;启发式
中图法分类号: TP311 文献标识码: A

目前,J2EE(Java 2 platform enterprise edition)[1]已经成为支撑大型分布式企业应用的基础中间件平台,事务服务(transaction service)是J2EE平台提供的一项重要服务,也是保证流程完整性和数据一致性的关键技术.

大量J2EE采用基于实体bean(entity bean)[2]的持久化方式.实体Bean本质上是对关系型数据的一种面向对象封装,应用服务器通常采用封锁机制来管理对实体Bean的并发访问,并应用缓存等技术提高数据访问性能.

一个 J2EE 事务实例为完成其工作必须获取一系列资源,对资源的竞争体现为对实体 Bean 的锁竞争.当服务器处于重负载时,频繁的访问冲突造成了一个事务的完成时间很大部分消耗在等待所需资源的锁上.而考察当前 J2EE 平台上事务管理器的实现,各个事务实例在竞争锁时仅采用简单的 FCFS(first come first served)机制,FCFS 的优点是简单、无饥饿;缺点是当服务器处于重负载时,次要任务与关键任务争夺有限的资源,因而严重降低了系统的表现.

在资源紧张的情况下,应优先保证关键任务事务的执行.这就启发我们可以引入优先级驱动的调度算法来区分任务类型,从而集中有限资源为关键事务提供更好的服务质量,即QoS(quality of service).目前,关于事务服务的QoS尚未有统一的定义[3],但一般认为,事务QoS指标应包括响应时间、事务成功率等.本文并不讨论事务QoS的具体执行机制,如QoS规约的指定、协商、反馈等,仅定义基本的QoS指标来量化衡量调度算法的优劣.

在实时系统和数据库等领域,已经提出很多种成熟的优先级驱动调度算法,然而,在 J2EE 中间件中提交的任务为交互式非实时任务时,缺乏必要的调度信息,如任务的预期执行时间、任务的价值等,使得这些传统算法无法适用或效果不佳.

为解决上述问题,本文根据 J2EE 环境特点,提出一种事务调度算法,称为 TMPBP(threaded multi-queue priority-based protocol),该算法能够高效、安全地分配系统资源,在重负载情况下显著提高关键事务的服务质量.TMPBP 基于一种启发式优先级分派算法,称为 HRS(heuristic resource statistics),能够在运行期动态地评估各个事务实例的价值,从而自动地调整事务实例的优先级.

本文第 1 节比较相关研究工作.第 2 节给出事务调度算法 TMPBP 的设计,包括其优先级分派算法 HRS.第 3 节证明 TMPBP 调度算法的若干性质.第 4 节给出调度算法的性能评价与参数调整.最后是全文的结论.

1 相关工作

对优先级驱动的调度算法研究,大多来自实时系统和数据库领域,在中间件领域的研究相对较少.

优先级驱动的调度算法主要有空闲时间最短最优先(least slack first,简称LSF)[4,5]、截止期最早最优先(earliest deadline first,简称EDF)[6,7]、单调速率(rate monotonic,简称RM)[6]、可达截止期最早最优先(earliest feasible deadline first,简称EFDF)[8]、最早放行最优先(earliest release first,简称ERF)[8]、价值最高最优先(highest value first,简称HVF)[9]、价值密度最大最优先(greatest value density first,简称GVDF)[4,9]等算法.

在上述算法中,包括 EFDF,LSF,RM 在内的一些算法,在 J2EE 中间件环境下缺乏必要的调度信息或一些前提不成立,因而无法适用.例如,由于缺乏任务的预期执行时间调度信息,无法计算其空闲时间,使得 LSF 算法无法应用;另一些算法能够应用但存在不足,这类算法包括 EDF,ERF 等.例如,虽然可以根据任务提交时间和超时时间计算出任务截至期来应用 EDF 算法,但 EDF 没有考虑到任务的关键性特征,关键任务在调度中并不能占到优势;HVF 算法认为,每个任务都有一个与时间 t 相关的价值函数,价值最高者最优先.而在 J2EE 环境下,由于缺乏必要的调度信息,无法构造传统的基于紧急度和截止期的价值函数;GVDF 算法也存在同样的问题.

在并发控制策略上,大多使用基本的两阶段封锁方法,一些乐观并发控制策略也被尝试应用,如JBoss[10]实现了数据版本机制.这些乐观并发控制协议适用于低负载和只读事务多的情形,在重负载时,由于访问冲突频繁,在事务提交的验证阶段失败率较高,性能严重下降.

优先级驱动的调度算法往往会带来一些问题.优先级倒置(priority inversion)[11]违背了优先级驱动调度的

初衷,造成了不公平的现象.优先级继承(priority inherence,简称PI)[12]是广泛使用的防止优先级倒置的方法.PI基于一种朴素的思想:当一个事务实例阻塞了更高级别事务实例的执行时,将该实例的优先级调整为所有被阻塞的实例中最大的优先级.此外,优先级驱动的调度算法可能产生饥饿,目前,对此问题的解决方法大多是重启事务并限制重启的次数[11].然而在J2EE中,事务以交互方式执行,事务管理器不了解应用程序的具体操作语义,不能直接重新启动事务和重做所有的操作.

在死锁方面,一些并发控制协议能够破坏死锁的一些必要条件,从而是无死锁(deadlock-free)的.例如,Data-Priority-Based Locking Protocol(DP)[11]是一种基于数据元素优先级的封锁协议.但DP假设了事务实例到达时即向调度器提交其待访问的数据元素列表,如上文所提,这一点在J2EE中是不成立的.这一类无死锁的协议还包括Priority Ceiling Protocol[12,13]等,但这些协议在运行期都严重降低了执行效率.传统的方法不能预防死锁,而是在等待图[14]中检测到死锁后,回滚其中至少一个实例,应用得更加广泛也更加实际.我们的算法中应用了这种事后检测方案,并将环中优先级最低的事务实例选为牺牲者.

2 算法描述

2.1 约 定

为便于后文的讨论和分析,现定义如下记号与概念:

定义 1(资源表示). 所有的实体Bean记为集合BS.每个实体Bean有权值属性,记为$wt(bean)$,它代表用户对该实体Bean资源重要性的评估,$wt(bean)\in N^+$,其中N^+为正整数.系统中所有权值构成一个有限集合,记为WT,最大权值记为$\max(WT)$.

定义 2(事务表示). 事务实例α的到达时刻记为$at(\alpha)$,完成时刻为$et(\alpha)$.每个事务实例有优先级属性,事务实例α在t时刻的优先级记为$priority(\alpha,t)$,$priority(\alpha,t)\in N^+$,其访问过的资源集合记为$rh(\alpha,t)$,$rh(\alpha,t)\subseteq BS$.

定义 3(事务 QoS 指标). 设给定时间内共到达 M 个事务,这 M 个事务中最后成功提交 N 个事务(分别记为1,2,…,n),因超时而回滚了 K 个事务.事务的 QoS 指标定义如下:

(1) 平均完成时间 ACT,$ACT=\left(\sum_{i=1}^{n}(et(i)-at(i))\right)\Big/N$.

(2) 加权平均完成时间 $WACT$,$WACT=\left(\sum_{i=1}^{n}((et(i)-at(i))\times priority(i,et(i)))\right)\Big/\sum_{i=1}^{n}priority(i,et(i))$.

(3) 截止期错失率 MDP,$MDP=(K/M)\times 100\%$.

定义 4(阻塞者). 设事务实例α等待获取某个资源上某种类型的锁,而事务实例β持有该资源上与该类型相冲突的锁,则称实例β为实例α的阻塞者(blocker).一个事务实例可能有多个 Blocker,每个 Blocker 可能又有它自己的 Blocker.用节点表示一个事务实例,节点间的有向边表示阻塞关系,那么,所有的节点形成一个有向无环图(directed acyclic graph,简称 DAG).从该节点开始,沿着有向边经过的一个节点序列称为该事务的一个 Blocker 链.

2.2 启发式优先级分派HRS

J2EE 中间件中提交的任务缺乏必要的调度信息,我们也不能期待用户为所有的任务都指定一个静态事务优先级,因为任务在执行期间其价值会发生变化,同时,很多已有的遗留代码都没有指定优先级.

困难之处在于,如何在缺少显式信息的情况下确定事务的价值或类型.特定类型的事务为完成任务需要访问特定类型的资源,在我们对自主研发的J2EE应用服务器OnceAS[15]的性能研究中,我们也观察到了事务类型与其访问的资源类型有着明显的相关性.这就暗示我们可以从事务资源访问历史中获取事务类型信息.HRS算法的基本思想是:通过给每个资源指定一个权值,根据事务实例的资源访问历史,启发式地估算该实例对系统的价值.此外,事务优先级也应当允许用户在明确知道任务类型时手工静态指定.我们综合了动态和静态两方面的需求,提出了如下启发式优先级分派算法HRS.

定义 5(HRS). 按照下列公式计算事务实例α在 t 时刻的优先级 $priority(\alpha,t)$:

$$priority(\alpha,t)=StaticPriority(\alpha,t)+AgePriority(\alpha,t)+ResourcePriority(\alpha,t).$$

其中,*StaticPriority* 为该实例的静态优先级,是选自某个固定取值范围的非负整数,如[0,1000].该值在创建时是可选的,默认值为 0.通常,*StaticPriority* 在创建时指定,可以隐式地取决于启动该事务的工作站地址,或用户显式地通过初始化参数来明确指定.

*AgePriority*是时间差的线性函数,标志着事务的年龄.$AgePriority(\alpha,t)=k\times(t-at(\alpha))$,其中,$k$为线性增长系数,$k\in N^+$.运行时间越长的事务,回滚的代价也越大;另一方面,大量事务超时值相同情况下,运行时间越长也就离截止期越近,紧急度越高.许多研究表明[16,17],优先级驱动的调度算法应该综合任务的关键性与截止期两个独立的特征参数,k实际上是对这两个特征参数重要性的权衡.

$ResourcePriority(\alpha,t)$定义为事务实例$\alpha$截止$t$时刻,所访问过的资源集合权值之和(如果一个资源被多次访问,只计算一次).我们在实体Bean描述符文件[2]中加入自定义的条目*BeanWeight*,它表示该Bean的权值,由应用部署人员根据业务系统的具体情况予以指定,初始值默认为 0.

2.3 调度算法

在优先级分派算法HRS基础上,我们提出一种基于优先级和线程标记方法的多级调度算法TMPBP,它是针对 J2EE 特点对基本两阶段加锁算法的改进.在 J2EE 中间件环境下,每个事务实例一定属于一个线程,TPPBP 依赖线程标记技术,事务创建时在线程上下文中定义两个变量,*TimeoutTimes* 和 *LastPriority*,分别记录了回滚次数和最近一次回滚前的优先级.Bean 的等待队列按照 *TimeoutTimes* 值的不同而组织成多个队列,调度时首先选择 *TimeoutTimes* 值最高的等待队列,调度该队列中优先级最高的实例,如果存在两个实例优先级相同,则随机挑选一个.每个 Bean 除权值 $wt(bean)$属性以外,引入 $priority(bean)$属性,它等于阻塞在该 Bean 上的最高事务实例优先级.注意:Bean 的优先级和其权值是不同的,权值在部署时指定,是静态的;而优先级是在运行时动态产生和变化的.在死锁方面,TMPBP 通过一个独立线程来定期检测等待图中的环来检测死锁.当死锁出现时,回滚环中优先级最低的事务实例.

完整的 TMPBP 调度算法的伪码描述如下(其中,事务所属的自身线程记为 *self*,当前时刻记为 *now*):

```
on schedule(tx,bean)://Transaction tx would like to visit bean
    lookup the bean's lock table;
    if (tx is permitted to grant the lock) {//go ahead
      if (bean∉rh(tx, now))
          tx.ResourcePriority=tx.ResourcePriority+wt(bean);
      grant tx the lock;
    }
    else {//blocked, waiting…
        if (tx.priority>bean.priority) {
          bean's priority=tx.priority;
          for all tx's Blocker chains, do //update the Blocker chain
              from the first node to the last node of the chain, do {
                if (node's Blocker's priority<node's priority)
                    node's Blocker's priority=node's priority;
              }
        }
        put tx to the waiting queue according to its timeoutTimes;
    }
on release lock(bean): //pick a tx to run
    select the queue Q with the highest timeoutTimes;
    select instance tx with the highest priority from Q, if there are
    several instances with the same priority, select a random instance from them;
    if (bean∉rh(tx, now))
        tx.ResourcePriority=tx.ResourcePriority+wt(bean);
    grant tx the lock;
```

```
    bean's priority=max priority of instances in bean's waiting queue. //adjust bean's priority
on transaction create( initPriority):
    ResourcePriority=0;
    if (has no timeoutTimes defined in Thread Context) {//init the Thread Context
        self.timeoutTimes=0;
        self.lastPriority=0;
     }
    StaticPriority=initPriority+self.lastPriority;
on transaction commit(tx):
    release tx's all locks;
    self.timeoutTimes=0;
    self.lastPriority=0;
on transaction rollback(tx):
    release tx's all locks;
    if ((reason==TIMEOUT)||(reason==DEADLOCK)) {
       self.timeoutTimes=self.timeoutTimes+1;
       self.lastPriority=priority(tx,now);
     }
    else{
       self.timeoutTimes=0;
       self.lastPriority=0;
    }
on calculate priority(α,t):
    return   StaticPriority(α)+k*(t–at(α))+ResourcePriority(α,t).
```

3 算法分析

在本节中,分析证明 TMPBP 调度算法的各项性质,包括时间复杂度、饥饿、优先级倒置等.

3.1 时间复杂度

从上述 TMPBP 调度算法的伪码描述中可以看出,在 TMPBP 算法中,除 on *schedule*(*tx*,*bean*)节中查找更新锁表、出入等待队列、更新 Blocker 链这 3 个操作以外,其余部分均为线性操作、无循环,故其时间复杂度为 $O(1)$.这 3 个操作的时间复杂度依赖于具体的数据结构设计.在我们的实现系统中,所有 Bean 以 Bean ID Hash 值为索引组成向量,其中每个节点包含该 Bean 的锁表及该 Bean 的等待队列.等待队列组织为双向链表,事务实例在链表内按优先级排序.设系统中并发事务实例数为 n,可以看出,检索和更新锁表的时间复杂度为 $O(n)$;等待队列出队时间为 $O(1)$;入队时需要排序,时间为 $O(n)$.

在Blocker链更新和死锁检测方面,我们使用了JBoss EJB Container中提供的数据结构与算法[10],其中,等待图组织为一个序偶⟨*tx*,*tx's Blocker*⟩的向量,放入新边后,从头扫描,如果某个实例出现两次说明构成了环.可以看出更新Blocker链和检测环的时间复杂度均为$O(n^2)$.

综上分析可知,TMPBP算法的时间复杂度为$O(n^2)$.与FCFS相比,TMPBP带来了一些性能负担,因此,如果应用系统中任务类型彼此相似,或负载程度较小从而冲突不严重,那么应优先选择较简单的FCFS调度.

3.2 调度性质

性质 1. TMPBP 算法产生的调度没有饥饿.

首先我们证明,在不考虑事务超时的情况下,TMPBP 调度算法不存在饥饿.

静态优先级集合和权值集合WT都是一个有限集合,设静态优先级取值区域中所有可能的值为集合$S\{StaticPriority_1,StaticPriority_2,...,StaticPriority_n\}$.取任意一个事务实例$\alpha$,必然存在时刻$t(t>at(\alpha))$,使得:

$$AgePriority(\alpha,t)>\mathrm{Max}(S)+\sum WT \tag{1}$$

同时,设t时刻系统中活跃的事务实例为集合$P\{\alpha,\beta_1,\beta_2,\ldots,\beta_n\}$,显然,$P$也是一个有限集.当时间大于$t$时,由于

*AgePriority*是时间*t*的线性函数,由*priority*(α,*t*)的定义和式(1)可知,事务α的优先级高于系统中所有新产生的事务实例的优先级.

比α优先级高的事务实例都属于集合 *P*,而集合 *P* 是一个大小为 *n*+1 的有限集.所以即使事务实例阻塞在某个资源的等待队列中,最坏情况下,也会在该资源的第 *n*+1 轮调度中获得锁.同理可得,在剩余的执行流程中,不会阻塞在任何一个资源的等待队列中,所以它不会产生饥饿.

以上讨论没有考虑到死锁,如果发生死锁,环中优先级最低的事务实例被强行回滚,释放资源.如果被强行回滚的是其他事务实例,不影响上述结论;如果是本事务实例被回滚,问题等同于下文的超时回滚.

现在我们考虑超时问题.事务可能在 *t* 到来之前就已经超时,从而造成饥饿现象,即客户一次次重启事务,但每次都在 *t* 到来之前就已经超时,被强制回滚,从而无法完成任务.在 TMPBP 中,我们将调度信息与客户的线程上下文绑定,根据线程上下文中的信息来辨认被重启的事务.当超时的事务重新到达系统时,它继承了回滚前的优先级,优先级延续增长,从而避开了超时限制.此时,问题归结为无超时的情况,而上文已证明了无超时情况下不会产生饥饿.

性质 2. TMPBP 调度不存在优先级倒置.

在 TMPBP 调度算法中,Bean 本身的优先级等于它等待队列中所有事务实例的最高优先级.每个事务实例在进入阻塞前计算自身的优先级,如果大于该 Bean 的优先级,那么分别更新 Bean 的优先级和该事务所有 Blocker 链上事务的优先级.通过 Bean 优先级标记机制,保证了沿着该事务实例的任意一个 Blocker 链,其优先级是单调非降的,从而防止了优先级倒置现象.

由于被阻塞的事务优先级随时间不断增长,而其 Blocker 链上各事务的优先级并不随着阻塞事务优先级的改变而实时更新.下面我们证明这一点并不影响调度结果,不会产生优先级倒置现象:

设被阻塞的事务实例为α,其所有 Blocker 链上的节点构成集合 *T*.由于α处于阻塞状态,其事务的 *StaticPriority* 和 *ResourcePriority* 不变,优先级的唯一变化来源是 *AgePriority* 的改变,但 *AgePriority* 是时间 *t* 的线性函数,*T* 中任一事务实例,其 *AgePriority* 也同步增长,且相同时间内增长幅度相同,所以沿任意一个 Blocker 链,单调非降属性仍然成立,不影响调度结果.

4 性能评价与调整

本节通过模拟不同的负载情况,研究了 TMPBP 和现有的 FCFS 两种调度算法下,第 2.1 节定义的几个 QoS 指标的对比,同时探讨了 HRS 算法中参数选值对调度结果的影响.实验环境为 PC 机,硬件为 CPU Pentium4 2.4G、内存 512M;软件环境为 Windows 2000 Professional Service Pack 4.

4.1 调度算法对比

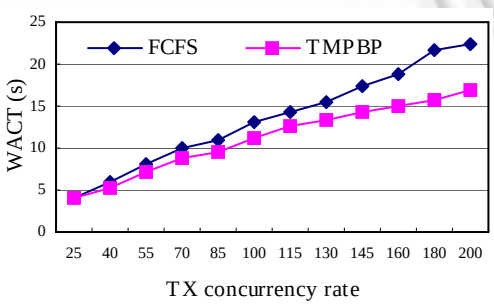


Fig.1 WACT comparison on concurrency

图 1 不同并发数 WACT 对比

我们模拟了不同数目事务实例的并发执行,如图 1 所示.

由于系统繁忙情况下事务往往集中访问一个工作集,我们将资源总数定为 30 个.在这 30 个资源中,前 10 个权值从 30~90 不等,后 20 个权值为 0.每个事务在生命周期内均完成 5 次资源访问,每次在资源上执行 500ms 的操作.由于事务较简单,我们将超时时间缩小为 30s,当一个事务的完成时间超过限制后,我们将其标记为失败.

从图 1 可以看出,随着并发数的增大,两种调度算法的 WACT 差别在逐渐增大.主要由于,随着负载程度的增大,严重的访问冲突造成等待队列较长,事务大部分时间消耗在等待锁上,所以在 TMPBP 调度中关键事务占很大优势.

图 2 和图 3 更清楚地显示了 TMPBP 算法的优势,两幅图分别显示了并发事务数为 200 时,不同优先级下事

务实例的 ACT 对比和 MDP 对比.

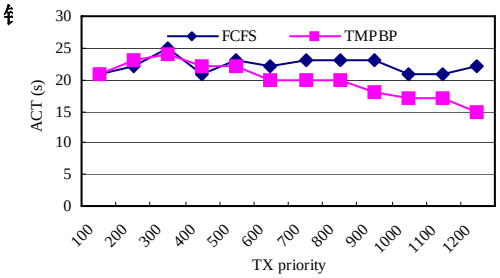


Fig.2 ACT comparison on priority

图 2 不同优先级 ACT 对比

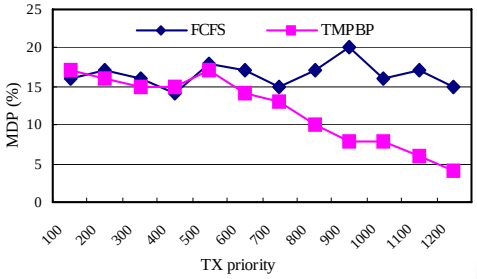


Fig.3 MDP comparison on priority

图 3 不同优先级 MDP 对比

4.2 参数调整

在第 2.2 节中,*AgePriority* 的线性参数 *k* 并没有指定.通常,*k* 应当根据关键事务完成时间长度分布、超时时间分布、等待队列的长度分布等参数而调整,*k* 的选取会显著影响调度结果.

我们分别观察了 *k*=5 和 *k*=500 两种较极端取值情况下,对调度结果的影响,如图 4 和图 5 所示.其余实验参数同上.

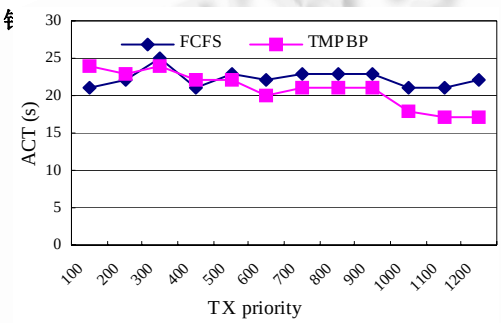


Fig.4 ACT comparison on *k*=5

图 4 *k*=5 时的 ACT 对比

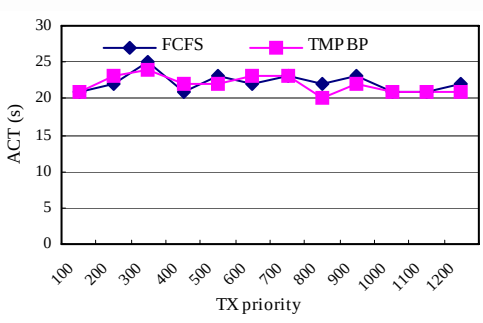


Fig.5 ACT comparison on *k*=500

图 5 *k*=500 时的 ACT 对比

从图 4 和图 5 的对比中可以看出,在 *k* 的两种取值下,TMPBP 调度算法的表现差别很大.图 4 中,由于本例中静态优先级较低,超时时间较短,*k*=5 时偏小,导致 ACT 优势相较 *k*=20 时(如图 2 所示)降低.而图 5 中 *k*=500 时,调度结果与普通的 FCFS 接近,TMPBP 失去了区分任务类型的能力.

上述两图可以解释 *k* 的作用:当 $k\to 0$ 时,HRS 不再考虑事务年龄,适用于超时设置较长、超时可能性较低的环境,它对事务年龄不敏感;反之,当 $k\to\infty$时,由资源访问历史所评估出的任务类型信息重要程度降低,该部分信息被淹没,导致 HRS 退化为年龄最长最优先的调度算法,关键事务不再占优势,所以调度结果与 FCFS 算法相近.

5 结束语

本文研究如何在 J2EE 环境下引入优先级驱动的事务调度算法,提高关键任务的服务质量.我们面临两个问题:(1) 在缺乏显式调度信息的环境下怎样动态发现关键事务;(2) 怎样针对 J2EE 特点,为关键事务提供一种安全、高效的调度算法.

本文的贡献是对上述两个问题的解决:首先,提出了新的优先级分派算法 HRS,它可以在运行期动态识别关键事务.HRS 的基本思想是,给每个资源赋予一个权值,从事务的资源访问历史中启发式地计算任务价值;其次,针对 J2EE 中间件特征,在 TMPBP 中改进了传统的两阶段封锁并发控制方法,例如,调度信息与线程上下文的绑定等.我们证明了调度算法是安全的.同时,仿真实验结果表明,TMPBP 显著地提高了关键事务的服务质量.

References:

[1] Sun Microsystems Inc. Java™ 2 platform enterprise edition (J2EE) specification. 2003.

[2] Sun Microsystems Inc. Enterprise JavaBeans specification. 2003.

[3] Xie WX, Navathe SB, Prasad SK. Supporting QoS-aware transaction in system on a mobile device(SyD). In: Proc. of the 23rd Int'l Conf. on Distributed Computing Systems Workshops (ICDCSW2003). Providence: IEEE Computer Society Press, 2003. 498−502.

[4] Abbott R, Garcia-Molina H. Scheduling real-time transactions: A performance evaluation. ACM SIGMOD Record, 1988, 17(1):71−81.

[5] Dertouzos ML, Mok AK. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. on Software Engineering, 1989, 15(12):1497−1506.

[6] Liu C, Layland J. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 1973,20(1): 46−61.

[7] Haritsa JR, Livny M, Carey MJ. Earliest deadline scheduling for real-time database systems. In: Proc. of the 12th IEEE Real-Time Systems Symp. Los Alamitos: IEEE Computer Society Press, 1991. 232−243. http://citeseer.ist.psu.edu/haritsa91earliest.html

[8] Liu YS, He XG, Tang CJ, Li L. Special Type Database Technology. Beijing: Science Press, 2000 (in Chinese).

[9] Jensen ED, Locke CD, Toduda H. A time-driven scheduling model for real-time operating systems. In: Proc. of the IEEE Real-Time Systems Symp. Washington: IEEE Computer Society Press, 1985. 112−122. http://www.cse.ucsc.edu/~sbrandt/courses/Winter00/290S/jensen.pdf

[10] JBoss Group. JBoss Administration and Development. 2004.

[11] Ulusoy Ö, Belford GG. Real-Time transaction scheduling in database systems. Information Systems, 1993,18(9):559−580.

[12] SHA L, Rajkumar R, Lehoczky J. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. on Computers, 1990,39(9):1175−1185.

[13] SHA L, Rajkumar R, Plehoczky J. Concurrency control for distributed real-time databases. ACM SIGMOD Record, 1988,17(1): 82−98.

[14] Bernstain PA, Hadzalacos V, Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesly, 1987.

[15] Huang T, Chen NJ, Wei J, Zhang WB, Zhang Y. OnceAS/Q: A QoS-enabled Web application server. Journal of Software, 2004,15(12):1787−1799 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1787.htm

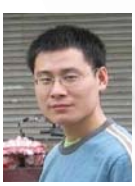
[16] Buttazzo G, Spuri M, Sensini F. Value vs. deadline scheduling in overload conditions. In: Proc. of the 19th IEEE Real-Time Systems Symp. Pisa: IEEE Computer Society Press, 1995. 90−99. http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=495199

[17] Huang JD, Stankovic JA, Towesly D, Ramamritham K. Experimental evaluation of real-time transaction processing. In: Proc. of the 10th Real-Time Systems Symp. Santa Monica: IEEE Computer Society Press, 1989. 144−153. http:// ieeexplore.ieee.org/iel2/268/2318/00063565.pdf?arnumber=63565

附中文参考文献:

[8] 刘云生,何新贵,唐常杰,李霖.特种数据库技术.北京:科学出版社,2000.

[15] 黄涛,陈宁江,魏峻,张文博,张勇.OnceAS/Q:一个面向 QoS 的 Web 应用服务器.软件学报,2004,15(12):1787−1799. http://www.jos.org.cn/1000-9825/15/1787.htm



丁晓宁(1979－),男,安徽庐江人,博士生,主要研究领域为网络分布计算,软件工程.

金蓓弘(1967－),女,博士,副研究员,CCF 高级会员,主要研究领域为网络分布计算,软件工程,数据库实现技术.

张昕(1977－),男,博士生,主要研究领域为网络分布计算,软件工程.

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程,网络分布计算.