一种基于排队系统的启发式中间件动态线程池管理机制

陈宁江 林 盘
(广西大学计算机与电子信息学院 南宁 530004)

摘 要 以Web应用服务器为代表的中间件需要为Internet应用提供有效的运行时性能保障和优化服务。线程池技术是一种常见的性能优化方法。针对Internet应用的特征,中间件的线程池管理需要在感知运行时上下文的基础上进行动态调整,然而如何挖掘有效的影响因素以使调整效果更具有适应性,仍值得深入研究。首先基于M/M/1/K/∞/FCFS排队系统提出了一个应用服务器的动态线程池模型,然后在此基础上研究了引入一组启发因素和规则,这将更有效地反映运行时上下文,实现驱动线程池大小的动态调整过程,使线程池规模适应资源的变化。以上机制通过原型实验验证了启发式因素的变化对线程池规模调整的有效影响,并表明该机制能够有效改善系统性能。

关键词 线程池,中间件,排队系统,启发式

Heuristic and Dynamic Thread Pooling Mechanism Based on Queuing System in Middleware

CHEN Ning-jiang LIN Pan
(College of Computer, Electronics and Information, Guangxi University, Nanning 530004, China)

Abstract Middleware in Internet environment, typically Web Application Server(WAS), needs to provide the performance guarantee and optimization services effectively for Internet applications. Thread pooling technique is a common method of performance optimization. With the consideration of the features of Internet applications, the management of thread pool in middleware needs to adjust dynamically on the basis of perceiving the context at run-time. However, how to find out effective influencing factors which make the adjusting results having better adaptability remains to be discussed further. The paper firstly presented a dynamic thread pool model of application server based on M/M/1/K/∞/FCFS queuing system; then, the paper studied a mechanism which imports heuristic factors and rules for reflecting the context at run-time. It can realize dynamic adjustment of thread pool size more effectively so that thread pool size is well adapted to resources change. The experiments verify the effective influence of heuristic factor that exert on adjustment of thread pool size and show that the presented management mechanism can significantly improve the performance of system.

Keywords Thread pool, Middleware, Queuing system, Heuristic

以Web应用服务器(WAS)为代表的中间件,为面向Internet的分布式应用的开发、运行和管理提供了有效的软件基础设施服务,对应用的服务质量(QoS)有着重要的影响[1]。对于规模较大和复杂性较高的Web应用,在负载数量迅速增长、系统资源动态变化的情况下,如何保持或者改善关键性能指标(如吞吐量、响应时间),让客户请求得到满意的服务性能,对中间件层次(如Web应用服务器)的性能保障和优化技术都提出了较高的要求。例如目前广泛应用的在线购物系统,用户量以千万级甚至亿级计算,如何解决其性能问题,让用户得到较高的满意度成为关键。近年的研究热点主要有高速缓存、线程池、负载平衡、消息处理等技术[2]。

针对Internet应用的多线程、分布式、并发性等特征,线程池是Web应用服务器常用的一种技术。它通过对多个任务复用已有线程,为线程创建和销毁的开销、系统资源不足等问题提供解决方案,适合处理大量负载情况下的应用服务器性能保障问题。当前,线程池管理的研究热点是尽量摆脱以往手工配置线程池的弱点,目标是在应用服务器运行过程中能够动态调整线程池大小。从已有的工作来看,存在的难点之一在于面对负载多变、资源状态动态变化的Internet计算环境,如何合理地、适时地评估和确定影响线程池配置的因素,以使线程池规模的运行时调整能够动态地达到或者接近最佳状态。本文就此问题展开讨论,以Web应用服务器的线程池管理为上下文,在M/M/1/K/∞/FCFS排队系统[4]的基础上,提出一种动态调整线程池大小的管理模型(HDTP-QS, Heuristic and Dynamic Thread Pooling model based on Queuing System)。HDTP-QS利用排队系统对大量负载请求进行排队,减少请求的等待时间;然后基于实时监控数据,在扩展线程池中引入响应系数等启发式因素,以此驱动线程池的动

到稿日期:2009-11-24 返修日期:2010-02-11 本文受国家863课题(2007AA01Z134),广西高校人才小高地建设创新团队资助计划(桂教人[2007]71号)资助。

陈宁江(1975—),男,博士,副教授,CCF会员,主要研究方向为软件工程、网络分布计算,E-mail:chnj@gxu.edu.cn;林 盘(1986—),男,硕士生,主要研究方向为软件工程、中间件技术等。

态调度过程，使得线程池规模与系统应用需求和计算资源相适应。实验表明，HDTP-QS 模型在运行时可以有效地自动调整线程池大小，缩短系统的响应时间和改善服务性能。

1 相关工作

关于线程池的研究，在操作系统等多个领域都有不少工作。传统线程池模型主要有工作组模型、主从模型和管道模型[5]等。中间件特别是 Web 应用服务器中的线程池管理研究历史相对较短。早期的线程池管理采用静态模型，主要采取人工方式离线调整线程池的规模。这种静态模型在大量负载动态变化的情况下并不适用，往往会致使应用服务器性能下降[5]。因此，线程池的动态管理受到关注，有必要根据系统负载请求、资源的动态变化、系统当前工作状态等因素来自动调整线程池大小。Ling 等人[3]建立了一个数学模型，分析了线程池中线程数对系统性能的影响。文献[8]以时间因素为特征，分析了线程池处理一个任务的各个阶段和线程池中一个任务完成所需的各部分时间，根据任务的完成时间来动态改变系统线程池的大小。但是他们的工作尚缺乏充分的原型验证，仍需要结合中间件的上下文来考虑具体的影响因素。文献[9]提出了一个多级线程池中间件体系结构，它可以控制网络和请求使用的资源，用于提高系统的时间可预测性，但是没有考虑 Internet 环境下大量负载的情况。文献[7]提出一种基于反馈技术的线程池调度管理框架($FFAT_pM$)，其以系统管理人员的先验性知识为基础，通过反馈运行时线程池状态实现在线调整线程池规模，但是该模型过多依赖于系统管理人员。文献[10]提出一种基于预测的、有效利用空闲资源提高服务器的响应时间的动态线程池模型，但其并没有考虑线程池大小对系统性能的影响。上述相关工作表明：1)研究者们认识到动态线程池管理需要考虑特定的运行时状态因素，然而有效的影响因素挖掘及其使用仍有待于进一步讨论；2)采用排队系统、反馈控制理论等来模拟和实现中间件的线程池管理是有效手段，但是仍需要结合具体的影响因素方能有针对性地给出解决方案。针对以上问题，本文以 Web 应用服务器为研究平台，研究线程池动态管理机制。与以上相关工作相比，特点在于：以 M/M/1/K/∞/FCFS 排队系统为建模基础，引入若干启发式因素以更好地反映运行时状态，通过启发式规则驱动线程池大小的动态调整，使调整效果具有更好的环境适应性和合理性。

2 基于排队系统的线程池模型

2.1 线程池模型的组成

首先给出本文工作所依赖的线程池管理模型。鉴于线程池管理的主从模型具有可移植、可管理、易于开发和部署的特性，我们基于主从模型重新设计了一个 Web 应用服务器的线程池模型 HDTP-QS，如图 1 所示。它利用 M/M/1/K/∞/FCFS 排队思想对大量负载请求进行排队；在基本线程池的基础上引入扩展线程池，通过池管理模块进行配置参数和控制消息的传递，以获取系统性能指标，并根据动态优化策略，创建新线程和阻塞扩展线程。

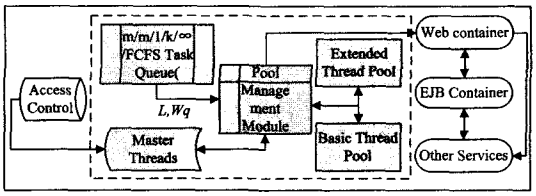


图 1 HDTP-QS 模型

2.2 排队系统

在不同数量负载请求下，系统达到最佳性能时的线程池大小是不同的。当负载请求和线程处理能力不协调时(包括负载请求过多、线程处理能力不足和负载请求过少、空闲线程过多两种情况)，系统性能必定会下降。为此，HDTP-QS 使用了请求排队机制。

负载请求增多，利用排队模型将请求插入到任务队列中等待，一方面可以减少在没有排队时过多负载请求获得线程资源的开销，另一方面可以减少服务器对请求的抛弃，最大限度地处理更多的请求。随着负载请求逐渐增加，任务队列的等待请求数量随之增大。然而，由于单个服务器系统处理请求的数量有限，必定会抛弃部分请求。利用适应于容量有限单服务器的 M/M/1/K/∞/排队模型进行负载请求的排队(第一个 M 为按照 Poisson 流到达的请求率，第二个 M 为请求服务时间，服务器台数为 1；排队请求设为∞)，可以有效降低请求的抛弃率。我们将通过 M/M/1/K/∞/排队模型所获得的系统中平均请求数和平均响应时间作为动态调整线程池大小的输入参数。由于负载过多，统一采用 FCFS(First Come First Served)服务规则，屏蔽按照任务种类进行服务的复杂性。以下我们利用排队论的 M/M/1/K/∞/FCFS 模型[4]描述线程池管理问题。

假设请求到达间隔和请求服务时间服从负指数分布，请求到达间隔使用到达率 λ 作为参数，请求服务时间使用服务率 μ 作为参数；系统容量 K 表示队列和线程池的总大小。

设 $p_n=P\{N=n\}$，$n=0,1,2,\cdots$ 为系统平稳后系统有 n 个请求的概率。因容量为 K，服务台为 1，有 $K-1$ 个等待位置，故到达率和服务率为：

$$\lambda_n=\begin{cases}\lambda, & n=0,1,2,\cdots,K-1\\ 0, & n\geqslant K\end{cases} \tag{1}$$

$$\mu_n=\mu, n=1,2,\cdots,K \tag{2}$$

由式(1)和式(2)可以求得系统中 n 个请求的概率为：

$$p_n=\rho^n p_0, n=1,2,\cdots,K \quad \rho=\lambda/\mu \text{ 和 } p_0=\frac{1}{1+\sum_{n=1}^{K}\rho^n} \tag{3}$$

系统中平均请求数为：

$$L=\sum_{n=0}^{K}np_n=p_0\rho\sum_{n=1}^{K}n\rho^{n-1}=p_0\rho\frac{\mathrm{d}}{\mathrm{d}\rho}\left(\frac{\rho(1-\rho^K)}{1-\rho}\right)$$

$$=\frac{\rho}{1-\rho}-\frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} \tag{4}$$

由式(4)得平均排队长为：

$$L_q=\sum_{n=1}^{K}(n-1)p_n=L-(1-p_0)=\frac{\rho}{1-\rho}-\frac{\rho(1+K\rho^K)}{1-\rho^{K+1}} \tag{5}$$

我们可以得出平均等待时间：

$$W_q=L_q/\lambda_e=L_q/\lambda(1-p_K) \tag{6}$$

HDTP-QS 模型中的池管理模块根据排队系统中的平均请求数 L 和平均等待时间 W_q 确定系统的响应系数等启发式

信息，构造负载请求执行过程中的启发式规则。池管理模块获取这些信息，实时监控系统的性能和线程资源的变化情况，在系统性能下降时根据启发式规则触发扩展线程池中线程的创建或者阻塞活动，从而达到动态改变线程池规模的目的。下面将对该线程池动态管理机制进行阐述。

3 线程池动态管理机制

3.1 请求执行过程中的线程池调整策略

在一段运行时间内，Web 应用服务器处理负载请求的过程中，根据运行时管理模块的监测，负载请求存在两种变化趋势：负载渐增过程和负载渐减过程。负载渐增过程对应于需要创建新线程的过程；负载减少过程对应线程阻塞过程。对此，我们采取不同的策略分别处理。

3.1.1 系统负载请求渐增过程

基本线程池中的线程都处于工作状态时，池管理模块激活扩展线程池。假设一个请求处理完成的时间为 T，线程调度过程中的开销、线程的上下文切换、消息分派过程中的内存拷贝所需要的时间为 T_s[13]，线程处理请求花费的时间为 T_p，则可以认为 $T=T_s+T_p$。为了衡量整个过程中系统的性能，主要考虑服务器的响应速度，我们认为它与请求在队列中的等待时间 W_q 和线程处理请求时间 T_p 有关。在服务请求处理完成时间 T 内，尽可能减少等待时间，提高请求处理时间的占有率，故定义响应系数 T_C：

$$T_C=\frac{T_p}{T}*\frac{1}{W_q} \tag{7}$$

式中，T_p/T 表示在请求处理的过程中请求处理的时间在处理过程中的比例。前述 M/M/1/K/∞/FCFS 排队模型减少了负载请求的等待时间 W_q，式(7)中引入 W_q 对响应系数进行调优。T_C 反映了应用服务器中所有负载请求得到响应的速率。当 T_C 越大时，系统的性能越好；反之，系统的性能越差。在请求运行过程中，池管理模块根据请求数、资源变化情况不断更新 T_C 值。

池管理模块根据 T_C 值感知某一时刻应用服务器的性能。性能下降时，池管理模块做出相应的调整。当扩展线程池没有空闲线程时，池管理模块发出消息，在扩展线程池中创建新的线程。请求增大且处理能力不足时，T_C 值较小；创建新的线程后 T_C 逐渐增大，但是在此过程中由于请求数迅速增加，新创建的线程并不能满足过多请求，导致 T_C 值会有所减小，池管理模块继续触发新线程的创建；系统中请求数和线程资源趋于稳定时，T_C 值也趋向于稳定，一段时间后开始逐渐减小，池管理模块停止创建线程的活动。我们认为应用服务器中平均请求数为 L 的情况下，T_C 处于最大值时线程池的大小为适应请求数 L 的最佳大小，此时线程池性能处于最佳状态。文献[3]中指出线程池性能达到最大值时的线程数 n 应满足下列公式：

$$\frac{\lfloor n\rfloor}{N}\leqslant 1-\varepsilon,\frac{\lfloor n\rfloor+1}{N}>1-\varepsilon \tag{8}$$

式中，N 为并发用户数量，$\varepsilon=T_o/T_g$（创建一个线程的开销为 T_o，单个线程的管理开销为 T_g）。池管理模块可以获取 ε 的值。我们用式(8)作为规则来验证并发用户数量为 L、T_C 处于最大值时线程池的大小 n 是否满足此公式，从而验证 T_C 值的有效性。

3.1.2 系统负载请求渐减过程

在负载请求处于稳定，应用服务器稳定处理请求一段时间后，系统中的请求数 L 逐渐减少。而线程池中的线程数由于之前的增大过程处于过剩的状态，此时线程池中的线程很多处于空闲状态，维护线程的开销反而大于处理请求的开销。应用服务器必须具有随着负载请求数的减少，工作线程随之减少的能力。

当负载请求数减少，平均等待时间 W_q 也相对变化。设 T_{max} 为请求处理过程中 T_C 的最大值，池管理模块可以从记录 T_C 值的数组中查询出 T_{max} 值；L 为应用服务器中最大的请求数，定义 T_a 为阻塞线程指标：

$$T_a=\frac{T_{max}*L}{\sum_{n=0}^{L}(W_q)_n} \tag{9}$$

在整个运行过程中，W_q 是不断变化的，池管理模块计算出请求数到达最大时 W_q 的平均值，由 T_{max} 除以 W_q 的平均值得到阻塞线程指标 T_a。当 T_C 值从 T_{max} 值开始减少时，系统中的线程处理能力大于请求数所需资源。当 T_C 值减少到 T_a 时，池管理模块触发阻塞线程活动（采用阻塞线程而不是直接删除掉线程。这样当新的任务请求到达时，不用为创建新线程耗费相应的时间和资源）；当 T_C 值大约减少为 T_{max} 的一半时（$T_C=T_{max}/2\pm\Delta,\Delta\to 0$），停止阻塞线程活动。这样避免了负载请求数过少或为 0 时线程池的规模不会过少或减少为 0，以确保系统中总有一定数量的线程在运行。

3.2 线程动态调度算法

基于前述的 M/M/1/K/∞/FCFS 排队模型和两个动态调整过程，我们给出一个线程调度算法。它的主要步骤如下：

①接收客户负载请求，由 M/M/1/K/∞/FCFS 模型进行排队，输出请求数 L、等待时间 W_q；

②池管理模块开始记录响应系数 T_C 值，存放于数组 array_ response 中；

③While(array_ response[i]<array_ response[$i-1$])

creatNewThread();//创建新的线程

T_C 值稳定于 K 值时，if($T_C\to k$)

跳出步骤③；

④池管理模块记录阻塞线程指标 T_a 值；

⑤While(array_ response[i]<=T_a)

阻塞扩展线程池中的线程；

If(扩展线程池中线程全部未激活)

停止阻塞线程活动；

Else if($T_C==T_{max}/2\pm\Delta,\Delta\to 0$)

停止阻塞线程活动并跳出步骤⑤；

⑥重复执行步骤①—步骤②；

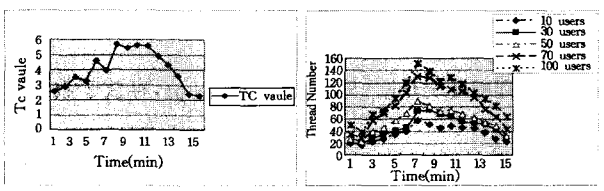
在以上调度机制中，本文的 HDTP-QS 利用响应系数和阻塞指标作为启发因素，对线程池规模的调整进行指导。由于响应系数和阻塞指标引入了平均等待时间和请求数，合理准确地评估了影响线程池性能的因素，有助于提升调整效果，对于改善应用服务器的响应时间和吞吐量会起到积极作用。我们可以从下面的实验中看到其效果。

4 实验与分析

我们在开源的 Web 应用服务器 Jboss 上对本文的线程池管理机制进行原型实验。具体的实验方案是以 Jboss 为平台搭建一个网上书店购物系统，采用性能测试工具 LoadRun-

ner[14]来模拟不同数量的用户访问书店系统。我们分别模拟10,30,50,70,100个不同用户量的情况,同时迭代运行模拟万级的并发量。实验分为3个阶段:加载所有用户量阶段,此阶段模拟了用户请求数不断增加的过程;运行阶段,分别迭代运行各用户量一定时间,此阶段模拟了系统正常情况下用户不断访问的过程;停止用户访问阶段,此阶段模拟了用户请求数不断减少的过程。实验结果从线程变化情况和系统性能(响应时间和吞吐量)两个方面进行说明。实验环境为:PC机,AMD Athlon(tm)64 *2 Dual Core Processor 4400+2.30 GH,896M内存;windowsXP操作系统;jboss-4.2.1.GA;LoadRunner8.0。

图2(a)为T_C值的整体变化趋势图;图2(b)是整个实验过程中不同用户量在不同时刻对应的线程池中的线程数量。从图2(a)中可以看出,用户加载过程中,T_C值增大;随着用户请求迭代运行,T_C值达到最大值;稳定运行大约3min后,T_C值逐渐减小。从图2(b)中可以看出:前5min,请求数不断增加以及当前线程池中线程数量不足,因此系统在扩展线程池中创建新的线程;大约10min后用户请求减少,线程池中线程数量减少。结合图2(a)和图2(b)知,T_C值减少至大约T_{max} 2/3时,系统自动阻塞线程;所有请求运行完成时,线程池中的线程并没有减少至0,因为T_C值减少至大约$T_{max}/2$时,系统停止阻塞线程活动。显然,基于HDTP-QS模型的应用服务器根据负载请求的数量以及系统资源的变化情况动态调整线程池大小。通过实验得知,在T_C处于最大值时,线程池中的线程数量完全符合式(8)的要求。



(a)T_C值变化趋势图　　(b)不同用户请示下的线程池大小

图2

为了进行工作比较,我们选取在Jboss上实现了FFAT$_p$M模型。图3给出JBoss原有线程池、使用FFAT$_p$M模型和使用HDTP-QS模型的JBoss的表现比较。从图3可以看出,由于HDTP-QS模型在整个负载请求处理过程中利用适应于容量有限单服务器的M/M/1/K/∞/排队模型进行负载请求的排队,减少了请求的等待时间。并且根据式(7)和式(9)进行动态调整,线程池的处理能力与负载和系统资源处于一个更加合理、协调的状态,使得基于HDTP-QS模型的JBoss比其它两种模型的平均响应时间有所提高。同时响应时间的减少也验证了文中定义的响应系数和阻塞线程指标合理地评估了影响线程池性能的因素,有效地调整了线程池规模。

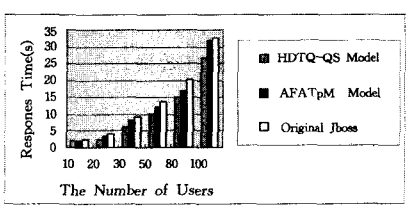


图3　平均响应时间比较图

图4给出了在吞吐量方面的比较情况。由于M/M/1/K/∞/FCFS排队模型和动态调整两个方面的作用,如图3所示,响应时间的提高使得相同时间内处理的请求数大大增加,致使基于HDTP-QS模型的JBoss吞吐量也随之提高。但是由于单机的硬件条件和JBoss的承载能力,较多客户同时访问系统时,JBoss无法同时处理大量的负载请求。图4反映了请求数超过万级时吞吐量没有显著的提高。

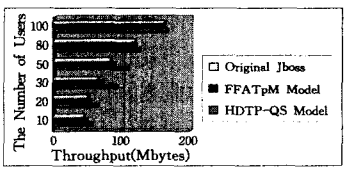


图4　吞吐量比较图

结束语　本文提出了一种Web应用服务器的动态线程池管理机制,目标是通过有效地调整线程池的规模,使得应用该模型的服务器性能保障能力得到提升。本文的工作贡献主要在于:在基于排队系统的线程池管理模型中,提出一种启发式因素(包括响应系数和阻塞指标)驱动的线程池动态调整机制,这些启发因素能够比较准确地反映线程池的运行时状态,从而驱动线程池规模能够自动地合理地动态调整。原型实验验证了启发式因素的变化对线程池规模调整的有效影响,并表明本文的管理机制能够有效改善系统性能,使服务器的平均响应时间和吞吐量得到改善。下一步工作将深入探讨线程池管理在应用服务器中的实现机制,如何与应用服务器的其他模块更好地协调工作;通过与其它线程池管理模型进行更深入、更广泛的比较实验,进一步挖掘启发因素的作用,以及将已有工作结果扩展到容量有限单服务器的M/M/s/K/∞/FCFS排队模型之中。

参考文献

[1] Geibs K. Middleware challenges ahead [J]. IEEE Computer, 2001,34(6):24-31

[2] Mattern F, Sturm P. From distributed systems to ubiquitous computing The state of the art, trends, and prospects of future networked systems [C] // Klaus Irmscher, Klaus-Peter Fähnrich, eds. Proceedings of KIVS. 2003:3-25

[3] Ling Y, Mullen T, Lin X. Analysis of optimal thread pool size [J]. ACM SIGOPS Operating Systems Review, 2000,34(1):42-55

[4] Servi L D, Finn S G. M/M/1 queue with working vacations(M/M/1/WV) [J]. Perform Evaluation, 2002,50(10):41-52

[5] Harkema M, Gijsen B M M, van der Mei R D, et al. Performance comparison of middleware threading strategies[C]// Proceedings of International Symposium on Performance Evaluation of Computer and Communication Systems. SCS Press, 2004:727-732

[6] Stonebraker M. Too much middleware[J]. ACM SIGMOD Record, 2002,31(1):97-102

[7] 杨刚,周兴社,潘惠芳. 基于反馈的自适应线程池管理框架[J]. 计算机工程,2006,32(5):65-67

[8] 刘云生,王刚,王卫国. 实时线程池性能研究与动态优化[J]. 计算机工程与科学,2007,29(12):123-126

[9] 郭长国,王怀明,邹鹏,等. 实时中间件的研究与实现[J]. 电子学报,2002,30(12A):2094-2098

(下转第201页)

结束语 PSO算法的搜索机制是随机的，通过迭代计算来找到近似最优值或是问题的近似最优解决方案，无论适应度函数是不是连续的或是可导的，都可以应用PSO来解决。因此，PSO的应用范围就更广泛，更有可行性。本文提出的IDPSO是改进的离散型PSO算法，在IDPSO算法中融入了遗传算法的部分操作算子，包括变异算子和交叉算子。普通的PSO算法一般以一定的概率按照式(1)、式(2)更新粒子的位置和速度，可以比较迅速地得到次优适应度函数值，但是有时候会因为忽略的粒子有较好的优化信息，从而陷入局部最优。IDPSO通过粒子群优化算法本身的更新规则代替遗传算法中的选择规则，在每次的迭代过程中都以一定的概率对每个粒子进行交叉、变异操作，尽可能地保留种群中各个粒子的优化信息，尽可能避免迅速陷入局部最优。

本文采用的IDPSO算法通过多组实例的测试得到了比较不错的布线方案；通过与遗传算法在最优值进化曲线和平均适应度函数值的比较，表明IDPSO具有更快的收敛速度、简单的操作，更具可行性和有效性。在下一步的工作中，将寻找一种能够表示更为广泛的矩形Steiner树的编码，用以表示非连通布线模型中的矩形Steiner树。

参考文献

[1] Hashimoto A, et al. Wire Routing by Optimizing Channal Assignment Within Large Appertures[C]// Proc. of 8th IEEE/ACM Design Automation Workshop. 1971

[2] 邓爱姣，李强，张嘉为. 改进的Prim启发式算法在VLSI布线中的应用[J]. 沈阳工业大学学报，2006，28(5)：557-559

[3] 杨昌玲，严晓浪. 基于树形编码的MRST混合遗传算法及其并行处理[J]. 微电子学，1999，29(2)：89-95

[4] Wey Chin-Long. Efficient Rectilinear Steiner Tree Construction with Rectangular Obstacles[C]// Proc. 5th WSEAS International Conference on Circuits System Selectronics Control & Signal Processign. Dallas, USA, 2006：204-208

[5] Rita M H, Bryant A J. A Spanning-Tree-based Genetic Algorithm for Some Instances of the Rectilinear Steiner Problem with Obstacles[C]// Proceedings of the 2003 ACM Symposium on Applied Computing. 2003：725-729

[6] Eberhart R C, Kennedy J. A New Optimizer Using Particles Swarm Theory[C]// Proc. 6th International Symposium on Micro Machine and Human Science. Nagoya, Japan, 1995：39-43

[7] Clerc M, Kennedy J. The Particle Swarm-explosion, Stability, and Convergence in a Multidimensional Complex Space[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(1)：58-73

[8] Hanan M. On steiner's problem with rectilinear distance[J]. SIAM Journal of Applied Mathematics, 1996, 14(2)：255-265

[9] Kennedy J, Eberhart R C. A Discrete Binary Version of the Particle Swarm Optimization Algorithm[C]// Proc. of the IEEE International Conference on Systems Man and Cybernetics. Orlando, USA, 1997, V：4104-4109

[10] Clerc M. Discrete Particle Swarm Optimzation illustrated by the Traveling Salesman Problem[EB/OL]. http://www.maurice-clerc.net, 2000-02-29

[11] 郭文忠，陈国龙，夏添. 异构机群下数据流自适应分配策略[J]. 计算机辅助设计与图形学学报，2009，21(8)：1175-1181

[12] 郭文忠，陈国龙. 一种求解多目标最小生成树问题的有效离散粒子群优化算法[J]. 模式识别与人工智能，2009，22(4)：597-604

[13] 王晓东，吴英杰. Prufer编解码的最优算法[J]. 小型微型计算机系统，2008，29(4)：687-690

[14] 张会生，翁史烈，张小兵，等. 基于内弹道改进型零维模型的装药优化仿真[J]. 弹道学报，2000，12(3)：32-36

[15] 洪先龙，严晓浪，乔长阁. 超大规模集成电路布图理论与算法[M]. 北京：科学出版社，1998

[16] Shi Y H, Eberhart R C. A Modified Particle Swarm Optimizer[C]// IEEE International Conference of Evolutionary Computation. Piscataway, NJ：IEEE, 1998：69-73

(上接第164页)

[10] Kang Donghyun, Han Saeyoung, Yoo Seohee, et al. Prediction-based Dynamic Thread Pool Scheme for Efficient Resource Usage[C]// Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops. 2008：159-164

[11] Goetz B. Thread Pools Help Achieve Optimum Resource Utilization[EB/OL]. http://www.900.ibm.com/developerWorks/en/java/j-jtp0730/index_eng.shtml, 2002-07

[12] 邵鸣年，张昕. 一种分布式系统中线程池的设计与实现[J]. 计算机工程与设计，2005，26(1)：7-11

[13] Mercury Interactive. LoadRunner controller user's guide windows version8.0 [EB/OL]. http://www.mercury.com/us/products/performance-center/loadrunner/

(上接第172页)

[10] Dimitris S, Antonios D, Timos S. Hierarchically compressed wavelet synopses[J]. The VLDB Journal, 2009, 18(1)：203-231

[11] Franky Kin-Pong C, Ada Wai-chee F, Clement Y. Haar Wavelets for Efficient Similarity Search of Time-series：With and Without Time Warping[J]. IEEE Trans. on Knowl. and Data Eng., 2003, 15(3)：686-705

[12] Popivanov I, Miller R J. Similarity Search Over Time-Series Data Using Wavelets[C]// Proceedings of the 18th International Conference on Data Engineering. IEEE Computer Society, 2002：212-216

[13] Liabotis I, Theodoulidis B, Saraaee M. Improving Similarity Search in Time Series Using Wavelets[J]. International Journal of Data Warehousing and Mining, 2006, 2(2)

[14] Yingyi B, Lei C, Ada Wai-Chee F, et al. Efficient anomaly monitoring over moving object trajectory streams[C]// Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Paris, France, ACM, 2009：159-168

[15] Hao-Ping H, Ming-Syan C. Efficient range-constrained similarity search on wavelet synopses over multiple streams[C]// Proceedings of the 15th ACM International Conference on Information and Knowledge Management. Arlington, Virginia, USA, ACM, 2006：327-336