

基于平均时间的线程池尺寸自适应调整算法

黄智泉,刘正熙

(国家空管自动化系统技术重点实验室,四川 成都 610065)

摘要:为了实现自适应调整线程池尺寸,提高并发程序处理的运行速度、改善效率和降低系统的资源开销,提出了一种基于任务平均处理时间的线程池尺寸自适应调整算法。首先研究线程池的一些特征数据以及用户请求的任务时间周期,提出了任务平均处理时间的概念。然后研究了不同任务类型下,任务平均处理时间与线程池的尺寸的相关性,提出了一种自适应调整算法。实验结果表明,该算法能够自适应调整线程池尺寸到适当区域,有效地提高应用程序的整体性能。

关键词:线程池;线程池尺寸;平均处理时间;自适应调整

中图分类号:TP31

文献标识码:A

文章编号:1673-629X(2013)02-0037-04

doi:10.3969/j.issn.1673-2013.02.009

Pool Size Adaptive Adjusting Algorithm Based on Average Time

HUANG Zhi-quan, LIU Zheng-xi

(National Key Laboratory of Air Traffic Control Automation System Technology, Chengdu 610065, China)

Abstract: To realize adaptive adjusting the thread pool size, improve the operation speed of concurrent processing program, efficiency and reduce the system resource spending, a thread pool size adaptive adjusting algorithm based on task average time is presented. Firstly, some characteristics of the thread pool and the task time period of users request are studied, then the concept of task average processing time is put forward. After that, the correlation between the task average processing time and thread pool size under the different types of tasks is analyzed. A kind of adaptive adjustment algorithm is put forward. The experimental results show that the algorithm can adaptively adjust thread pool size to the appropriate area, effectively improving the overall performance of application program.

Key words: thread pool; thread pool size; average process time; adaptive adjusting

0 引言

服务器系统需要同时处理大量的用户请求,如何解决其性能问题,让用户得到较高的满意度成为关键。近年来研究热点主要有高速缓存、线程池、负载均衡、消息处理等技术^[1]。其中多线程技术是处理大量并发请求的首要选择之一。目前,该技术主要有两种模型^[2]:线程-请求模型和线程池模型。线程-请求模型为每个请求建立一个线程,完成任务后立即销毁。而线程池模型则预先建立一定数量的线程,每个线程处理完用户请求后转换为空闲线程,以备响应新的用户请求。相对于线程-请求模型,线程池模型减少了反复建立和销毁线程的操作,改善了系统的性能并减少了响应时间^[3,4]。所以,许多主流服务器系统都使用该技术,例如 Apache 和 Windows IIS 系统等。

线程池技术能够提高系统性能,但线程池技术要

预先创建一定数量的线程,系统为了维护这些创建的线程需要开销。当线程创建得过多,系统为了维护线程的开销过多;而且,当请求过多、并发执行的线程过多时,它们之间资源的竞争就会变得更加剧烈,导致系统的性能反而下降^[5]。所以线程池性能的主要决定因素在于线程池的尺寸,以前在很多线程池的应用中,线程池的大小都是由系统配置文件指定,随着系统的运行,需要管理员根据系统的运行情况手动地调整线程池的大小。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^[6],有必要根据系统的荷载请求、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^[7]。

1 线程池模型

线程池采用 POSIX C 标准和 Pthread 线程库实现,可以方便集成到已有的 C/C++ 系统中。线程池分为四个模块^[8],即线程队列、任务队列、线程管理与调度模块、性能检测与优化模块。线程池系统结构如图 1 所示。

(1) 任务队列和线程队列。任务队列和线程队列

收稿日期:2012-05-22;修回日期:2012-08-25

基金项目:国家“973”重点基础研究发展计划项目(2009CB320803)

作者简介:黄智泉(1988-),男,福建莆田人,硕士研究生,研究方向为计算机技术及应用。

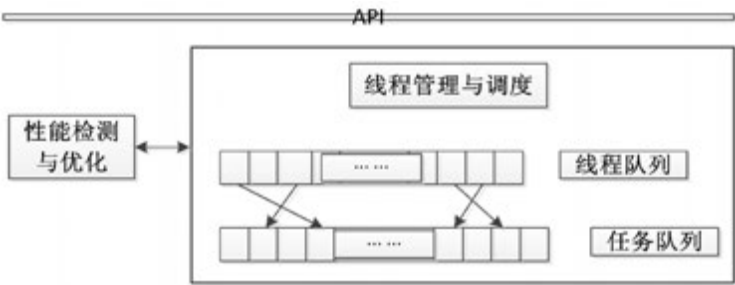


图 1 线程池系统结构

是两个分别用来存储工作任务和线程池线程的数据结构。用户提交过来的任务被存放在任务队列中;线程队列用来存储线程的信息,线程一般存储为 pthread_t 类型。线程队列中的线程有两种状态,即空闲状态和工作状态。在系统才启动的时候,所有线程都处于空闲状态,这些线程处于阻塞状态(线程处于阻塞状态不占用 CPU,只占用很少一部分内存^[9])。当有任务被放到任务队列时,系统发出一个 task_posted 信号。一旦信号量被设置,根据 Linux 线程调度机制,在阻塞的空闲线程中产生竞争,其中有一个线程被唤醒,开始执行任务^[10]。线程池中的线程执行完任务后并不退出,而是处于阻塞状态,等待下一个任务到达。

(2)线程管理与调度模块负责创建、保存、销毁以及调度线程。

(3)性能检测与优化模块动态地检测系统性能,如果性能由于用户载荷改变导致性能下降,该模块动态调整系统的参数,以达到优化的效果。该模块检测系统某些选定的数据,根据所得数据评估系统的性能状态;如果性能下降,适当调整线程池的尺寸,从而达到提高线程池性能的目的。

文中通过实验研究任务的时间流、任务的属性以及任务的处理时间与线程池尺寸的关系,提出了一种基于平均处理时间的线程池尺寸自适应调整算法。

2 线程池特征数据

2.1 任务存储结构

线程池系统中的每一个任务信息都存储在链式数据结构中,链式数据结构的时间属性如表 1 所示。这些时间属性会在程序执行过程中进行更新。

表 1 任务时间属性表

时间值	描述
submittedTime	客户端提交任务时间
acceptTime	任务队列接受任务时间
finishedTime	任务处理结束的时间

2.2 任务睡眠时间

在真实环境中,由于多线程系统应用不同,执行的任务类型也不一样。任务主要可分成两种类型,即计算密集型与 I/O 操作型^[11]。实验为了研究这些不同情况,定义了一个新的变量,即任务睡眠时间(taskSleep)。该变量用来调整任务的睡眠频率。这个值越大,任务睡眠的时间越长。即表示该系统处理的任务倾向为 I/O 操作型。

2.3 任务处理时间

从客户端提交任务到服务器端执行完该任务,任务的时间周期可以分成三个阶段,即响应时间、等待时间和执行时间^[12],如图 2 所示。



图 2 任务时间流

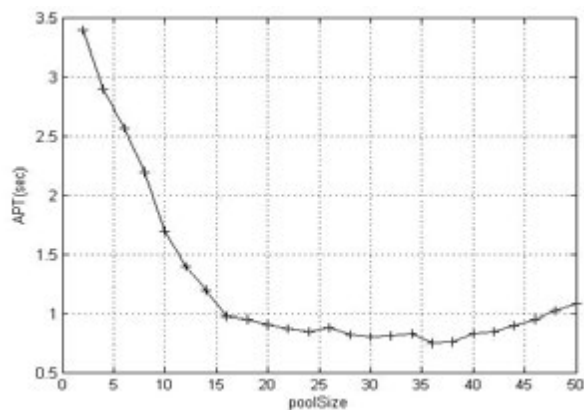
其中,响应时间与网络状况等有较大关系,本实验不予讨论。等待时间主要指任务在任务队列中的等待时间。执行时间与任务的类型有很大关系,I/O 操作型比计算密集型需要花费更多时间。

任务的总空闲时间由任务在任务队列中的等待时间和任务在执行过程中的等待或者挂起的时间组成^[3]。这个任务空闲时间很难计算出来,所以定义任务处理时间(task process time)即等待时间加上执行时间,任务处理时间可以很容易计算出来。由表 1 可知,任务处理时间由 finishedTime 和 acceptTime 之差得到。对于同一种类型的任务,当任务处理时间越小,任务空闲时间也就越小,线程池的性能也越好。

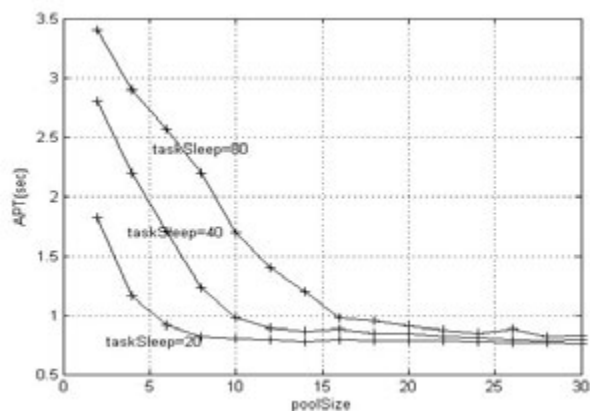
3 线程池尺寸自适应算法

3.1 平均处理时间 VS 线程池尺寸

实际应用中,用户的请求量,即系统的任务总数会随着时间不断变化,整体表现出波峰、波谷的特征。实验为了研究方便,设置用户请求量稳定在一定范围内。实验中定义执行完 5 个任务为一个循环,任务平均处理时间(average process time, APT)为一个循环内的平均任务处理时间,可以很方便计算出来。然后,做了关于 APT 与线程池尺寸的实验,设置 taskSleep = 80,实验结果如图 3(a)所示。接着又做了关于不同任务类型下的任务平均处理时间(APT)与线程池尺寸的实验,把 taskSleep 分别设置为 80、40 和 20。实验结果如图 3(b)所示。



(a) taskSleep = 80



(b) taskSleep = 20、40 和 80

图3 APT VS 线程池尺寸

图3(a)中,曲线表示了在 taskSleep = 80 的情况下,任务平均处理时间 (APT) 与线程池尺寸的关系。由图3(a)可知,在初始阶段,线程池尺寸较小的时候,即 poolSize = 2 的时候,任务平均处理时间数值比较大,此时线程池不能满足系统的需求,需要增大线程池尺寸。当增加线程池尺寸,即 poolSize = 4,任务平均处理时间会明显地减小。随后继续不断增大 poolSize,任务平均处理时间也会随着不断减小。但当 poolSize = 15 时,随后增加线程池尺寸,任务平均处理时间不会大幅减小,会稳定在一个范围内。这个点定义为稳定点 (stablePoint)。如果继续增大线程池尺寸,当 poolSize = 42 时,随后增加线程池尺寸,任务平均处理时间反而会有所上升,即线程池的性能下降。将其定义为衰减点 (reductionPoint)。定义稳定点与衰减点之间的区域为安全区域 (safeZone)。所以需要研究一种算法,使得线程池尺寸能够尽快地动态调整到安全区域,提高系统的整体性能。

图3(b)中,三条曲线分别表示了在不同的 taskSleep 下,任务平均处理时间与线程池尺寸的关系。由图4可知,当 taskSleep = 80 的时候,初始阶段,任务平均处理时间 (APT) 随着线程池尺寸增大而较为平缓地减小。线程池尺寸的稳定点比较大, stablePoint₈₀ = 15。

当把 taskSleep 设置为 40 的时候,初始阶段,在线程池尺寸相同的情况下,任务平均处理时间比 taskSleep = 80 的情况要大幅减少 (即这些任务睡眠时间短,执行较快)。初始阶段,任务平均处理时间 (APT) 随着线程池尺寸增加而减小的速度加快。而且线程池尺寸的稳定点也变小了, stablePoint₄₀ = 10。当继续减小 taskSleep,将其设置为 20 的时候,初始阶段,任务平均处理时间 (APT) 随着线程池尺寸增加而减小的曲线越来越陡峭。线程池尺寸的稳定点也越来越小, stablePoint₂₀ = 6。

通过分析上述实验结果与分析,可以得出以下结论:

(1) 线程池技术可以明显提高多线程系统性能。但当线程创建得过多,系统为了维护线程的开销过多,反而会影响系统性能;

(2) 线程池的性能与实际应用、任务性质也有很大关系。相对来说,I/O 操作型任务需要更多的线程。

综上,合理的线程池尺寸是充分发挥线程池技术优势的关键。接下来的实验通过研究任务平均处理时间,提出一种基于平均处理时间的自适应线程池尺寸算法。使得线程池尺寸能够尽快地调整到安全区域 (safeZone),提高系统的整体性能。

3.2 基于平均时间的自适应算法

计算当前任务平均处理时间 (curAPT) 与上一循环的任务平均处理时间 (preAPT) 的百分比。如果这个百分比大于 2%, 即平均处理时间有了较大改变的时候,对线程池的尺寸进行一个步长 (pace) 的调整。算法在开始的时候,通过比较 curAPT 与 preAPT 的百分比,不断增加 poolSize。当发现上一次的 poolSize 增加引起 curAPT 增大时,减小 poolSize。算法的流程图如图4所示。

preAPT 用来记录上一次循环的任务平均处理时间,标志位 mark 用来记录上一次是增加还是减小线程池尺寸,1 表示增加,0 表示减小。pace 指定每次调整 poolSize 的步长。初始化时候,设置 preAPT 为 0, mark 为 1, pace 为 2。

4 实验结果及分析

实验中,在每个循环结束时,即每执行完五个任务,执行本算法。调整线程池尺寸。设置 taskSleep = 80 的情况,由于线程池尺寸的初始化大小对算法有较大影响,选择两个不同的初始化尺寸,分别为 5 和 15。实验结果如图5所示。

图5中,两条折线分别表示了在不同的线程池尺寸初始化情况下,算法如何在每次循环之后调整线程池尺寸。图中显示,开始阶段,两条折线都是向上增长

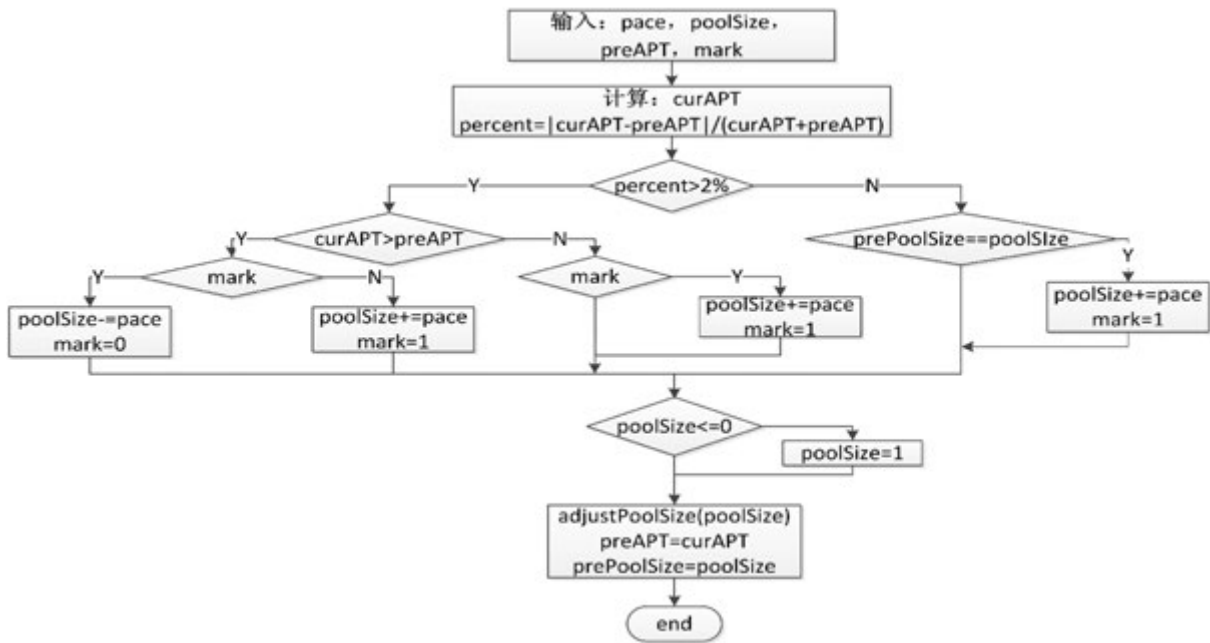


图 4 算法流程图

参考文献:

- [1] Mattem F, Sturm P. From distributed systems to ubiquitous computing: The state of the art, trends, and prospects of future networked system [C]//Proceedings of KIVS. [s. l.]: [s. n.], 2003: 3-25.
- [2] 孙晓东, 韩江洪, 刘征宇, 等. 基于分段的线程池尺寸自适应调整算法[J]. 计算机工程, 2011, 37(2): 43-45.
- [3] IBM Corp. . CellBroadband Engine Programming Handbook, Version 1.0 [M]. [s. l.]: IBM Corp. , 2006.
- [4] Eichenberger A E. Using Advanced Compiler Technology to Exploit the Performance of the Cell Broadband Engine Architecture[J]. IBM Systems Journal, 2006, 45(1): 59-84.
- [5] 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.
- [6] Harkema M, Gijzen 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. [s. l.]: SCS Press, 2004: 727-732.
- [7] 陈宁江, 林 盘. 一种基于排队系统的启发式中间件动态线程池管理机制[J]. 计算机科学, 2010, 37(10): 161-164.
- [8] 王 华, 马 亮, 顾 明. 线程池技术研究与应 [J]. 计算机应用研究, 2005, 22(11): 141-142.
- [9] Lu Juwei. Face Recognition Using Kernel Direct Discriminant Analysis Algorithms [J]. IEEE Transaction on Neural Networks, 2003, 14(1): 117-126.
- [10] 史蒂文斯. Unix 环境高级编程 [M]. 北京: 人民邮电出版社, 2006.
- [11] 杨开杰, 刘秋菊, 许汀荣. 线程池的多线程并发控制技术研究[J]. 计算机应用与软件, 2010, 27(1): 168-170.
- [12] Andrew S. 计算机网络 [M]. 潘爱民译. 北京: 清华大学出版社, 2004.

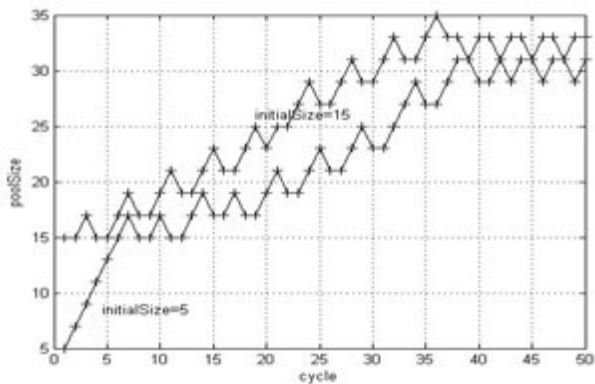


图 5 实验结果

的。等尺寸调整到了 30~35 这块区域,线程池的尺寸就不会再继续向上增长了,而是稳定在这块安全区域 (safeZone)。实验结果显示,对于两种不同的初始化线程池尺寸,算法都会尽快调整 poolSize 到安全区域 (safeZone)。并且之后 poolSize 一直在这个区域内波动。在真实系统中,APT 数据可能会受到其它因素的干扰,影响了算法的正确性。所以算法的少量误差是可以接受的。

5 结束语

线程池的应用越来越广泛,线程池技术可以有效提高系统的性能。但是,决定线程池性能的最关键因素是线程池的尺寸。文中通过实验研究不同类型应用程序给线程池系统性能带来的影响,同时研究线程池中的时间特征数据与线程池尺寸的相关性。提出了一种基于任务平均处理时间的线程池尺寸自适应调整算法。实验结果表明,该算法可以有效地提高系统整体性能。

基于平均时间的线程池尺寸自适应调整算法

作者: [黄智泉](#), [刘正熙](#)
作者单位: [国家空管自动化系统技术重点实验室, 四川 成都 610065](#)
刊名: [计算机技术与发展](#)
英文刊名: [Computer Technology and Development](#)
年, 卷(期): 2013(2)

本文链接: http://d.g.wanfangdata.com.cn/Periodical_wjtz201302011.aspx