可变负载动态反馈弹性调度模型及其算法研究*

陈 宇+, 戴琼海

(清华大学 深圳研究生院 宽带网与多媒体研究中心,广东 深圳 518055)

Research on Dynamic Feedback and Elastic Scheduling Model and Algorithm for Flexible Workload

CHEN Yu+, DAI Qiong-Hai

(Broadband Network & Multimedia Research Center, Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China)

+ Corresponding author: Phn: +86-755-26036765 ext 815, E-mail: cheny@sz.tsinghua.edu.cn, http://www.sz.tsinghua.edu.cn

Received 2003-03-13; Accepted 2003-10-08

Chen Y, Dai QH. Research on dynamic feedback and elastic scheduling model and algorithm for flexible workload. *Journal of Software*, 2004,15(3):379~390.

http://www.jos.org.cn/1000-9825/15/379.htm

Abstract: Due to the variation of the tasks' attributes, the behavior of soft real-time systems, such as multimedia application, is becoming increasingly unpredictable. Under this circumstance, the scheduling algorithms, which depend on the tasks' static attributes, can't give a usable and efficient resource allocation to those soft real-time systems. This paper presents an elastic scheduling algorithm for flexible workload. Based on sampling the total number and lost number of the task instances, this algorithm adjusts the number of task instances executing in the next sampling period to guarantee the tasks' basic QoS (quality of service) and to improve the system resource utilization and concurrency in the next sampling period. This paper analyzes the model and evaluates its performance. Simulation results show that, besides improving resource utilization, this algorithm has good stability and convergence.

Key words: feedback; real-time; scheduling; multimedia; QoS(quality of service)

摘 要: 由于工作负载的动态变化,以多媒体应用为代表的软实时系统的运行具有很大的不确定性.在这种情况下,依靠任务的静态属性进行调度分析和决策不足以为系统提供高效、实用的资源分配支持.提出一种弹性资源调度算法,该算法周期地采集系统的作业总数和作业丢失数,并以此为根据改变部分软实时任务的作业周期,以调整系统在下一个采样周期内的作业总数,达到满足任务的 QoS(quality of service)、接纳尽可能多的服务请求、提高系统的并发服务能力的目的.详细分析了模型结构和核心算法的实现机制,并利用模拟平台对该算法进行了验证.实验结果表明,该算法在提高资源利用效率的同时,还具有良好的稳定性和收敛性.

关键词: 反馈;实时;调度;多媒体;服务质量

中图法分类号: TP316 **文献标识码**: A

* **作者简介**: **陈宇**(1974－),男,四川大竹人,博士,主要研究领域为实时调度算法,多媒体计算;**戴琼海**(1964－),男,副教授,主要研究领域为宽带网络,图像处理.

实时多任务系统的首要特性是任务响应时间的确定性和高吞吐量,通用多任务系统以实现任务间公平地共享系统资源为主要目标.然而,随着计算机在人类生活中越来越普遍地得到应用,实时多任务系统和通用多任务系统的边界变得模糊起来,大量的应用同时具有实时系统和通用系统的特性.例如,流媒体服务器既要求能够在确定的时间范围内向客户发送一定数量的音、视频数据,又要求能够同时为尽可能多的用户提供服务.流媒体服务器的软实时任务的作业时间具有数据敏感性,即任务每次作业时间由处理数据量的多少决定.对于不同的数据集,同一任务作业时间的比值可以达到 5 倍,甚至更高,而且任务只是在少数情况下达到最坏作业时间.此外,流媒体服务器与生俱来的容错特性,使任务的一次或几次作业超越时限,并不会导致任务失效.与之类似的应用还包括网上交易系统、WWW 服务器等.对于上述应用,若沿用硬实时系统的调度算法,以任务的最坏作业时间作为调度的依据,以保证任务每次作业均能满足时限要求,将极大地浪费系统资源,降低系统的并发程度.

近年来,针对软实时应用的需求,国内外研究人员已经提出了许多软实时调度算法,并进行了大量的实践工作.通常,软实时调度算法用任务的平均作业时间代替最坏作业时间,以提高系统的实际资源利用率和任务接收率.在系统的运行过程中,任务的平均作业时间与实际作业时间存在偏差,系统负载可能在接近满载和过载间波动,造成任务的 QoS 的不确定和波动.

本文提出一种弹性资源调度算法.该算法周期地采集系统的作业总数和作业丢失数,并以此为根据改变部分软实时任务的作业周期,以调整系统在下一个采样周期内的作业总数,使任务在允许的 QoS 波动范围内稳定地运行,并接纳尽可能多的服务请求,达到在满足任务的 QoS 的前提下提高系统的并发服务能力的目的.本文第 1 节介绍软实时调度算法的研究概况.第 2 节介绍一种新颖的软实时调度算法——弹性调度.第 3 节详细论述动态反馈弹性实时调度.第 4 节利用模拟平台对该算法进行性能分析.最后给出结论.

1 相关工作

软实时调度的研究目标在于,在满足任务 QoS 的前提下,提高系统的并发程度,充分利用系统资源.目前,主要的软实时调度算法基础模型包括:

1) 非精确计算;

2) 可变工作负载的弹性周期调度;

3) 基于反馈控制理论的软实时调度.

非精确计算模型将任务分割为两个部分:强制执行部分和可选执行部分.强制执行部分必须在任务的时限到来之前结束,以保证该任务的输出满足最低 QoS 需求.可选执行部分的执行可以有两种选择:其一是要么完全执行,要么放弃;其二是在时限到来之前尽可能执行,以最大可能地提高运算精度.当可选执行部分在时限到来之前运行完成时,则称该任务实现精确计算.非精确计算对于软实时系统而言有着重要的意义.当系统出现超载时,调度器选择放弃部分任务的可选部分的执行,以尽可能小的性能代价保证系统继续运行,并在系统负载降低后重新获得理想的系统输出.目前,研究人员已经提出了多种非精确计算模型,如 IC[1],IRIS[2],m/k-firm deadline[3]等,并在许多实验和实际系统中得到应用[4~6].

所谓可变工作负载是指,在工程实际中,任务可以拥有多个不同的作业周期.例如,对于一个实时控制任务,若存在多个可用作业周期,则当采用小周期作业时,任务的控制精度高,当采用大周期作业时,任务的控制精度低,但仍旧满足任务的最低 QoS 要求.文献[7]通过修改降低任务的作业频率,使任务集能够满足选定调度算法的可调度条件.文献[8]通过降低任务的作业频率来降低任务的作业丢失率.文献[9,10]提出了一系列在系统过载情况下提高任务周期的策略,但没有讨论在系统处于轻载情况下如何减小任务周期.文献[11,12]在固定任务作业时间的前提下讨论了在系统过载或轻载条件下,如何提高或降低任务的作业周期,并讨论在存在资源限制的情况下调度策略所作的调整.

非精确计算和弹性周期调度为不同的应用定制调度算法,而基于反馈控制理论的软实时调度算法为实时调度算法的设计提供了一个通用模型.通过建立系统的反馈控制模型,获得系统输入和输出之间的传递函数,设计者能够依靠经典的控制理论解决软实时系统的调度问题[13].文献[14~16]通过引入反馈控制提高 Web 服务器

的性能.文献[17]通过引入反馈控制提高 Lotus Notes 的性能.文献[18]利用反馈控制理论实现 IP 路由器的拥塞控制算法.文献[19]提出了反馈-前馈实时调度算法,利用反馈控制响应任务作业时间在系统运行过程中的动态变化,利用前馈控制处理新加入的系统负载.

值得注意的是,上述各种软实时调度算法均可看作分层调度算法,其上层算法的工作主要是调整任务的属性,而下层算法根据任务的属性,确定任务的优先级,并依此进行调度.下层算法通常采用经典的实时调度算法,如 RM 和 EDF.

2 弹性周期调度模型

在文献[11]中,作者针对可变工作负载的软实时任务,提出了弹性周期调度算法.作者将任务的资源利用率比做弹簧,其大小就是弹簧的长度.弹性周期调度模型假设任务的周期可以在一定范围内连续变动,而任务的作业时间预先确定,在系统运行过程中保持不变.弹性周期调度算法采用 EDF 作为下层的基础算法.在任务集的资源利用率小于 1 时(弹簧长度小于规定值),任务采用最小作业周期;当任务集的资源利用率超过 1 时(弹簧长度超过规定值),弹性周期调度增大部分任务的作业周期,降低其资源利用率(压缩弹簧),使任务集的资源利用率保持在 EDF 的可调度条件之内.当任务不再运行时,弹性周期调度减小部分任务的作业周期,提高其资源利用率(释放弹簧),提高任务的 QoS.在文献[12]中,作者放宽对任务属性的限制,认为任务的作业时间在系统的运行过程中动态变化.因此,为了能够更精确地计算任务的资源利用率,算法对任务的作业时间进行周期性的采样,并以采样值和现有的估计值的加权和作为任务下一个采样周期内的作业时间.

弹性调度算法是小粒度的软实时调度算法,每当任务的作业超过时限时,弹性调度算法都需要进行响应.因此,弹性调度模型虽然能够提供较高的控制精度,但是以较大的执行开销为代价.实际上,由于软实时任务与生俱来的容错特性,软实时任务的作业连续 n 次(n 小于确定值)超越时限不会导致任务的致命失效,而只是降低任务的 QoS.利用这种特性,采用粗粒度的调度策略,根据任务集在一段时间内的作业状况,调整任务的周期属性.只要确定合理的采样周期,任何任务在一个采样周期内的作业丢失次数均不会超过规定的门限,因此能够保证任务的 QoS 不低于用户的最低需求.依靠合理的设计,粗粒度的周期调整策略能够达到既降低调度算法的执行开销,又保证任务最小可接受 QoS 的目的.

弹性调度算法采用预先定义任务作业时间,或者采用估计值和采样值的加权和作为任务资源利用率的计算参数.软实时系统中任务的作业时间与被处理的数据量相关,其变动十分剧烈.因此,预先定义的属性不能反映系统真实的运行情况,调度算法难以合理分配资源.通过采样获取的信息动态地修改任务属性,使之较真实地反映任务的运行情况,能够提高调度算法的性能.这是以引入新的执行开销为代价的,比如对任务每一次作业时间的采样、新的估计值的计算等等.此外,如何确定估计值和采样值的权值也是算法设计者需要面对的问题.显然,采用更容易获得,而且更稳定的任务相关参数作为调度策略的输入参数,是避免引入估算任务作业时间这一额外开销的可行方法.

3 动态反馈弹性调度模型

目前,流媒体应用已经成为网络服务的重要组成部分.随着网络容量和数据处理算法的迅速发展,作为流媒体服务的提供者,流媒体服务器接收用户请求并提供服务的能力是决定流媒体应用系统性能的至关重要的因素.流媒体系统的设计者要求流媒体服务器在满足已连接用户的 QoS 要求的前提下,能够并发尽可能多的服务连接.我们将向用户提供服务的线程称为任务.在流媒体服务器中,任务可以具有几个离散的作业周期,任务按照不同的作业周期运行,提供不同的 QoS,采用的作业周期越短,则 QoS 越高.在按照一定作业周期运行时,只要任务的作业连续丢失数低于一定阈值,作业的丢失对 QoS 的影响就可以忽略.流媒体服务器的任务作业时间具有数据敏感性,在系统运行过程中动态变化.根据对现有软实时调度算法和流媒体服务器应用环境的分析,我们提出了面向可变工作负载的动态反馈弹性调度模型.

3.1 任务模型

动态反馈弹性调度的基本假设是:每个任务拥有 n 个离散的作业周期,任务能够在系统运行过程中动态地选择合适的作业周期.利用这一特性,选择不同的作业周期可以改变任务在确定的采样周期内的作业次数,进而改变任务的资源需求.

在我们的模型中,每个任务 τ_i 可以由多元组 $\tau_i(n,P_{i1},...,P_{in},W_i)$ 表示,其中 n 代表任务拥有的作业周期数, P_{ij} 代表任务 τ_i 的第 j 级作业周期, $j\in(1,n)$,作业周期按由大到小顺序排列.每个作业周期映射任务 τ_i 的一个 QoS 级别,高 QoS 对应小周期. W_i 代表任务的关键性属性,用以表示放弃任务的作业需要付出的代价.任务模型不包括作业时间参数,我们将利用任务在确定采样周期中的作业次数作为调度策略的参数.同时我们假设,除了处理器之外,任务间不共享其他资源.表 1 给出了一个适用于动态反馈弹性调度的典型任务集.

Table 1 Task set with flexible workload

表 1 可变工作负载任务集

Task	n	L_{i1}	L_{i2}	L_{i3}	W_i
τ_1	3	33	25	16	1
τ_2	2	25	16		2
τ_3	3	33	25	16	2

从表 1 可以看出, τ_1 和 τ_3 拥有 3 个 QoS 级别,而 τ_2 只有两个 QoS 级别.处于同样 QoS 级别的任务,其作业周期可以不同. τ_2 和 τ_3 拥有同样的关键性,且高于 τ_1 .

3.2 反馈调度模型

在系统运行过程中,任务的一次执行被称为作业,作业的运行超越规定的时限称为丢失.动态反馈弹性调度算法的调度过程可以表述为:调度器记录每个采样周期内执行的作业总数、丢失的作业总数和系统资源利用率;当作业的丢失次数大于 0 时,调度器增大部分任务的作业周期,通过减少下一采样周期的作业执行次数来降低系统在下一个采样周期的资源请求,达到避免作业丢失的目的;如果作业的丢失数等于 0,且系统的资源利用率小于 1,调度器通过减小部分任务的作业周期,直到所有任务都能够提供最佳 QoS,或系统资源利用率趋近于 1.动态反馈弹性调度算法不记录采样周期中任务每次作业的时间,而是记录整个采样周期内处理器资源被占用的总数.

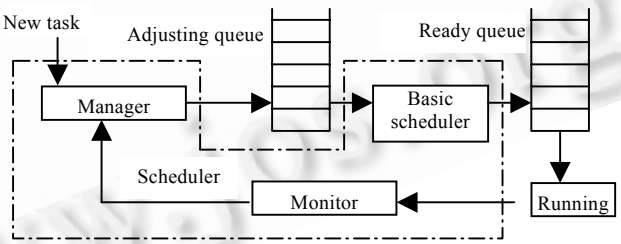

Fig.1 Model of dynamic feedback elastic scheduling

图 1 动态反馈弹性调度模型

如图 1 所示,动态反馈弹性调度由调度器、调整队列和就绪队列组成,而调度器又包括任务管理器、基本调度器和监视器.任务管理器是动态反馈弹性调度模型的核心,其功能包括访问控制和调整任务属性.任务管理器根据用户请求的资源需求和系统当前状态决定是否接受新的用户请求.任务管理器为被接受的请求创建相应的任务,并加入调整队列.任务管理器根据当前系统状态调整已接收任务的作业周期.当调整队列中的任务就绪时,基本调度器根据调整队列中任务的当前属性,将任务的作业加入就绪队列.监视器记录每个采样周期中执行的作业数目、丢失的作业数目和处理器被占用的时间.在每个采样周期结束时,监视器将采样结果传递给任务管理器.

3.3 模型结构和周期调整策略分析

3.3.1 任务队列

动态反馈弹性调度算法构建两种队列:就绪队列和调整队列.调整队列记录任务信息,包括任务拥有作业周期级别数、当前的作业周期和关键性属性.任务管理器按照任务的最大作业周期级别数 n,定义 n 个子队列,包括 1 个极限队列(队列号为 0)和 $n-1$ 个可变队列(队列号为 1~n–1).根据表 1,任务集的调整队列拥有 3 个子队列,包括 1 个极限队列和 2 个可变队列.如果 τ_2 当前的作业周期为 16,则 τ_2 属于子队列 1.如果 τ_1 当前的作业周期为 16,则 τ_1 属于子队列 2.极限队列中的任务,其周期为任务的最大作业周期,不可再增大.就绪队列记录作业信息,包括作业的就绪时间和时限.就绪队列中的作业按照优先级从高到低排列.

3.3.2 基本调度器

基本调度器根据作业的时限,确定就绪队列中作业的优先级.动态反馈弹性调度采用 EDF(最早时限优先)算法作为基本调度器的调度算法.EDF 根据当前时刻与作业时限的距离,由小到大排列作业.由于事先并未确定作业的执行时间,基本调度器不能确定当前资源是否满足作业要求,只能在作业超越时限时将作业放弃.当作业超越规定时限时,所在采样周期的作业丢失数加 1.显然,被放弃的作业浪费了一定数量的处理器时间,降低了系统资源的有效利用率.动态反馈弹性调度在满足用户 QoS 需求的前提下,通过增大任务的作业周期,减少下一采样周期中的作业数量,降低作业的丢失数,提高了系统资源的有效利用率.

3.3.3 监视器与采样周期

监视器记录每个采样周期内的作业执行数 N、作业丢失数 e 和资源利用率 U. N 是本采样周期内就绪的作业总数. e 是指本采样周期内超越时限的作业总数.监视器记录采样周期内被占用的系统资源(处理器时间)总数 t,然后计算该采样周期的系统资源利用率:

$$U=\frac{t}{T} \tag{1}$$

其中 T 为监视器的采样周期.通过式(1)可以获得上一个采样周期内的空闲资源状况,以此为根据,任务管理器可以决定是否接收新的服务请求,或提高现有任务的 QoS.需要强调的是,我们不关心每一次作业占用的处理器时间,而是从宏观上把握采样周期内的资源占用情况,因此动态反馈弹性调度的任务模型不需要引入作业的执行时间参数.

监视器的采样周期决定动态反馈弹性调度模型的控制粒度,是影响调度性能的重要因素.采样周期设置过大,任务在采样周期内的作业丢失数可能超过规定范围,导致任务失效.采样周期设置过小,则控制频率过高,增大了调度算法的运行开销.因此,在保证任意任务在采样周期内的作业丢失数小于任务能够容忍的最大作业丢失数的前提下,应该选取尽可能大的采样周期.设存在由 n 个任务组成的任务集 S,每个任务拥有 m 个周期级别,其任务属性如下:

$$\{\tau_1(m,P_{11},...,P_{1m},W_1),\tau_2(m,P_{21},...,P_{2m},W_2),...,\tau_n(m,P_{n1},...,P_{nm},W_n)\}.$$

设任务能够容忍的最大作业丢失数为 $Lost$,则动态反馈弹性调度的采样周期是

$$P=Lost\times\min_{(i,j)}(P_{ij}),\quad i\in n,j\in m \tag{2}$$

式(2)以任务的最小作业周期与 $Lost$ 的乘积作为采样周期,能够确保采样周期内任意任务的作业丢失数不超过能够容忍的最大作业丢失数 $Lost$.显然,这是较为悲观的假设.在实际的调度过程中,可以适当延长采样周期,即

$$P=\lfloor\alpha\times Lost\rfloor\times\min_{(i,j)}(P_{ij}),\quad i\in n,j\in m \tag{3}$$

其中 α 是设定动态反馈弹性调度算法的采样周期的参考系数,$\alpha\le 1.5$. α 越大,则采样周期越小,算法控制精度越高,运行开销越大.

3.3.4 任务管理器

任务管理器是动态反馈弹性周期调度器的核心部分.与文献[7~12]依靠任务的资源利用率和作业丢失率不同,本文的任务管理器将任务集在采样周期中的作业总数和作业丢失总数作为任务管理器的反馈值,任务集在

下一个采样周期中的作业总数作为被控参数.这是因为,我们的任务模型不提供任务的作业执行时间参数,任务管理器不能预先确定任务的每一次作业的运行时间,但能预先确定作业的周期.通过确定作业周期,任务管理器能够确定任务在下一个采样周期内的作业次数,进而控制系统资源利用率和任务丢失率.这里,我们定义作业丢失数影响因子 K,则下一个采样周期作业减少的次数是本采样周期作业丢失次数的 K 倍, K 大于或等于 1.

下面,我们将讨论影响动态反馈弹性周期调度性能的主要因素.为了简化讨论,我们假设任务的作业执行时间是固定的.设存在任务集 S,参数见表 2.

Table 2 List of tasks with fixed executing time

表 2 执行时间固定的任务列表

	Period 1	Period 2	Executing time
Taks 1	10	5	4
Taks 2	10	5	2
Taks 3	10	5	1

情况 1.对于任务集 S,当所有任务均采用最小作业周期时,必然出现作业丢失.但是作业执行顺序不同,丢失数不同.设采样周期为 20,由于作业优先级相同,当 EDF 首先调度最长作业时,则作业的丢失数为 8,若首先调度最短作业,作业丢失数为 4.

结论 1.当系统过载时,优先级相同但执行时间不同的作业,如果调度顺序不同,作业丢失数也会不同.首先调度短作业可以在同等条件下降低作业丢失数.

情况 2.设 $K=1$,对于任务集 S,当作业丢失数为 4 时,任务管理器延长任务 2 和任务 3 的作业周期至 10 时,任务集仍旧出现作业丢失,而延长任务 1 和任务 2 的作业周期至 10 时,任务集的资源利用率小于 1,下一采样周期不存在作业丢失.

结论 2.对于减少同样数量的作业执行次数,选择不同的任务进行周期调整,其调整效果不同.改变长任务的周期,调整效果更好.

情况 3.设 $K=1.5$,对于任务集 S,当作业丢失数为 4 时,任务管理器延长任务 2 和 3 的作业周期至 10 时,任务集的资源利用率小于 1,下一采样周期不存在作业丢失.

结论 3.作业丢失数影响因子是影响任务管理器性能的重要因素.

情况 4.对于任务集 S,如果调度算法的采样周期为 24,则作业的执行可能跨越两个采样周期.设所有任务都采用最小作业周期,任务的初次作业均在 0 时刻就绪,EDF 算法首先调度最短作业.当到达 24 时,作业 2 和作业 3 已经运行结束,而作业 1 正在运行,并将在 25 被放弃.在第 0 个采样周期,共执行 15 次作业,其中丢失 4 次.当 $K=1$ 时,任务管理器通过调整任务 2 和任务 3 的运行周期,使下一采样周期的作业执行减少 4.8 次,但仍会出现作业丢失.当 $K=1.5$ 时,任务管理器需要调整任务 1~任务 3 的作业周期,这将使作业执行次数的减少大于 6 次,所以在下一采样周期不会出现作业丢失.

结论 4.作业的执行跨越采样周期边界将影响作业丢失数采样的精度.

情况 5.对于任务集 S,当任务管理器调整所有任务的作业周期,使之采用最大作业周期时,根据表 2 给出的作业执行参数,不会出现作业丢失.然而,如果作业 3 的执行时间变为 5,任务集的总资源需求大于系统能够提供的系统资源,将会出现作业丢失.

结论 5.作业执行时间的动态变化使任务管理器通过一次丢失数采样和作业周期调整就能避免在以后的调度算法采样周期中不再发生作业丢失的希望变得渺茫.

由以上讨论可知,减少调度算法采样周期内的作业总数与降低和消除作业丢失数间存在着复杂的对应关系,该关系受多方面因素的影响,难以利用一个简单的数学公式加以描述.同时,我们认为,就动态反馈弹性调度而言,准确地描述这一关系是没有必要的.因为,流媒体应用的容错特性使任务具有在部分数据延迟或丢失的情况下继续提供满足用户基本 QoS 需求的能力.同时,流媒体服务器等软实时应用的性能评价并不是以对某个用户提供服务的质量优劣作为标准的,而是考察向一组用户提供服务时表现出来的综合能力.因此,动态反馈弹性调度的设计目标并非针对每一个独立的任务,也非通过一次采样和周期调整就完全消除任务丢失,而是通过连续的采样和控制,逐渐逼近当前任务集的最佳作业状态.从控制理论的角度看,评价动态反馈弹性调度的标准不

再只是任务丢失率和任务拒绝率,而是对系统动态特性和稳态特性的综合考虑.在本文中,动态反馈弹性调度的动态特性主要考察任务集从异常状态(存在作业丢失)进入正常状态(无作业丢失)的收敛速度,而稳态特性主要考察任务集在正常情况下运行时,真实作业状态与最佳作业状态间的偏差.

3.4 周期调整算法

任务管理器根据作业丢失数和资源利用率采样值,利用周期调整算法动态地调整部分任务的作业周期属性,达到避免作业丢失、提高系统的并发服务数量、充分利用系统资源的目的.周期调整算法的设计目标是:1)在异常状态下,系统的运行能够快速地收敛到正常状态;2) 在正常运行且存在空闲资源时,提高系统资源利用率和服务质量.

对于目标 1,周期调整算法以第 $k-1$ 个采样周期中的作业丢失数 e_k 作为输入,通过延长部分任务的作业周期,将第 k 个采样周期中将要运行的作业总数减少 Ke_k 个,使系统向正常状态收敛. K 是丢失数影响因子, K 大于或等于 1.算法 1 给出了周期延长算法.

算法 1. 周期延长算法.

```
task_queue[n];                  //n−1 个可变队列和一个极限队列
float K;                        //K 是丢失数影响因子
int min_num;                    //min_num 是最小调整数
int T,P;                        //T 是监视器采样周期,P 是被调整任务的作业周期
                                //周期随下标增长而减小
extend_period(lost)             //lost 是丢失任务数
{
    if (lost<min_num)       lost=min_num;
    e=K*lost;
    result = get a task node from task_queue[i];      // task_queue[i]是队列号最高的非空可变队列
    while(result==true&&i!=0)
        {
            e=e−(T/P_{i−1}−T/P_i);
            P=P_{i−1};
            insert the adjusted node into a task_queue[i−1];
        result = get a task node from task_queue[i];
        if (e<=0)    return;
        if (task_queue[i]为空)   i=i−1;
    }
    get a task node from task_queue[0] with lowest critical;
    do
    {
            e=e−T/P_0;
            give up this node;
            get a task node from task_queue[0] with lowest critical;
    }while(e>0)
}
```

周期延长算法从队列号最高的非空可变队列开始,逐一延长该队列中任务的作业周期,计算减少的作业数并更新 e,然后将调整后的任务加入低一级可变队列的末尾.当所有可变队列均为空,且 e 不为 0 时,周期延长算法通过放弃关键性最低的任务,减少下一采样周期的作业总数.对于周期延长算法,以下 3 点需要着重指出.

(1) 当作业丢失数较小时,任务管理器对任务的调整力度也较小,导致系统状态的收敛较为缓慢.为使系统状态在轻度过载的情况下也能较快地收敛,本文引入了最小调整数,

$$\begin{cases} e_k = e_k, & \text{如果} e_k \geq \min\_num \\ e_k = \min\_num, & \text{如果} e_k < \min\_num \end{cases} \tag{4}$$

(2) 由第 3.3.4 节中的结论 1 和结论 2 可知,EDF 算法在同等条件下首先调度短作业,任务管理器在同等条件下首先调整长任务,能够有效地提高调度算法的性能.由于动态反馈周期调度的任务模型不包括作业时间属性,因此上述手段失去了实现的基础.为弥补由于同等条件下调度顺序和调整顺序的不确定给算法性能带来的

影响,根据第 3.3.4 节中的结论 3,本文采用较大的作业丢失数影响因子.

(3) 动态反馈弹性调度要求用户定义请求的关键性.当依靠周期延长无法使系统收敛至正常状态时,任务管理器以任务关键性作为标准,放弃部分关键性低的任务以满足其他任务的作业的资源需求.需要指出的是,调整队列中的任务的关键性永远高于新的服务请求的关键性,任务管理器不能为了接收新的服务请求而放弃调整队列中的任务.

对于目标 2,周期调整算法以第 $k-1$ 个采样周期中的资源利用率为输入,压缩部分现有任务的作业周期,使第 k 个采样周期中的作业总数上升,达到提高第 k 个采样周期的资源利用率和服务质量的目的.算法 2 给出了周期压缩算法.

算法 2. 周期压缩算法.

```
task_queue[n];                  //n−1 个可变队列和一个极限队列
int count;                      //count 是记数值
int T,P;                        //T 是监视器采样周期,P 是被调整任务的运行周期
                                //周期随下标增长而减小
compress_period(count)
{
    j=count;
    while(j>0&&低于最高可变队列非空)
    {
        get a task node from task_queue[i];        //task_queue[i]是调整级别最低的非空队列
        P=P_{i+1};
        insert the adjusted node into a task_queue[i+1];
        j−−;
    }
}
```

周期压缩算法从队列号最低的非空队列开始,逐一压缩该队列中任务的作业周期,并更新记数值 count,然后将调整后的任务加入高一级队列的末尾.重复上述操作,直到 count 为 0,或除最高级可变队列外的所有队列均为空.

由第 3.3.4 节中的结论 4 可知,作业执行跨越采样边界影响了作业丢失数的采样精度.为解决此问题,我们将记录每个采样周期内作业就绪的次数改为记录每个采样周期内作业结束的次数.即通过控制下一个采样周期将要结束的作业的数量,达到控制资源需求总量的目的.与控制作业的就绪次数相比,控制作业的结束次数使系统运行能够更快地从异常状态收敛至正常状态.

除了调整当前任务集中任务的作业周期以外,任务管理器还需要处理新的服务请求.任务管理器只在每个采样周期的边界处理新的服务请求,以避免采样周期内任务的动态增加对原有任务的作业产生影响.算法 3 给出了周期调整算法针对不同系统状态,处理当前任务集和新的服务请求的伪代码实现.

算法 3. 周期调整算法.

```
request_queue;                  //服务请求队列
float ε;                        //ε是控制偏差
int step,T,P;                   //step 是调整步长,T 是监视器采样周期,P 是被调整任务或用户请求的运行周期
Adjust_algorithm(lost,U,request_queue)
{
    if (lost>0)
    {
        extend_period(lost);        //首先满足当前任务集的调整需求
        while(request_queue!=NULL)  //处理新的服务请求
        {
            get a request from request_queue;
            e=K*(T/P_0);
            While(可调整队列不为空&&e>0)
            {
```

```
                    get a adjustable node;
                e=e–T/P_i;
                insert the adjusted node into a task_queue[i–1];
                }
                if (e<=0) insert the new node into a task_queue[0];
                if (调整队列为空) return;
            }
        }
        else
        {
            if (lost=0&&U<=1–ε)
              {
              count=step;
              while(count>0&&rquest_queue!=NULL)          //首先处理新的任务请求
                    {
                        get a request from request_queue;       //task_queue[i]是调整级别最低的非空队列
                        P=P_0;
                        insert the requested node into a task_queue[0];
                        count––;
                    }
              if (count>0) compress_period(count);
              return;
            }
            else
                {
                    return;
            }
        }
    }
```

算法 3 显示了当存在作业丢失时,周期调整算法首先根据作业丢失数延长现有任务的作业周期,减少下一个采样周期的作业丢失数.如果现有任务均处于极限队列,且作业数的减少未达到需求,周期调度算法以任务关键性作为标准,通过放弃部分关键性低的任务以降低下一采样周期中的作业执行总数.上述操作结束后,周期调整算法通过延长处于可变队列中任务的作业周期减少作业数,以便容纳新的服务请求.服务请求按照最大周期计算新增作业数.在这一操作过程中,现有任务减少的作业数等于服务请求新增的作业数.当所有当前任务均处于极限队列时,周期调度算法停止接受新的服务请求.现有任务的关键性永远高于服务请求的关键性,任务管理器不能为了接收新的服务请求而放弃当前任务.

当资源利用率降到 $1-\varepsilon$ 以下时,周期调整算法首先处理新的服务请求.调整步长 step 是在采样周期边界被接受的用户请求数.如果用户服务请求总数小于 step,周期调度算法压缩现有任务的作业周期以提高其服务质量.被压缩的任务数等于 step 与用户服务请求总数的差,记为 count.

ε是周期调度算法的控制偏差.当系统资源利用率接近 1 时,由于作业执行时间的不确定,系统状态可能在接近满载和过载间波动,导致任务管理器在相邻采样周期交替地执行周期延长和压缩操作.为减少这一现象的出现概率,我们引入控制偏差ε.只有当资源利用率降到 $1-\varepsilon$以下,任务管理器才接受新的服务请求,或压缩当前任务的作业周期.虽然周期调度算法的任务模型不包含运行时间属性,运行过程中也不考虑作业的执行时间,但是系统的设计者能够预先估计任务的资源利用率峰值,并以此为根据确定控制偏差.当单个任务的资源利用率峰值较高时,如等于 0.1,周期调整算法的控制偏差大于 0.1.在这种情况下,任务集的资源利用率可能长期处于 90%左右,系统资源不能得到充分利用.然而,在流媒体等实际应用中,任务的资源利用率峰值等于甚至小于 0.01.对于上述应用,设 $\varepsilon=0.02$,任务集的资源利用率能够保持在 98%附近.

调整步长 step 控制每个采样周期内增加的任务数,以避免由于增幅过大而导致系统资源利用率饱和继而作业丢失.周期调整算法根据任务的资源利用率峰值的估计值,确定调整步长.周期调整算法按照调整步长创建新任务和压缩现有任务,经过多个采样周期,使系统资源利用率逐渐逼近 $1-\varepsilon$.

任务管理器将尚未得到处理的用户请求放在等待队列中.用户请求具有确定的等待时限.如果请求在等待时限到来时仍未得到处理,任务管理器将放弃该请求,并向用户发送相应的系统信息.

在第 4 节中,我们将通过模拟实验评价动态反馈弹性周期调度算法的调度参数——丢失数影响因子,以及调整步长的定义对系统性能的影响.

4 模拟实验及结果分析

动态反馈弹性周期调度算法是一种启发式的调度算法,算法参数和任务属性动态变化的幅度是影响算法性能的主要因素.在本节中,利用实时调度算法模拟平台[20],我们将通过模拟实验观察算法参数变化对算法性能的影响.

在模拟实验中,每个软实时任务的基本属性见表 3.每个任务拥有 3 个可用的作业周期.软实时任务的作业执行时间在最大执行时间和最小执行时间之间平均分布.任务就绪时,由调度模拟器在执行时间范围内随机选择本次作业的执行长度.

Table 3 The basic parameters of soft real-time tasks

表 3 软实时任务的基本属性

Parameters	Period 1	Period 2	Period 3	Executing time (Max)	Executing time (Min)
Value	600	450	350	4	1

调整步长对调度算法性能的影响如图 2 所示.任务集的初始最大资源需求是 200%,调度器将所有任务的作业周期均定义为 450.从图中可以看出,若调整步长大,则调度的收敛时间短.但是,大的调整步长也会给算法带来一些不利的影响.当控制偏差 ε =0.04,调整步长为 40 时,系统的资源利用率在 4800 处低于 96%,算法将继续调整部分任务的作业周期,导致在[4800,6000]间出现作业丢失.调整步长为 20 和 30 均能避免上述情况的发生.因此,在实际应用中,设计者可以对调整步长进行分段定义.资源利用率较低时采用较大的调整步长,提高收敛速度.资源利用率接近 1−ε 时,采用较小的调整步长,避免系统过载.

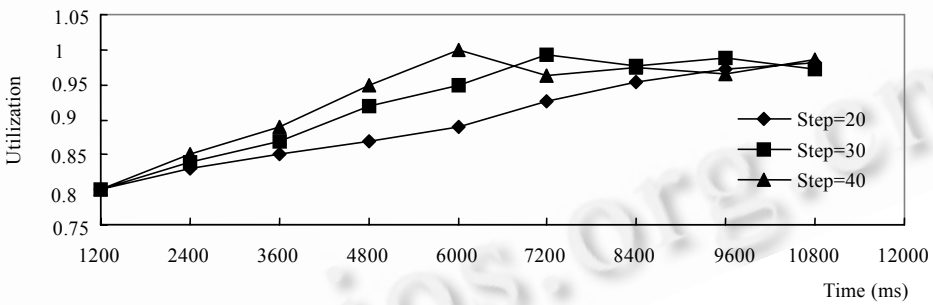

Fig.2 Resource utilization under different adjusting steps

图 2 不同调整步长下的资源利用率

任务丢失数影响因子对调度算法性能的影响如图 3 所示.任务集的初始最大资源需求是 200%,调度器将所有任务的作业周期均定义为 350.从图中可以看出,当 K=1 时,调度算法经过一次采样和调整并不能消除作业丢失的情况.如果算法对系统运行状态作更悲观的假设,当设 K=1.2 和 K=1.5 时,系统能够以更快的速度进入正常运行状态.这是以过度降低任务的 QoS 为代价的.

从图 2 和图 3 中我们还可以观察到,系统进入稳定状态后,系统资源利用率在饱和状态(利用率为 1)附近波动.当出现作业丢失时,调度算法通过微调,使系统返回正常状态.

5 结论及展望

本文提出的动态反馈弹性周期调度模型及其核心算法——周期调整算法,是以可变负载的软实时任务为调度对象,其设计目标是在满足软实时任务规定 QoS 的前提下,支持尽可能多的并发服务,达到系统资源的充分利用.周期调整算法利用周期采样获得的作业完成数、作业丢失数和资源利用率,通过修改任务的作业周期,达

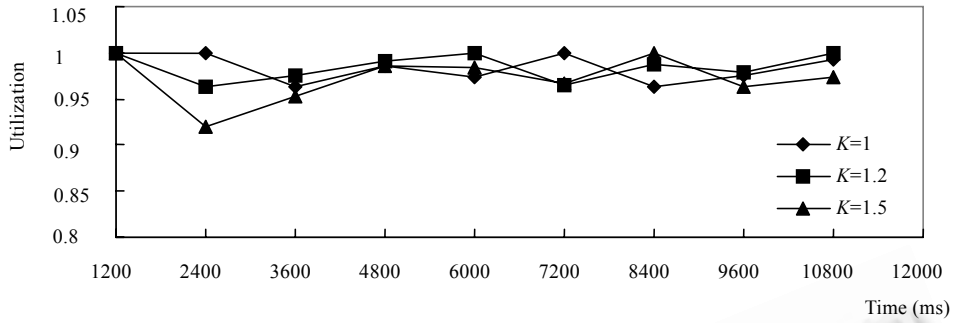

Fig.3 Resource utilization under different lost factors

图 3 不同丢失数因子下的资源利用率

到控制任务的资源需求、合理分配系统资源的目的.软实时系统在过载条件下的收敛速度和正常状态下的稳定性是评判动态反馈弹性周期调度算法性能的主要标准.

通过算法分析和模拟实验可知,算法参数和任务属性动态变化的幅度是影响动态反馈弹性周期算法性能的主要因素.获取较为准确的运行参数,使算法以较高的性能运行是算法投入实际运行面临的主要问题.动态反馈弹性周期调度算法是一种启发式的调度算法,参数的获得需要通过大量的现场实验.那么参数的获取是否存在更简单的方法呢?建立流媒体服务器的调度模型,依靠反馈控制理论设计调度控制器,通过理论分析获取算法参数和评价算法性能将是我们下一步工作的重点.

作业丢失数与作业完成数之间存在着十分复杂的非线性关系,本文采用启发式调度,而没有对上述关系进行深入研究.显然,过于粗糙的简化是影响算法性能的一个重要因素.在今后的工作中,我们将通过大量实验,利用系统辨识的方法,尽可能准确地掌握作业丢失数与作业完成数之间的关系,作为进一步优化算法、提高算法性能的基础.

致谢 在此,我们向对本文的工作给予支持和建议的同行表示感谢.

References:

[1] Liu JWS, Shin WK, Lin KJ, Bettati R, Chung JY. Imprecise computations. Proc. of the IEEE, 1994,82(1):83~94.

[2] Dey JK, Kurose J, Towsley D. On-Line processor scheduling for a class of IRIS real-time tasks. IEEE Trans. on Computers, 1996,45(7):217~228.

[3] Hamdaoui M, Ramanathan P. A dynamic priority assignment technique for streams with (*m*,*k*)-firm deadlines. IEEE Trans. on Computers, 1995,44(12):1443~1451.

[4] Aydin H, Melhem R, Mossé D, Mejía-Alvarea P. Optimal reward-based scheduling for periodic real-time tasks. IEEE Trans. on Computers, 2001,50(2):111~130.

[5] Chen Y, Xiong GZ. Imprecise computation fault-tolerant rate-monotonic scheduling. In: Zhou WL, ed. Proc. of the 5th Int'l Conf. on Algorithms and Architectures for Parallel Processing. Beijing: IEEE Computer Society, 2002. 278~285.

[6] Chen Y, Yu X, Xiong GZ. Fault-Tolerant earliest deadline first scheduling with resource reclaim. In: Zhou WL, ed. Proc. of the 5th Int'l Conf. on Algorithms and Architectures for Parallel Processing. Beijing: IEEE Computer Society, 2002. 290~293.

[7] Kuo T-W, Mok AK. Load adjustment in adaptive real-time system. In: Jeffay K, ed. Proc. of the 12th IEEE Real-Time System Symp. San Antonio: IEEE Computer Society, 1991. 160~170.

[8] Nakajima T, Tezuka H. A continuous media application supporting dynamic QoS control on real-time mach. In: Dittai Z, ed. Proc. of the ACM Multimedia'94. San Francisco: ACM Press, 1994. 289~297.

[9] Lee C, Rajikumar R, Mercer C. Experiences with processor reservation and dynamic QoS in real-time mach. In: Aigrain P, ed. Proc. of the ACM Multimedia'96. Boston: ACM Press, 1996. 22~31.

[10] Beccari G, Caselli S, Reggiani M, Zanichelli F. Rate modulation of soft real-time tasks in autonomous robot control systems. In: Burns A, ed. Proc. of the 11th Euromicro Conf. on Real-Time Systems. New York: IEEE Computer Society, 1999. 21~28.

[11] Buttazzo GC, Lapari G, Caccamo M, Abeni L. Elastic scheduling for flexible workload management. IEEE Trans. on Computers, 2002,51(3):289~302.

[12] Buttazzo G, Abeni L. Adaptive workload management through elastic scheduling. Real-Time Systems, 2002,23(3):7~24.

[13] Lu CY, Stankovic JA, Son SH, Tao G. Feedback control real-time scheduling: Framework, modeling and algorithms. Real-Time Systems, 2002,23(3):85~126.

[14] Abdelzaher TF, Bhatti N. Web server QoS management by adaptive content delivery. In: Bhatti S, ed. Proc. of the Int'l Workshop on Quality of Service. London: IEEE Computer Society, 1999. 216~225.

[15] Abdelzaher TF, Lu CY. Modeling and performance control of internet servers. In: Bitmead B, ed. Proc. of the 39th IEEE Conf. on Decision and Control. Sydney: IEEE Computer Society, 2000. 2234~2239.

[16] Lu CY, Abdelzaher TF, Stankovic JA, Son SH. A feedback control approach for guaranteeing relative delays in Web servers. In: Ho JM, ed. Proc. of the IEEE Real-Time Technology and Application Symp. Taipei: IEEE Computer Society, 2001. 51~62.

[17] Parekh S, Gandhi N, Hellerstein J, Tilbury D, Jayram TS, Bigus J. Using control theory to achieve services level objectives in performance management. In: Zimmer W, ed. IFIP/IEEE Int'l Symp. on Integrated Network Management. Seattle: IEEE Computer Society, 2001. 841~854.

[18] Hollot CV, Misra V, Towsley D, Gong WB. A control theoretic analysis of RED. In: Izmailov R, ed. Proc. of the IEEE Infocom. Anchorage: IEEE Computer Society, 2001. 1510~1519.

[19] Cervin A, Eker J, Bernhardsson B, Årzén KE. Feedback-Feedforward scheduling of control tasks. Real-Time Systems, 2002,23(3):25~53.

[20] Chen Y. Research on supporting techniques for high dependable fault tolerant real-time systems [Ph.D. Thesis]. Chengdu: University of Electronic Science and Technology of China, 2002 (in Chinese with English abstract).

附中文参考文献:

[20] 陈宇.高可靠容错实时系统的支撑技术研究[博士学位论文].成都:电子科技大学,2002.