集群系统中实现软实时服务机制的研究与评估

常光辉[1],陈蜀宇[2],汤 羽[3],卜长清[1]
CHANG Guang-hui[1],CHEN Shu-yu[2],TANG Yu[3],BU Chang-qing[1]

1.重庆大学 重庆计算机学院,重庆 400045
2.重庆大学 软件工程学院,重庆 400045
3.美国华盛顿大学 计算机系,美国
1.College of Computer Sciences,Chongqing University,Chongqing 400045,China
2.College of Software Engineering,Chongqing University,Chongqing 400045,China
3.Department of Computer Science,The George Washington University
E-mail:sychen@cqu.edu.cn

CHANG Guang-hui,CHEN Shu-yu,TANG Yu,et al.Scheduling scheme and performance assessment for timely responsive service on cluster server.Computer Engineering and Applications,2007,43(25):133-139.

Abstract: This paper presents three real-time scheduling schemes for cluster servers:centralized scheme,distributed scheme and pull-based distributed scheme.Distributed scheme deploys scheduling on the end-node,while the scheduling work of centralized scheme is on the front-nodes,and that makes the former has a greater scalability.Pull-based distributed scheme makes end-nodes ask for events and direct load balance by themselves,to improve the scalability and efficiency of real-time cluster system.Lots of experimental results have been presented,illustrating the three scheduling schemes are benefit for decreasing the response time and maximizing the system throughout.

Key words: cluster server;real-time service;scheduling schemes;real-time response characteristic

摘 要:研究了集群架构服务器的三种实时调度策略,即集中式策略、分布式策略和主动型全分布策略。与调度功能主要集中在前端机的集中式策略比较,分布式策略将调度功能部分离散至后端节点,改善了系统的可扩展性。而主动型全分布式调度则进一步实现了调度分派功能的完全离散化,通过后端机的主动拉取事件和频率调控实现了负载平衡的自调节,极大地改善了系统的扩展性和服务效率。通过试验研究和试验表明,这三种都有效地降低了集群的响应服务请求的时间,还保证了系统的输出最大化。

关键词:集群服务器;实时服务;调度策略;实时响应特性
文章编号:1002-8331(2007)25-0133-07 **文献标识码**:A **中图分类号**:TP393

1 引言

高速网络系统(如千兆、万兆网络)和高性价比商用处理器的出现,使集群体系结构成为高性能计算和重大应用领域的强有力的竞争者。集群结构基于开放式的软件体系标准,采用规模化生产的通用商品化硬件,这使其具有了很好的性价比。并且,根据用户对计算需求的增加,能够便利地扩展集群系统的规模,最充分地满足用户对计算能力的要求。目前,集群体系结构不仅用于高负载的互联网服务器和搜索引擎(如 Yahoo,MSN 和 Google 等网站),同时还在高性能科学计算,QoS 服务,高速路由器和生产自动控制领域[1-4]得到广泛应用。在传感器网络(Sensor Network)[5]这类环境中,中央服务器不仅需要支持大容量、高密度的荷载,更要求对具有不同时限要求的传感器事件提供即时响应。比如,监测战场敌方车辆移动的传感器信息就应得到最优先的服务。对这类重要信息的响应延迟,就可能导致灾难性的后果。由此,引发了对集群架构服务器上实现实时服务的思考。

以支持多用户、资源共享、系统产出最大化为目标的现代通用型操作系统(如 UNIX,Linux)由于设计目标的不同,缺乏有效的内核实时支持功能,难以满足不同类型任务对响应时限的要求,不能直接支持集群架构下实时服务程序的开发。而传统的应用于航空控制,生产自动化,情报指令系统,金融在线交易领域的实时操作系统(RTOS)往往设计用于嵌入式系统,采

基金项目:国家高技术研究发展计划(863)(the National High-Tech Research and Development Plan of China under Grant No.2003AA116010);国家教育部新世纪人才支持计划(the New Century Excellent Talent Foundation from MOE of China under Grant No.NCET-04-0843);重庆市自然科学基金(the Natural Science Foundation of Chongqing City of China under Grant No.2005BB2192)。

作者简介:常光辉(1980-),男,硕士研究生,主要研究方向为分布式计算,实时系统等;陈蜀宇(1963-),男,博士导师,教授,主要研究方向为计算机网络与通信、智能计算机、容错与诊断、信息安全;汤羽,计算机科学博士候选人,计算机科学硕士,美国DRS Electronic Systems 公司高级软件工程师,主要从事计算机网络协议,集群体系,实时计算,分布式系统性能评估等方面的研究;卜长清(1983-),男,硕士研究生,主要研究方向为分布式计算,实时系统等方面的研究。

用非通用标准,在扩展性和性价比上存在着严重缺陷。另外,嵌入式系统内存和储存空间的限制也导致系统扩展性差,不能够支持分布式多节点的集群服务器。

在算法方面,实时操作系统的固定优先级算法(Rate-Monotonic)和最早时限优先算法(Earliest-Deadline-First)[6]在集中系统资源保证实时任务时限要求的同时,也导致了系统资源利用率低下的缺陷[4],且有可能过分忽略非实时任务而使其与服务无缘(service starvation)[7,8]。另一方面,现有的集群服务器(如Linux虚拟服务器[9])调度算法,如加权轮循(Weighted Round Robin),最少连接优先(Least-Connection-First),和加权最少连接优先(Weighted Least-Connection-First)等,主要目标是负载平衡(Load Balance),实现系统输出最大化,但对于实时和非实时任务缺乏区分,无法满足实时任务对响应时限的要求。

上述种种不足促使我们决定研究集群实时调度机制,在对实时任务提供软实时即时服务的同时也兼顾非实时任务,实现系统总体性能的优化。

文章其他部分组织如下:第二章描述实时集群服务器的体系结构,第三章提出三种调度策略,第四章给出这三种调度策略的实验结果和分析,并与现今流行的负载均衡算法进行了比较,第五章给出小结并讨论下一步的工作。

2 系统结构

实时集群服务器(RTCS)系统由三部分组成:传感器、前端机和后端机(图1)。传感器向集群系统前端机发送服务请求(又叫事件)。前端机可以看作是集群系统与外部的接入点,它接收外部传感器发来的事件,并按照设定的策略分发给后端机。后端机是实际处理任务的节点,类似于LVS系统的Real Server。前端机和后端机之间通过局域网连接并采用TCP套接字实现数据传递。

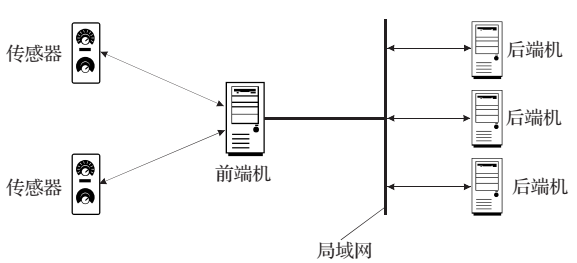


图1 实时集群服务器(RTCS)体系结构

目前常见的集群架构模式有三种:地址翻译(Network Address Translation,NAT),IP隧道(IP Tunneling)和直接转发模式(Direct Routing)。在地址翻译(NAT)模式[10]中,前端机需要安装双网卡,一个赋予集群虚拟地址 用于对外通信,另一个赋予本地局域网地址用于与后端机联接。前端机收到用户请求包后,将请求包中虚拟服务器的IP地址转换为某个选定后端机的IP地址,转发给后端机;后端机将响应包发回给前端机,再由前端机将回送给用户。集群服务器研究目前采用的是具有单一接入点的NAT构型。

3 集群调度策略

资源管理和调度策略是在集群体系上建立实时服务系统需要考虑的最关键的两个问题。资源管理考虑的重点是如何将系统资源(包括服务节点,网络带宽,CPU时间等)分配给各种时限要求不同的任务类型,调度策略则决定各个任务(事件)接受服务的顺序和时间。在NAT模式下,系统资源管理的有两种方式,即服务器分组(server partition)[11,12]和服务分隔(service isolation)[13]。服务器分组是把后端机分成几个子集群,每个子集群专门处理某一类型任务,仍然属于资源预留的方式。服务分隔不再对后端节点分组,而是让每一个后端机服务各种类型任务,但在每一个后端机上通过优先级区分的调度策略对不同性质的任务提供优先权区分的服务,从而达到优化系统资源分配,满足不同时限要求的目标。

对应于这两种资源管理方式,设计了三种在集群体系上提供软实时服务的调度策略,即集中式策略(Centralized Scheme,CS),分布式策略(Distributed Scheme,DS)和主动型全分布策略(Pull-Based Distributed Schedule,PBDS)。下面分别对这三种调度策略进行阐述。

3.1 集中式调度策略

集中式调度策略属于被动式类型,其特点是所有或大部的任务分派和资源调度的工作均由前端机完成,后端机只是被动地接受前端机分派的任务,提供相应的服务。因此前端机必须掌握完整的系统状态数据,并频繁地与后端机交换信息,这往往造成前端机负荷过重,成为系统性能扩展的瓶颈。集中式调度策略将后端服务器分成实时和非实时两个子集群,前端机上的实时和非实时两个调度器分别将实时和非实时事件对应分发给两个后端子集群,后端机则以简单的先进现出(FIFO)策略提供服务(图2)。

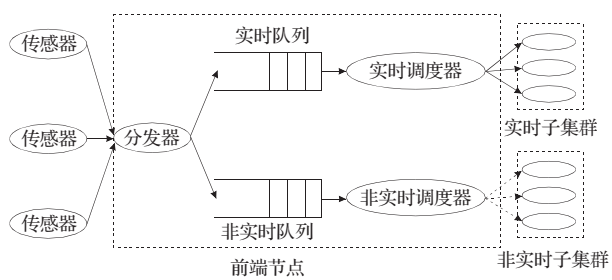


图2 集中式调度策略

传统的集群负载均衡算法在分发任务时,采用的是负载均衡机制,以后端节点载荷平衡为目标,追求系统产出的最大化。这样的算法不适合需要提供实时响应服务的体系。因此在设计RTCS系统时,考虑到实时和非实时事件响应时限的差别,有针对性地制定了不同的调度策略。实时事件希望能得到最快响应,因此对实时子集群采用最短响应时间优先算法(Shortest-Response-Time-First),总是将实时任务分派到当前最短响应时间的服务节点上;而对非实时任务则使用最少连接数优先算法(Least-Connection-Server-First),总是将任务分派到当前负载最小的节点上,从而使非实时子集群的产出达到最大化。

集中式调度策略具有易实现,后端服务程序简单,安全性好的优点,但同时也存在其固有的缺点。首先,前端机作为集群的唯一的进入点,并承担了所有的调度任务,负载过重,往往成为系统的性能瓶颈,影响了系统扩展性。再者,将后端服务节点硬性划分为服务不同类别任务的子集群的作法,往往导致子集群之间的负载不均衡,降低系统的产出率。虽然文献[11]提出了一种根据系统状态变化,动态调节服务节点分区的改进方法,但集中式的调度模式还是很难有效地实现分区之间的负载均衡,特别是在高荷载情况下,系统的利用率明显下降。一般认

为,从效能的角度,集中式调度模式只适用于中小型集群服务器。

3.2　分布式调度策略

不同于集中式的服务节点分区作法,分布式调度策略等价看待每一台后端机,让每一台后端机接受各种类型任务,而通过后端机上的服务区分算法对实时和非实时任务提供不同特征的服务。文献[13]提出了一种双层调度的概念,即在应用层使用负载均衡算法分发任务,而在每个后端节点采用服务区分算法。在 RTCS 研究中借用了这种双层调度的概念,但根据要求进行了修改。在前端机上,实时和非实时调度器针对不同任务采用了两种不同算法。实时调度器采用最短响应时间优先算法(Shortest-Response-Time-First)分派实时任务,力求使响应时间达到最短;而非实时调度器则使用最少连接数优先算法(Least-Connection-Server-First),从而使非实时任务的产出达到最大,如图 3 所示。在每一个后端机上,分派来的实时和非实时事件分别进入两个不同的缓冲队列,节点调度器则采用我们设计的队列长度比例算法(Queue length Proportional Scheduling)[14]来决定两个队列中任务的执行顺序。

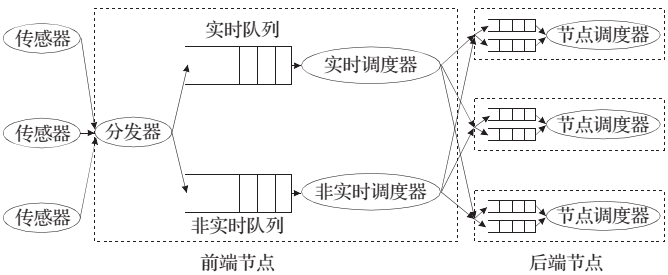


图 3　分布式调度策略

分布式调度策略把部分调度工作从前端机移到了各后端机,一定程度上缓解了前端机的负担,使之不易成为系统瓶颈,增强了系统的扩展性,同时后端机不分区也提高了资源的利用率。但是这种双层调度仍属被动型调度模式,前端机仍然必须保持系统全局状态参数(各节点负载数据,各节点响应时间参数等),以便任务分发时选择后端节点。这导致了前端机与后端机之间频繁的数据交换,影响了响应速度。与集中式策略比较,其系统输出率较高,但平均响应时间却略低,这在进行的集群实验中得到了证实。

集中式策略可扩展性差,资源利用率低;分布式策略改善了扩展性和资源利用率,但时间响应性不尽理想。我们思考能否实现一种全分布策略,将调度功能全部移到后端机上,彻底消除前端机作为性能的瓶颈,实现系统最优化。在下一章中,提出一种主动型全分布调度模式对此进行了尝试。

3.3　主动型全分布调度策略

这个调度策略的提出是基于如下两点考虑:

(1)集群前端机承担的调度分派任务加大了前端机的载荷,随着集群规模的增大,负载过重的前端机必将成为系统性能的瓶颈;

(2)主要由前端机决定的被动型调度模式,要求前端机随时保持系统全局状态量的记录,这导致了前、后端之间数据交换量的增大,消耗网络带宽资源,影响了系统的响应速度。

鉴于上述两点,提出如下的主动型全分布策略,实现集群调度功能的完全离散化,在保证系统对实时任务响应速度的同时,优化系统整体功能。这一全分布式调度策略的结构见图 4,其具体内容如下:

(1)前端机不再提供任何调度功能,而只根据后端机的拉取信号转发相应的事件,任务单一,负载降至了最低程度。

(2)事件的转发由后端机主动控制,后端机拉取线程与前端机派发线程之间通过一个三步骤的拉取协议(Pulling Protocol)实现事件的传递(见图 5)。具体而言,后端机根据本机状态决定拉取的任务类别(实时任务或非实时任务),向前端机发送拉取信号(Pulling Signal),此为第一步骤;前端机接到信号后,向该后端机送出相应任务,此为第二步骤;后端机完成相应服务后,再向前端机送出答复信息,由前端机转发给远端的传感器,此为第三步骤,至此一个服务循环结束。

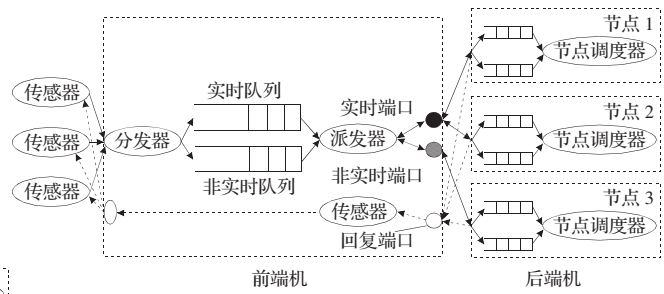


图 4　主动型全分布式策略(PBDS)

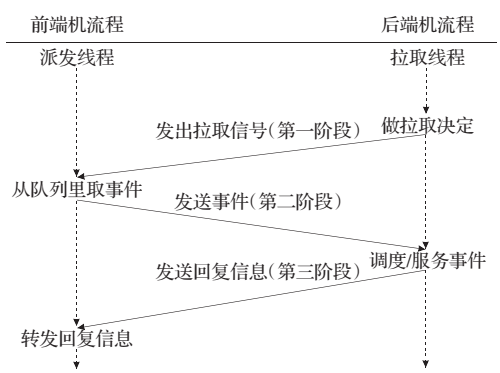


图 5　拉取协议流程图

(3)后端机上有三个主要功能件来实现任务的拉取,负载自平衡,和对任务的性能区分服务,这在下面有进一步阐述。

PBDS 模式将全部调度工作从前端机移至到了后端机,整个系统不再有一个负责总体协调的中心节点,这就必须解决两个问题:

(1)后端机在作拉取决定时,如何决定取哪一类事件?实时还是非实时?这将决定系统资源的分配,影响对不同类别事件的服务质量。

(2)如何确定拉取事件的频率?即各后端节点的拉取速度该如何决定?事件过快拉取将导致本节点任务累积,影响响应速度;拉取速度过慢又将导致节点间荷载不均衡,影响系统整体输出。

对此设计了后端机的三个子线程程序来实现上述功能,即:完成任务拉取的拉取线程(Pulling Thread),调节拉取频率的监控线程(Monitor Thread),和实现性能区分服务的调度线程(Scheduling Thread)。这三个子线程职责不同,并行运行,相互协调,完成节点负载自我平衡及任务调度的工作,如图 6 所示。

拉取线程的工作是基于拉取决定算法(Pulling Decision Algorithm)(图 7(a))。这一算法的基本思想是:根据本节点两个等待队列的长度比决定拉取类别,如果这一比值低于设定门

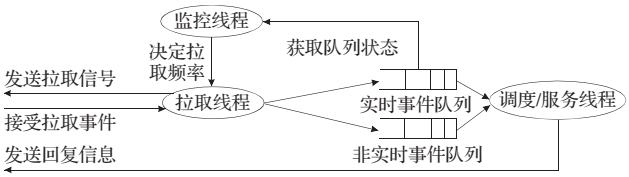


图 6 后端机子线程工作示意图

槛值(表明实时任务等待量不足),则向前端机索取实时任务;否则索取非实时任务。算法的目标在于让实时任务得到尽快响应的同时,分配一定数量的资源给非实时任务,达到总体性能的优化。

监控线程则是根据延迟设置算法(Set Delay Algorithm)(图 7(b))决定拉取任务的频率。后端节点每隔一定时间间隔(称为延迟时间量)发出拉取信号,这个延迟时间量是一个变化值,由监控线程根据本机负载状态进行动态调节。延迟设置算法设置了两个门槛值 MAX_DELAY_TIME 和 MIN_DELAY_TIME 来表征最大和最小延迟时间量,并设定队列长度门槛值 QUE_LEN_LOW 来判断系统是否处于低负载状态。当该机满载时,延迟时间量设为最大值,减慢索取频率;当实时和非实时队列的长度都不到长度门槛值时,该机目前负载不足,延迟时间量应设为最小值,加快索取频率;如果介于这两者之间,则延迟时间量设置成与队列长度成反比,由此对索取频率进行动态调整,从而达到系统负载在各个后端机的均衡。

```
Pulling_Decision_Algorithm
{
  if(RT queue is full)
    pull NRT event;
  else if(NRT queue is full)
    pull RT event;
  else
    if(rt_que_len/nrt_que_len<RT_BOUND)
      pull RT event;
    else
      pull NRT event;
}
```

(a)拉取决定算法

```
Set_Delay_Algorithm
{
  if(either RT queue or NRT queue is full)
    set_delay(MAX_DELAY_TIME);
  else if(both RT and NRT queue length<QUE_LEN_LOW)
    set_delay(MIN_DELAY_TIME);;
  else
    que_len_ratio=(rt_que_len+ nrt_que_len)/que_size;
    set_delay(que_len_ratio);
}
```

(b)延迟设置算法

图 7 PBDS 算法原理

在队列的另一端,节点调度线程则按照队列长度比例算法(Queue Length Proportional Scheduling)[14]对实时和非实时事件提供性能区分(Performance-Differentiated)的服务,并将回复信息返回给前端机。

整个 PBDS 策略是在拉取算法和延迟算法这两个原理的基础上实现,拉取算法和 QLP 调度原理决定了系统资源在任务类别之间的调配,延迟算法则通过控制各节点加载速度来实现后端节点负载的自平衡。在系统加载的初期阶段,各节点负载低于门槛值,都以高频率索取任务,可能造成节点间负载不平衡,但系统的低载使得系统响应性不受影响。随着负载水平增高,超过门槛值的节点开始减慢索取速度,荷载向低负载节点倾斜,由此实现了负载均衡的自我调节。在随后的大集群高负载实验中,证实了 PBDS 体系在经历加载初期进入稳定期后,实现了节点间的负载平衡,表现出优良的系统总体性能。

4 性能评估分析

根据前面的体系结构和调度策略设计,开发了一套基于 Linux 系统的集群服务器开发工具包(RTCS Application Framework)。这一 RTCS 工具包采用中间层型式,将集群管理调度功能封装为功能模块,覆盖了体系结构复杂性,为用户程序提供了界面友好、简单易用的编程接口(API),极大地方便和简化了集群平台上实时服务应用程序的开发。

在开发的 RTCS 中间件界面上,编写了一个集群服务器验证程序(Prototype Application)来测试评估所提出的集群实时服务调度策略,对其时间响应性和系统总输出作出评价。图 8 给出了 Linux 操作系统,RTCS 中间件及集群服务器验证程序之间的层次结构关系。

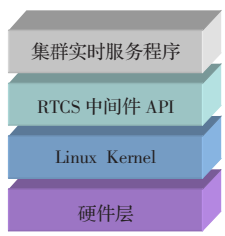


图 8 软件层次结构图

4.1 实验设计

在局域网环境中进行了集群服务器的验证实验,测试环境如下:

硬件平台:40 台 PC,Pentium IV 1.8 GHz 处理器

操作系统:Red Hat 9.0(Linux 内核 2.4.20)

网络连接:10/100 Mb/s LAN

系统构形:NAT 构形(见图 1)

测试系统由远端传感器(由 2-20 台 PC 机组成),集群前端机(装有双网卡),和集群后端机(20 台 PC)组成。传感器通过套接字接口发送固定格式固定长度的短信息(称为事件 event)给集群前端机,并接收回复信息。集群服务器由前端机和后端服务节点群组成。前端机在收到传感器事件后,对事件进行分类,并按照预设的集群调度策略将事件分发至后端机。后端机在完成服务后,将回复信息送回前端机,再转送至原传感器。

传感器发送事件可以有固定式和指数分布形式两种模态,发送速率则可通过设置发送间隙时间来设定,测试采用的速率为 0.73,1.46,2.19,3.65,4.38,5.11,5.84(Kb/s)。在实验中,传感器每次发送一个数据包,一个数据包含 100 个事件,而实验数据则是基于多个数据包的平均值。后端机的服务模式也有固定和指数两种模式,实验中事件的平均服务用时设为 2 秒。

为了评估实时调度策略在各种集群构形和各种负载下的响应特性,我们设计并完成了如下的三个实验。

(1)变化节点数性能对比实验

该实验以常见的集群负载平衡算法[15,16]做样本,比较评估集中式和分布式调度策略在不同传送速率和不同节点数下的

实时响应特性和系统输出率。实验中传感器数目为 2 个,分别以 0.73,1.46,2.19,3.65,4.38,5.11,5.84(Kb/s)的速率发送事件。在 9 个测试态中,后端节点数从 2 增长到 10。每个测试态完成 3 个数据包,取其平均值。

(2)大数量传感器对比实验

该实验用来评估三种实时调度策略在大数目传感器下的响应特性。实验中传感器数目从 4 增加到 20,后端机的数目固定为 10 台。事件发送则采用了了较低的 1.46 Kb/s 和较高的 5.84 Kb/s 两种速率。

(3)高强度负载 PBDS 实验

测试 PBDS 策略在大集群(节点数目增至 20),高负载(传感器数目达到 40)下的响应能力。

4.2　性能参数

采用实时平均往返时间 (real-time average round trip time)和系统总输出(system overall throughput)两个参数来评估系统的响应特性,其定义如下:

实时平均往返时间:$T_{ave}^{rt}=\sum_{i=1}^{N}t^{rt}(i)/N$。

其中 $t^{rt}(i)$是第 i 个实时事件的往返时间,定义为事件 i 从发出到收到回复的时间差;N 是回收到的实时事件总数;T_{ave}^{rt} 为平均响应时间。

系统总产出:$P^{all}=(N*l)/\sum_{i=1}^{N}t(i)$。

其中 P^{all} 代表系统总产出,N 表示被服务的事件总数目(包括实时和非实时),l 为事件信息的长度(以 byte 计算),而 $t(i)$是第 i 个事件的往返耗时。

实时平均往返时间反映出集群服务器对实时事件的响应能力,而系统总输出是系统单位时间内的输出量,是衡量系统资源利用率高低的重要指标。我们的目标是在大大降低实时响应时间的同时,保持或提高系统的产出率。

4.3　变化节点数下的性能对比实验

图 9 给出了集中式策略下,不同发送速率对平均响应时间的影响。可以看出,不管后端节点数如何,一开始响应时间随传送速率增加有陡然上升。但在发送速率达到某一值后,响应时间曲线会趋于平坦,表明系统进入稳定状态,对传送速率不再敏感。负载均衡算法和分布式算法也呈现出同样趋势。图 10 则给出了分布式策略下,后端机数量对响应时间的影响,可看出,无论传送速率高低, 后端机数量的增加将显著降低实时响应时间。

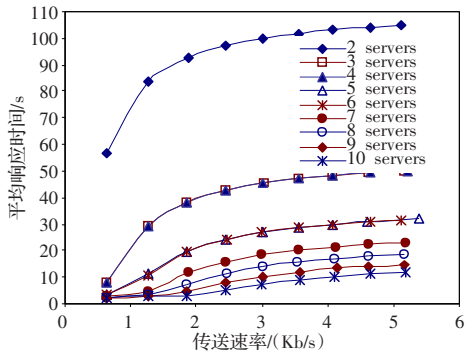


图 9　平均响应时间/发送率的坐标图
(集中式调度,指数模式)

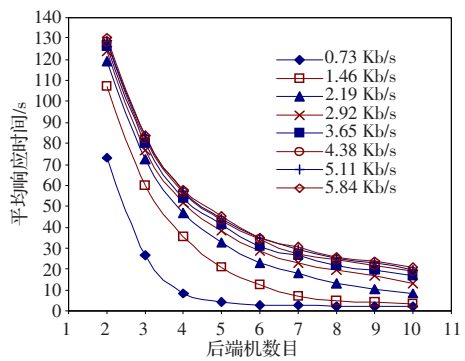
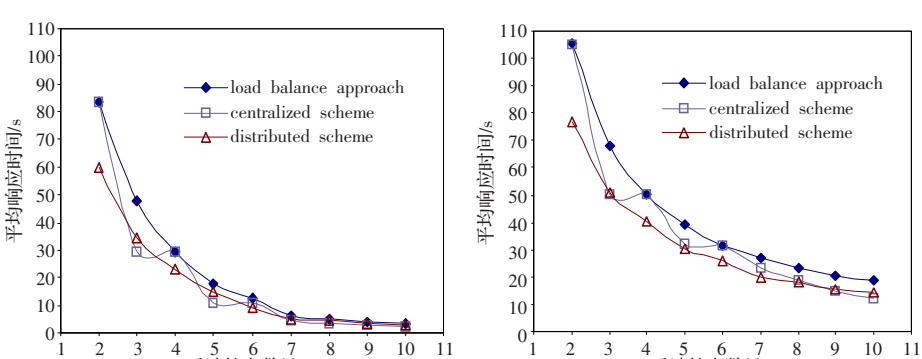


图 10　平均响应时间/后端机数目的坐标图
(分布式调度,指数模式)

图 11(a)和(b)给出了两种传送速率下,集中式调度和分布式调度与负载平衡算法的性能比较。可看出,不管在低传送率(1.46 Kb/s)或高传送率(5.84 Kb/s)下,集中式实时调度和分布式实时调度都比传统的负载平衡算法给出较短的响应时间。表 1 列出了传送率为 5.84 Kb/s 时两种实时调度策略与常规负载平衡算法比较,响应时间下降的百分比。分布式策略在后端机数目较低时(<9),时间降幅显著优于集中式策略,但在后端机数目升高后(≥9),这一数据反而低于集中式策略,这表明,被动型的分布策略的扩展性不尽理想。系统总产出也是衡量集群服务器性能的一个重要指标。图 12 比较了集中式和分布式两种实时调度与负载平衡算法的系统总产出率。从图中看出,尽管大大降低了实时响应时间,两种实时调度策略保持了与常规算法相同水平的系统总产出。仅仅当后端机数目增至 9 以上时,集中式调度的产出率才显现了 8.28%的降幅,这说明集中式调度策略在集群规模增大时,其节点分区的作法导致资源利用率低,总产出率下降。但总体上看,实时调度策略以系统产出



(a)发送速率=1.46 Kb/s　　(b)发送速率=5.84 Kb/s

图 11　平均响应时间对比(负载平衡/集中式/分布式)

表 1 响应时间下降百分比(与负载平衡算法比较,传送速率=5.84 Kb/s)

#servers	2	3	4	5	6	7	8	9	10
centralized	0.63%	26.46%	0.17%	18.39%	0.52%	13.87%	18.89%	27.90%	35.86%
distributed	27.09%	25.70%	20.07%	22.71%	17.97%	26.03%	19.99%	23.46%	23.62%

下降 4%–8%的微小代价,换来了实时响应时间改进 14%–35%的结果,效果还是十分显著的。

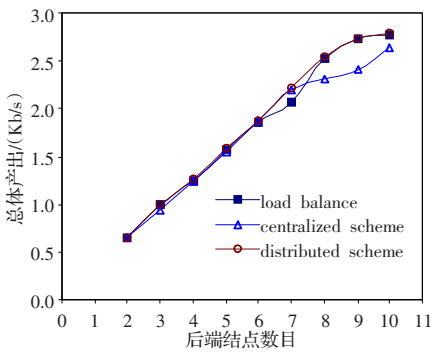


图 12 系统总产出率

(负载均衡/集中式/分布式,传送速率=5.84 Kb/s)

4.4 大数量传感器对比实验

前面的实验只用了两个传感器发送事件,测量到的只是小数量传感器下的数据,还需要检验集群服务器在大数量传感器,即高负载情况下的响应性能。在此实验中,传感器数量从 4 逐渐增加到 20,发送速率为 1.46 Kb/s 和 5.84 Kb/s,后端机数目固定为 10。针对三种实时策略进行了测试,并与相同构型、相同荷载下常规平衡算法得到的结果进行了比较。

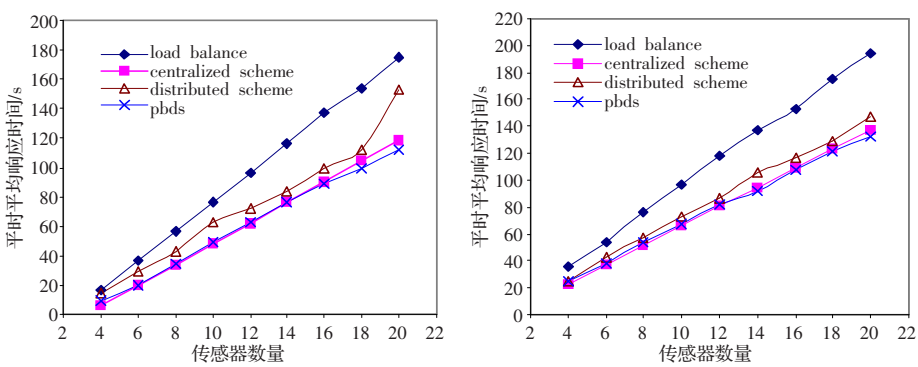
图 13 给出了大数量传感器下各种调度策略的时间响应性的对比。可以看出,不管是高发送率还是低发送率,集中式调度和 PBDS 在平均响应时间方面表现最好。当传感器数目超过 18 时,PBDS 的响应性能甚至比集中式还略好一点,这证实了 PBDS 能够很好地适应大数量传感器荷载。集中式策略给出较好的实时响应数据,但是以低系统资源利用率为代价,在后面的总产出图中它的产出率最低。

分布式调度的实时响应特性较集中式调度略差在预计之中。这是由于分布式和集中式策略都在前端机执行一个时间复杂度为 O(n),n 为后端机数目的搜索算法,以寻找负载最少或响应时间最短的服务器,而分布式策略还需在后端机多执行一个 QLP 长度比算法,这就增加了时间延迟。但分布式策略减轻了前端机负载,其系统扩展性较集中式调度要好。

系统总产出的结果见图 14。当传感器数量小于 8 时,PBDS 的产出比其他策略显著要低,这是由于三步骤拉取协议通信的额外消耗;当传感器数量大于 8 时,PBDS 的产出就上升,接近

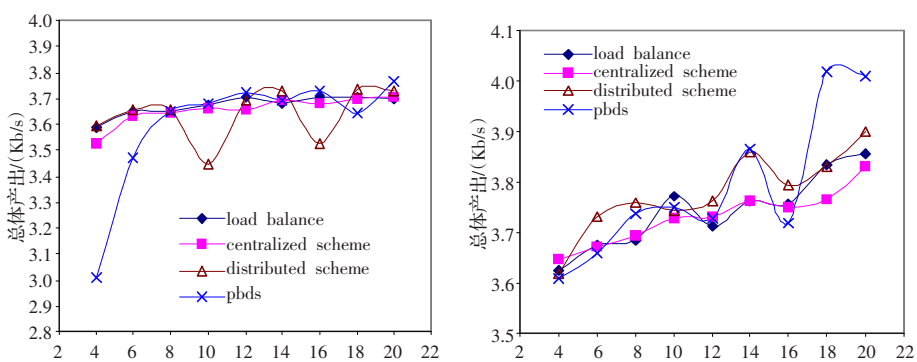
表 2 总产出率变化百分比(与负载平衡算法比较,传送速率=5.84 Kb/s)

# 后端结点数目	2	3	4	5	6	7	8	9	10
集中式	–0.34%	–5.57%	6.59%	–0.60%	0.93%	6.63%	8.61%	–8.28%	–4.80%
分布式	0.52%	0.10%	1.98%	1.15%	1.07%	7.56%	0.35%	–0.04%	0.78%



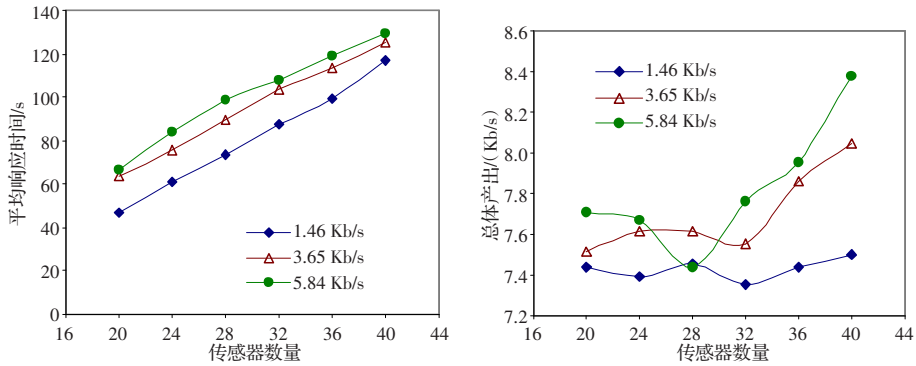
(a)传送速率=1.46 Kb/s (b)传送速率=3.84 Kb/s

图 13 大数量传感器下实时响应时间的比较(平衡算法/集中式/分布式/全分布式)



(a)传送速率=1.46 Kb/s (b)传送速率=3.84 Kb/s

图 14 大数量传感器下总产出率的比较(平衡算法/集中式/分布式/全分布式)



(a)实时响应特性　(b)系统产出率

图 15　PBDS 高负载下 PBDS 的性能

于产出最高的分布式策略；当传感器数量大于 18,PBDS 就远远超过了分布式(传送速率=5.84 Kb/s 时),成为产出率最高。由此看出 PBDS 这种全分布策略自我负载平衡的优势,PBDS 策略在实时响应性和系统产出率两个方面俱给出了令人鼓舞的结果,表现出了支持高负载大集群的优良性能。

总产出图中分布式和 PBDS 策略的几个不稳定点,如传送率=1.46 Kb/s,传感器数量=10 和 6 时,分布式的数据,及传送率=5.84 Kb/s,传感器数量=12 和 16 时,PBDS 的数据都出现非理性下降。这是测试程序的问题或是调度策略本身的缺陷,还有待进一步的研究和实验。

4.5　高强度负载 PBDS 实验

PBDS 全分布式策略在前述的大数量传感器实验中表现出了优异的实时响应特性和高产出率。我们关心在进一步的高强度负载、大集群的情况下,PBDS 策略能否保持其实时响应性能,并使产出率随集群规模成比例增加?为此进行了高强度负载实验,后端机数目增加为 20,传感器的数目最高增加到 40,单机发送事件速率分别为 1.46 Kb/s,3.65 Kb/s,最高达 5.84 Kb/s。这样，传感器向集群前端机的总体发送速率最高可达 40*5.84=233.6 Kb/s，对一台 Pentium IV 1.8 GHz 的 PC 机而言,这已是相当高的输入强度了。我们希望看到 PBDS 策略在高负载情况下的响应特性，以及在加载过程中性能的稳定性情况。

从图 15(a)看出,在低中高三种传送速率下,PBDS 均保持了近乎线性的响应特性,即平均响应时间随传感器数目线性增加，而非大幅跳跳性上升。当传送速率从 1.46 Kb/s 加到 3.65 Kb/s 时,响应时间增幅为 7%-18%,而从 3.65 Kb/s 增加到 5.84 Kb/s 时,增幅只有 4%-10%,这表明 PBDS 具有更好的应付高速率荷载的能力。图 15(b)给出了总产出率数据。在传感器量低于 32 时，系统产出呈一定幅度的扰动，尤其是传送速率为 5.84 Kb/s 时,对应于传感器数目=28 的数据不正常,这需作进一步调查。但传感器数目达到 32 以后,系统产出率稳定上扬,传送率高时增幅更大,进一步证实了 PBDS 策略通过调度功能离散化,负载均衡自调节和优化资源调配所达到的处理高强度荷载的优异性能。

5　结语

本文研究了在集群架构服务器上实现软实时服务的三种调度策略。这种软实时服务的目的在于最大程度上降低实时任务的响应时间,同时保持系统的高输出。提出的三种实时调度策略中,集中式策略的调度功能全部集中于前端机,实时响应性好,易于实现,但资源利用率低,系统扩展性差,只适用于小型集群。分布式策略通过将部分调度功能移至后端机,提高了资源利用率,改善了扩展性,但支持高负载大集群仍然不足。针对前述两种调度策略的不足,提出了一种主动型全分布式调度策略(PBDS),把集群调度分派功能完全离散到后端节点,将前端机负担降至最低,消除了系统功能的瓶颈。在后端机上,通过拉取算法和延迟算法,实现了针对不同时限要求的资源调配和节点负载的自平衡,表现出优异的实时响应特性和系统扩展性。

以上述工作为基础,开发了基于 Linux 系统的集群开发工具包，为集群构架上开发实时应用程序提供了一个模块化,简单易用的开发平台。在此基础上编写了集群实时服务器验证程序，在 40 个节点的集群体系上对上述调度策略进行了对比实验。实验证实,对比于常规负载平衡算法,集中式和分布式策略对实时响应时间有 0.17%到 35.86%(平均为 19.41%) 的改进,而付出的系统产出降低代价仅为 0.34%到 8.28%(平均为 3.27%),达到了既对实时任务提供即时响应,又兼顾非实时任务,优化系统性能的目的。随后的大传感器数量实验和高强度 PBDS 实验证实了全分布式 PBDS 算法在高负载下优异的实时响应特性和稳定的系统产出率,表现出这种新算法在支持大型集群体系,提高系统扩展性上的巨大潜力。

RTCS 集群服务器实时调度策略研究的初步结果是令人鼓舞的,在集群实时服务机制这一领域这一工作具有前瞻性。下一步将着眼于建立 PBDS 调度模式的概率分析模型,探索调度参数(如拉取速率,队列长度比等)与系统响应特性的关系,进一步优化系统的性能。另外,还将加入 Linux 内核部份工作,在内核层面与上层服务器调度程序之间建立一种关联机制。当系统处于超负荷状态时,及时通过这种通信机制,协调下层内核与上层程序的调度与执行,避免集群系统由于内核超载而陷于死机。这一工作对于集群服务器用于大强度高负载苛刻的工作环境具有重要意义。(收稿日期:2007 年 5 月)

参考文献:

[1] Buyya R.High performance cluster computing:architectures and systems:volume I[M].New Jersey:Prentice Hall PTR,1999.

[2] Aron M,Druschel P,Zwaenepoel W.Cluster reserves:a mechanism for resource management in cluster-based network servers[C]//Proc of 2000 ACM SIGMETRICS Intl Conf on Measurement and Modeling of ComputerSystems,Santa Clara,CA,2000-06.

[3] Pradham P,Chiueh T.Implementation and evaluation of a QoS-capable cluster-based router[C]//Proc of Super Computing 2000,Dallas,2000-11.

(下转 146 页)