ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.3, March 2009, pp.766−778
doi: 10.3724/SP.J.1001.2009.03313

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

集群软件无线电系统中实时信号处理调度研究*

朱晓敏+, 陆佩忠

(复旦大学 计算机科学与工程系,上海 200433)

Scheduling of Real-Time Signal Processing in Cluster-Based Software Radio Systems

ZHU Xiao-Min+, LU Pei-Zhong

(Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China)

+ Corresponding author: E-mail: xmzhu@fudan.edu.cn

Zhu XM, Lu PZ. Scheduling of real-time signal processing in cluster-based software radio systems. *Journal of Software*, 2009,20(3):766−778. http://www.jos.org.cn/1000-9825/3313.htm

Abstract: In this paper, according to the characteristics of signal processing in cluster-based SR systems, the following scheduling issues are investigated: First, a universal scheduler model suitable for signal processing in cluster-based SR systems is proposed. The model is simple, the efficient and it avoids the bottleneck problem. Second, a novel three-step scheduling strategy-RQBB is put forward, where the existing DASAP algorithm is used in step 1. Third, two heuristic algorithms MQB and MSD are proposed. They are used in step 2 and step 3 of RQBB, respectively. The MQB is a fair algorithm that strives to make all the accepted tasks have high QoS benefit (high mean of QoS levels and small difference of QoS levels). The MSD is designed to guarantee the system with high throughput and achieve load balancing without violating the timing constraints of accepted tasks. Extensive simulation experiments are performed to compare RQBB with RQRB, DASAP and DALAP. Experimental results indicate RQBB improves QoS benefit better than others and achieve load balancing while guaranteeing high schedulability.

Key words: cluster; software radio system; real-time; scheduling; heuristic algorithm; quality of service (QoS)

摘 要: 在集群软件无线电系统中,当宽带大容量信号数据进入系统后通过在节点上的并行计算实现对强衰弱信号的高增益、低延迟处理.结合集群软件无线电系统中信号处理的特点,研究了以下任务调度方面的问题:1) 提出了一种适合集群软件无线电系统中信号处理的调度器模型.该模型简单、高效,避免了瓶颈问题.2) 提出了一种新的包含 3 个步骤的调度策略——RQBB,其中第 1 步采用已有的 DASAP 算法.3) 提出了两种启发式算法——MQB 和 MSD,分别用在 RQBB 的第 2 步和第 3 步操作.MQB 是一种公平算法,用于使所有接收的任务具有较高的 QoS 收益(较高的 QoS 级别和较小的 QoS 级别差异),MSD 算法用于使系统具有较高的吞吐率并达到负载均衡.通过大量实验对 RQBB 与 DASAP, DALAP 算法和 RQRB 策略进行了比较.实验结果表明,RQBB 具有较高的调度成功率,使得所接收任务具有最优的 QoS 收益,同时使得系统具有较高的吞吐率并达到负载均衡.

关键词: 集群;软件无线电系统;实时;调度;启发式算法;服务质量

* Supported by the National Natural Science Foundation of China under Grant No.60673082 (国家自然科学基金); the Foundation for the Authors of National Excellent Doctoral Dissertation of China under Grant No.200084 (全国优秀博士学位论文作者专项基金)

Received 2007-12-10; Accepted 2008-03-14

中图法分类号: TP301 文献标识码: A

近年来,集群计算技术发展飞快,在成本和体积迅速下降的同时,计算能力大幅度提升.对于计算密集型和数据密集型应用,集群计算技术是较为经济和可靠的手段.本文研究如何有效应用集群计算技术实现软件无线电思想.当采集的宽带大容量数据进入系统后,通过集群中节点并行计算方式对强衰弱信号实现高增益和低延迟处理,以达到实时接收和解译通信信号的目的,这是极具发展前景的研究方向[1].

通常,高速采集设备采集到的大容量信号是需要进行解调或译码的采样信号数据.这些数据进入系统后,首先被分割成多个数据块(任务),然后这些任务被分发到各个节点上进行并行处理.处理后的结果需要进行拼接以形成完整的数据序列,因此任务之间存在关联.多个任务在节点上的并行处理过程中,彼此不需要通信,也不存在优先顺序,所以任务在处理过程中是彼此独立的[1].而最终完整数据序列的获取时间点则取决于最后一个任务的完成时间.因此,最小化所有任务的完成时间即最小化调度跨度(makespan)将有助于提高系统的吞吐率,减小延迟,达到负载均衡,这是本文研究的一个主要内容.

目前,有许多时间复杂度不同的信号处理算法可以用于同一信号处理.通常,时间复杂度高的信号处理算法具有较好的信号处理质量,但需要较长的处理时间.而时间复杂度低的信号处理算法正好相反.例如,对于分组 turbo 码的译码,文献[2]提出了一种接近最优的迭代算法,但是译码的时间复杂度非常高.为了在译码质量和时间复杂度之间进行折衷,许多学者进行了大量研究并提出了多种具有不同时间复杂度的译码算法[3,4].

当信号数据分割成多个数据块(任务)后,需要通过一定的调度策略将这些任务分配到不同的处理节点上进行处理,这直接影响到了系统的吞吐率、负载均衡和任务的处理质量等等.传统的容纳控制模式采用“二元”的调度策略来决定是否接受或是拒绝任务[5].由于信号在处理过程中,可以选择多个不同的处理算法.因此,本文研究的另一个主要内容是如何通过调度策略在满足任务截止期的前提下,最优化任务的处理质量(本文称其为 QoS 收益),即任务具有较高的 QoS 级别同时具有较小的 QoS 差异.

本文提出了一种适合集群软件无线电系统中信号处理的调度策略——RQBB(real-time–Qos enhancement adopting maximizing QoS benefit–load balancing).该策略采用 3 个步骤实现我们的调度目标,即系统具有较高的调度成功率、较高的吞吐率并要达到负载均衡,同时,任务在满足截止期的前提下具有最优的处理质量.第 1 步采用 Qin 等人提出的 DASAP 算法[6],大体上决定任务的分配;第 2 步采用本文提出的一种公平的 QoS 提升算法——MQB(maximizing QoS benefit)算法,在第 1 步的基础上最大化任务的 QoS 收益;第 3 步采用本文提出的一种新的负载均衡算法——MSD(minimizing standard deviation of Nodes’ latest finish time)算法,在第 2 步的基础上最小化所有节点完成时间的标准差以提高系统的吞吐率并达到负载均衡.

1 相关工作

目前已有许多应用于集群计算的调度算法.然而,多任务的最优分配调度问题是一个 NP 难题[7],只有在极少数情况下才存在最优调度算法.因此,在实际应用中,通常建立与实际相吻合的模型,采用尽可能接近最优的启发式算法来解决调度问题[8,9].Branu 等人对 11 种常用的启发式算法进行了评估[13].这些启发式调度算法包括 OLB,UDA,Fast Greedy,Min-min,Max-min,Greedy,Genetic algorithm(GA),Simulated Annealing(SA),GSA,Tabu 和 A*.文献[10]的实验结果表明,Min-min,GA 和 A*具有较好的性能.这 3 种算法的调度跨度的差异在 5%以内,但 GA 和 A*的运行速度比较慢.在文献[10]的实验中,对于 512 个任务,16 个处理器的异构系统,Min-min 算法需要运行 1s,GA 算法需要运行 100s,而 A*则需要运行 1 200s.Maheswaran 等人提出了一种 Sufferage 算法,具有比 Min-min 算法更好的负载均衡性[11].Subramani 等人提出了一种相邻节点伙伴模式的启发式调度算法[8],等等.尽管这些调度算法能够使系统具有较高的吞吐率并达到负载均衡,但是它们没有考虑任务的时间限制,不适合实时应用.尤其是 GA 和 SA 等采用人工智能的方法,由于这些方法的计算时间具有非常高的可变性,因此采用最长调度时间将极大地增加系统延迟,甚至使系统变得不可调度[12].

实时系统可以分为硬实时系统[5]和软实时系统[12]两类.在硬实时系统(如病人监控系统、飞机控制系统等)

中,当任何一个任务不能满足时间要求时,都将可能产生灾难性的后果.而在软实时系统(如视频传输系统、电话交换系统等)中,若干任务不能在截止期内完成就不会对系统产生太大的影响,但要尽量提高任务的调度成功率.根据信号处理的特点,集群软件无线电系统属于软实时系统.通常实时调度算法可以分为静态调度算法(static/offline)[13]和动态调度算法(dynamic/online)[6]两类.静态调度算法适合周期性(periodic)任务调度,而动态调度算法用于非周期(aperiodic)任务调度.由于信号数据动态到达,因此本文采用的调度算法属于动态调度算法.同时,一些实时任务的调度算法是抢占式的(preemptive)[14,15],而另一些实时任务的调度算法是非抢占式(non-preemptive)的,即任务在执行过程中不会被其他任务中断.本文采用后者,这是因为采用非抢占式的调度算法可以极大地减少执行任务间的切换代价,尤其适合软实时应用[16].另外,一些调度算法用于调度彼此存在依赖关系(dependent)的任务[13,17],这些任务之间的关系通常用有向无环图(DAG 图)来描述;而另一些算法用于调度彼此不存在依赖关系(independent)的任务[20].如前面所述,由于信号在节点上的并行处理过程中,彼此之间是独立的,因此我们的调度策略属于独立任务调度.

以往很多研究主要侧重于考虑集群中所有节点完全相同的情况[19],即集群是同构的.但这种同构集群最终将被异构集群所取代,主要是因为目前很多正在使用的节点还具有较强的处理能力,因此从经济角度考虑,不可能一次性地将所有节点全部升级.目前已有一些在异构集群上处理实时任务的调度算法.例如,Qin 等人提出了一种在实时异构系统中提高系统可靠性的调度算法[6].Auluck 等人提出了一种在实时异构多处理机系统中采用可选择任务复制的方法[20],等等.但上面的调度算法只把任务的调度成功率作为调度的主要目标,忽略了系统的负载均衡和吞吐率,同时也没有考虑应用领域背景,如 QoS 需求等.

对于一些有 QoS 需求的实时应用,也出现了一些相应的调度算法.例如,Mittal 等人提出了量化 QoS 降级的集成实时任务调度算法[21].Abdelzaher 等人提出了一种通过 QoS 协商方式用于飞机控制应用的中间件服务[5],等等.本文研究的一个主要内容是如何在集群软件无线电系统中,通过调度算法动态提供信号处理 QoS 需求,使得任务具有最大的 QoS 收益.目前还未见类似的文章.

2 调度器模型和任务模型

2.1 调度器模型

调度器模型可以分为两类:集中式调度器模型和分布式调度器模型[6].与分布式调度器模型相比,集中式调度器模型有两个最明显的优点:1) 通过对中心调度器的备份,便于实现容错处理;2) 实现比较简单、容易.缺点是在节点数目较多、数据量较大的集群系统中,节点和调度器的通信容易形成瓶颈,调度开销比较大.

本文在集中式调度器模型的基础上进行了改进,给出了一种适合集群软件无线电系统中大容量信号处理的通用调度器模型,该模型如图 1 所示.

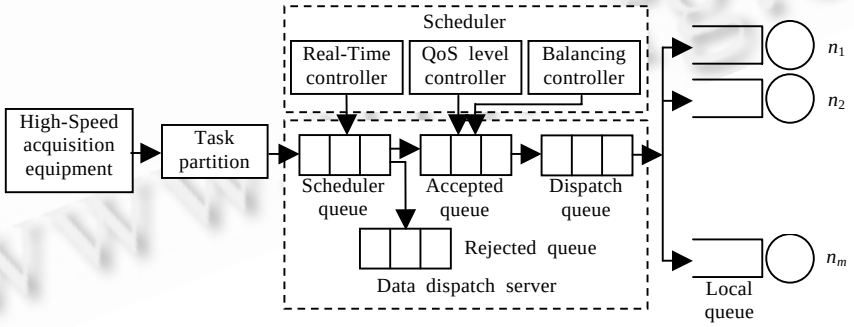


Fig.1 Scheduler model

图 1 调度器模型

在该调度器模型中,各设备间通过高速网络进行连接.高速采集设备采集到的信号数据被分割成多个任务后,存放在调度队列中.数据分发服务器实际上是一个带有多个网口(例如:4 个千兆网口)并与网络交换机(例

如:48 口千兆网络交换机)连接的服务器.

实时控制器首先大体上决定任务的分配方案.值得注意的是,实时控制器可以采用任何已有或新提出的用于提高调度成功率的调度算法.任务首先被设置为最低的 QoS 级别(即采用时间复杂度最小的信号处理算法),目的在于提高任务的调度成功率.实时控制器根据任务的截止期和执行时间决定任务是否被接受.QoS 控制器用来提高接收队列中任务的 QoS 收益.均衡控制器在保证任务接收率和 QoS 收益不变的基础上,调整部分任务分配以达到负载均衡.之后,任务被发送到分发队列,继而分发到指定节点的局部队列中等待该节点进行处理.

本文提出的实时调度器模型具有以下优点:1) 数据分发速度快.数据分发服务器的每个网口负责向指定的某些节点分发任务.因此,当节点数增加时,通过增加网口来保证数据的高速传输,从而避免了瓶颈问题;2) 中心调度器只用来调度任务,而不再分发任务,因此减轻了中心调度器压力,提高了系统的可靠性与稳定性;3) 调度器模型中增加了 QoS 控制器和均衡控制器,提高了系统吞吐率,达到了负载均衡,同时提高了任务的 QoS 收益.

2.2 任务模型

令 $T=\{t_1,t_2,\ldots,t_n\}$为任务集合.$N=\{n_1,n_2,\ldots,n_m\}$为节点集合.执行时间用矩阵 $E=(e_{ij})_{n\times m}$ 表示,其中,元素 e_{ij} 表示任务 t_i 在节点 n_j 上的执行时间.d_i 表示任务 t_i 的截止期.任务最早开始时间用矩阵 $EST=(est_{ij})_{n\times m}$ 表示,其中,元素 est_{ij} 表示任务 t_i 在节点 n_j 上的最早开始时间.M 为调度跨度,$M=\max_{1\le i\le n}\{f_i\}$,其中 f_i 表示任务 t_i 的完成时间.$X=(x_{ij})_{n\times m}$ 为一个二元矩阵,元素 x_{ij} 为 1 当且仅当任务 t_i 被分配到节点 n_j 上,否则 x_{ij} 为 0.$CE=(ce_{ij})_{n\times m}$ 也为二元矩阵,元素 ce_{ij} 为 1 当且仅当在节点 n_j 上的任务 t_i,其 QoS 级别可以被提升,否则 ce_{ij} 为 0.$Q=\{q_1,q_2,\ldots,q_k\}$表示 QoS 级别集合,其中,$q_1<q_2<\ldots<q_k$.X_i 表示任务 t_i 的所有可行调度.$x_i\in X_i$ 为 t_i 的调度选择.任务 t_i 采用 x_i 调度的 QoS 级别可以表示为 $q(x_i)$.x_i 是一个可行调度,需满足下面两个条件:1) 满足 x_i 的截止期 d_i,即 $f_i\le d_i$;2) 满足 QoS 需求,即 $q_1\le q(x_i)\le q_k$.最大化节点 n_j 上所有任务的 QoS 收益应在时间限制的前提下满足下面两个式子:

$$\max_{x_i\in X_i}\left\{\alpha=\frac{\sum_{i=1}^{n}(x_{ij}q(x_i))}{\sum_{i=1}^{n}x_{ij}}\right\},$$

$$\min_{x_i\in X_i}\left\{\beta=\left(\sum_{i=1}^{n}\left(x_{ij}q(x_i)-\frac{\sum_{i=1}^{n}(x_{ij}q(x_i))}{\sum_{i=1}^{n}x_{ij}}\right)^2\right)\Big/\sum_{i=1}^{n}x_{ij}\right\}.$$

令 $\chi=\frac{\alpha}{\varepsilon+\sqrt{\beta}}$($\varepsilon$为任意小正实数)为节点 n_j 上任务的 QoS 收益.$\chi'_k=\frac{\alpha'}{\varepsilon+\sqrt{\beta'}}$ 为节点 n_j 上某一任务 t_k 的 QoS 被提高一个级别后,节点 n_j 上任务的新 QoS 收益.最大化 n_j 上任务的 QoS 收益可定义为下面的目标函数:

$$QB_j=\max\{\chi'_k\},(\exists ce_{kj}=1\wedge\chi'_k\ge\chi)\vee(\forall ce_{kj}=1\wedge\chi'_k<\chi) \quad (1)$$

例 1:假设有 5 个 QoS 级别(1,2,3,4,5).在某一节点上有 5 个任务(t_1,t_2,t_3,t_4,t_5),其当前相应的 QoS 级别分别为 2,3,5,2,1.令ε为 0.1,则χ的值为 1.78.如果每个任务的 QoS 级别分别提高 1(t_3 除外,因为 t_3 级别已经最高),则得到新的 QoS 收益,分别为 $\chi'_1=1.96$,$\chi'_2=1.78$,$\chi'_4=1.96$ 和 $\chi'_5=2.20$.$QB_j=\max\{\chi'_1,\chi'_2,\chi'_4,\chi'_5\}=2.20$,因此,这 5 个任务的新 QoS 级别分别为 2,3,5,2 和 2.

例 2:假设有与例 1 相同的 5 个任务和 QoS 级别数.当前这 5 个任务的 QoS 级别均为 1.令ε=0.1,则χ的值为 10.若每个任务的 QoS 级别分别提高 1,则新的 QoS 收益均为 2.4.尽管 $\chi'_k<\chi(k=1,2,3,4,5)$,但是如果当前任何一个任务 QoS 级别的提升都不会使这 5 个任务错失其截止期,则这 5 个任务的新 QoS 级别是其中一个为 2,其他为 1.虽然新 QoS 收益小于原 QoS 收益,但是避免了局部极值问题,使任务有进一步的 QoS 级别提升机会.负载均衡是本文所研究的另一个重要问题.每个节点的完成时间可以表示为 $LF_j=\max_{1\le i\le n}\{x_{ij}f_i\}$.调度跨度

$M=\max_{1\le j\le m}\{LF_j\}$.本文通过节点完成时间的标准差来衡量系统的负载均衡程度:

$$LB=\sqrt{\frac{\sum_{j=1}^{m}\left(LF_j-\overline{LF}\right)^2}{m}} \tag{2}$$

其中, $\overline{LF}=\frac{\sum_{j=1}^{m}LF_j}{m}$.负载均衡的目标即最小化所有节点完成时间的标准差:

$$\min\{LB\} \tag{3}$$

3 MQB 和 MSD 算法

3.1 MQB算法

MQB 算法按照目标函数(1)来最大化同一节点上任务的 QoS 收益.下面首先介绍与其相关的一些性质.

性质 1. 如果一个任务的 QoS 级别可被提高,那么一定保证 QoS 级别提高后,该任务和在同一节点上其后面执行的所有任务满足截止期.

假设在节点 n_j 上,如果任务 t_i 的 QoS 级别可被提高,即 ce_{ij}=1,一定满足下面两个不等式:

$$est_{ij}+e_{ij}(q(x_i))\le d_i \tag{4}$$

$$\forall t_k,s_{kj}>s_{ij}:est'_{kj}+e_{kj}(q(x_k))\le d_k \tag{5}$$

其中,est_{ij}是任务 t_i 在节点 n_j 上的最早开始时间.$e_{ij}(q(x_i))$是任务 t_i 在节点 n_j 上 QoS 需求为 $q(x_i)$的执行时间.s_{ij} 表示任务 t_i 在节点 n_j 上的执行顺序. $est'_{kj}=est_{kj}+it_{ij}$,其中 it_{ij} 表示在节点 n_j 上的任务 t_i,其 QoS 提高一个级别后的时间增量.性质 1 表明,任务 QoS 级别的增加在满足自身任务截止期的同时,不能违反任何已经按照 DASAP 算法分配到该节点上,在其后面执行的所有任务的截止期.注意,一个节点上任务 QoS 级别的提升不会影响到其他节点上的任务.任务 t_i 在节点 n_j 上的最早开始时间 est_{ij} 可如下计算:

$$est_{ij}=a_i+\sum_{s_{kj}<s_{ij}}e_{kj}(q(x_k)) \tag{6}$$

其中,a_i 表示任务 t_i 的到达时间, $\sum_{s_{kj}<s_{ij}}e_{kj}(q(x_k))$ 表示在节点 n_j 上,执行顺序小于 t_i 的所有任务的执行时间之和.

性质 2. 如果一旦一个任务的 QoS 级别不能被提升,那么这个任务的 QoS 级别以后也不可能被提升.

在节点 n_j 上,如果任务 t_i 的 QoS 级别 $q(x_i)$不能被提升,有以下两种情况:

(1) $q(x_i)=q_k$,此时 t_i 的 QoS 级别已经最高,以后也不可能再被提升.

(2) $q(x_i)<q_k$,但 $q(x_i)$提高一个级别后,不满足性质 1,即 $\exists t_k,s_{kj}>s_{ij}:est'_{kj}+e_{kj}(q(x_k))>d_k$,其中 $est'_{kj}=est_{kj}+it_{ij}$ 或者 $est_{ij}+e_{ij}(q(x_i))>d_i$.由于节点 n_j 上的任务不能被丢弃,如果$\exists t_p,p\ne i$,使得 t_p 的 QoS 级别可以被提升,当 $s_{pj}<s_{ij}$ 时, $\forall t_q,s_{qj}>s_{pj}:est'_{qj}=est_{qj}+it_{pj}$,使得在同一节点上 t_i 后面执行的所有任务最早开始时间增加,任务 t_i 更不能再满足性质 1,因此不可能再被提升;当 $s_{pj}<s_{ij}$ 时, $\exists t_q,s_{qj}>s_{pj}:est'_{qj}=est_{qj}+it_{pj}$,使得在同一节点上,t_i 后面执行的部分任务最早开始时间增加,任务 t_i 也不能再满足性质 1,因此不可能再被提升.

性质 2 表明,如果一旦一个任务的 QoS 级别不能被提升,那么这个任务以后就不必再进行 QoS 级别提升处理,从而降低了调度的时间复杂度.

MQB 算法的伪代码如图 2 所示.

MQB 算法的目标是在满足同一节点上所有任务时间要求的基础上最大化这些任务的 QoS 收益.值得注意的是,QoS 收益目标函数不是单调递增的,从而避免了局部极值问题,见例 2.因此,MQB 算法选择进行 QoS 级别提升的任务是该任务能够使得其 QoS 级别提高后新 QoS 收益最大(见图 2 的第 19~第 22 行).

根据性质 2,如果任务的 QoS 级别为 q_k,那么这些任务将被从集合 S 中删除(见第 7、第 8 行).因为 q_k 是最高

的 QoS 级别,不可能再提高.所以没有必要再处理这些任务.根据性质 1,在增加节点 n_j 上的任务 t_i 的 QoS 级别之前,MQB 算法判别是否由于 t_i 级别的增加满足该任务的截止期,同时不违反所有已接收任务的截止期(见第 12 行).如果不能满足,表明任务 t_i 的 QoS 级别不能再增加,因此,把该任务从结合 S 中删除(见第 14 行).如果第 18 行能够被执行到,说明至少有一个任务的 QoS 的级别可以提升.下面分析 MQB 算法的时间复杂度.

```
1. for each node n_j in the cluster{
2.    S←NULL; /*initialization*/
3.    put each task t_i into set S; /*this allocation of t_i on n_j is determined by Step 1*/
4.    while(S is not empty){
5.       max_reward←0; flag←FALSE;
6.       for each task t_i with QoS level q_m in set S{
7.          if(q_m=q_k){ /*t_i has the highest QoS level*/
8.             remove t_i from S; (Property 2)
9.             continue; /*to deal with next task in set S*/
10.         }else{
11.            increase QoS level of t_i by 1; calculate the increased time it_ij;
12.            if est_ij + e_ij(q_{m+1}) > d_i  (or ∃t_k, s_kj > s_ij : est_kj + e_kj(q(x_k)) + it_ij > d_k ){ (Property 1)
13.               decrease the QoS level of t_i by 1;
14.               remove t_i from S; (Property 2)
15.               continue; /* to deal with next task in set S*/
16.            } /* end if */
17.         } /* end if */
18.         flag←TRUE; /*QoS level can be enhanced*/
19.         calculate the new QB_j;
20.         if(QB_j>max_reward){
21.            max_reward←QB_j; /*find the maximal QB_j */
22.         }/* end if */
23.      } /* end for each task t_i in set S*/
24.      if(flag){
25.         record the task t_i with max_reward, selected_task←t_i;
26.         update the execution time of task selected_task;
27.         for each task t_k whose s_kj is later than that of selected_task{
28.            update the est_kj of task t_k on node n_j;
29.         } /* end for */
30.      } /* end if */
31.   } /* end while */
32.} /* end for each node n_j */
```

Fig.2 Pseudocode of MQB algorithm

图 2 MQB 算法伪代码

定理 1. MQB 算法的平均时间复杂度为 $O(n^2k/m)$,其中 n 为任务数,m 为节点数,k 为 QoS 级别个数.

证明:将同一节点上所有任务放入集合 S 的平均时间复杂度为 $O(n/m)$(见第 3~第 5 行).判断一个任务的 QoS 级别是否最高,其时间复杂度为 $O(1)$(见第 7~第 10 行).任务 t_i 的 QoS 级别提高 1,并计算时间增量 it_{ij},其时间复杂度为 $O(1)$(见第 11 行).判断一个任务的 QoS 级别是否可以提高的平均时间复杂度为 $O(n/m)$(见第 12 行).计算新的 QoS 收益 QB_j,其平均时间复杂度为 $O(n/m)$(见第 19 行).更新执行顺序大于 *selected_task* 的那些任务的最早开始时间,其平均时间复杂度为 $O(n/m)$(见第 27~第 29 行).因此,MQB 算法的平均时间复杂度可如下计算:

$$O(m)\left(O(n/m)+O(n/m)\left(O(k)\left(O(1)+O(1)+O(n/m)+O(n/m)\right)+O(n/m)\right)\right)=O(n^2k/m).$$ □

为减小等待延迟和调度延迟,在实际信号处理过程中,执行一次的任务分割数在 50~200 之间.QoS 级别在 2~3 之间,因此时间复杂度不高.

3.2 MSD算法

性质 3. 任务可以从一个节点移动到其他节点执行,必然同时满足:(1) makespan 减小,(2) 在任务 QoS 级别不变的情况下,满足该任务截止期.当没有任务可以移动时,认为系统已经达到负载均衡.

假设节点 n_j 的完成时间 LF_j 最大,其上最后执行的任务为 t_i.t_i 不能移动到其他节点,n_k 有两种情况:

(1) 节点 n_k 是 t_i 移动到该节点后完成时间最小的节点.如果 $LF_k+e_{ik}(q(x_i))\geq LF_j$,则 $\forall n_p, p\neq k\neq i$: $LF_p+e_{ip}(q(x_i))\geq LF_j$,这使得 makespan 保持不变或增加,不能减小所有任务的整体完成时间,因此 t_i 不能移动.

(2) 节点 n_k 是 t_i 移动到该节点后满足 $LF_k + e_{ik}(q(x_i)) < LF_j$ 的节点,但 $est_{ik} + e_{ik}(q(x)) > d_i$.虽然 t_i 移动到 n_k 上可以使得 makespan 减小,但不能使得任务在 QoS 级别不变的情况下按时完成,因此,t_i 不能移动.

MSD 算法伪代码如图 3 所示.为了最小化所有节点完成时间的标准差,MSD 算法将完成时间最晚的节点上最后一个执行的任务移动到另一个节点后,使得该目的节点的新完成时间最小(见第 7 行),同时,满足该任务的截止期并保证其 QoS 不变(见第 6 行).下面分析 MSD 算法的时间复杂度.

```
1. do{
2.    balanceFlag←FALSE
3.    find the node n_j, where LF_j is latest; min_LF←LF_j
4.    find the last task t_i to be executed on node n_j;
5.    for each node n_k{
6.       if( LF_k + e_ik(q(x_i)) < min_LF && LF_k + e_ik(q(x_i)) < d_i ){(Property 3)
7.          min_LF←LF_k; /*find the node on which finish time is minimal*/
8.          balanceFlag←TRUE;
9.       } /* end if */
10.   } /* end for*/
11.   if( balanceFlag ){
12.      allocate task t_i to node n_q whose LF_q=min_LF;
13.      update the values of LF_q and LF_j, LF_q ← LF_q + e_iq(q(x_i)); LF_j ← LF_j − e_ij(q(x_i))
14.   } /* end if */
15.}while(balanceFlag)
```

Fig.3 Pseudocode of MSD algorithm

图 3 MSD 算法伪代码

定理 2. MSD 算法的平均时间复杂度为 $O(mn+n^2/m)$,其中 n 为任务数,m 为节点数.

证明: 找到完成时间最晚的节点 n_j,其时间复杂度为 $O(m)$(见第 3 行).在节点 n_j 上,找到后一个执行的任务,其平均时间复杂度为 $O(n/m)$.找到目标节点 n_q 使得该节点的新完成时间最小的时间复杂度为 $O(m)$(见第 5~第 10 行).更新节点 n_j 和 n_q 完成时间的时间复杂度为 $O(1)$(见第 12、第 13 行).因此,MSD 算法的平均时间复杂度为

$$O(n)(O(m)+O(n/m)+O(m)+O(1))=O(mn+n^2/m).$$ □

最坏情况下,do-while 循环次数为 n–1.但由于采用第 1 步策略已经将任务大体上分配到了各个节点,因此循环的次数大为减少,使得 MSD 算法的时间复杂度不高.

4 实验测试

本文通过大量的模拟实验来测试 RQBB 策略的性能.将其与 DASAP[6],DALAP[6]算法和 RQRB 策略(包含 3 个步骤,第 1 步和第 3 步与 RQBB 相同,第 2 步采用 Round-Robin 算法)进行比较.为公平起见,将 DASAP 和 DALAP 算法进行了略微修改,即它们随机选取任务的 QoS 级别.尽管修改后的两种算法也提供了 QoS 需求,但是它们没有最大化 QoS 收益并使得系统达到负载均衡.下面简要介绍 DASAP 和 DALAP 算法.

1. DASAP 算法:任务具有最早截止期执行最优先,并且该任务被分配到使其具有最早开始时间的节点上.

2. DALAP 算法:任务具有最早截止期执行最优先,并且该任务被分配到在满足其截止期的前提下使其具有最晚开始时间的节点上.

本文主要从以下几个方面比较 RQBB、RQRB、DASAP 和 DALAP 的性能:

1. 调度成功率(guarantee ratio,简称 GR);
2. QoS 收益均值(QoS benefit average,简称 QBA);
3. QoS 级别均值(QoS level average,简称 QLA);
4. QoS 级别标准误差(QoS level standard deviation,简称 QLSD);
5. 调度跨度(Makespan,简称 MS);
6. 节点完成时间标准误差(Finish time standard deviation,简称 FTSD of nodes).

4.1 模拟方法和参数

异构性可以分为节点异构性和任务异构性[11].本文在实验中充分考虑了节点和任务的异构性.下面给出实验的模拟方法:

1. 为了体现节点的异构性,用 p_j 表示节点 n_j 的处理能力.p_j 是一个正实数,p_j 越大,节点处理能力越强.参数 *powerAverage* 和 *powerSpan* 分别表示所有节点的平均处理能力和节点处理能力的浮动范围.p_j 均匀地分布在 *powerAverage–powerSpan* 和 *powerAverage+powerSpan* 之间.

2. h_i 表示任务 t_i 的处理难度,用以体现任务的异构性.h_i 是一个正实数,h_i 越大,任务 t_i 在同一节点上的执行时间越长.参数 *hardnessAverage* 和 *hardnessSpan* 分别表示所有任务的平均处理难度和处理难度的浮动范围.h_i 均匀分布在 *hardnessAverage–hardnessSpan* 和 *hardnessAverage+hardnessSpan* 之间.

3. 文献[9]将任务执行矩阵分为两类:一致(consitent)矩阵和不一致(inconsisten)矩阵.本文中执行时间矩阵属于一致矩阵.任务 t_i 在节点 n_j 的执行时间 e_{ij} 可以表示为 $e_{ij}=\left(1+\frac{q(x_i)}{10}\right)\times baseTime\times(h_i/p_j)$,其中,$0\le q(x_i)\le 9$ 为任务 t_i 的当前 QoS 级别,$q(x_i)$为一正整数.参数 *baseTime* 为一随机正实数.

4. 任务 t_i 的截止期可以计算为 $d_i=a_i+\max(e_{ij})+baseDeadline$,其中,$a_i$ 是任务 t_i 的到达时间,$\max(e_{ij})$为任务 t_i 在所有节点上执行时间的最大值.参数 *baseDeadline* 是一个随机正实数,*baseDeadline* 越大,任务截止期越宽松.

5. r_j 表示节点 n_j 的开始工作时间.r_j 是在 0 和 *readyTime* 之间的一个随机数.参数 *readyTime* 为一个随机正实数.表 1 给出了实验中的参数值.

Table 1 Lists of experimental parameters

表 1 实验参数列表

Parameter	Value(Fixed)-(Min, Max, Step)
nodeNumber (Node number)	(27) - (15,45,5)
taskNumber (Task number)	(2000)
powerAverage (Power average)	(700)
powerSpan (Power span)	(400) - (200,600,50)
hardnessAvergae (Hardness average)	(190)
hardnessSpan (Hardness span)	(100) - (40,160,10)
baseDeadline (Base deadline)	(50) - (20,90,10)
baseTime (Base time)	(3) - (1,6,0.5)
readyTime (Ready time)	(9)
ε	(0.1)

4.2 节点数对性能的影响

本节将通过一组实验观察节点数对 RQBB、RQRB、DASAP 和 DALAP 的影响.实验结果如图 4 所示.

图 4(a)显示了 RQBB 和 RQRB 具有相同的 GR,平均高于 DASAP 和 DALAP 的 16.79%和 13.54%.这是因为它们都在第 1 步采用了 DASAP 算法并设定所有任务的 QoS 级别为最低(本实验中为 0),而 DASAP 算法为任务随机选取 QoS 级别.此外,RQBB 和 RQRB 在第 2 步和第 3 步都没有放弃或增加任务,因此二者始终具有相同的 GR.随着节点数目的增加,更多的节点可以用来处理更多的任务,因此所有方法的 GR 随着节点数的增加而提高.

图 4(b)~图 4(d)显示了 DASAP 和 DALAP 的 QBA、QLA 和 QLSD 基本保持不变,这是因为它们没有采用 QoS 级别提升方法,而 RQBB 和 RQRB 在这几方面会随着节点数的增加而趋近目标函数(1).图 4(b)显示了当节点数目小于 25 时,RQBB 的 QBA 保持在 1 左右,这是因为在节点较少时,RQBB 把 GR 作为调度的主要目标,所以从图 4(c)~图 4(d)可以看出,RQBB 的 QLA 略有增加,QLSD 的值一直很大.当节点数大于 25 时,RQBB 的 GR 增长迅速,这是因为约束变得较为宽松(见性质 1).同时,RQBB 的 QBA 和 QLA 增加,而 QLSD 相应降低,这是因为 RQBB 利用任务截止期前的空闲时间提高了所接收任务的 QoS 级别,并减小了 QoS 级别之间的差异.当节点数超过 25 以后,系统有能力接收所有任务,同时约束变得更加宽松,所以所有任务的 QoS 级别达到最高、QBA 达到最大,QLSD 为 0.对于 RQRB,由于它在第 2 步采用了 Round-Robin 方法,因此 QLSD 基本保持不变.而 RQRB 的 QBA 和 QLA 变化的原因与 RQBB 相同.

图 4(e)显示了 RQBB 和 RQRB 的调度跨度随着节点数目的增加而减小,而 DASAP 和 DALAP 的调度跨度基本保持不变,这是因为当节点数目增加时,接收任务数也随之增加,而由于 RQBB 和 RQRB 采用了负载均衡方法,其调度跨度在节点数增加时小于 DASAP 和 DALAP.图 4(f)显示了 RQBB,RQRB 和 DASAP 都具有较好的负载均衡,而由于 DALAP 将任务分配到开始时间最晚的节点上,导致有些节点的负载很轻,而有些节点负载很重,当节点数目增加时,这种现象更为明显,甚至有些节点处于空闲状态,因此,DALAP 不能保证系统的负载均衡.

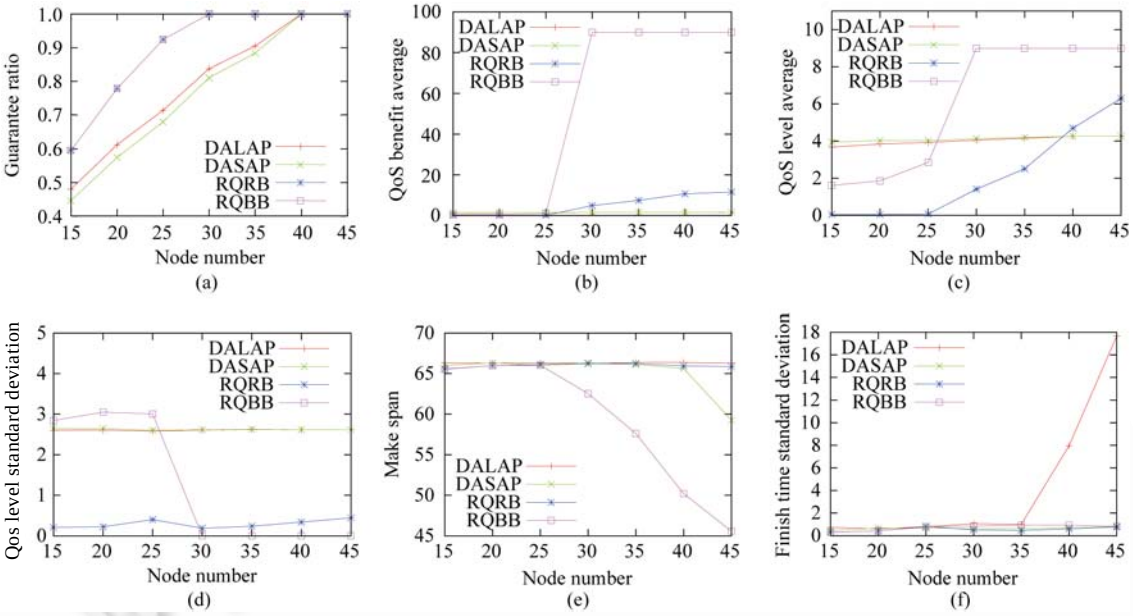


Fig.4 Performance impact of node number

图 4 节点数对性能影响

4.3 节点异构性对性能的影响

本节将通过一组实验观察节点异构性对 RQBB,RQRB,DASAP 和 DALAP 的影响.实验中用参数 *powerSpan* 代表节点的异构性,*powerSpan* 增大,则节点的异构性增大,反之则减小.实验结果如图 5 所示.

从图 5(b)~图 5(d)可以看出,RQBB 的 QBA,QLA 和 QLSD 随着节点异构性的增大而改善,这是因为节点异构性的增大使得接收的任务数略有减少,所以所接收任务的空闲时间增多.尽管节点的异构性发生了变化,但是节点的总体性能保持不变,因此,RQBB 利用增加的任务空闲时间提高了 QoS 级别.

图 5(e)显示了 RQBB 的调度跨度略有减小,这是因为接收的任务数略有减少.但是 DALAP 的调度跨度却一直保持不变,这表明 DALAP 具有较差的灵活性.图 5(f)显示了尽管节点的异构性改变了,但是 RQBB,RQRB 和 DASAP 都能使系统具有较好的负载均衡.由于 RQBB 和 RQRB 采用了负载均衡方法,使得系统达到最好的负载均衡.而 DALAP 随着异构性的增加,使得性能好的节点负载较重,而性能差的节点负载很轻,因此,节点异构性越强,DALAP 使得系统的负载越不均衡.

本组实验表明节点异构性对 RQBB 略有影响,使得 GR 略有下降,但是 QBA,QLA 和 QLSD 却得到了改善,并且负载仍保持均衡,这说明 RQBB 具有很强的自适应性.

4.4 任务截止期对性能的影响

本节通过一组实验来观察任务截止期对 RQBB,RQRB,DASAP 和 DALAP 的影响.当任务截止期变得宽松时,我们考察这些方法如何利用这些空闲时间的.实验中用参数 *baseDeadline* 改变任务截止期,*baseDeadline* 增大,则任务截止期增大,反之,则减小.实验结果如图 6 所示.

图 6(a)显示,当任务截止期增大时,所有方法的 GR 都随之增加.这是因为任务的时间限制变得宽松.尽管任务截止期增大后任务具有较多的空闲时间,但是从图 6(b)~图 6(d)可以看出,DASAP 和 DALAP 并没有使任务的 QoS 的级别提高,所以这些接收任务的空闲时间被浪费了.但是,RQRB 在任务截止期内充分利用了这些空闲时间,提高了任务的 QoS 收益.图 6(b)~图 6(d)显示了随着截止期的增加,RQBB 的 QBA,QLA 和 QLSD 随之改善.

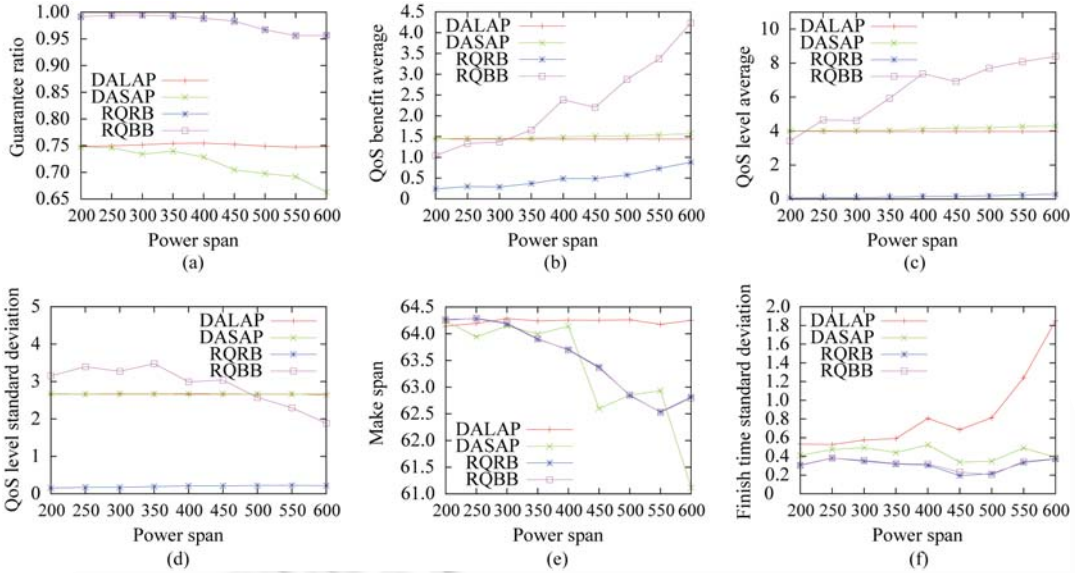


Fig.5 Performance impact of node heterogeneity

图 5 节点异构性对性能影响

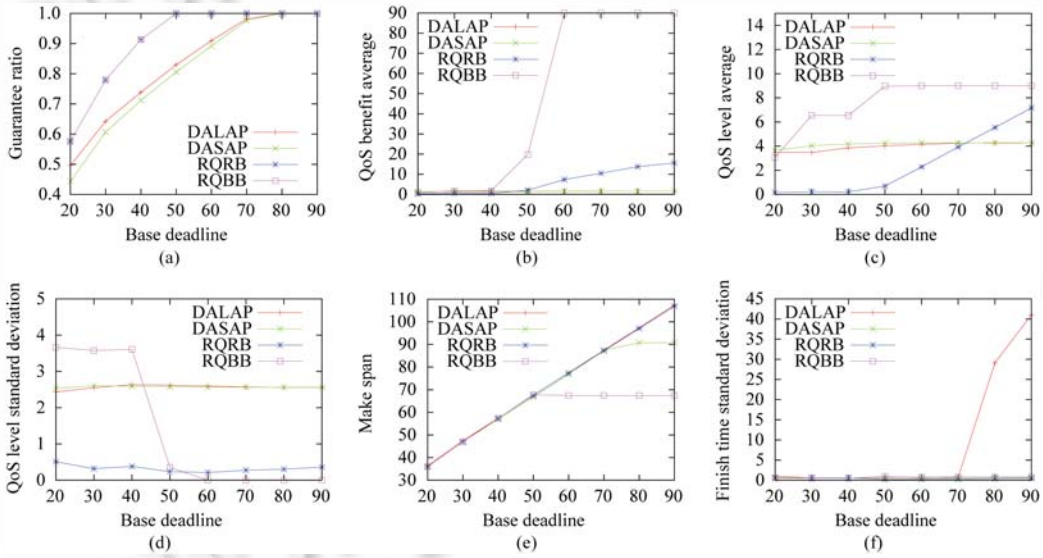


Fig.6 Performance impact of tasks' deadlines

图 6 任务截止期对性能影响

图 6(e)显示了当 *baseDeadline* 小于 50 时,所有方法的调度跨度随截止期的增加而增大.这是因为系统接收了更多的任务.而当 *baseDeadline* 大于 50 后,RQBB 的调度跨度基本保持不变且小于其他方法.这是因为 RQBB 在第 3 步采用了负载均衡方法,使得系统具有较高的吞吐率.

图 6(f)反映出随着任务截止期的增加,RQBB,RQRB 和 DASAP 使得系统具有较好的负载均衡,而 DALAP 把更多任务分配到了某些性能好的节点上,使得系统负载极不均衡.

4.5 任务粒度对性能的影响

本节我们将通过一组实验观察任务粒度对 RQBB,RQRB,DASAP 和 DALAP 的影响.实验中用参数 *baseTime* 代表任务的粒度,*baseTime* 增大,则任务粒度增大,反之,则减小.实验结果如图 7 所示.

图 7(a)显示出当任务粒度增大时,所有方法的 GR 减小.这是因为随着任务粒度的增大,任务具有较长的执行时间,而在这些任务后面的任务,其最早开始时间推迟了,因此导致后面执行部分任务不能满足其截止期.由于 RQBB 和 RQRB 在第 1 步采用了最低的 QoS 级别,所以其 GR 最高,分别高于 DASAP 和 DALAP 的 18.56%和 15.07%.图 7(b)~图 7(d)显示了 RQBB 和 RQRB 的 QBA,QLA 和 QLSD 随着粒度的增大而变差.这是因为任务粒度增大后,提高该任务 QoS 级别所需要的时间也随之增大,因此,能在任务截止期内提高 QoS 级别的任务数有所减少.由于 RQRB 采用轮询方法来提高任务 QoS,而没有考虑任务之间的差异,因此,RQBB 比 RQRB 具有较高的 QoS 收益.

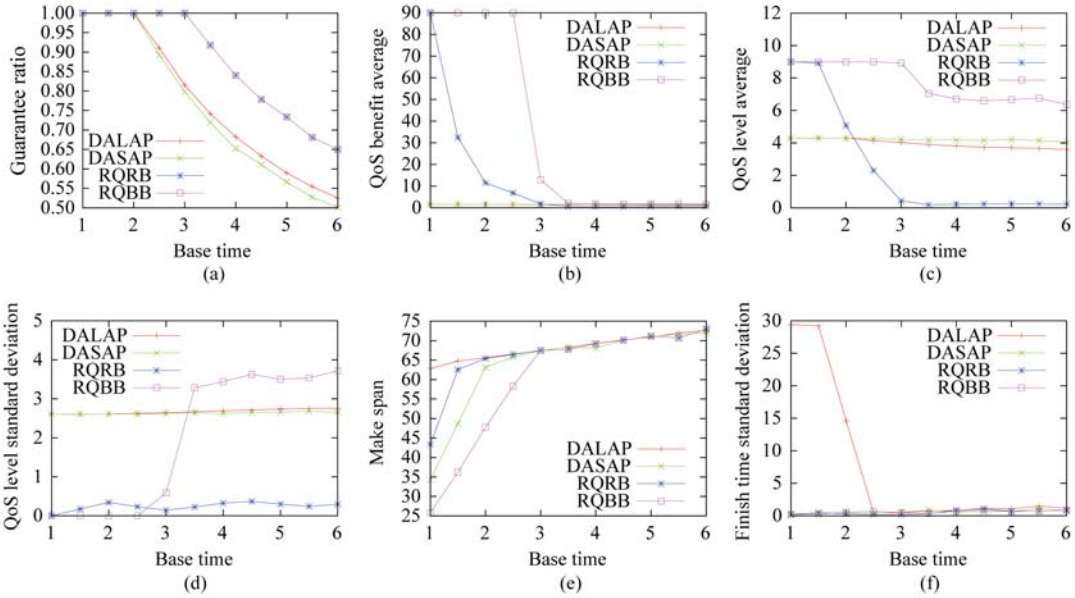


Fig.7 Performance impact of tasks' granularity

图 7 任务粒度对性能的影响

图 7(e)显示出当粒度增大时,所有方法的调度跨度增加,这是因为任务粒度的增大使得任务具有更长的执行时间.尽管如此,RQBB 仍优于其他方法.图 7(f)反映了尽管任务粒度发生变化,但 RQBB,RQRB 和 DASAP 使系统的调度跨度基本保持不变,而 DALAP 使系统的调度跨度变化很大.

本组实验表明,任务分割是一个重要的内容,直接影响着系统的性能.

5 结论及下一步工作

本文提出了一种适合集群软件无线电系统的调度器模型,有效地避免了瓶颈问题.同时,将实时控制器、QoS 控制器和均衡控制器整合在该模型中,达到了信号的高增益、低延迟处理.根据该调度器模型,提出了一种包含 3 个步骤的调度策略——RQBB.第 1 步可以采用任何已有或新提出调度算法以满足任务的时间要求,极大地提高了系统的灵活性和可扩展性.本文中采用的是已有的 DASAP 算法.另外,提出了两种启发式算法——MQB 和 MSD,分别用在 RQBB 的第 2 步和第 3 步.MQB 是一种公平算法,使得所有分配在同一个节点上的任务具有较高的 QoS 级别和较小的 QoS 级别的差异,同时很好地避免了局部极值问题.MSD 算法通过最小化节点完成时间的标准差来达到系统的负载均衡.最后,我们通过大量的实验对 RQBB,RQRB,DASAP 和 DALAP 进行了比较,实验结果表明,RQBB 的性能优于其他方法,具有很强的适应性,适合集群软件无线电系统中的信号处理.

下一步的研究工作主要包括:第一,在 RQBB 策略的基础上,除了考虑任务的处理时间外,进一步考虑任务的通信时间、调度时间和分发时间,使得调度策略更为有效.第二,研究针对集群软件无线电系统的容错调度方法来提高系统的稳定性和可靠性.第三,较为准确地预测任务的执行时间,考虑通过反馈方式使得调度更为精确,同时,定量地给出节点的性能差异.第四,研究自适应的任务分割策略,以提高节点的利用率,减少系统开销.

References:

[1] Zheng K, Wang J, Huang L, Decarreau G. Open wireless software radio on common PC. In: Proc. of the 17th Annual IEEE Int'l Symp. on Personal, Indoor and Mobile Radio Communications. Helsinki: IEEE Press, 2006. 707−716.

[2] Pyndiah R, Glavieux A, Picart A, Jacq S. Near optimal decoding of product codes. In: Proc. of the IEEE Global Telecommunications Conf. San Francisco: IEEE Press, 1994. 339−343.

[3] Yu NY, Kim Y, Lee PJ. Iterative decoding of product codes composed of extended hamming codes. In: Samir T, Mehmet U, eds. Proc. of the 5th IEEE Int'l Symp. on Computers and Communications. Antibes-Juan Les Pins: IEEE Press, 2000. 732−737.

[4] Chi Z, Song L, Parhi KK. A study on the performance, complexity tradeoffs of block turbo decoder design. In: Proc. of the IEEE Int'l Symp. on Circuits and Systems. Sydney: IEEE Press, 2001,4:65−68.

[5] Atdelzater TF, Atkins EM, Shin KG. QoS negotiation in real-time systems and its application to automated flight control. IEEE Trans. on Computers, 2000,49(11):1170−1183.

[6] Qin X, Jiang H. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. Journal of Parallel and Distributed Computing, 2005,65(8):885−900.

[7] Garey MR, Johnson DS. Strong NP-completeness results: motivation, examples, and implications. Journal of Association for Computing Machinery, 1978,25(3):499−508.

[8] Subramani V, Kettimuthu R, Srinivasan S, Johnston J, Sadayappan P. Selective buddy allocation for scheduling parallel jobs on clusters. In: Gropp B, Pennington R, Reed D, Baker M, Brown M, Buyya R, eds. Proc. of the IEEE Int'l Conf. Cluster Computing. Chicago: IEEE Press, 2002. 107−116.

[9] Vallee G, Morin C, Berthou JY, Rilling L. A new approach to configurable dynamic scheduling in clusters based on single system image technologies. In: Proc. of the Int'l Parallel and Distributed Processing Symp. Nice: IEEE Press, 2003. 22−26.

[10] Braun TD, Siegal HJ, Beck N, Boloni LL, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B, Hensgen D, Freund RF. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In: Prasanna VK, ed. Proc. of the 8th Heterogeneous Computing Workshop. San Juan: IEEE Press, 1999. 15−29.

[11] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 1999,59(2):107−131.

[12] Beccari G, Caselli S, Zanichelli F. A technique for adaptive scheduling of soft real-time tasks. Journal of Real-Time Systems, 2005,30(3):187−215.

[13] Topcuoglu H, Hariri S, Wu MY. Performance-Effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Parallel and Distributed Systems, 2002,13(3):260−274.

[14] Palis MA. Online real-time job scheduling with rate of progress guarantees. In: Hsu DF, Ibarra OH, Saldaña RP, eds. Proc. of the Int'l Symp. on Parallel Architectures, Algorithms and Networks. Metro Manila: IEEE Press, 2002. 57−62.

[15] Manimaran G, Murthy CSR. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1998,9(3):312−319.

[16] Li W, Kavi K, Akl R. A non-preemptive scheduling algorithm for soft real-time systems. Journal of Computers and Electrical Engineering, 2007,33(1):12−29.

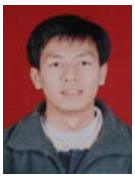
[17] Boyer WF, Hura GS. Non-Evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. Journal of Parallel and Distributed Computing, 2005,65(9):1035−1046.

[18] Baruah SK, Goossens J. Rate-Monotonic scheduling on uniform multiprocessors. IEEE Trans. on Computers, 2003,52(7):966−970.

[19] Xie T, Qin X. Scheduling security-critical real-time applications on clusters. IEEE Trans. on Computers, 2006,55(7):864−879.

[20] Auluck N, Agrawal DP. A scalable task duplication based algorithm for improving the schedulability of real-time heterogeneous multiprocessor systems. In: Huang CH, Ramanujam J, eds. Proc. of the Int'l Conf. on Parallel Processing Workshops. IEEE Press, 2003. 89−96.

[21] Mittal A, Manimaran G, Murthy CSR. Integrated dynamic scheduling of hard and QoS degradable real-time tasks in multiprocessor systems. In: Proc. of the 5th Int'l Conf. on Real-Time Computing System and Applications. Hiroshima: IEEE Press, 1998. 127−136.



朱晓敏(1979－),男,辽宁盘锦人,博士生,主要研究领域为集群计算,实时系统.

陆佩忠(1961－),男,博士,教授,博士生导师,主要研究领域为通信信号处理,高性能计算,信息安全.