一种面向同构集群系统的并行任务节能调度优化方法

李 新[1] 贾智平[1] 鞠 雷[1] 赵衍恒[1] 宗子良[2]

[1](山东大学计算机科学与技术学院 济南 250101)

[2](德克萨斯州立大学计算机科学系 圣马科斯 78666 美国)

摘 要 节能调度算法设计是高性能计算领域中的一个研究热点.复制调度算法能够减少后继任务等待延时,缩短任务总体调度时间,但是耗费了更多的能量.为此,作者提出一种启发式处理器合并优化方法PRO.该方法按照任务最早开始时间和最早结束时间查找处理器时间空隙,将轻负载处理器上的任务重新分配到其它处理器上,从而减少使用的处理器数目,降低系统总体能耗.实验结果表明,和已有的复制任务调度算法TDS、EAD和PEBD相比,优化后的调度算法在不增加调度时间的条件下,能够明显减少使用的处理器数和系统总体能耗,从而更好地实现性能和能耗之间的平衡.

关键词 绿色网络;集群;并行;同构;前驱约束;节能调度;绿色计算

中图法分类号 TP393 **DOI号**: 10.3724/SP.J.1016.2012.00591

Energy Efficient Scheduling and Optimization for Parallel Tasks on Homogeneous Clusters

LI Xin[1] JIA Zhi-Ping[1] JU Lei[1] ZHAO Yan-Heng[1] ZONG Zi-Liang[2]

[1](*School of Computer Science and Technology, Shandong University, Jinan* 250101)

[2](*Department of Computer Science, Texas State University, San Marcos TX* 78666, *USA*)

Abstract The design of energy-efficient scheduling algorithms has become a hot research topic in high performance computing. To shorten schedule length of parallel tasks with precedence constraints, scheduling algorithms could duplicate tasks on critical paths to avoid communication delay caused by inter-task dependence. However, task duplications incur more energy consumption. In this paper, we propose a heuristic Processor Reduction Optimizing (PRO) approach to reduce the number of processors used to run parallel tasks, thereby decreasing system energy consumption. The PRO approach can find appropriate time slots to accommodate tasks from low-utilized processors according to their earliest start time and earliest complete time. Extensive experimental results show that the proposed PRO approach, compared to existing duplication-based scheduling algorithms, such as Task Duplication Scheduling (TDS), Energy-Aware Duplication (EAD) and Performance-Energy Balanced Duplication (PEBD) algorithms, can effectively decrease the number of used processors and save energy without performance degradation.

Keywords green network; cluster; parallel; homogenous; precedence constraint; energy-efficient scheduling; green computing

---

收稿日期:2011-08-25;最终修改稿收到日期:2012-01-10.本课题得到国家自然科学基金(60903031,61070022)、U.S. National Science Foundation(CNS-0915762,CNS-1118043)、山东省自然科学基金(ZR2010FM015)、山东省优秀中青年科学家科研奖励基金(BS2010DX017)和山东大学自主创新基金(2009TS032)资助.**李 新**,男,1978年生,博士,副教授,主要研究方向为高效能计算、任务调度和嵌入式系统.E-mail: lixinsdu@gmail.com; lx@sdu.edu.cn. **贾智平**,男,1964年生,博士,教授,博士生导师,主要研究领域为嵌入式系统和可信计算. **鞠 雷**,男,1982年生,博士,副教授,主要研究方向为嵌入式实时系统分析与优化. **赵衍恒**,男,1987年生,硕士研究生,主要研究方向为嵌入式系统和任务调度. **宗子良**,男,1979年生,博士,助理教授,主要研究方向为高效能计算、并行编程和分布式存储系统.

1 引 言

随着高性能计算的发展,大型集群(机群)系统的能量消耗越来越多,绿色节能成为高性能计算必须考虑的重要因素之一[1].例如,2006 年美国境内的服务器和数据中心的总能耗为 614 亿千瓦时,几乎等于 580 万美国家庭用电的总和①.如何降低现有计算机系统的能耗,设计能量有效利用的高效能集群系统,已经成为高性能计算亟需解决的问题之一[2].

集群系统中节能调度问题是为系统中每一个并行任务分配处理器等资源,并指派占用这些资源的起止时间,在满足依赖关系的条件下,使得所有任务较早完成,并且能量消耗尽量少.节能调度与传统的并行任务调度相比,主要目标不是减少任务总体完成时间(调度长度),而是尽量减少使用资源数目和占用时间,提高资源利用率,达到整个集群系统性能和能耗之间的平衡.

多处理器/机上并行任务最优调度问题已经证明是 NP 难的[3].国内外学者普遍采用启发式算法生成调度方案,减少任务总体完成时间,降低系统能耗.TDS(Task Duplication Scheduling)调度[4]将所有任务按照依赖关系图的拓扑结构分成多条路径,每个路径上的任务分为一组,分配到一个处理器上执行.该方法通过在数据相关的任务分组上复制执行前驱任务减少处理器间的数据传输延迟,缩短所有任务的总体执行时间.然而,任务复制执行在提高性能的同时,带来了能耗增加问题.为此,Zong 等人[5]在 TDS 的基础上,提出了两种非抢占式离线节能调度算法——EAD(Energy-Aware Duplication)调度和 PEBD(Performance-Energy Balanced Duplication)调度,分别通过设置任务能量阈值和能量/时间比阈值,减少低能效任务复制次数,从而降低系统的总体能量开销.

本文在以上研究的基础上,设计了一种启发式处理器合并优化方法,简称 PRO(Processor Reduction Optimizing).该方法通过合并任务数较少的任务分组,减少处理器使用数目,从而降低系统总能量开销.PRO 方法与 TDS、EAD 和 PEBD 调度结合形成了 3 种优化的调度算法 TDS-PRO、EAD-PRO 和 PEBD-PRO.实验结果表明,优化后的 3 种算法与优化之前的相应算法相比,任务总体执行时间略微减少或相等,而总体能量开销平均减少 23.71%、21.28%和 23.51%.

与已有的研究工作相比,本文的创新之处在于:

(1)在文献[5]的基础上,提出一种改进的集群系统能耗统计模型,考虑了任务复制情况下能耗统计.该模型将处理器、网卡和网络交换机作为系统能耗的主要构成,给出了详细的统计方法.

(2)提出了一种处理器合并优化方法 PRO.该方法与现有的任务复制执行策略相结合,能够得到优化的调度方案,减少使用的处理器数目,降低系统总能耗.

(3)开发了一个并行任务调度模拟程序,实现了 TDS、EAD、PEBD 和本文提出的 3 个改进调度算法,并使用来自于真实应用的任务集对算法进行了测试和评估.

本文第 2 节介绍相关工作;第 3 节给出节能调度问题的形式化描述,包括任务模型、处理器模型和能耗统计模型;第 4 节详细描述节能调度方法;第 5 节对算法的性能进行测试和分析;最后总结全文.

2 相关工作

在集群上运行的并行程序需要大量的数据处理和数据交换,集群管理系统需要对这些并行程序分配资源和调度.分配和调度策略对整个系统性能具有重要的影响.总体而言,并行调度策略可以分为 3 种:基于优先级的调度、基于分组(聚类[6],cluster)的调度和基于复制的调度[5].首先,基于优先级的调度[7]为每个任务分配一个优先等级,根据等级将任务分配到处理器上执行.基于分组的调度算法[8]尽可能将有数据通信的任务划分到一个分组,然后按分组指派处理器,从而减小处理器间通信消耗.基于复制的调度 TDS[4]利用处理器的空闲时间复制前驱任务,避免某些前驱任务的通信数据传输,从而减少传输延迟.复制策略往往和分组策略共同使用,在不同分组中复制关键路径上的前驱任务,减少数据传输延迟[5].在大多数情况下,基于复制的调度在性能(调度长度)上优于非复制的调度方法,尤其当任务之间传输数据量较多时.然而,由于任务被复制后在不同处理器上重复执行,这增加了系统的总能耗.为了解决这个问题,Zong 等人[5]提出了两种能量敏感的复制调度算法(EAD 和 PEBD),减少任务复制次

① http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf. 2011,12,20

数，实现性能提升和能耗节约的折中. 本文提出的处理器任务合并方法是对 TDS、EAD 和 PEBD 算法的改进，通过减少处理器使用数目，进一步降低系统总能耗.

另外，动态功率管理机制[9]也是一种有效的节能方法. 该方法通过及时关闭或休眠某些不使用的部件，减少系统资源占用和能量消耗. 文献[10]将反馈控制理论引入到异构集群系统中，提出一种基于处理器使用率的反馈控制算法，实现动态的负载平衡. 动态电压/频率调整(Dynamic Voltage and Frequency Scaling, DVFS)技术[11-16]通过降低芯片内部电压或频率，减小处理器芯片的功率，从而减小集群系统的能量开销. 在通信密集型应用中，网络通信的能耗在总能耗中占比重较大，因此动态电压/频率调整的好处可能会降低[5]. Soteriou 等人[9]研究了如何根据通信情况变化，动态打开或关闭网络连接. Gunaratne 等人[17]提出了一种自适应连接速率方法，根据网络利用率动态调整网络连接状态，减少以太网能量开销. 文献[18]表明，采用 DVFS 的互连技术能够明显减少能量开销. 但是，他们的方法要求网络设备(网卡和交换机等)具有多个通信速率和功耗等级，并且能够动态切换. 然而，大多数以太网设备在完全空闲和充分利用两种情况下，几乎消耗同样多的能量[17].

3　节能调度问题模型

节能调度问题由任务模型、服务器模型和能耗统计模型三部分构成. 表 1 中给出了本文使用的主要变量和参数.

表 1　本文使用的主要变量和参数

名称	含义	名称	含义
n	任务数量	$EST(v_i)$	最早开始时间
m	计算机节点数量	$ECT(v_i)$	最早完成时间
N_{switch}	交换机数量	$LAST(v_i)$	最迟开始时间
v_i	第 i 个任务	$LACT(v_i)$	最迟完成时间
t_i	任务 i 的执行时间	$FP(v_i)$	关键前驱
e_{ij}	任务 j 依赖于任务 i	w_{ij}	任务 i 在处理器 j 上的作业
c_{ij}	任务 i 向任务 j 传输数据网络耗时	f_{ij}	作业 w_{ij} 实际结束时间

3.1　任务模型

本文中的任务集合由 n 个具有依赖关系的并行任务构成. 任务调度不可抢占(non-preemptive). 任务集合和依赖关系可以使用任务图来表示.

定义.　任务图 G 是一个有向无环图 DAG (Directed Acyclic Graph)，$G=(V,E)$，其中顶点集合 V 用于表示任务集合，$V=\{v_i \mid 1\leq i\leq n\}$；有向边集合 $E=\{e_{ij} \mid 1\leq i\leq n, 1\leq j\leq n\}$ 表示任务之间的数据流向或依赖关系. 任务 $v_i=(t_i,\Omega_i)$，其中，t_i 表示任务执行时间，$\Omega_i\subseteq V$ 表示任务 i 的前驱任务集合. 任务图中，顶点内部的数字表示任务编号，顶点边上的数值表示任务执行时间. 有向边 e_{ij} 上数值 c_{ij} 表示：如果任务 i 和 j 不在一台计算机上执行时，任务 i 的运算结果传输给任务 j 需要的网络耗时. 如果两个任务在一台计算机上执行，网络传输耗时为 0. 例如，图 1(a)给出了一个具有 5 个任务的集合. 如果任务 3 和任务 1 不在一台计算机上执行，那么将耗用 1 个单位的时间才能将相关结果发送到任务 3 所在的处理器.

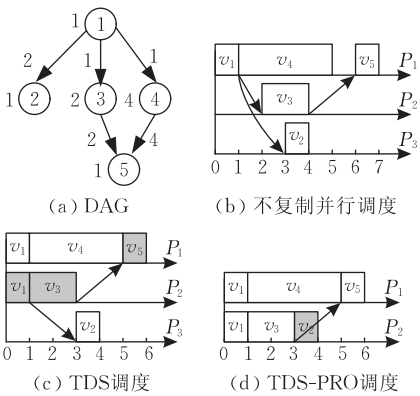


图 1　任务图及调度结果举例

3.2　服务器模型

假设服务器 Π 由 m 个同构计算机节点(下文简称节点)组成，每个节点具有一个单核处理器，$\Pi=\{P_1,P_2,\cdots,P_m\}$. m 的个数根据应用需要动态确定. 任务可以在任意节点上执行，而且执行时间相等. 每个处理器都具有一个固定运行频率. 假设 PC_{busy} 和 PC_{idle} 分别表示处理器执行任务时和空闲时的功率.

分配矩阵 $\boldsymbol{X}$ 定义为 $\boldsymbol{X}=\{x_{ij} \mid 1\leq i\leq n, 1\leq j\leq m\}$，其中，$x_{ij}=1$ 表示任务 i 分配到处理器 j 上执行；$x_{ij}=0$ 表示任务 i 不在处理器 j 上执行. 任务 v_i 在处理器 j 上的一次执行称之为 v_i 的一个作业 w_{ij}，任务 v_i 的所有作业表示为 $\{w_{ij} \mid x_{ij}=1\}$.

3.3　能耗模型

处理器是服务器机柜中最耗能的部分[5]. 而在局域网中，switch 和 hubs 大约占整个网络设备总能耗的 80%[19]. 因此，本文仅统计 CPU、网卡和交换机三部分耗费的能量，作为系统能量消耗.

3.3.1 CPU 能耗

任务 v_i 在节点上执行一次耗费的能量 Ev_i 等于：

$$Ev_i = PC_{\text{busy}} \times t_i \tag{1}$$

考虑到一个任务可能在不同处理器上重复执行，所有处理器处于运行状态所消耗的能量 EC_{busy} 可以用式(2)来计算.

$$EC_{\text{busy}} = PC_{\text{busy}} \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} \cdot t_i \tag{2}$$

处理器处于空闲状态的能耗是空闲功率 PC_{idle} 和空闲时间的乘积. 式(3)表示集群中第 j 个处理器空闲时的能耗.

$$EC_{\text{idle}}^{j} = PC_{\text{idle}} \left(L_{\max} - \sum_{i=1}^{n} (x_{ij} \cdot t_i) \right) \tag{3}$$

其中，$\sum_{i=1}^{n} (x_{ij} \cdot t_i)$ 是第 j 个节点上任务执行总时间；$L_{\max}$ 表示调度长度(schedule length)，即最后一个任务的完成时间. $L_{\max} = \max_{i=1}^{n} \max_{j=1}^{m} (f_{ij})$，其中 f_{ij} 为任务作业 w_{ij} 的结束时间. 任务执行时间之和最大的路径构成关键路径. 关键路径决定调度长度. 例如，图 1(a)中路径 1→4→5 为关键路径.

所有节点空闲时的总能耗 EC_{idle} 为

$$EC_{\text{idle}} = PC_{\text{idle}} \left(m \cdot L_{\max} - \sum_{j=1}^{m} \sum_{i=1}^{n} (x_{ij} \cdot t_i) \right) \tag{4}$$

因此，处理器总能耗 EC 可以表示为

$$EC = EC_{\text{busy}} + EC_{\text{idle}} \tag{5}$$

需要说明的是，这个能耗模型与 DVFS 技术是兼容的. 支持 DVFS 的处理器可能有多个电压频率等级，调度算法可以选择合适的电压或频率作为处理器功率 $PC_{\text{best-fit}}$. 在这种情况下，可以使用 $PC_{\text{best-fit}}$ 代替式(2)中的 PC_{busy}，分段计算处理器运行状态的总能耗.

3.3.2 网卡能耗

假设每个节点上只有一块网卡，PN_{busy} 表示网卡忙碌(收发数据)时的功率，PN_{idle} 表示网卡空闲时的功率. 第 j 个节点中网卡接收数据耗费的能量可以表示为

$$EN_{\text{busy}}^{j} = PN_{\text{busy}} \cdot \sum_{i=1}^{n} \left(x_{ij} \cdot \sum_{vk \in \Omega_i} (1 - x_{kj}) \cdot c_{ki} \right) \tag{6}$$

假设发送和接收同一批数据耗时相等，所有网卡忙碌(发送和接收)时间总能耗为

$$EN_{\text{busy}} = 2 \cdot PN_{\text{busy}} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_{ij} \cdot \sum_{vk \in \Omega_i} (1 - x_{kj}) \cdot c_{ki} \right) \tag{7}$$

类似式(5)，网卡总能耗 EN 等于

$$EN = m \cdot PN_{\text{idle}} \cdot L_{\max} + 2(PN_{\text{busy}} - PN_{\text{idle}}) \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_{ij} \cdot \sum_{vk \in \Omega_i} (1 - x_{kj}) \cdot c_{ki} \right) \tag{8}$$

3.3.3 交换机能耗

本文中，集群互联是同构的，也就是说所有设备以相同的速率传递数据. 文献[11,14]表明空闲和充分利用两种情况下，网络设备几乎消耗同样多的能量. 例如，在 Myrinet-2000 网络中，交换机在不同的通信流量状态下消耗的能量几乎是相等的[20]. 因此，可以假设交换机忙碌时功率与空闲时功率一致，统一使用 PS 表示.

所有节点连接到交换机两层级联网络. 网络中交换机的台数 N_{switch} 与每台交换机端口数 N_{port} 和处理器数 m 有关. 当 $m \leq (N_{\text{port}})^2$ 时，

$$N_{\text{switch}} = \left\lceil \frac{m}{N_{\text{port}}} \right\rceil + 1 \tag{9}$$

交换机总能耗 ES 等于

$$ES = N_{\text{switch}} \cdot L_{\max} \cdot PS = \left(\left\lceil \frac{m}{N_{\text{port}}} \right\rceil + 1 \right) \cdot L_{\max} \cdot PS \tag{10}$$

最后，可以得到服务器的总能耗开销 E：

$$E = EC + EN + ES \tag{11}$$

在以上定义的基础上，调度问题可以描述为：给定一个具有前驱约束的并行任务集合和一个处理器节点集合，寻找一种调度方案使得在满足前驱约束的前提下，所有任务都能较快完成，并且服务器总能量开销最小.

4 节能调度方法

本文提出的节能调度方法先计算各个任务的调度参数，再执行 TDS、EAD 或 PEBD 算法，对任务进行初步分配，然后更新作业开始执行时间和结束时间，最后执行处理器合并优化算法.

4.1 计算任务调度参数

调度算法必须保证前驱任务先执行，后继任务后执行. 为了满足这个条件，使用 *level* 的概念产生任务分配次序. 一个任务的 *level* 被定义为从该任务开始执行到最后一个任务完成的时间. 本文使用文献[4]中提出方法，自下而上计算任务的 *level*.

$$level(v_i) = \begin{cases} t_i, & \text{如果} v_i \text{的后继为空} \\ \max\limits_{k \in v_i \text{的后继集合}} (level(k)) + t_i, & \text{否则} \end{cases} \tag{12}$$

按 *level* 从小到大顺序排列任务，形成任务处理

序列Q. 然后，计算任务的最早开始时间 EST(Earliest Start Time)、最早完成时间 ECT(Earliest Completion Time)、最晚允许开始时间 $LAST$(Latest Allowable Start Time)和最晚允许完成时间 $LACT$(Latest Allowable Completion Time).

没有前驱的任务的 EST 等于 0，其它任务的 EST 可以根据式(13)递归计算出来.

$$EST(v_i)=\begin{cases}0, & \text{如果}v_i\text{的前驱为空},\\ \min\left(\max\limits_{e_{ji}\in E}\left(\max\limits_{e_{ki}\in E,v_k\neq v_j}(ECT(v_j),ECT(v_k)+c_{ki})\right)\right), & \text{否则}\end{cases}\tag{13}$$

任务 v_i 的最早完成时间 ECT 等于：

$$ECT(v_i)=EST(v_i)+t_i\tag{14}$$

每个有前驱的任务都至少有一个关键前驱任务 FP(Favorite Predecessor). $FP(v_i)$ 定义如下：

$$FP(v_i)=v_j,\ \text{其中}\ \forall e_{ji}\in E,e_{ki}\in E,\ j\neq k\,|\,ECT(v_j)+c_{ji}\geqslant ECT(v_k)+c_{ki}\tag{15}$$

一个任务和它的关键前驱任务会被分配到同一个节点上执行，从而得到较短的调度长度.

最后一个任务的最晚允许完成时间 $LACT$ 等于它的最早完成时间. 其它任务的 $LACT$ 用自下而上的方法按照式(16)计算.

$$LACT(v_i)=\begin{cases}ECT(v_i), & \text{如果}v_i\text{的后继为空},\\ \min\left(\min\limits_{e_{ij}\in E,v_i\neq FP(v_j)}(LAST(v_j)-c_{ij}),\right.\\ \left.\min\limits_{e_{ij}\in E,v_i=FP(v_j)}(LAST(v_j))\right), & \text{否则}\end{cases}\tag{16}$$

任务的最晚允许开始时间 LAST 定义为

$$LAST(v_i)=LACT(v_i)-t_i\tag{17}$$

调度算法根据以上参数决定任务执行位置和是否复制.

4.2　任务调度算法

任务复制执行能够减少后继任务的等待延迟，减少调度长度. 有关学者先后提出了任务复制分配算法 TDS 和能量可感知的调度算法 EAD、PEBD.

4.2.1　任务复制调度算法 TDS

TDS 调度[4]通过复制关键路径上的前驱任务，避免前驱任务的数据通信延迟，从而使得关键路径上的任务尽早开始. 在 TDS 算法中，任务分组过程从后继任务为空的任务(假设为 v_i)开始，通过以任务 v_i 为根节点的逆向深度优先搜索，在 DAG 图中从下向上找到多条从任务 v_i 到前驱为空任务的路径. 一条路径上的任务被分到一个任务分组，在同一个处理器上执行. 下一个任务分组从任务处理序列 Q 中第一个未分配的任务开始. 如果前驱任务的复制，不能缩短调度长度，那么不复制该前驱任务. 如果所有任务都被分配过了，那么算法终止. 如图 1(c)所示，通过在处理器 P_2 上复制执行任务 v_1，后继任务 v_3 和 v_5 的开始执行时间可以提前，从而缩短调度长度.

4.2.2　能量可感知的调度算法 EAD

TDS 算法只是缩短了调度长度，而没有考虑能量消耗问题. 任务复制执行会增加 CPU 忙碌时间，也会消耗更多能量. 为此，EAD 算法[5]在 TDS 算法中增加了对任务能耗的判断. EAD 算法首先检查关键前驱任务的复制是否会使当前任务的开始时间提前. 如果能提前，再将执行前驱任务的能耗与节省的网络传输能耗之差(ΔE)作为参数，如果该参数小于或等于所设定的 EAD 阈值，则复制任务；否则，不复制该任务. EAD 算法仅复制增加的能耗不大于 EAD 阈值的任务，所以比 TDS 节省能耗.

4.2.3　性能-能耗平衡的调度算法 PEBD

PEBD 算法[5]和 EAD 算法类似，将任务复制策略和能量优化策略结合在一起. 不同之处在于，PEBD 算法对性能和节能做出了平衡，将执行前驱任务的能耗与节省的网络传输能耗之差/缩短等待时间的比值($\Delta E/\Delta t$)作为参数. 只有该参数小于或等于所设定的 PEBD 阈值时，才复制前驱任务. 与 EAD 相比，PEBD 算法不仅考虑复制执行增加的能耗，而且考虑复制执行缩短的时间. 因此，PEBD 算法更合理. EAD、PEBD 阈值成为决定任务是否被复制的重要因素. 第 5.1 小节对不同阈值的影响进行了测试和分析.

4.3　更新作业调度时间

任务调度算法执行后，需要更新每个处理器上所有作业的调度时间，为执行处理器合并优化算法做准备. 为此，调度时间更新函数从上向下广度优先遍历 DAG 图，依次对每一个任务进行处理，计算任务所有作业的开始时间和完成时间. 如果任务 v_i 是 DAG 图中根节点任务，则设定 v_i 所有作业的开始时间为 0. 否则，一个作业 w_{ij} 的最早开始时间等于 v_i 所有前驱任务的到达时间(完成时间＋传输时间)和处理器 j 中 w_{ij} 前面邻接作业的完成时间的最大值.

图 2 给出了调度时间更新函数，其中 v_k 为任务 v_i 的一个前驱任务，$startTime$ 保存作业最早开始时间的中间过程值.

```
算法 1. UpdateTaskScheduleTime(v_i).
输入：任务 v_i
输出：v_i所有作业的 EST 和 ECT
1. for w_ij ∈ v_i的作业集合
2.   if v_i是 DAG 图中没有前驱的任务 then
3.     EST(w_ij)=0
4.   else then
5.     startTime=-1  //设置开始时间的下边界值
6.     for v_k ∈ v_i的前驱任务集合 Ω_i
7.       if v_k的作业 w_kj 在处理器 j 上执行 then
8.         preECT=ECT(w_kj)
9.       else then  //如果不在一个处理器上
10.        temp1=+∞  //前驱任务完成时间的上界
11.        for P_l ∈ v_k的作业所在处理器集合
12.          if ECT(w_kl) 还没有获得 then
13.            UpdateTaskScheduleTime(v_k)  //递归计算
14.          temp2=ECT(w_kl)+c_ki
15.          if temp2<=temp1 then
16.            temp1=temp2  //v_k结果的最早到达时间
17.        preECT=temp1
18.      if preECT>startTime then
19.        startTime=preECT //前驱任务最晚到达时间
20.  if w_ij不是处理器 P_j上第 1 个作业 then
21.    z=处理器 P_j作业队列中排在 w_ij 前面的作业
22.    if ECT(z)还没有获得 then
23.      UpdateTaskScheduleTime(z)
24.    if ECT(z)>startTime then
25.      startTime=ECT(z)
26.  EST(w_ij)=startTime
27.  ECT(w_ij)=startTime+t_i
```

图 2 任务调度时间更新函数

4.4 处理器合并优化方法

与 TDS 算法相比，EAD 和 PEBD 算法避免了低能效的任务复制，减少了任务复制次数，节约了系统总能耗. 但是这 3 种算法对于分配任务数较少的处理器都没有做合并处理. 如果一个任务集 DAG 图的并行分支比较多，那么许多处理器上分配的任务数会很少，处理器利用率不高，造成处理器资源的浪费.

为此，在上述任务调度算法的基础上，笔者提出一种处理器合并优化 PRO 方法，在维持依赖关系和作业调度时间的前提下，对作业的执行位置进行调整，把处理器上的任务作业尽可能集中到少量处理器上，提高 CPU 利用率，减少使用的处理器数目. PRO 方法的主要思路是使用更新后的 EST 和 ECT 作为参数，将负载轻的处理器上的作业(假设为 w_{ij})合并到在$[EST(w_{ij}),ECT(w_{ij})]$时间范围内空闲的其它处理器上，减少使用的处理器数目，从而降低系统总能耗.

PRO 方法与 TDS、EAD、PEBD 调度算法相结合，可以生成 3 种优化的调度算法 TDS-PRO、EAD-PRO 和 PEBD-PRO. 如图 1(d)所示，TDS-PRO 算法将任务 v_2 合并到处理器 P_2 上，释放了处理器 P_3，减少了处理器个数.

PRO 方法的具体过程如下：首先，将处理器按照分配任务数量由多到少排序. 然后，尝试将任务数最少的处理器(假设为 P_i)上的任务插入到任务数较多的处理器上. 如果这些处理器上没有一个合适的时间空隙接纳该任务，那么停止 P_i 上剩余任务的合并，开始尝试合并下一个处理器 P_{i-1} 上的任务. 如果除了接收合并任务的处理器和无法被合并的处理器之外，没有其它处理器，那么算法结束. 如果某个处理器上的任务都移动到其它处理器上，那么该处理器节点就可以被释放，不再使用. 图 3 给出了处理器合并算法的伪代码，其中$|P_i|$为处理器 P_i 上的作业数.

```
算法 2. ProcessorReductionOptimizing.
输入：(1)初步调度分配方案；(2)更新后的任务调度时间参数
输出：优化后的调度方案
1. 按照作业数量递减顺序生成处理器序列{P_1,P_2,…,P_m}
2. for i←m to 2 //先优化作业少的处理器
3.   if P_i接收过插入作业 then
4.     continue//接收过插入作业的处理器无法被合并
5.   for k←|P_i| to 1//先处理队列尾部的作业
6.     w=P_i上第 k 个作业
7.     bTag=false
8.     for j←1 to i-1 //先从作业多的处理器上查找空隙
9.       if TryMoveJob(w,P_j,P_i) then
10.        bTag=true
11.        break
12.    if bTag=false then
13.      break//遇到无法插入的作业，停止优化 P_i
```

图 3 处理器合并算法

在图 3 中，作业优化检查函数($TryMoveJob$)是一个重要的函数，该函数检查是否能将处理器 P_i 上的作业 w 插入到处理器 P_j 的空隙时间中. 在进行尝试插入之前，首先判断 w 是否在其它处理器上有复制执行. 如果有复制执行，则 w 已经没有执行的意义，将 w 直接从处理器 P_i 上删除，并且返回 true. 如果 w 没有复制执行，判断如果将 w 插入到处理器 P_j，作业 w 的前驱是否会延迟 w 的开始时间. 如果存在延迟，则返回 false. 因为延迟 w 的开始时间，可能会增加调度长度. 否则，查找处理器 P_j 上是否有合适空隙(slot)容纳 w.

作业优化检查函数的伪代码如图 4 所示，其中 $STslot$ 为待检查空隙的开始时间，$ETslot$ 为空隙的结束时间，判断作业 w 是否能够插入到$[STslot,ETslot]$空隙中.

4.5 时间复杂度分析

定理 1. 给定一个受依赖关系约束的并行任务集，优化调度算法的时间复杂度为 $O(h^2n^2)$，n 是任务数，h 是 DAG 图的高度.

算法 3. $TryMoveJob(w,P_{to},P_{from})$.
输入：(1)待插入的作业 w；(2)待插入处理器 P_{to}
(3)作业 w 所在的处理器 P_{from}
输出：如果能插入则返回 true；否则，返回 false
1. if w 已经被其它节点执行 then
2. 从处理器 P_{from} 上删除 w
3. return true
4. for $preTask \in w$ 的前驱任务集合
5. if P_{to} 上没有 $preTask$ 的作业 then
6. if P_{from} 上有 $preTask$ 的作业 then
7. return false //如果 w 被推迟执行，则不移动
8. for $i \leftarrow 0$ to $|P_{to}|$
9. if $i=0$ then //待检查 slot 在作业队列开头
10. $STslot=0$ //slot 开始时间
11. $ETslot=EST(P_{to}$ 上第 1 个作业)
12. else if $i=|P_{to}|$ then //slot 在作业队列结尾
13. $STslot=ECT(P_{to}$ 上最后一个作业)
14. $ETslot=L_{max}$ //处理器调度长度
15. else
16. $STslot=ECT(P_{to}$ 上第 i 个作业)
17. $ETslot=EST(P_{to}$ 上第 $i+1$ 个作业)
18. if $STslot \le EST(w)$ and $ECT(w) \le ETslot$ then
19. 从处理器 P_{from} 上删除 w
20. 移动作业 w 到处理器 P_{to} 上
21. return true
22. else return false

图 4　作业优化检查算法

证明. 优化的节能调度算法执行了 4.1～4.4 节中 4 个阶段的操作.

(1)在计算任务调度参数过程中，首先遍历 DAG 图中所有任务，计算任务的 $level$，复杂度为 $O(e+n)$，其中 e 是 DAG 图中有向边数. 将任务的 $level$ 按非递增的顺序进行快速排序的时间复杂度为 $O(n\lg n)$. 然后，遍历任务计算 EST、ECT、FP、$LACT$ 和 $LAST$ 参数，复杂度为 $O(e+n)$. 因此，第一阶段的时间复杂度为 $O(e+n\lg n)$.

(2)在任务调度算法中，根据复制策略将任务分配到一个或多个节点上执行. 最坏的情况下，关键路径上所有任务都被复制执行，时间复杂度为 $O(hn)$.

(3)更新作业调度时间函数采用广度优先顺序遍历所有任务，调用 $UpdateTaskScheduleTime$ 函数. 最坏情况下，$UpdateTaskScheduleTime$ 函数内部访问所有作业一次，函数时间复杂度为 $O(hn)$. 因此，这一阶段的时间复杂度为 $O(hn(e+n))$.

(4)在处理器合并优化过程，先将处理器按作业数递减进行快速排序，复杂度为 $O(n\lg n)$；然后将任务数少的处理器上的作业尝试插入到任务多的处理器上. 最坏的情况下，$TryMoveJob$ 函数内部需要比较 P_{to} 处理器上所有空隙，其时间复杂度为 $O(hn)$；$TryMoveJob$ 函数最多被调用 hn 次. 因此，此阶段的时间复杂度为 $O(h^2n^2)$.

综合考虑以上分析，由于 $e<hn$，节能调度算法的总时间复杂度为 $O(h^2n^2)$.　　证毕.

5　实验评估

为了比较不同算法的性能，笔者开发了一个模拟调度程序. 该程序可以模拟并行任务的分配和调度过程，并能通过 DAG 图和 Gantt 图形象化显示任务之间的依赖关系和各处理器上任务执行过程. 模拟程序实现了本文提到的 6 个算法 TDS、TDS-PRO、EAD、EAD-PRO、PEBD 和 PEBD-PRO，并统计调度长度、处理器使用个数和系统总能耗开销 3 个评价指标.

不同算法的性能会受到应用程序规模的影响. 在实验中，测试任务集选用两个真实的并行应用——机器人控制(Robot)和 FPPPP 应用①. Robot 和 FPPPP 任务数分别为 88 和 334，其它参数如表 2 所示. Robot 任务集平均出度小，属于计算密集型；FPPPP 任务集平均出度大，属于通信密集型[5]，

表 2　Robot 和 FPPPP 任务集的属性

任务集	任务数目	平均执行时间/s	平均出度
Robot	88	28.2	1.5
FPPPP	334	21.3	3.6

依赖任务之间的数据传输时间通过通信-计算时间比 CCR(Communication-Computation Ratio)和网络消息延迟系数 K 计算出来. 数据传输时间＝$K\times CCR\times$前驱任务执行时间. 网络消息延迟系数 K 是以 Myrinet② 网络消息延迟时间为基准的比例系数.

表 3　不同实验的参数设置

实验编号	任务集合	处理器型号	网络	EAD 阈值	PEBD 阈值	CCR
1	Robot/FPPPP	AMD 85W	Ethernet	0～2000	0～500	0.1
2	Robot	4 种处理器③	Ethernet	200	80	0.1
3	Robot/FPPPP	AMD 35W	Ethernet	200/1500	80/150	0.1
4	Robot	AMD 35W	Ethernet	200	80	0.1～2
5	Robot	AMD 35W	4 种网络④	200	80	0.1

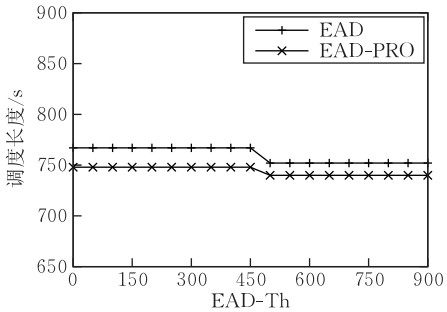
实验分为 5 组，分别对不同的算法阈值、处理器类型、任务集合、CCR 参数和网络类型进行测试. 表 3 列出了具体的实验参数.

① Standard Task Graph Set web site [EB/OL]. http://www.kasahara.elec.waseda.ac.jp/schedule/2011, 12, 20
② http://www.myricom.com/scs/myrinet/overview/2011, 12, 20
③ 详见表 4.
④ 详见表 5.

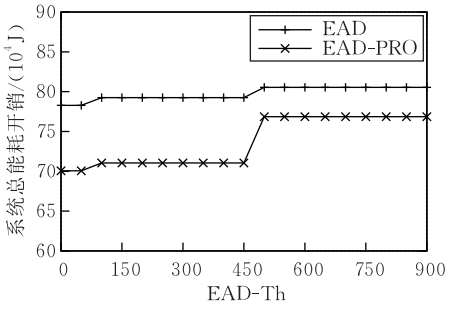
5.1 实验 1：不同算法阈值的影响

EAD、PEBD 阈值增大意味着任务复制执行可能性增加. 本实验通过不同阈值对算法的影响进行分析比较，得到后续实验中使用的阈值.

首先，测试 EAD 阈值(EAD-Th)变化对 EAD 和 EAD-PRO 算法的影响. 从图 5(a)可以看出，在 Robot 任务集上，当阈值增加到 500 时，两个算法的调度长度都略微减少，这说明此时，复制关键路径上前驱任务可以在一定范围内缩短调度长度. 与 EAD 相比，EAD-PRO 的调度长度较少. 这说明通过路径合并，关键路径上有些任务等待时间减少，从而使得整体调度长度缩短.



(a) 调度长度

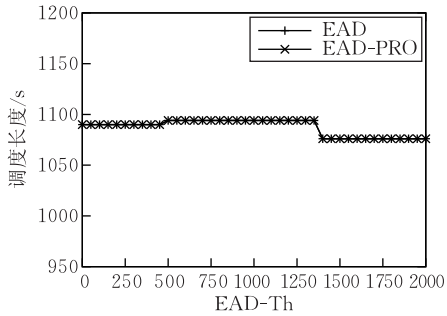


(b) 系统总能耗开销

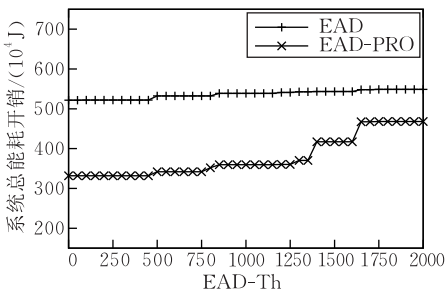
图 5 Robot 任务集上 EAD 阈值对算法的影响

从图 5(b)、图 6(b)可以看出，随着 EAD 阈值的增大，系统的总能耗增加. 同时可以看到 EAD-PRO 的总能耗小于 EAD 的总能耗，这是因为 EAD-PRO 算法将负载轻的处理器上的任务合并到其它处理器上，减少了使用的处理器数目，从而减少了系统总能耗.

图 6(a)表明，FPPPP 任务集的调度长度开始不变，在 EAD 阈值等于 500 时增加. 这是因为关键任务被复制，复制的任务虽然缩短了其后继任务的开始时间，然而其复制之后推迟了关键路径上其它任务的开始，从而使的整个任务集的完成时间延长. 在 EAD 阈值增加到 1400 时，更多的复制任务使得关键路径上的某些任务开始时间提前，从而缩短了调度长度. EAD 和 EAD-PRO 的调度长度相等，这是因为 EAD-PRO 算法在选择任务合并到别的处理器上时，如果合并会延长其后继任务的开始时间，则放弃此任务的复制，所以合并后的执行时间不会晚于原来的执行时间. 后面的实验也能明显体现出改进算法的这一优点.



(a) 调度长度



(b) 系统总能耗开销

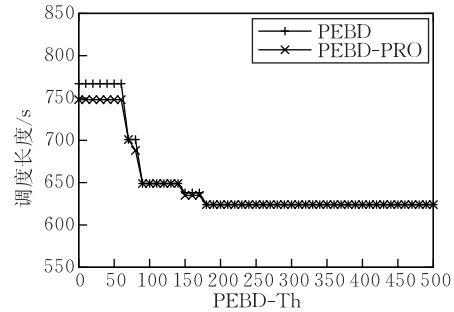
图 6 FPPPP 任务集上 EAD 阈值对算法的影响

综合图 5 和图 6 的数据，可以看出 EAD 阈值在不同任务集上影响不一样. 在图 5(b)中，当 EAD 阈值等于 200 时，总的能量开销处于平均水平，因此选择 200 作为 Robot 任务集上后续实验的 EAD 阈值. 在图 6(b)中，EAD 阈值在 1500 附近，能量消耗处于平均水平，所以选择 1500 作为 EAD 算法在 FPPPP 任务集上的阈值.

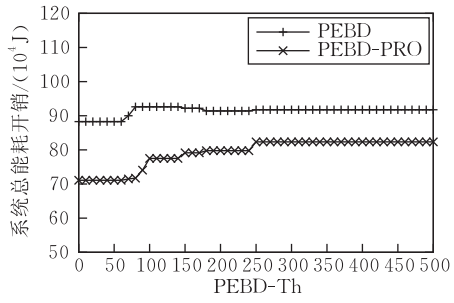
由图 7(a)可以看出，当 PEBD-Th 由 50 逐渐增加到 200 时，Robot 任务集的调度长度明显减少. 这说明前驱任务复制可以使关键路径上任务的开始时间提前，从而缩短了调度长度. 然而在图 8(a)中，FPPPP 任务集上调度长度减少不大.

在图 7(b)中，阈值由 150～200 变化时，PEBD 算法能耗出现了略微减少，这是由于某些任务复制执行减少的网络传输能耗大于复制执行增加的能耗. 图 7(b)、图 8(b)整体来看，随着阈值的增大，PEBD-PRO 算法总体能耗逐渐增加. 与 PEBD 相比，PEBD-PRO 算法的节能效果显著，大约减少 10%～40%.

在后续实验中，PEBD 阈值在 Robot 和 FPPPP 任务集上分别设为 80 和 150，使得两个算法的总能量消耗都处于各自的平均水平.

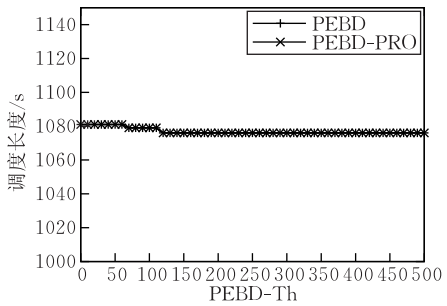


(a) 调度长度

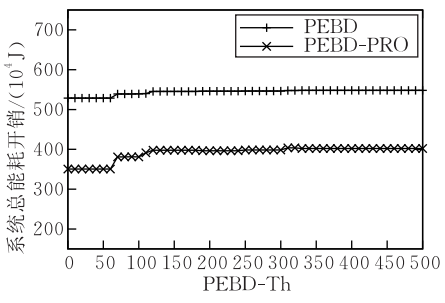


(b) 系统总能耗开销

图 7　Robot 任务集上 PEBD 阈值对算法的影响



(a) 调度长度



(b) 系统总能耗开销

图 8　FPPPP 任务集上 PEBD 阈值对算法的影响

5.2　实验 2：不同处理器功率的影响

本实验采用表 4 中给出的 4 种不同处理器[①]运行功率和空闲功率对算法进行测试. 实验采用 Robot 任务集.

表 4　不同处理器的功率参数

CPU 型号	运行功率/W	空闲功率/W
AMD Athlon 4600+85W	104	15
AMD Athlon 4600+65W	75	14
AMD Athlon 4600+35W	47	11
Intel Core 2 Duo E6300	44	26

由图 9 看出，TDS-PRO、EAD-PRO 和 PEBD-PRO 算法的 CPU 总能耗均比相应的基准算法少. 在 4 种处理器功率情况下，TDS-PRO 相对于 TDS 算法分别节能 15%、16.5%、18% 和 24%；EAD-PRO 相对于 EAD 算法的节能为 12%、13%、6% 和 7%；PEBD-PRO 相对于 PEBD 算法节能 28%、20%、17% 和 22%.

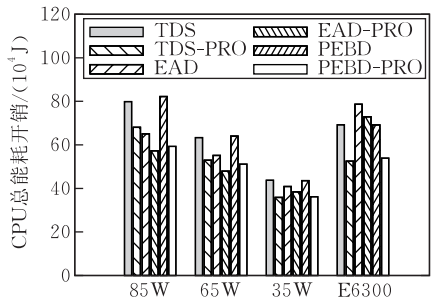


图 9　不同功率组合下 CPU 总能耗开销

5.3　实验 3：不同任务类型的影响

实验 3 使用 Robot 和 FPPPP 两个任务集对算法进行测试，实验结果如图 10 所示.

图 10(a)显示了各算法的调度长度. PRO 方法要求任务重新分配时不能延长后继任务的开始时间，所以新算法的调度长度等于或略小于原算法. 如图 10(b)所示，改进算法在 FPPPP 任务集上明显减少了处理器数目. TDS-PRO、EAD-PRO 和 PEBD-PRO 算法在 FPPPP 任务集上节约处理器数分别为 51%、52% 和 40%，而在 Robot 任务集上节约处理器数分别为 27%、5.4% 和 24%. 这说明，路径分支较多的任务集(如 FPPPP)占用处理器较多，处理器平均利用率较低，任务合并的机会也就越多.

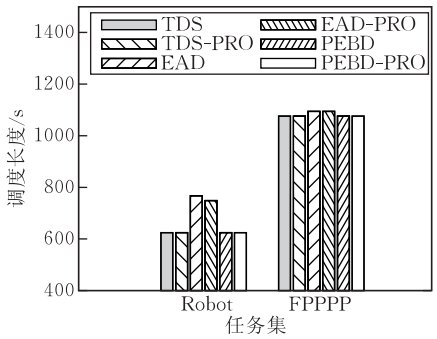
由于任务数和平均出度不同，在不同任务规模下，系统的总能量开销存在着很大的差别. 从图 10(c)可以看出 FPPPP 任务集的能耗开销远大于 Robot 的开销. TDS-PRO、EAD-PRO、PEBD-PRO 在 FPPPP 上的节能效率分别为 33%、35%、27%，在 Robot 上的节能效率分别为 14%、5%、13%. 这验证了，PRO 方法在分支多的通信密集型任务集上不仅能够更多的减少处理器数目，而且具有更好的节能效果.

5.4　实验 4：不同通信-计算比(*CCR*)的影响

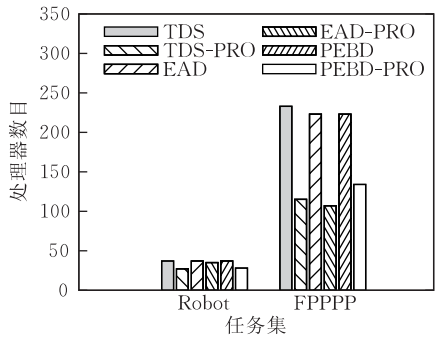
这组实验比较了不同 *CCR* 取值对算法的影响. 实验中，*CCR* 从 0.1～2 变化，间隔 0.1.

从图 11(a)可以看出，当 *CCR* 从 0.4 到 2 逐渐变化时，各算法的调度长度不断增加，这是由于在任

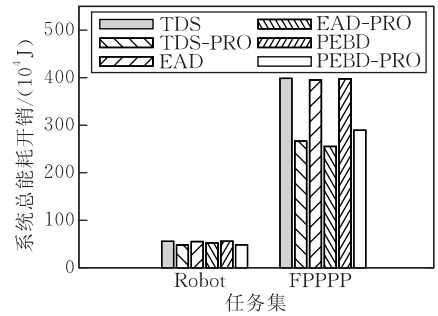
① http://www.xbitlabs.com/articles/cpu/display/amd-energy-efficient_6.html 2011, 12, 20



(a) 调度长度

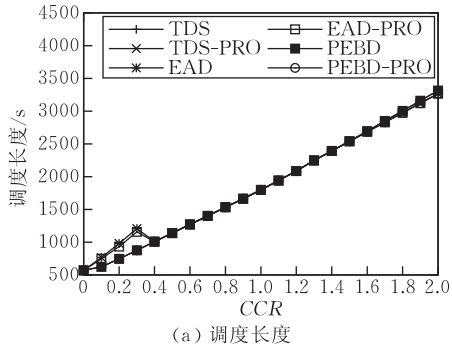


(b) 使用处理器数目

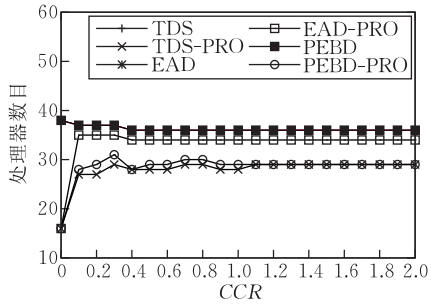


(c) 系统总能耗开销

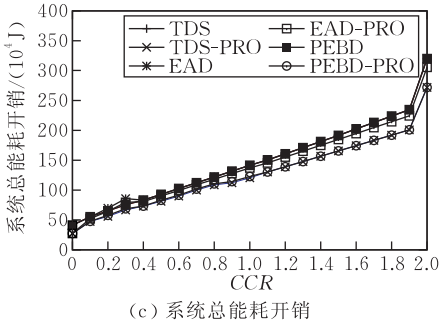
图 10 不同任务类型对算法的影响



(a) 调度长度



(b) 使用处理器数目



(c) 系统总能耗开销

图 11 不同 CCR 参数下算法性能比较

务执行时间不变的情况下,*CCR* 增加引起网络延迟增大,调度长度也会增加.与此同时,传输能耗增大,总能耗也相应增加.

CCR 的变化影响到网络传输延迟,*CCR* 在 0～0.3 范围内变化时,TDS、EAD、PEBD 算法使用的 CPU 个数略微减少,而 3 个改进算法使用的 CPU 个数变化比较大.这说明 *CCR* 参数不仅影响处理器间的网络通信时间,而且影响改进算法的合并效果.当 *CCR* 在[0,1]范围内变化时,改进算法受 *CCR* 参数的影响比较大.

由图 11(c)可得,随着 *CCR* 的增加,各个算法的系统总能耗不断增加.改进算法的能耗比相应的基准算法的能耗略少.

5.5 实验 5:不同网络的影响

不同类型的网络具有不同的设备功率和网络延迟.与文献[5]类似,本实验中采用千兆以太网、Infiniband①、QsNet[21] 和 Myrinet 4 种不同类型网络,测试网络设备参数对算法性能的影响.表 5 给出了这 4 种类型网络的主要参数.

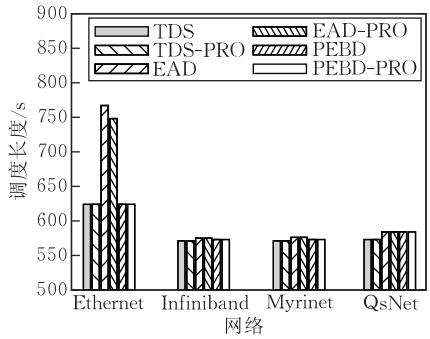
表 5 不同网络设备参数

网络类型	交换机功率/W	网卡功率/W	每台交换机端口数	相对于 Myrinet 的消息延迟系数 K
GB Ethernet	75	5	24	13
Infiniband	25	10.6	24	0.9
Myrinet	55.2	9.3	32	1
QsNet	42	12	8	1.73

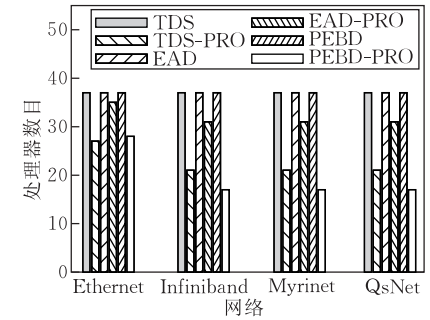
图 12(a)表明,算法在 Ethernet 上执行时间最长,QsNet 次之,Infiniband 和 QsNet 的执行时间最小.这是由消息延迟系数引起的.延迟系数越大,网络延迟越大,执行时间越长.由图 12(b)看出,3 种改进算法在 Infiniband、Myrinet 和 QsNet 网络中占用处理器数目差不多,具有类似的性能.图 12(c)表明改进算法在不同网络下,总能耗比相应基准算法要

① http://www.mellanox.com/pdf/products/silicon/InfiniScaleIII.pdf 2011,12,20

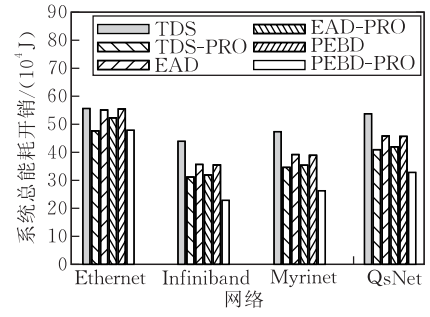
小，节能效果明显.



(a) 调度长度



(b) 使用处理器数目



(c) 系统总能耗开销

图 12　不同网络下算法性能比较

总体来看，改进算法在 Ethernet 上使用的处理器数目和总能耗开销相对较大. 这是由于 Ethernet 的网络延迟系数较大造成的. 在这 4 种网络上，改进算法都具有较好的节省资源效果.

5.6　实验小结

根据以上 5 个实验结果和分析，我们可以看到 PRO 类优化算法与相应的基准算法相比，调度长度略微减少或相等，而使用处理器个数和总体能量开销明显减少. 综合统计 5 个实验中所有结果，TDS-PRO、EAD-PRO 和 PEBD-PRO 算法的能耗开销比优化前的算法平均减少 23.71%、21.28% 和 23.51%.

6　结束语

本文提出了一种考虑处理器、网卡和交换机的能耗统计模型，对集群服务器系统的能耗进行建模. 在此基础上，对带前驱约束的并行任务调度问题进行了讨论，提出了一种减少处理器数目的调度优化方法. 该方法能够找到合适的处理器空闲时间，将轻负载处理器上的任务合并到其它处理器上，减少处理器使用数目，从而降低系统总能耗. 实验结果表明，在没有增加处理器调度长度的情况下，优化后的算法能够明显减少使用处理器数目和系统总能耗开销.

参　考　文　献

[1] Lin Chuang, Tian Yuan, Yao Min. Green network and green evaluation: Mechanism, modeling and evaluation. Chinese Journal of Computers, 2011, 34(4): 593-612(in Chinese)
(林闯，田源，姚敏. 绿色网络和绿色评价：节能机制、模型和评价. 计算机学报，2011，34(4)：593-612)

[2] Elnozahy E N M, Kistler M, Rajamony R. Energy-efficient server clusters. Power-Aware Computer Systems, 2003, 2325: 179-197

[3] Huang Jin-Gui, Chen Jian-Er, Chen Song-Qiao. Parallel-job scheduling on cluster computing systems. Chinese Journal of Computers, 2004, 27(6): 765-771(in Chinese)
(黄金贵，陈健二，陈松乔. 网络机群计算系统中的并行任务调度. 计算机学报，2004，27(6)：765-771)

[4] Ranaweera S, Agrawal D P. A task duplication based scheduling algorithm for heterogeneous systems//Proceedings of the Parallel and Distributed Processing Symposium. Atlanta, USA, 2000: 445-450

[5] Zong Ziliang, Manzanares Adam, Ruan Xiaojun, Qin Xiao. EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. IEEE Transactions on Computers, 2011, 60(3): 360-374

[6] Du Xiao-Li, Jiang Chang-Jun, Xu Guo-Rong, Ding Zhi-Jun. A grid DAG scheduling algorithm based on fuzzy clustering. Journal of Software, 2006, 17(11): 2277-2288(in Chinese)
(杜晓丽，蒋昌俊，徐国荣，丁志军. 一种基于模糊聚类的网格 DAG 任务图调度算法. 软件学报，2006，17(11)：2277-2288)

[7] Sih G C, Lee E A. A compile time scheduling heuristic for interconnection-constrained heterogeneous processors architectures. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(2): 175-187

[8] Pande S S, Agrawal D P, Mauney J. A scalable scheduling method for functional parallelism on distributed memory multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1995, 6(4): 388-399

[9] Soteriou V, Peh L S. Dynamic power management for power optimization of interconnection networks using on/off links//Proceedings of the High Performance Interconnects. Stanford, USA, 2003: 15-20

[10] Wang Jie, Wang Hong-An, Fu Yong, Li Xin. Feedback utilization control for heterogeneous real-time clusters. Journal of Computer Research and Development, 2009, 46(10): 1626-1633(in Chinese)

(王洁，王宏安，傅勇，李新. 一种面向异构实时集群系统的使用率反馈控制方法. 计算机研究与发展，2009，46(10)：1626-1633)

[11] Bianchini R，Rajamony R. Power and energy management for server systems. Computer，2004，37(11)：68-76

[12] Ge R，Feng X Z，Cameron K W. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters//Proceedings of the ACM/IEEE Supercomputing Conference. Seattle，USA，2005：34

[13] Hsu C H，Feng W C. A power-aware run-time system for high-performance computing//Proceedings of the ACM/IEEE Supercomputing Conference. Seattle，USA，2005：1

[14] Hsu C H，Feng W C. A feasibility analysis of power awareness in commodity-based high-performance clusters//Proceedings of the IEEE Cluster Computing，Burlington，USA，2005：1-10

[15] Hotta Y，Sato M，Kimura H，Matsuoka S，Boku T，Takahashi D. Profile-based optimization of power performance by using dynamic voltage scaling on a PC cluster//Proceedings of the Parallel and Distributed Processing Symposium. Rhodes Island，Greece，2006：340-347

[16] Kappiah N，Lowenthal D K，Freeh V W. Just in time dynamic voltage scaling：Exploiting inter-node slack to save energy in MPI programs//Proceedings of the ACM/IEEE Supercomputing Conference. Seattle，USA，2006：33

[17] Gunaratne C，Christensen K，Nordman B，Suen S. Reducing the energy consumption of Ethernet with adaptive link rate (ALR). IEEE Transactions on Computers，2008，57(4)：448-461

[18] Shang L，Peh L，Jha N K. Power-Efficient Interconnection networks：Dynamic voltage scaling with links. Computer Architecture Letters，2002，1(1)：6-9

[19] Gupta M，Singh S. Greening of the Internet//Proceedings of the ACM Conference on Applications，Technologies，Architectures and Protocols for Computer Communications. Karlsrhue，Germany，2003：19-26

[20] Zamani R，Afsahi A，Qian Y，Hamacher C. A feasibility analysis of power-awareness and energy minimization in modern interconnects for high-performance computing//Proceedings of the IEEE Cluster Computing. Austin，USA，2007：118-128

[21] Beecroft Jon，Addison David，Hewson David et al. QsNetII：Defining high-performance network design. IEEE Micro，2005，25(4)：34-47

LI Xin，born in 1978，Ph. D.，associate professor. His research interests include energy-efficient computing，task scheduling and embedded system.

JIA Zhi-Ping，born in 1964，Ph. D.，professor，Ph. D. supervisor. His research interests include embedded system and trust computing.

JU Lei，born in 1982，Ph. D.，associate professor. His research interests include design，analysis，and optimization of embedded and real-time systems.

ZHAO Yan-Heng，born in 1987，M. S. candidate. His research interests include embedded system and task scheduling.

ZONG Zi-Liang，born in 1979，Ph. D.，assistant professor. His research interests include energy-efficient computing，parallel programming and distributed storage systems.

Background

This paper is a research product from our projects mainly supported by the National Natural Science Foundation of China (NSFC) under grant No. 60903031 and 61070022，the U. S. National Science Foundation under Grant No. CNS-0915762 and CNS-1118043. These projects explore energy-efficient techniques or thermal-aware methods for scheduling parallel tasks in multi-core or cluster systems. We also develop simulator programs that can analyze the run-time power consumption of different system components (e. g. processors，network and disks). Now we are designing novel scheduling algorithms to achieve high performance，reliability and energy efficiency.

Many research projects have taken measures to reduce power cost of data centers. For example，IBM Blue Gene®/L supercomputer makes use of low-frequency，low-power processors with modest performance to save energy. The other well known idea is to utilize the Dynamic Voltage and Frequency Scaling (DVFS) technology in power-scalable clusters，in which the power level will be scaled down when processors are not fully utilized and scaled up when processor workloads are heavy. Intel SpeedStep and AMD PowerNow! are typical examples of DVFS approaches. Researchers in Princeton University investigated the possibility of introducing DVFS technology to interconnections. In addition，researchers from UC San Diego，Duke University，HP labs and Arizona State University proposed a series of cyber-physical approaches to reduce the power cost caused by cooling systems.

In recent years，we have proposed several scheduling algorithms for soft or mixed real-time systems，especially for data stream management systems. We also have designed energy-efficient resource management mechanisms for large-scale supercomputing systems，mobile clusters and wireless networks. These algorithms have been published in a number of prestigious journals and conferences.

This paper is focused on energy-aware task scheduling on homogeneous cluster servers. We propose a novel heuristic Processor Reduction Optimizing (PRO) algorithm to decrease the number of processors. Simulation results show that the proposed PRO algorithm can significantly reduce energy consumption without performance degradation.