基于自适应功耗管理的高性能计算机作业调度策略的研究

王 洁[1] 曾 宇[2]
(首都师范大学管理学院 北京 100089)[1] (北京市计算中心 北京 100094)[2]

摘 要 作业调度系统是高性能计算机的核心组件,其目标是在满足性能要求的前提下,使得所有任务消耗的总功耗最低。提出了一种自适应功耗管理策略,该策略采用遗传算法作为功耗调度算法,采用作业队列的能效比作为调度因素,与面向资源效率的传统作业调度算法相比,在确保提升资源利用率、减少资源碎片、提升作业吞吐率、减少饥饿作业的前提下,大幅提升了系统的能效比。实验证明该策略能有效提高整机能效,与传统作业调度策略相比能节约9%以上的能耗。

关键词 自适应功耗管理,作业调度,遗传算法,高性能计算
中图法分类号 TP312 **文献标识码** A

Research of Job Scheduling Strategy of High-performance Computer Based on Adaptive Power Management

WANG Jie[1] ZENG Yu[2]
(School of Management, Capital Normal University, Beijing 100089, China)[1] (Beijing Computing Center, Beijing 100094, China)[2]

Abstract The job scheduling system is a core component of high-performance computer, and the goal of job scheduling is to meet the performance requirements under the premise, and get the lowest total power consumption of all tasks. We presented a job scheduling strategy based on adaptive power management. The strategy is based on genetic algorithm, and takes the performance/power ratio of the job queue as the scheduling factor. Comparing with the traditional job scheduling algorithms, it largely increases the system's energy efficiency with ensuring the resources utilization rate and the job throughput as well as decreasing the resources pieces and the pending jobs. The experiments show that this strategy can effectively improve productivity, and reduce energy consumption about 9% compared with traditional strategy.

Keywords Adaptive power management, Job scheduling, Genetic algorithm, High-performance computing

1 引言

作业调度系统是高性能计算机系统的核心组件,它负责接收用户的作业请求,并选择适当的资源运行用户作业[1]。调度理论与方法是一类NP完全问题[2],当前衡量作业调度系统的主要指标有两个层面,从系统层面来看,主要包括作业吞吐率与资源利用率;从用户层面来看,主要包括作业最大响应时间与作业平均等待时间、饥饿作业与资源碎片等。作业调度算法用来决定采用何种调度策略进行资源-作业匹配。目前现有作业调度系统(如LSF、PBS)及作业调度算法主要以提升资源利用率、减少资源碎片、提升作业吞吐率、减少饥饿作业为目的,并没有考虑作业在不同节点服务器上运行时功耗的变化,也没有考虑作业的能效需求差异[3]。

自适应功耗管理基于作业的周期性和连续性分析,根据任务之间的相关性和出现概率对负载变化进行预测,根据通讯需求和预测的负载确定任务在不同节点的运行时间和功耗,最小化两者的乘积,即能耗。本文中所提出的自适应功耗管理框架是对高性能计算机系统级作业功耗监控分析以及最优化功耗调度策略的有益探索。在此基础上,力求总结符合实际应用情况的功耗分析和管理算法。我们将基于遗传算法的自适应功耗调度策略作为作业调度的算法,来达到提高资源利用率和降低系统整体功耗的双重目的。通过基于遗传算法的功耗调度来达到最佳节能效果,使得高性能计算机的整体能效最佳。

测试表明,该调度策略能在保证作业的响应时间、作业吞吐率的基础上提高整机能效,基于遗传算法的自适应功耗调度策略与传统作业调度相比能节约9%以上的能耗。

2 相关研究

作业调度算法用来决定采用何种调度策略进行资源-作业匹配。主流作业调度算法包括First come first serve, First fit, Best fit, Greedy, Random, Largest processing time, Shortest processing time, Priority, Machineprio (扩展算法), Minresource(空间共享), Contiguous(连续分配), Maxbalance(性

到稿日期:2012-01-20 返修日期:2012-03-20 本文受国家863计划(2011AA0405)资助。

王 洁(1977—),女,博士,讲师,主要研究方向为并行计算、数据挖掘等,E-mail:wangjie@cnu.edu.cn;曾 宇(1973—),男,博士后,研究员,主要研究方向为高性能计算机、并行计算、云计算等。

能接近),Fastest (node speed)等诸多算法[4]。

常用的资源调度算法有:FIFO 算法:按先进先出的策略执行作业,作业的执行顺序是可以预测的,能够充分保证系统的公平性;FirstFit 算法:按作业到达的顺序扫描作业队列,选择第一个系统资源能够满足的作业运行,它克服了 FIFO 中大作业的阻塞问题;Greedy 算法:扫描整个作业队列,并将各个作业进行组合,从中选出系统资源能够满足的最大作业组合来运行;MinMin 算法:将最好的资源分配给最小的作业,来提高系统的吞吐率;MaxMin 算法:将最好的资源分配给最大的作业,从而获得总体作业的最小完成时间;Sufferage 算法:计算作业最优和次优主机分配的完成时间差,将时间差最大的作业分配给使它最早完成的机器;Xsufferage 算法:与 Sufferage 算法类似,只是将输入、输出数据的传输考虑了进去;基于经济学的资源调度算法:通过价格竞争来达到资源供需之间的平衡。

当前高性能计算机主流作业调度系统包括 SUN SGE[5]、LSF、OPEN PBS、PBS PRO、IBM Loadlevel 等。其中,应用最为广泛的是 LSF(Load Sharing Facility)[6] 和 PBS(Portable Batch System)[7] 作业调度系统。

在作业的调度模式上,LSF 提供了可扩展的作业选取策略框架,支持多种作业选取策略,并允许用户自行确定策略,并提供抢占式调度和关键资源保障,保证紧急作业的调度。在资源分配上提供公平共享和独占式策略。PBS 在作业调度策略上提供了默认的 FIFO 调度策略,还提供了 TCL、BACL、C 3 种过程语言和调度类,定义了一些调度需要的函数,以及提供了完整的 API,方便实现新的调度策略。作业调度和节点分配策略是可配置的,对于节点分配策略,提供了公平共享和独占两种策略。

3 自适应功耗管理

由于作业队列中的任务共享高性能计算机的计算资源,任务的相互影响不能忽略。作业调度的目标是在满足性能要求的前提下,所有任务消耗的总功耗最低,因此高性能计算机系统节能管理调度策略研究必须考虑如下方面[3]:

1) 周期性:分析系统总体负载的变化,发现其中的周期性现象,据此预测负载变化并预留相关资源。一般通过计算负载变化的自相关函数进行分析。

2)连续性:分析系统负载变化中的连续性特点,在任务开始执行前,通过分析之前几个任务周期中的负载变化趋势预测任务产生的负载,以便配置计算资源。

3) 节点服务器间的通讯开销:在根据任务功耗进行调度时,预测任务间的通讯量,考虑不同节点的通讯能力对任务处理时间的影响。

4) 数据相关性:在科学计算中,任务的处理时间与输入数据与采用的计算方法有很大关系。预测任务处理时间和所需功耗时,应该考虑到输入数据的影响。

5)节点异构性:管理程序分发任务时,可能会使用一组异构的节点共同处理任务,由于处理能力的差别会造成性能占优节点等待低速节点的情况,增加任务总处理功耗,故需要考虑节点的相互影响。

6)时限:借鉴 Pering[8] 和 Grunwald 等人[9] 在 DVS(dynamic voltage scaling)控制中提出的时限概念,优先级较低的任务并不需要服务器使用极限速度处理,可以通过调整节点服务器峰值功率的方式来降低服务器的处理速度,只要能够在时限结束之前完成任务就能够满足需求。

3.1 系统框架结构

综合考虑上述因素,自适应功耗管理功能模块图如图 1 所示,其目的是要实现高性能计算机的能效比最高。

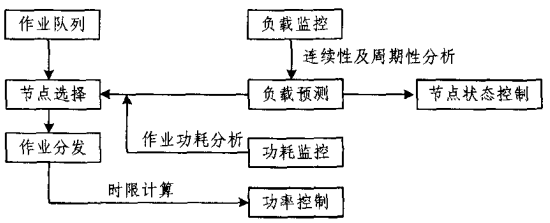


图 1 自适应功耗管理功能模块图

3.2 作业能效的计算

作业的负载除以运行作业时产生的能耗,就是作业的能效。其中,能耗可以通过监测获得,而作业负载包括计算、通信、I/O 等方面的负载之和,可以通过计算获取。

任务通讯开销主要是指在并行计算时的节点间通讯需求,与应用相关。例如,石油行业计算任务在数据输入和最后结果汇总整理阶段通讯量很大,在计算过程中各节点通讯量较小,而一些工业计算任务在计算过程中通讯非常频繁,对节点间的通讯延迟非常敏感。除此以外,根据应用和输入数据文件的大小也可以对负载进行估算。

定义 1 CPU 相对性能为 μ,表示 CPU 处理速度的差别。

定义 2 时间片长度为 t,任务执行时间 $T=\sum t$。

定义 3 监控得到的节点功耗为 P_{Online},抽样时间为 T_{sensor}。

定义 4 节点相对处理能力 $\alpha=(P_{\sigma\text{-CPU}}+P_{\sigma\text{-I/O}}/C)\times\lambda$,其中 $P_{\sigma\text{-CPU}}$ 和 $P_{\sigma\text{-I/O}}/C$ 代表节点的计算和通讯能力,C 代表任务的通讯开销,λ 代表输入数据的影响。

根据上述定义,每颗 CPU 负载 $J_{\text{CPU}}=\sum$CPU 使用率$\times t$,归一化任务负载 $J=\sum\mu J_{\text{CPU}}/(\sum\alpha\times T)$。$J_{\text{Tasknow}}$ 为当前任务 Task 产生的负载,扫描已执行的任务队列寻找同一任务 J_{Taskold},预测该任务下一次出现时的负载 $J_{\text{Tasknext}}=(k\times J_{\text{Taskold}}+J_{\text{Tasknow}})/(k+1)$,$k$ 为任务执行时间片数。任务执行节点的功耗 $P_{\text{Task}}=\sum P_{\text{Online}}\times T_{\text{sensor}}$,总功耗 $P_{\text{Total}}=\sum P_{\text{Task}}$,能效比 $E=J_{\text{Tasknow}}/P_{\text{Total}}$,根据定义 4 知,节点执行任务的预期能效比 $E_{est}=P_{avg}/\alpha$,P_{avg} 是该节点运行时的平均功耗,由于 α 与任务相关,因此 E_{est} 随任务而变化。为了保证负载预测的准确性,可以继续向前寻找更多的同类任务,根据多个负载数据进行预测,时间越早,任务负载在计算中的比重越小。这不

同于前面提到的基于任务的DVS算法，即侧重于分析任务可能产生的总负载而不是处理过程中负载的变化情况。任务执行时服务器的能效比，可以反映作业调度系统的优劣。

3.3 获取周期性

并不是所有作业都有周期性，对于负载均衡类应用程序，时间周期性较为明显，需要根据负载变化情况确定时间周期。

为了快速得到周期的长度，可以通过各峰值间的时间间隔来估算。过程如下：

1)确定需要分析的时间段，例如过去一周；

2)确定时间周期的上限 M，例如两天；

3)选择 N 个极限值，N 应该是时间长度的整数倍，例如14(7×2)个最大值；

4)取任意两个极限值的时间间隔 m，计算离散自相关函数：

$$R(m)=\frac{1}{N}\sum_{n=0}^{N-1}s_1(n)s_2(n+m),0\leqslant m\leqslant M$$

其中，R 越大则周期性越强，s 代表任务负载，N 代表取样数量。如果 R 的最大值大于阈值，则使用相应的 m 作为周期长度。

5)根据周期长度预测负载变化。

当同一任务连续出现时，需要分析连续性因素，根据之前该任务运行时的负载变化趋势，预测当前任务产生的负载。例如，最近任务周期的负载为100，在最近5个任务周期，负载的最大相邻周期间波动为20，则当前周期开始的任务负载为120(100+20×100%)，据此计算任务的功耗和运行时间并为之分配资源。

3.4 时限控制

时限是从用户角度分析问题，考虑任务的紧迫性，通过调整节点运行速度，在满足需求的前提下降低功耗。时限可以由用户指定，也可以由管理程序自动选择，最优化运行时间和功耗，过程如下：

1)预测任务产生的负载 S；

2)计算任务在编号为1的节点运行时间 T_1 和编号为2的节点运行时间 T_2；

3)计算任务在不同节点运行产生的功耗 P_1 和 P_2；

4) 选择功耗较低的节点运行任务，有可能出现 $T_1>T_2$，但 $P_1<P_2$ 的情况，此时尽管节点1运行速度较慢，但由于其功耗低，因此更适合运行该任务。

4 基于遗传算法的自适应功耗调度策略

作业调度问题可描述为：高性能计算机拥有 M 个节点，作业调度队列中有 N 个任务，怎样分配方案可以达到最高能效比？本文采用了基于遗传算法(Genetic Algorithm)[10]的自适应功耗调度策略。遗传算法是模拟生物在自然环境中的遗传和进化过程而形成的一种自适应全局优化概率搜索算法。它将问题的求解表示成染色体的适者生存过程，通过染色体群的逐代不断进化，包括复制、交叉和变异等操作，直到满足一定性能指标和收敛条件终止，来求得问题的最优解或满意解。

定义5 集合 $MAP=\{map_1,map_2,\cdots,map_n\}$ 表示由映射方案构成的映射空间集合。一个映射方案为一个二元组 $map=(t_i,r_j)$，表示任务 t_i 分配到节点 r_j 上执行。

定义6 节点处理时间偏差 $q=\max(S_1/\alpha_1,S_2/\alpha_2,\cdots)-\sum S_i/\sum\alpha_i$，代表最大节点处理时间和高性能计算机各节点平均处理时间的偏差。

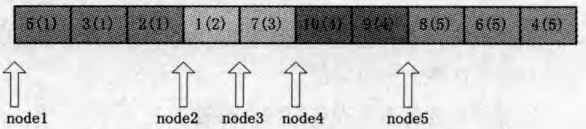
初始种群生成：采用随机调度算法(Gen_Random)。

输入：作业队列 Task，节点全集 R，节点处理时间偏差上限 q

输出：MAP

```
While Task 非空 do
  随机选择资源 r_j
  随机选择作业 t_i∈Task
    If 将作业 t_i 分配到节点 r_j 上满足处理时间偏差 q
  then
      分配 t_i 至 r_j 的作业队列末尾
    End if
    从 Task 中删除 t_i
End while
```

染色体编码：采用直接编码的方式，参数编码如图2所示。



(a) 二维编码

节点	作业分发序列
node1	5,3,2
node2	1
node3	7
node4	10,9
node5	8,6,4

(b) 一维编码

图2 染色体编码

适应度评价：采用作业队列的能效比 E 作为评价标准。遗传算法对一个个体 i 的好坏用适应度函数值 $Fit(i)$ 来评价，适应度函数值越大，解的质量越好。

$Fit(i)=k*TotalPerPow(i)$ 式中，$TotalPerPow$ 表示在第 i 个解下高性能计算机总的能效比，k 为大于1的系数。

选择操作：选择算子的目的是把优化的个体遗传到下一代或者通过交叉产生新的个体遗传到下一代。选择算子的操作建立在群体中个体的适应值基础上。为了保证种群的多样性，在此采用轮盘赌选择算子，它的基本思想是：各个个体被选中的概率与其适应度函数值大小成正比。设群体大小为 n，个体 i 的适应度为 $Fit(i)$，则个体 i 被选中遗传到下一代群体的概率为：

$$P_i=Fit(i)/\sum_{i=1}^{n}Fit(i)=E_i/\sum_{1}^{N}E_j$$

杂交率确定为0.7。从种群(MAP)中选择两个方案，产生一个介于0和1之间的随机数，如果其小于杂交率，则进行交叉操作。

交叉操作：采用单点交叉，随机选择一个位置，交换选择操作选中的两个种群位置后的所有调度方案，如随机产生 map_3，选择操作选择了 MAP_1 和 MAP_2，则新的 MAP_1 包含

原有$\{map_1, map_2, map_3\}$和原来 MAP_2 中的$\{map_4, map_5, \cdots\}$，新的 MAP_2 包含原有$\{map_1, map_2, map_3\}$和原来 MAP_1 中的$\{map_4, map_5, \cdots\}$。

选择操作和交叉操作的随机性可能影响新一代调度方案的有效性，为此采用贪心修补算法(Greedy Repair)恢复方案的有效性。当一个任务被多次分配时，随机保留一个调度，删除其它调度；当一个任务没有被分配时，则对其重新分配(初始种群生成中的随机调度算法)。

变异操作：交叉操作完成之后进行变异操作，本章的变异概率选择为 0.03，新一代中的个体以此概率变异，随机选择调度方案中的两个节点，在它们的作业队列中随机选择两个作业进行交换，变异操作示意图如图 3 所示。

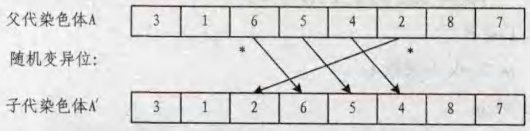


图 3　变异操作

终止条件：满足以下 3 个条件中的任意一个，计算过程结束。

1)循环次数大于设定值；

2)结果在连续多次循环中没有改进；

3)结果达到设定值。

实际使用中，除了限制节点峰值功耗，高性能计算机系统负载较低时，在保留一定冗余计算资源的前提下，可以使剩余服务器节点进入待机状态，以降低能耗；负载增大时，可以通过管理网络快速唤醒待机服务器(唤醒时间小于 5s)，继续处理任务。

5　性能评价

我们对 PBS+Maui[11] 作业管理和采用基于遗传算法的功耗调度策略的作业管理进行了对比测试。对同一组作业分别采用上述两种方法测试其作业响应时间、作业最大响应时间、系统吞吐率、系统能耗等指标。

测试平台如表 1 所列，共 24 颗 CPU 核。

表 1　基于遗传算法的功耗调度策略测试平台

主机名	功能	IP	架构	主频(GHz)
Node0	计算节点(管理节点)	192.168.1.177	双路四核	1.596～2.395(可调频率)
Node1	计算节点	10.0.38.93	双路四核	1.000～2.000(可调频率)
Node2	计算节点	10.0.38.95	双路四核	1.000～2.000(可调频率)

PBS+Maui 采用 Machineprio 调度算法，即根据系统所有节点的配置，从主频、内存、硬盘、网络带宽等方面对节点性能进行评价，得出相应的节点性能优劣指标，以此评价节点优先级。调度时，每次根据作业需求，找出能满足作业需求的优先级最高的节点，将其作为该作业运行节点。在对目前不能运行的作业预约后，对之后的作业进行扫描，在不影响预约作业执行的情况下，将其回填到预约作业执行之前执行，以提高系统利用率及系统性能，回填策略的回填深度为 5。

表 2　PBS+Maui 调度与基于遗传算法的功耗调度对比

	运行时间	作业吞吐率	平均响应时间	最大响应时间	能耗
第一组 PBS+Maui 测试	1:19:26	15.11/Hour	2100.75S	4286S	0.82503KW.h
第二组 PBS+Maui 测试	1:16:11	15.75/Hour	2062.10S	4151S	0.79859KW.h
第三组 PBS+Maui 测试	1:12:08	16.64/Hour	1898.35S	3848S	0.84983KW.h
第四组 PBS+Maui 测试	1:15:04	15.99/Hour	1810.75S	3608S	0.78673KW.h
第五组 PBS+Maui 测试	1:15:24	15.92/Hour	1588.60S	3564S	0.78227KW.h
基于遗传算法的功耗调度测试	1:08:12	18.00/Hour	1609.10S	3132S	0.73010KW.h

基于遗传算法的功耗调度策略，根据过滤后的大作业构建初始种群，每个基因包含作业编号、作业运行的节点服务器和使用的内核数量、作业开始运行时间等信息；评价每个作业的能效比并根据交叉概率选择需要进行交叉的个体，对个体进行交叉操作，对每个基因中的节点进行能效评价。每个节点被该基因保留的概率是该节点的能效比除以该基因所有节点的能效比总和，能效比越高的节点被保留的概率就越大，找出相对较优的可以使系统能耗最小的作业序列进行提交。

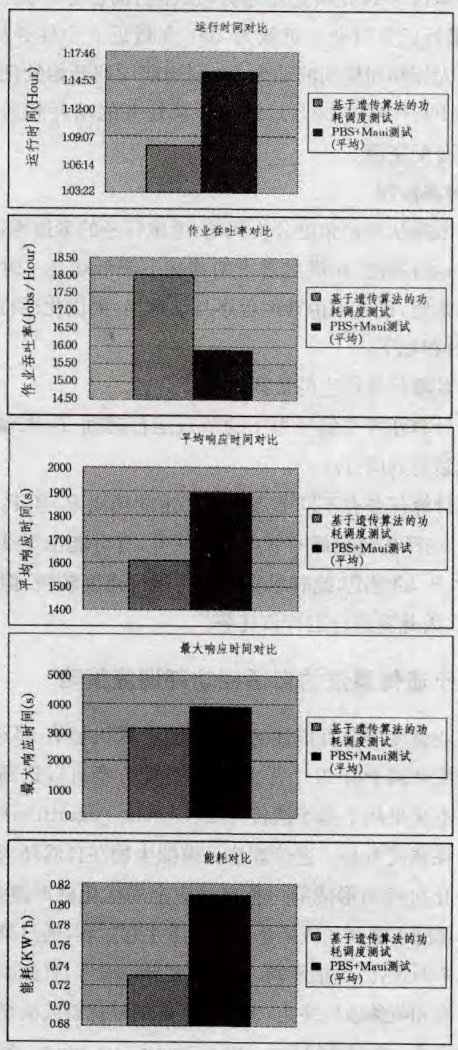


图 4　PBS+Maui 调度与基于遗传算法的功耗调度策略结果对比

定义 10 个不同大小作业，构建作业序列，对测试中使用的作业进行多线程算术运算，线程数量可控，每个线程可以将一个内核的利用率提升至 100%，通过线程数量可以控制节点的 CPU 平均利用率。

基于 PBS+Maui 调度运行 5 组测试，基于遗传算法的功耗调度运行一组测试，测试数据如表 2 所列。

5 组 PBS+Maui 测试结果均值与基于遗传算法的功耗调度测试结果对比如图 4 所示。

可以看到，基于遗传算法的功耗调度与 PBS+Maui 的作业调度相比，平均响应时间减少 14.96%，最大响应时间减少 19.51%，作业吞吐率提高 13.29%，运行时间减少 9.85%，节能 9.67%，具有明显的优势。

图 5 为 PBS 选择作业运行时的功耗和资源利用率变化情况，图 6 为基于遗传算法的功耗调度策略选择作业运行时的功耗和资源利用率变化情况。分析两条曲线的相似度，可知采用遗传算法的功耗调度方式能够使资源利用率和功率的变化更加一致，提高了系统资源和能源利用率，系统能随时保持最佳能效比。

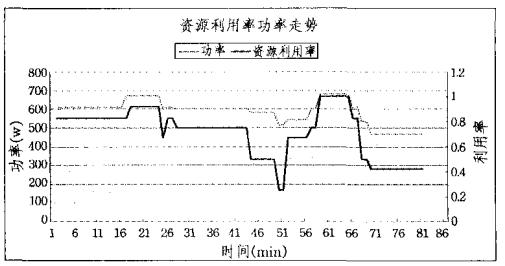


图 5 PBS+Maui 调度作业队列功耗和资源利用率

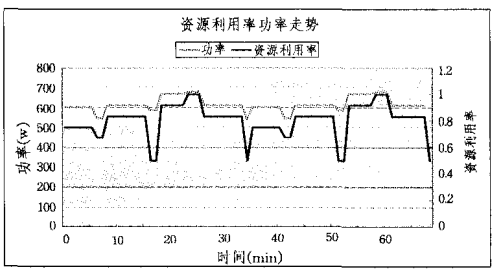


图 6 基于遗传算法的功耗调度作业队列功耗和资源利用率

结束语 本文提出了一种基于自适应功耗管理的高性能计算机作业调度策略，它采用遗传算法作为功耗调度算法，采用作业队列的能效比作为作业调度参数，计算过程中综合考虑了作业类型、预期能效、提交顺序、运行节点、运行时间、系统功耗等因素对最终调度结果的影响。实验证明该策略能有效提高整机能效，与传统作业调度策略相比能节约 9%以上的能耗。

参 考 文 献

[1] Sherwani J, Ali N, Lotia N, et al. Libra: a computational economy-based job scheduling system for clusters[J]. Software-practice and Exoerueice, 2004, 34: 573-590

[2] Weiser M, Welch B, Demers A, et al. Scheduling for reduced CPU energy[C]// Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation. New York: The Advanced Computing Systems Association, 1994: 13-23

[3] 曾宇. 高效能计算机若干关键技术的研究与实现[D]. 北京：中国科学院，2009

[4] Buyya R, et al. High Performance Cluster Computing: Architectures and Systems[M]. Prentice Hall, USA, Volume 1, 1999

[5] Gentzsch W. Sun Grid Engine (SGE): A cluster resource manager[OL]. http://gridengine.sunsource.net/

[6] Platform. Load Sharing Facility (LSF)[OL]. http://www.platform.com/products/wm/LSF/

[7] Systems V. OpenPBS v2.3: The portable batch system software. Veridian Systems, Inc., Mountain View, CA[OL]. http://www.openpbs.org/scheduler.html

[8] Pering T, Burd T, Brodersen R W. The simulation and evaluation of dynamic voltage scaling algorithms[C]// Proceedings of the 1998 International Symposium on Low Power Electronics and Design. Monterey, CA: ACM, 1998: 76-81

[9] Grunwald D, Levis P, Farkas K I, et al. Policies for dynamic clock scheduling[C]// Proceedings of the 4 th Symposium on Operating Systems Design and Implementation. San Diego, 2000

[10] Deb K, Pratap A, Agarwal S, et al. A fast and elitist multi objective genetic algorithm: NSGA-II[J]. IEEE Transactions Evolutionary Computation, 2002, 6: 182-197

[11] Bode B, Halstead D M, Kendall R, et al. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters[C]// Proceedings of 4th Annual Linux Showcase & Conference. Atlanta USA, 2000

(上接第 271 页)

[4] Lin T, Granular Y. Computing on binary relations I: Data mining and neighborhood systems[C]// Skoworn A, Polkowshi L, eds. Proc. of the Rough Sets in Knowledge Discovery. Physica-Verlag, 1998: 107-121

[5] Yao Y Y. Relational interpretation of neighborhood operators and rough set approximation operators[J]. Information Sciences, 1998, 111(198): 239-259

[6] Wu W Z, Zhang W X. Neighborhood operator systems and approximations[J]. Information Sciences, 2002, 144(1-4): 201-217

[7] 胡清华，于达仁，谢宗霞. 基于邻域粒化和粗糙逼近的数值属性约简[J]. 软件学报，2008, 19(3): 640-649

[8] van der Maaten L J P, Postma E O, van den Herik H J. Dimensionality Reduction: A Comparative Review[R]. TiCC TR 2009-005. http://www.uvt.nl/ticc, 2009-10-26

[9] Camastra F. Data Dimensionality Estimation Methods: A survey [J]. Pattern Recognition, 2003, 36(12): 2945-2954