# 基于多种资源的负载平衡算法的研究

蒋 江, 张民选, 廖湘科
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 系统资源的有效利用是集群系统的关键问题, 负载平衡是实现资源有效利用的重要手段. 本文, 提出了两种基于资源使用率和向量负载指数的、采用进程迁移机制的负载平衡算法, 并通过踪迹驱动的方法进行了大量的模拟和分析.

关键词: 集群; 异构性; 负载平衡; 资源使用率; 进程迁移; 向量负载指数
中图分类号: TP316 文献标识码: A 文章编号: 0372-2112 (2002) 08-1148-05

# Study on Load Balancing Algorithms Based on Multiple Resources

JIANG Jiang, ZHANG Min-xuan, LIAO Xiang-ke
(School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China)

Abstract: The key problem of the computing cluster is how to utilize all the resources effectively. Load balancing is a primary means of making the best of all the resources in a cluster. We propose two load balancing algorithm, which are based on the resource utilization and the vector load index. These algorithms adopt the process migration mechanism. We conduct a lot of trace-driven simulations.

Key words: cluster; heterogeneity; load balancing; resource utilization; process migration; vector load index

## 1 引言

### 1.1 概述

集群计算的主要目标是通过网络互连实现全系统范围内的资源的共享, 并且通过高效的资源管理和任务调度技术实现资源的有效共享, 从而提高资源利用率, 获得高性能. 因此, 资源的有效利用是集群系统软件研究的关键问题.

负载平衡(Load Balancing)是实现资源有效共享的重要手段, 它一直是集群计算技术研究的热点问题. 在异构集群系统中, 由于节点和应用的动态性以及资源类型的多样性, 负载平衡问题就更为复杂.

传统的研究是将不同的资源分离地看待, 没有考虑资源之间的相互关系, 如文[1~3]等. 我们的研究是从资源之间的相互关系入手, 将多种资源以向量的方式表示, 通过向量的模和向量运算体现资源间的协调关系, 从而能够更为有效地协调利用多种资源.

### 1.2 系统模型及异构性

我们的研究对象是异构集群系统. 它是由高速互联网络连接起来的多台自治计算机组成的高性能计算系统; 计算节点是开放式的, 用户可以自由地向任何一个计算节点提交不同资源需求的任务, 计算节点为所有用户共享.

集群系统的异构性包含两个方面: 类型异构性和资源异构性. 类型异构性是指计算节点指令集结构(ISA)的不同和操作系统(OS)类型; 资源异构性是指计算节点的多种资源(如CPU、内存和I/O资源)拥有量的不同.

对于资源异构性, 定义:

CPU 资源异构性 $H_{CPU}=\sqrt{\frac{\sum_{i=1}^{N}(W_{CPU}(i)-\overline{W_{CPU}})^2}{N}}$. 其中,

$W_{CPU}(i)=\frac{V_{CPU}(i)}{\max_{j=1}^{N}V_{CPU}(j)}$, $V_{CPU}(i)$ 为计算节点 i 的处理器的速度;

$\overline{W_{CPU}}=\frac{\sum_{j=1}^{N}W_{CPU}(j)}{N}$; N 为系统中计算节点的数目.

内存资源异构性 $H_{MEM}=\sqrt{\frac{\sum_{i=1}^{N}(W_{MEM}(i)-\overline{W_{MEM}})^2}{N}}$.

$W_{MEM}(i)=\frac{TUM(i)}{\max_{j=1}^{N}TUM(j)}$; $TUM_i$ 节点i 的用户可用内存容量(节点的部分内存被 OS 占用); $\overline{W_{MEM}}=\frac{\sum_{j=1}^{N}W_{MEM}(j)}{N}$.

$H_{cpu}$和 $H_{MEM}$的值越大, 系统的资源异构性越大. 对于同构系统, 有 $H_{cpu}=H_{MEM}=0$.

类似地可以定义其他资源的异构性.

收稿日期: 2001-07-20; 修回日期: 2001-12-26
基金项目: 自然科学基金(No. 69933030); 国家"863"高技术研究发展计划基金(No. 863-306-ZD1-20-1); 教育部"5·跨世纪优秀人才培养计划"基金6项目(教技函[2000]1号文件)

## 2　负载平衡算法中的几个关键问题

### 211　进程生命时间分布

通过对集群计算环境中大量进程的生命时间数据的统计和分析,Harchol2Balter[4] 采用统计学方法,得出了进程生命时间的概率分布形式: P{Lifetime> T}= T^k. 其中,k 的值随不同的系统在- 113 到- 018 之间变化,通常靠近- 1. 基于这个分布,可用进程的当前生命时间来预测进程的未来运行时间.

同时,Harchol2Balter 和 Downey 通过分析模拟,认为进程迁移机制能够有效地提高负载平衡算法的性能. 基于他们的工作,在异构集群系统中,对于具有相同类型(ISA 和 OS)的两个节点,可以推导出适合于迁移的进程的最小运行时间为

$$a_t = \frac{f + L/b}{n - V_s/V_d\#(m+1)} \qquad (1)$$

其中,$V_s$ 和 $V_d$ 为源节点和目标节点的处理器速度; n 表示迁移前源节点的进程数目(包含被迁移进程), m 为迁移前目标节点的进程数目;进程迁移代价 c= f+ L/b,f 为固定迁移代价, L 为迁移进程的内存大小, b 为内存传输带宽.

### 212　陈旧信息问题

由于负载信息的收集、散布、使用以及进程真正到达目标节点之间具有时间差,因此在负载平衡决策时使用的负载信息都是陈旧(stale)的信息[5,6]. 陈旧信息可能导致错误的决策. 解决信息陈旧问题的一种有效方法是采用 k 子集算法(k2subset algorithm)[5]. 本文提出了一种最小 k 子集随机算法 SKR,主要步骤为: 当位置策略进行目标节点选择时,首先从所有节点中挑选 k 个负载最少的节点,然后再从这 k(1[ k[ N)个节点中随机选择一个节点作为目标节点. 一方面,SKR 算法能够比 k 子集算法更有效地保证选择到负载相对较少的节点,另一方面它也能够有效地解决位置策略中的群聚效应(Herd Effect)问题. 实验表明,k 的取值一般为7 $\log_2 N$ô.

## 3　基于多种资源的负载平衡算法

### 311　基本原理

对于资源异构性而言,负载平衡算法应当在考虑有效使用全系统范围内同种资源的同时,充分考虑多种资源之间的协调使用问题,从而减少和避免资源使用瓶颈的产生. 因此,负载平衡算法的目标为:

ô 不同节点的同种资源具有相同的资源使用率.

ô 同一节点上的不同资源具有相同的资源使用率.

负载平衡算法有两种实现机制: 初始放置和进程迁移. 在异构集群系统中,由于系统状态和应用特性具有高度可变性,本文的研究将主要集中于进程迁移机制. 本文主要采用在相同类型(ISA 和 OS)的计算节点间进行进程迁移的方法来解决类型异构性问题.

我们的算法主要适用于 I/O 需求和进程间通信需求不强的应用. 强 I/O 应用可以采用 migration2to2data 算法,强通信的应用一般采用 coscheduling 算法.

### 312　向量负载指数

本文主要考虑 CPU 和内存两种资源.

在异构集群系统中,定义节点 h 的负载向量 L(h)为: L(h)= (type(h), res(h)). 其中,节点类型向量 type(h)= (ISA(h), OS(h)),表示计算节点的指令集结构(CPU 类型)和操作系统类型. 节点资源负载向量 res(h)= ($U_{CPU}$(h), $U_{mem}$(h)),表示节点当前的资源使用情况, $U_{CPU}$(h)为 CPU 资源使用率, $U_{mem}$(h)为内存资源使用率. 采用资源使用率作为资源负载指数是解决资源异构性的一种有效手段. 资源负载向量的各个分量表示不同资源的资源使用情况,向量的角度能够显示资源使用的平衡状态.

节点 h 的资源负载向量 res(h)为: $res(h) = \left(\frac{Q(h)}{LCPU_h}, \frac{\sum_{i=1}^{num(h)} mem(i)}{TUM_h}\right)$. 其中, mem(i)为进程 i 的稳定工作集的大小. num(h)为节点 h 上的进程数目. $TUM_h$ 为节点 h 的用户可用内存总量. $LCPU_h$ 为节点 h 的有效 CPU 队列长度, $LCPU_h$ 的值与 CPU 的处理能力成正比. Q(h)为节点 h 上的、采用指数平滑算法的 CPU 队列的长度.

为了衡量进程迁移对源节点的影响,定义迁移进程 i 相对于源节点 h 的资源负载向量 res(h, i)为: $res(h,i) = \left(\frac{1}{LCPU_h}, \frac{mem(i)}{TUM_h}\right)$.

为了衡量迁移进程对目标节点的影响,定义源节点 h 上的迁移进程 i 相对于目标节点 r 的资源负载向量 res(h, i, r)为: $res(h,i,r) = \left(\frac{1}{LCPU_r}, \frac{mem(i)}{TUM_r}\right)$. 其中, $LCPU_r$ 为节点 r 的有效 CPU 队列长度; $TUM_r$ 为节点 r 的用户可用内存总量.

通过资源负载向量的向量加减运算,可以方便地获得迁移进程对源节点和目标节点资源使用情况的影响. 同时,资源负载向量的模提供了一种对节点负载状况进行比较的有效方法. 根据负载平衡目标,系统平衡时的单个节点负载向量的角度为 45 度.

### 313　基于资源的负载平衡算法

从资源协调使用的角度出发,本文提出了两种基于进程迁移机制的负载平衡算法,它们都是发送者初始(Sender Initiated)算法.

当+ res($h_s$) + > Threshold (Threshold 为负载阀值)时,过载的源节点 $h_s$ 启动负载平衡算法.

#### 3131　算法 1

算法的具体步骤如下:

(1)位置策略(目标节点的选择)

第一步　选择与原节点 $h_s$ 类型向量相同的节点的集合 $H_E$,

$H_E$= {$h_i$| $h_i$ I H,且 type($h_i$)= type($h_s$)}. 其中, H 为集群系统中所有节点的集合.

若 $H_E$ 为空,算法停止. 否则,采用 SKR 算法选择目标节点.

第二步　根据 $H_E$ 中所有节点的资源负载向量(资源使用率)的模,从中选择出 k 个负载最轻节点的集合 $E_k$,

$E_k$= {$h_i$| $h_i$I $H_E$,且+ res($h_i$) + 为所有资源负载向量的模中最小的 k 个之一}.

第三步 从 $E_k$ 中选择目标节点 $h_d$,

$h_d = \{h_j \mid h_j \in E_k$, 且 $h_j$ 为 $E_k$ 中随机的任意一个节点$\}$.

(2) 选择策略(迁移进程的选择)

第四步 确定适合迁移的进程的集合. 根据公式(1), 适合迁移的进程集合为:

$$P_E = \{p_i \mid p_i \in P_S, \text{且 } age(p_i) > \frac{1}{num(h_s) - V_s / V_d \#(num(h_d) + 1)} \#(f + mem(p_i)/b)\}$$

其中, $P_s$ 为源节点 $h_s$ 的所有进程的集合; $age(p_i)$ 为进程 $p_i$ 的当前生命时间; $num(h_s)$, $num(h_d)$ 分别表示进程迁移前源节点的进程数目(包含迁移进程)和迁移前目标节点的进程数目;

第五步 从 $P_E$ 中选择迁移进程 $p_m$. 在选择迁移进程时, 需要同时迁移进程对源节点和对目标节点的资源负载向量的影响. 因此, 迁移进程 $p_m$ 应满足:

$$p_m = \left\{ p_i \;\middle|\; p_i \in P_E, \text{且} + res(h_s) - res(h_s, p_i) + + + res(h_d) + res(h_s, p_i, h_d) + = \min_{p_j \in P_E} (+ res(h_s) - res(h_s, p_j) + + + res(h_d) + res(h_s, p_j, h_d) +) \right\}$$

3.3.1.2 算法 2

算法的具体步骤如下:

(1) 选择策略(迁移进程的选择)

第一步 确定适合迁移的进程的集合 $P_E$. 迁移进程的选择的基本要求是: 进程的未来运行时间大于进程的迁移代价. 根据 Harchol-Balter 和 Downey[4] 的研究, 可以用进程的当前生命时间来预测进程的未来运行时间. 因此,

$P_E = \{p_i \mid p_i \in P_s$, 且 $age(p_i) > f + mem(p_i)/b\}$.

若 $P_E$ 为空, 算法停止.

第二步 从源节点 $h_s$ 的 $P_E$ 集合中选择迁移进程 $p_m$. 理想的迁移进程应当具有最大的资源负载向量模、最小的迁移代价和最长的当前生命时间. 因此, 迁移进程 $p_m$ 应当满足:

$$p_m = \left\{ p_i \mid p_i \in P_E, \text{且} + res(h_s, p_i) + @\frac{age(p_i)}{f + mem(p_i)/b} = \max_{p_j \in P_E} (+ res(h_s, p_j) + @\frac{age(p_j)}{f + mem(p_j)/b}) \right\}.$$

(2) 位置策略(目标节点的选择)

第三步 选择符合迁移进程 $p_m$ 类型异构性要求的节点的集合 $H_E$,

$H_E = \{h_i \mid h_i \in H$, 且 $type(h_i) = type(h_s)\}$

若 $H_E$ 为空, 算法停止. 否则, 采用 SKR 算法选择目标节点.

第四步 进程 $p_m$ 迁移到 $H_E$ 的节点后, 形成的资源负载向量的模的集合为:

$$Mmig(p_m, H_E) = \{+ res(h_i) + res(h_s, p_m, h_i) + \mid h_i \in H_E\}$$

从 $Mmig(p_m, H_E)$ 中选择出 k 个模最小的节点构成集合 $E_k$, 有:

$E_k = \{h_j \mid h_j \in H_E$, 且 $+ res(h_j) + res(h_s, p_m, h_j) +$ 为 $Mmig(p_m, H_E)$ 中最小的 k 个之一$\}$.

最后, 选择目标节点. 目标节点 $h_d$ 应当满足:

$h_d = \{h_i \mid h_i \in E_k$, 且 $h_i$ 为 $E_k$ 中随机任意一个节点$\}$

## 4 模拟及分析

本文的性能模拟, 采用踪迹驱动的模拟方法.

### 4.1.1 集群系统模拟器的设计

为了进行模拟和性能评价, 设计并实现了一个集群系统模拟器. 这个模拟器的设计非常灵活, 实现了算法机制与策略的分离. 在模拟中, 采用了如下的假设:

• 计算节点的局部调度采用 round-robin 策略.

• 每个节点都能够获取其他所有节点的负载信息.

• 当计算节点上的进程的稳定工作集之和大于或者等于节点的可用内存时, 节点上的每个任务都将按一定的页故障率 R(每百万条指令的平均页故障次数)发生页故障. 假定页故障的发生在任务执行过程中均匀分布. 页故障时, 进程切换.

### 4.1.2 工作负载

模拟中所用的工作负载来源于 Harchol-Balter 和 Downey[4] 的 8 个踪迹. 为了满足模拟需求, 对这些踪迹进行了合理的处理, 形成了 16 个异构节点的踪迹数据, 数据格式为: < arrival time, arrival node, requested memory size, number of instructions> .

其中, 任务的指令数目是根据原踪迹中任务的 CPU 时间与节点的速度换算出来的; 内存需求采用 Pareto 分布(文[4], 一种 Heavy-tailed 分布)生成, 任务的平均内存需求分别为 1MB, 4MB 和 8MB.

### 4.1.3 模拟结果及分析

#### 4.1.3.1.1 性能评价指标和对比算法

负载平衡算法的主要性能评价指标是平均减速(Mean Slowdown). 减速定义为任务的 wall-clock 时间与它的 CPU 执行时间的比值. 任务的减速主要来自于 CPU 队列的等待时间、内存页故障的调页时间、进程间通信时间以及进程迁移的开销等因素. 平均减速是进行集群系统性能评价的一种有效手段.

为了衡量基于资源的负载平衡算法的效果, 本文模拟了以下五种算法, 它们都采用进程迁移机制:

• NO_LS: 不进行负载平衡.

• CPU_PM: 基于 CPU 队列长度的负载平衡算法, 这是集群系统中最常用的算法.

• MEM_CPU_PM: 优先考虑内存资源, 同时兼顾 CPU 资源的负载平衡算法. 我们主要模拟 X. Zhang 等人[3] 的算法.

• RB_H_PM: 这是我们提出的算法 1.

• RB_P_PM: 这是我们提出的算法 2.

统称 RB_P_PM 算法和 RB_H_PM 算法为 RB_PM 算法.

#### 4.1.3.1.2 对资源异构性的作用

为了验证算法的在异构集群系统中的有效性, 设计了四个集群系统, 如表 1 所示:

表 1

| | 系统 1 | 系统 2 | 系统 3 | 系统 4 |
|---|---|---|---|---|
| $H_{CPU}$ | 0.10 | 0.114 | 0.128 | 0.14 |
| $H_{MEM}$ | 0.10 | 0.10 | 0.138 | 0.14 |

其中, 系统 1 为同构集群系统, 系统 4 具有最大的资源异构性.

我们选择了踪迹 0 来衡量算法的效果. 模拟中我们分别测试了任务平均内存需求为 1MB, 4MB 和 8MB 三种不同情况. 模拟结果如图 1、2、3 所示.

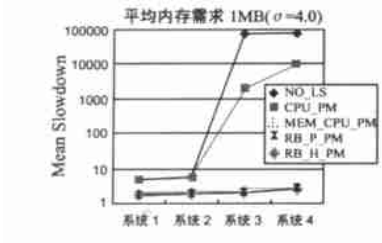


图 1

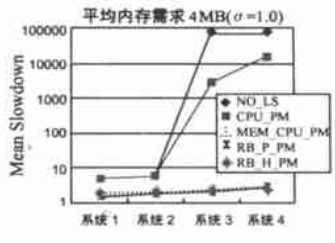


图 2

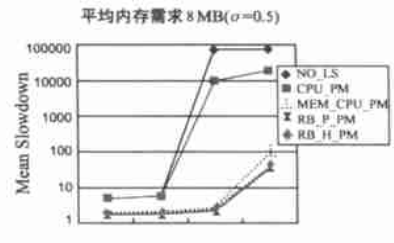


图 3

从图中可见, 当任务的平均内存需求一定、系统的资源异构性增加时( 从系统 1 到系统 4) , NO. LS 算法和仅仅考虑 CPU 资源使用情况的 CPU. PM 算法将导致非常高的平均减速, CPU. PM 算法能够获得比 NO. LS 更低的系统平均减速; 由于 MEM. CPU. PM 算法和基于资源的 RB. P. PM、RB. H. PM 算法同时考虑了 CPU 资源和内存资源, 因此能够很好地适应系统异构性的变化, 获得较低的平均减速. 当任务的内存需求越大时( 图 1 到图 3) , 两种 RB. PM 算法越能够体现出比其他算法更好的性能效果.

同时, 当系统的资源异构性和任务平均内存需求增加时, RB. PM 的两种算法能够获得比 MEM. CPU. PM 算法更好的性能效果. 图 1 中 RB. P. PM 算法比 MEM. CPU. PM 算法的性能提高为 1215% ~ 1815% , 图 2 中为 15% ~ 48% , 图 3 中为 1812% ~ 214% . 这是由于 RB. PM 的两种算法能够协调使用 CPU 和内存资源, 可以更为有效地利用所有资源, 因此能够比优先考虑内存资源的 MEM. CPU. PM 算法获得更低的平均减速.

模拟结果还表明, RB. P. PM 算法能够获得比 RB. H. PM 算法平均 10% 左右的性能提高, RB. P. PM 算法能够迁移比 RB. H. PM 算法更多数目的进程. 在后面的研究中, 我们主要考虑 RB. P. PM 算法.

### 4131 3　所有踪迹的性能比较

为了合理比较在所有 8 个踪迹中; 四种负载平衡算法的效果, 本文模拟了任务平均内存需求为 4MB 的中间情况. 为了提供直观可比性, 对不同系统中不同踪迹的页故障率 R 进行了相应调整, 使得所有的平均减速都在 20 以内. 模拟结果如图 4~ 6 所示.

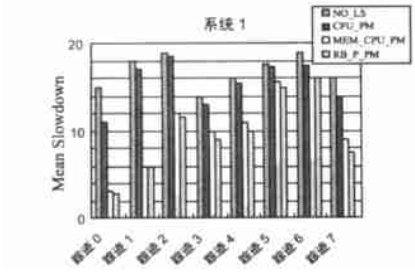


图 4

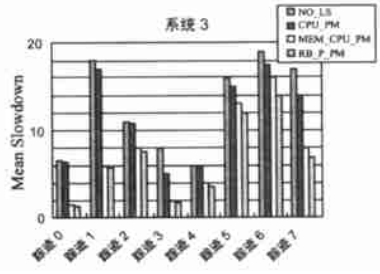


图 5

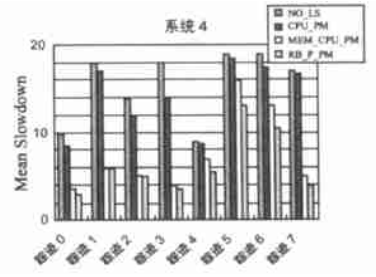


图 6

从图中可以看到, 在模拟的 3 个系统中, 对于所有的踪迹 NO. LS 算法都具有最高的平均减速; 位于第二的是仅考虑 CPU 资源的 CPU. PM 算法; 优先考虑内存资源, 兼顾 CPU 资源的 MEM. CPU. PM 算法能够获得比 CPU. PM 算法更低的系统平均减速; 协调考虑 CPU 和内存资源的 RB. P. PM 算法能够有效适应系统的资源异构性变化, 获得最低的平均减速.

## 5　结论及未来的工作

本文针对异构集群系统的资源共享和有效利用问题, 提出了两种基于资源使用率和向量负载指数的、采用进程迁移机制的负载平衡算法.

通过踪迹驱动的模拟和分析, 结果表明: 在异构集群系统中, 本文提出的基于多种资源之间相互协调关系的负载平衡算法能够有效地平衡系统负载, 提高资源使用率, 从而能够降低系统的平均减速, 提高系统的整体性能. 而且, 系统的资源异构性越高, 我们的算法比传统算法的性能优势就越明显.

由于时间和条件关系, 本文现在的工作还不十分完善. 未来还需要继续进行的工作有:

Ó 当前的模拟仅仅考虑了 CPU 和内存两种资源, 这实际上不足以完全体现基于资源的负载平衡算法的效果. 下一步将进行更多种类资源情况的性能模拟.

Ó 深入研究集群系统中的陈旧负载信息问题.

参考文献:

[ 1 ] Barak A, Braverman A. Memory ushering in a scalable computing clus2 ter [ J] . Journal of Microprocessors and Microsystems, 1998, 22( 324) : 175- 182.

[ 2 ] X Zhang, Y Qu, L Xiao. Improving distributed workload performance by

sharing both CPU and memory resources [A]. Proc. of 20th Inter. Conf. on Distributed Computing Systems [C]. Taipei: 2000. 233- 241.

[3] L Xiao, X Zhang, Y Qu. Effective load sharing on heterogeneous networks of workstation [A]. Proc. of the 2000 Inter. Parallel and Distributed Processing Symposium [C]. Mexico: 2000. 431- 438.

[4] M Harchol2Balter, A B Downey. Exploiting process lifetime distributions for dynamic load balancing [J]. ACM Transactions on Computer Systems, 1997, 15(3): 253- 285.

[5] M Mitzenmacher. How useful is old information? [A]. Proc. of the 16th ACM Symposium on Principles of Distributed Computing [C]. 2000. 6 - 20.

[6] Michael Dahlin. Interpreting stale load information [A]. The 19th IEEE Inter. Conf. on Distributed Computing Systems [C]. Austin: IEEE Computer Society, 1999. 285- 296.

[7] M Mitzenmacher. The power of two choices in randomized load balancing [D]. Berkeley: University of California, 1996.

[8] Mor Harchol2Balter. The effect of heavy2tailed job size distributions on computer system design [A]. Proc. of ASA2IMS Conf. on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics [C]. Washington: 1999.

[9] F Douglis, J Ousterhout. Transparent process migration: design alternatives and the sprite implementation [J]. Software: Practice and Experience, 1991, 21(8): 757- 785.

[10] M Nuttall, M Sloman. Workload characteristics for process migration and load balancing [A]. Proc. of the IEEE Inter. Conf. on Distributed Computing Systems [C]. 1997. 133- 140.

[11] Mark P Nuttall. Cluster load balancing using process migration [D]. London: Department of Computing, University of London, 1997.

作者简介:

蒋 江 男, 1973 年 11 月出生于云南宣威, 博士, 主要研究领域为高性能集群系统软件、高性能计算机体系结构等. Email: jjlt@sina.com

张民选 男, 1954 年生于湖南邵东, 教授, 博士生导师, 主要研究领域为高性能计算机体系结构、并行处理技术与算法、微处理器及 ASIC 技术.