ISSN 1000-1239/CN 11-1777/TP
50(3): 481-489, 2013

一种面向网格计算的自适应动态冗余预留策略

肖 鹏[1,2] 胡志刚[2]
[1](湖南工程学院计算机与通信系 湖南湘潭 411104)
[2](中南大学信息科学与工程学院 长沙 410083)
(efn4623@126.com)

An Adaptive Dynamic Redundant Reservation Strategy in Grid Computing

Xiao Peng[1,2] and Hu Zhigang[2]
[1](*Department of Computer and Communication*, *Hunan Institute of Engineering*, *Xiangtan*, *Hunan* 411104)
[2](*School of Information Science and Engineering*, *Central South University*, *Changsha* 410083)

Abstract Redundant mechanism has been widely applied as an effective means to enhance the reliability of grid systems. However, it will lead to higher ineffective utilization, which in turn brings about many negative effects on applications' performance. To address this problem, an adaptive redundant reservation strategy is proposed with the aim to mitigate the negative effects of conventional strategies. It provides a mechanism that enables the grid systems adjust redundant degree at runtime without decreasing the reliability. The quantitative relationship between redundant degree and reliability is presented theoretically, and extensive experiments based on real-world workload are conducted to examine the performance of the proposed strategy compared with other redundant strategies. The experimental results show that the proposed redundant strategy outperforms other existing strategies in terms of effective resource utilization. Also it brings about tradeoff between the reliability and the execution performance of applications, which significantly mitigates the negative effectives of conventional redundant strategies.

Key words grid computing; workflow; redundant mechanism; co-reservation; reliability

摘 要 针对网格环境中冗余机制在系统可靠性和任务执行效率之间难以平衡的问题,提出一种冗余度可动态调整的自适应冗余策略.该策略以可靠性指标为约束条件,依据负载变化自适应地优化系统冗余度,在不降低可靠性的前提下减少冗余度过高对网格任务执行效率的负面影响.理论分析给出了冗余度与可靠性之间的量化关系,实验分析对比了该动态冗余策略与其他冗余策略的性能差异,并对策略关键参数进行了对比分析.实验结果显示,当系统面临较高的任务负载或负载变化剧烈时,自适应的动态冗余预留策略能够显著提高资源有效利用率,并在任务执行效率和系统可靠性之间实现动态平衡,从而降低传统冗余策略对系统性能的负面影响.

关键词 网格计算;工作流;冗余机制;协同预留;可靠性

中图法分类号 TP393

网格计算[1]环境中的冗余机制主要用于提高系统可靠性和任务执行效率[2-7].通过对任务进行冗余资源分配和调度,由资源故障导致的任务执行失败率将显著降低[2-3,6,8-12];网格任务也可以从多个返回

收稿日期:2011-04-12;**修回日期**:2011-12-26
基金项目:国家自然科学基金项目(60673165,60970038,61272148);湖南省科技计划项目(2012GK3075)

结果中选择 QoS 满意度最高(如响应时间最短[2]、负载均衡度最优[9]、精确度最高[13])的作为最终执行结果.因此,冗余机制已经成为实用网格系统的一个基础服务.

冗余度 K 是关系到策略效率和开销的核心参数.一般而言,当冗余度 K 较高时,系统可靠性增强,反之则减弱.但较高的冗余度容易导致过重的无效负载,而过高的无效负载将导致系统拒绝服务率上升,反而降低任务的执行效率和可靠性.文献[2-3,9,11-15]中关于实际网格的性能测评报告都同时指出了冗余机制的这一缺陷.已有研究显示,为开放性和分布性很高的网格系统提供高效的冗余机制的困难主要在于:1)资源可获得性难以预测,导致冗余策略所能提供的可靠性难以保证[2,13,16];2)冗余策略可能加剧负载的波动性,从而对任务执行造成负面影响[7,17-18];3)粗粒度的冗余策略难以有效满足不同任务的个性化 QoS 需求[2,5,18-21].针对以上问题,本文从资源可获得性与冗余度之间的关系出发,提出一个冗余度可动态调整的自适应冗余策略,目的是在确保不损失可靠性的前提下,减少冗余度过高所产生的负面影响.

1 相关研究

文献[2]中 Casanova 全面总结了冗余机制对任务执行效率、负载均衡、调度公平性等方面的影响,其实验结果显示:1)冗余机制带来的无效负载对调度性能、负载均衡、可靠性等指标造成难以预知的负面影响;2)任务倾向过度使用冗余机制.针对无效负载的问题,Zhang 等人[21]提出一种分层冗余控制模型,目的是在保证数据一致性的前提下降低冗余开销;Elghirani 等人[22]提出采用非合作博弈理论来分析最优冗余度;付伟等人[18]则提出一种基于逻辑环结构的冗余副本存储算法,用于降低任务 QoS 受限情况下的冗余开销.

相对存储资源而言,计算资源受冗余机制的影响更为复杂.在已有研究中,冗余机制对调度性能影响的研究最为丰富.Rood 等人[3]从计算资源的动态可用性出发,提出一种基于可获得性预测的冗余策略,其核心思想是分析资源的短期可获得性,用于确定是否进行冗余请求;Tang 等人[19]则针对工作流执行期可靠性问题提出了一种可信度驱动的工作流调度算法,目的在于降低资源执行期安全性对工作流执行效率的影响;Bozdag 等人[8]则从压缩目标资源集合的角度,提出一种降低冗余度的两阶段工作流调度算法.

针对任务过度使用冗余机制的特点,Prodan 等人[12]在 ASKALON 系统中测试了工作流任务的各种延迟开销,其实验数据显示:过度使用冗余机制导致的任务执行延迟约占整体执行时间的 7%～23%,且该比例随冗余度增加而显著上升.为控制任务的这种倾向,Haddad[23] 和 Pathan[24] 同时提出利用计算经济理论通过资源费用来约束任务过度使用冗余机制的倾向,但涉及多资源协同分配时,各类经济模型的定价机制和交易机制容易造成过多的通讯开销,从而加剧任务执行延迟[13].文献[9]则提出一种基于负载均衡的冗余度可调策略,其核心思想是以负载均衡指标来约束冗余度,在系统负载出现不均衡的情况时,通过动态调整冗余度来实现负载均衡的目标.Desprez 等人则在文献[6]中针对数据密集型工作流任务提出了一种基于"数据临近"的动态冗余策略,通过分析计算节点与数据节点之间的距离来确定适当的冗余度,目标是降低任务执行期间的通信开销.

在以上研究中,文献[3,6,9]明确提出了降低冗余度的思想,与本文所提的自适应动态冗余策略思想最为接近.但文献[3]主要是采用实验方式来验证资源可获得性与冗余度之间的关系,缺少两者之间的定量分析.而文献[6,9]则是采用启发式函数的方式动态调整冗余度,并未分析冗余度对资源有效利用率的影响,也未分析可靠性与任务执行效率之间量化关系.本文则从资源预留成功率出发,定量分析了多资源协同预留成功率与冗余度之间的量化关系,并实现了具有负载感知能力的冗余预留策略.

2 问题描述

设子任务集合为 $J=\{t_1,t_2,\cdots,t_n\}$,每个子任务 t_i 由三元组 $\langle a_i,c_i,v_i\rangle$ 表示,其中 a_i 为子任务执行时间,c_i 为资源需求量,v_i 为预留资源量;资源站点表示为集合 $R=\{r_1,r_2,\cdots,r_m\}$,r_i 的可用资源状况表示为时间槽集合 $Slot(r_i)=\{s_{i,1},s_{i,2},\cdots,s_{i,k}\}$.

定义 1. 任务 t_i 的预留请求表示为三元组 $\langle st_i,et_i,v_i\rangle$,$st_i$,为预留起始时间,$et_i$,为预留截止时间,$v_i$,为资源预留量.

定义 2. 资源站点 r_i 的第 j 个时间槽 $s_{i,j}$ 表示为三元组 $\langle s\_st_{i,j},s\_et_{i,j},s\_c_{i,j}\rangle$,$s\_st_{i,j}$ 为时间槽的起始时间,$s\_et_{i,j}$ 为截止时间,$s\_c_{i,j}$ 为可用资源量.

定义 3. 协同预留矩阵为任务集合 J 到资源站点 R 的映射,记为 $\boldsymbol{S}:J\times R\rightarrow\{0,1\}$.

定义 4. 设随机事件 $\Psi_{i,j}$ 表示子任务 t_i 在资源节点 r_j 上成功预留到所需资源,$\Psi_{i,j}$ 发生的概率记为 $P(\Psi_{i,j})$.

定义 5. 设系统冗余度为 K,协同预留矩阵 $\boldsymbol{S}$ 能被系统接纳的概率记为 $P(\boldsymbol{\Psi},\boldsymbol{S})$,其中 $\boldsymbol{\Psi}$ 为 $\Psi_{i,j}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$组成的 $n\times m$ 型随机事件矩阵.

综合以上定义,针对给定的预留矩阵 $\boldsymbol{S}$,任务能否成功预留到资源成为保证任务顺利执行的关键.当 $K=1$ 即系统并不采用冗余机制时,网格系统需要保证预留资源在实际分配时刻的可获得性;当 $K>1$ 时,实际网格系统并不需要保证所有预留资源在实际分配时刻的可获得性,而往往只是采用"尽力请求"策略.因此,当冗余机制中的冗余度固定不变时,多资源协同预留问题可以转化为以下约束条件下的最优规划问题:

$$\begin{aligned}&\max\ P(\boldsymbol{\Psi},\boldsymbol{S})=\prod_{i=1}^{n}P\Big(\bigcup_{j=1}^{m}\Psi_{i,j}\Big);\\&\text{s.t. }\boldsymbol{S}=J\times R\rightarrow\{0,1\};\\&\forall j\in\{1,\cdots,m\},\ |\{S_{i,j}=1\}|=K.\end{aligned}\tag{1}$$

当冗余机制中的冗余度可以动态调整时,冗余度 K 可以视为 $\boldsymbol{\Psi}$ 和 $\boldsymbol{S}$ 的函数,因此多资源协同预留问题可以转化为以下最优规划问题:

$$\begin{aligned}&\min\ K(\boldsymbol{\Psi},\boldsymbol{S});\\&\text{s.t. }\boldsymbol{S}=J\times R\rightarrow\{0,1\};\\&P(\boldsymbol{\Psi},\boldsymbol{S})=\prod_{i=1}^{n}P\Big(\bigcup_{j=1}^{m}\Psi_{i,j}\Big).\end{aligned}\tag{2}$$

本文主要针对(2)所示问题进行求解,因此必须分析 $P(\Psi_{i,j})$ 的计算方法,即子任务 t_i 在资源站点上 r_j 成功预留所需资源的概率.

3　动态冗余预留策略

3.1　预留成功率分析

如前文所述,当网格系统采用冗余机制时($K>1$),冗余预留请求采用"尽力请求"策略或者"随机请求"策略.为此,本文引入一个基于概率的广义预留接纳定义.

定义 6. 资源站点 r_j 上的时间槽 $s_{j,k}$ 满足 $(st_i\geqslant s\_st_{j,k})\wedge(et_i\leqslant s\_et_{j,k})\wedge(s\_c_{j,k}\geqslant v_i)$ 的概率为 $P(s_{j,k},t_i)$,并称时间槽 $s_{j,k}$ 以概率 $P(s_{j,k},t_i)$ 满足 t_i 的预留请求.

显然,若系统必须满足 $P(s_{j,k},t_i)=1$,则退化到标准化文献 GFD-E.5[25] 所定义的传统预留定义.结合定义 5 和定义 6 可知:

$$P(\Psi_{i,j})=\max\{P(s_{j,k},t_i)\mid s_{j,k}\in r_j\}.\tag{3}$$

为分析 $P(s_{j,k},t_i)$ 的计算方法,图 1 给出了一个资源节点的时间槽分布图:

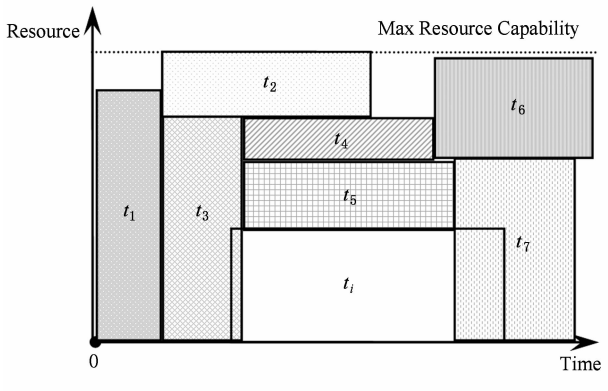


Fig. 1　Illustration of resource time slots.

图 1　资源站点的时间槽示例

显然,影响 $P(s_{j,k},t_i)$ 的是已经存在的预留请求($t_1\sim t_7$),依据这些预留请求对 t_i 的影响方式不同,可将其分为如下两个集合.

定义 7. 在资源站点 r_j 的时间槽集合上,与子任务 t_i 的预留起始时间叠交的预留请求集合记为 $\alpha_j(t_i)$:

$$\begin{aligned}\alpha_j(t_i)=\{t_k\mid\ &\forall k\in[1,\cdots,i-1],\\&st_k<st_i<et_k\}.\end{aligned}\tag{4}$$

定义 8. 在资源站点 r_j 的时间槽集合上,与子任务 t_i 的预留截止时间叠交的预留请求集合记 $\beta_j(t_i)$:

$$\beta_j(t_i)=\{t_k\mid\forall k\in[1,\cdots,i-1],st_i<st_k<et_i\}.\tag{5}$$

若以图 1 为例,则 $\alpha_k(t_i)=\{t_2,t_3\}$ 且 $\beta_k(t_i)=\{t_6,t_7\}$.显然,若 t_2 或者 t_3 提前释放其资源,则 t_i 的预留起始时间不会被干扰;同理若 t_i 提前结束,则 t_6 和 t_7 的预留起始时间不会被干扰.

定理 1. 资源站点 r_j 上的空闲时间槽 $s_{j,k}$ 能够完全满足子任务 t_i 的预留起始时间的概率为

$$P^{\alpha}(s_{j,k},t_i)=\prod_{l\in\Omega^*}Pr\{T_l\leqslant st_i\}.\tag{6}$$

其中,随机变量 T_l 表示子任务 $t_l\in\alpha_j(t_i)$ 的实际完成时间,$Pr\{T_l\leqslant st_i\}$ 表示子任务 t_l 的实际完成时间早于 st_i 的概率;集合 $\Omega^*\subseteq\alpha_j(t_i)$ 且满足以下条件:

$$\begin{cases}\Omega^*=\{\varphi\mid\varphi\subseteq\alpha_j(t_i),\max\{\prod_{l\in\Psi}Pr\{T_l\leqslant st_i\}\}\};\\v_i\leqslant s\_c_{j,k}+\sum_{l\in\Omega^*}v_l.\end{cases}\tag{7}$$

证明. 由定义 7 可知,集合 $\alpha_j(t_i)\neq\varnothing$是导致任务 t_i 的预留请求发生起始时间违约的充要条件. 若集合 $\alpha_j(t_i)$中若干任务在 t_i 的预留起始时间 st_i 之前完成,不妨设这些提前完成的任务集合为 $\varphi\subseteq\alpha_j(t_i)$,当集合 φ 满足如下条件:

$$v_i\leqslant s\_c_{j,k}+\sum_{l\in\phi}v_l, \tag{8}$$

则时间槽 $s_{j,k}$在 st_i 时刻的空闲资源量完全可以满足 t_i 的预留资源量,此时时间槽 $s_{j,k}$能够完全满足子任务 t_i 的预留起始时间的概率为

$$P^{\alpha}(s_{j,k},t_i)=\prod_{l\in\phi}Pr\{T_l\leqslant st_i\}. \tag{9}$$

结合图 1 可知,满足(8)所示条件的 φ 不是唯一的,例如 $\varphi=\{t_3\}$或$\{t_2,t_3\}$都满足条件. 为此,必须找到一个使得式(6)所示概率最大的集合,记为 Ω^*,而式(7)所示的条件就是该集合必须满足的要求. 求取集合 Ω^* 的问题可以归结为如下所示的规划问题.

$$\begin{aligned}&\max\prod_{l\in\Omega^*}Pr\{T_l\leqslant st_i\};\\&\text{s.t. }v_i\leqslant s\_c_{j,k}+\sum_{l\in\Omega^*}v_l;\\&\Omega^*\subseteq\alpha_j(t_i).\end{aligned} \tag{10}$$

该规划问题与经典的 0-1 背包问题相同,文献[26]中有详细的求解算法描述,本文不作重复. 证毕.

定理 2. 设资源站点 r_j 的资源总量为 $C^{\max}$,集合 $\beta_j(t_i)$中的预留请求按其预留起始时间升序排序为$\langle t_1,t_2,\cdots,t_m\rangle$,则空闲时间槽 $s_{j,k}$能够完全满足 t_i 的预留截止时间的概率为

$$P^{\beta}(s_{j,k},t_i)=Pr\{T_i\leqslant st_l\}. \tag{11}$$

其中,$Pr\{T_i\leqslant st_l\}$表示子任务 t_i 的实际完成时间早于 $t_l\in\beta_j(t_i)$的预留起始时间 st_l 的概率,且 t_l 满足条件

$$\sum_{j=1}^{l-1}v_j\leqslant C^{\max}-v_i<\sum_{j=1}^{l}v_j. \tag{12}$$

证明. 由定义 8 可知,当 $\beta_j(t_i)\neq\varnothing$时 t_i 的截止时间将干扰集合$\beta_j(t_i)$中的已有预留请求的起始时间. 由于集合 $\beta_j(t_i)$中的预留请求按预留起始时间升序排序为$\langle t_1,t_2,\cdots,t_m\rangle$,若 t_i 的实际完成时间早于 st_1,则 t_i 实际并不会干扰集合 $\beta_j(t_i)$中的任何预留请求. 此时,空闲时间槽 $s_{j,k}$能够完全满足 t_i 的预留截止时间的概率为

$$P^{\beta}(s_{j,k},t_i)=Pr\{T_i\leqslant st_1\}. \tag{13}$$

结合图 1 可知,式(13)所示的概率并非最优的计算方式. 例如图 1 所示,$\beta_j(t_i)=\{t_6,t_7\}$,只要 t_i 的实际完成时间早于 st_7,则实际上不会出现预留违约. 因为 $st_7>st_6$,由概率分布函数的递增特性可知

$$Pr\{T_i\leqslant st_7\}>Pr\{T_i\leqslant st_6\}. \tag{14}$$

因此,当 t_l 满足式(14)所示条件时,$\langle t_1,t_2,\cdots,t_l\rangle$都不会和 t_i 的预留请求发生任何交叠,所以不存在预留违约. 同时,由于 t_i 提前释放资源,此时系统能够保证$\langle r_{l+1},r_{l+2},\cdots,r_m\rangle$不会由于空闲资源不够而导致预留违约. 证毕.

综合以上定理 1 和定理 2,依据冗余度是否可变的两种情况可以得到以下两个推论.

推论 1. 设网格系统采用固定冗余度为 K 的策略时,且任意两个冗余预留请求都不同时分配到相同资源节点,则任务 t_i 能够成功预留资源的概率为

$$P(\bigcup_{j=1}^{m}\Psi_{i,j})=1-\prod_{j=1}^{k}[1-P^{\alpha}(s_j,t_i)\times P^{\beta}(s_j,t_i)]. \tag{15}$$

推论 2. 若网格系统采用可变冗余度策略,并设定预留请求成功率不低于 W^* 时,冗余度 K 的取值上限和下限分别为

$$K_{\min}=\left\lceil\frac{\ln(1-W^*)}{\ln(1-\min\{P^{\alpha}(s_j,t_i)\times P^{\beta}(s_j,t_i)\})}\right\rceil; \tag{16}$$

$$K_{\max}=\left\lceil\frac{\ln(1-W^*)}{\ln(1-\max\{P^{\alpha}(s_j,t_i)\times P^{\beta}(s_j,t_i)\})}\right\rceil. \tag{17}$$

3.2 动态冗余预留算法

推论 1 给出了固定冗余策略下单一子任务的预留成功率和冗余度 K 之间的关系;推论 2 则给出了单一子任务在预留成功率 W^* 约束下,其冗余度 K 的上限和下限. 为求解式(2)所示的约束优化问题,本文设计了一个冗余度依据实时资源利用率动态调整的协同预留接纳算法.

算法 1. 基于自适应冗余策略的协同预留算法.

输入:任务集合 $J=\{t_1,t_2,\cdots,t_n\}$,资源站点集合 $R=\{r_1,r_2,\cdots,r_m\}$,预留保证度下限 W^*,资源利用率下限和上限 $U^{\min}$,$U^{\max}$;

输出:协同预留矩阵 $\boldsymbol{S}$,预留保证度 $P(\boldsymbol{\Psi},\boldsymbol{S})$.

步骤 1. 初始化 $\boldsymbol{S}$ 中的所有元素 $S_{i,j}$为 0,并令 $P(\boldsymbol{\Psi},\boldsymbol{S}):=1$.

步骤 2. 计算每个子任务 t_i 在资源站点 r_j 上的预留保证度 $\Psi_{i,j}$.

① 若 r_j 满足 $\alpha_j(t_i)=\varnothing$且 $\beta_j(t_i)=\varnothing$,则令 $\Psi_{i,j}:=1$,否则执行以下步骤;

② 若 $\alpha_j(t_i)\neq\varnothing$,则采用 0-1 背包算法求得集合 Ω^*,并依据式(6)计算出每个时间槽 $s_{j,k}$所对应

的 $P^{\alpha}(s_{j,k},t_i)$;

③ 若 $\beta_j(t_i)\neq\varnothing$,则依据式(11)计算出每个时间槽 $s_{j,k}$ 所对应的 $P^{\beta}(s_{j,k},t_i)$;

④ 令 $\Psi_{i,j}:=\max\{P^{\alpha}(s_{j,k},t_i)\times P^{\beta}(s_{j,k},t_i)\}$.

步骤 3. 计算每个子任务 t_i 在资源站点 r_j 上的最佳冗余度 K.

① 以降序方式排列 $\{\Psi_{i,1},\Psi_{i,2},\cdots,\Psi_{i,m}\}$;

② 依据式(16)计算出 $K_{\max}$,依据式(17)计算出 $K_{\min}$;

③ 若 r_j 的资源利用率低于 $U^{\min}$,则令 $K:=K_{\max}$;若 r_j 的资源利用率高于 $U^{\max}$,则令 $K:=K_{\min}$;其他情况下,则令 $K:=(K_{\max}+K_{\min})/2$.

步骤 4. 生成协同预留矩阵 $\boldsymbol{S}$,并计算其能提供的预留保证度 $P(\boldsymbol{\Psi},\boldsymbol{S})$.

① 若 t_i 在站点 r_j 上的预留保证率满足 $P(\boldsymbol{\Psi},\boldsymbol{S})\times\Psi_{i,j}\geqslant W^{*}$,则令 $S_{i,j}:=1$,$P(\boldsymbol{\Psi},\boldsymbol{S}):=P(\boldsymbol{\Psi},\boldsymbol{S})\times\Psi_{i,j}$;

② 针对所有任务重复上面步骤,直至生成任务集合 J 的协同预留矩阵 $\boldsymbol{S}$.

以上算法 1 的复杂度为 $O(n\times(m+K))$,n 为子任务数目,m 为网格系统的资源站点数目,K 为动态冗余度($K_{\min}\leqslant K\leqslant K_{\max}$).

冗余预留策略的通讯开销主要集中:1)冗余任务的传输开销;2)冗余任务输入/输出数据的传输开销.在传统的固定冗余策略中,设冗余度为 K_{fix},则完成任务的总体通讯开销为

$$W_{\text{com}}=\sum_{i=1}^{n}\sum_{j=1}^{K_{\text{fix}}}\left(\frac{S_{i,j}}{B_{i,j}}+\frac{D_{i,j}^{\text{in}}}{B_{i,j}^{\text{in}}}+\frac{D_{i,j}^{\text{out}}}{B_{i,j}^{\text{out}}}\right).\quad(18)$$

其中,$S_{i,j}$ 为第 i 个任务的第 j 个副本的大小,$D_{i,j}^{\text{in}}$ 和 $D_{i,j}^{\text{out}}$ 分别为输入、输出数据量,$B_{i,j}$ 为调度器到副本执行站点之间的带宽,$B_{i,j}^{\text{in}}$ 和 $B_{i,j}^{\text{out}}$ 则分别为执行站点到数据目标站点之间的带宽.若系统需要提供的可靠性保证不变,则必然满足 $K_{\min}<K_{\max}<K_{\text{fix}}$.因此,动态冗余策略所能节省的通讯开销为

$$W_{\min}=\sum_{i=1}^{n}\sum_{j=1}^{K_{\text{fix}}-K_{\max}}\left(\frac{S_{i,j}}{B_{i,j}}+\frac{D_{i,j}^{\text{in}}}{B_{i,j}^{\text{in}}}+\frac{D_{i,j}^{\text{out}}}{B_{i,j}^{\text{out}}}\right);$$

$$W_{\max}=\sum_{i=1}^{n}\sum_{j=1}^{K_{\text{fix}}-K_{\min}}\left(\frac{S_{i,j}}{B_{i,j}}+\frac{D_{i,j}^{\text{in}}}{B_{i,j}^{\text{in}}}+\frac{D_{i,j}^{\text{out}}}{B_{i,j}^{\text{out}}}\right).\quad(19)$$

其中,$W_{\min}$ 为节省通讯开销的下限,$W_{\max}$ 为节省通讯开销的上限.式(19)中的带宽参数($B_{i,j}$,$B_{i,j}^{\text{in}}$ 和 $B_{i,j}^{\text{out}}$)在调度方案确定后才能获得,因此本文在实验部分将分析动态冗余策略对不同通信计算比(communication computation ratio,CCR)任务的执行效率的影响.

4　实验与性能分析

实验构建的异构多集群计算网格包括了 12 个高性能计算集群作为资源站点,各集群节点的静态性能指标参考网格测试床 DAS-2 的参数配置,集群节点之间通过 1 Gbps 的网络连接,集群内部采用 100 Mbps 的局域网连接,系统网络连接如图 2 所示:

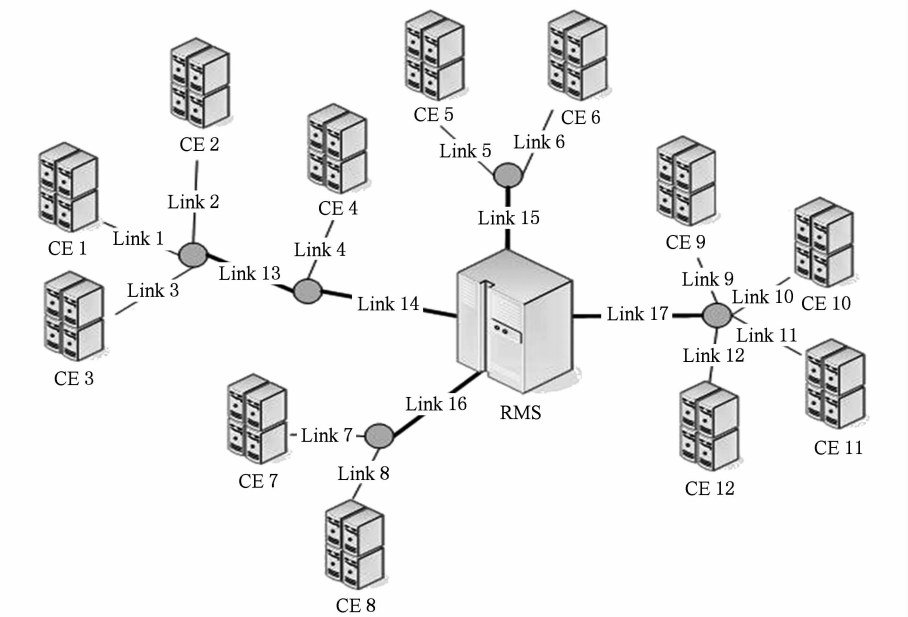


Fig. 2　Topology of multi-cluster grid in experiments.

图 2　实验网格模型的网络拓扑

4.1 对比实验分析

本次实验采用随机 DAG 作为工作负载,对比分析本文所提出的基于预留可靠性的动态冗余策略(reservation-reliability based dynamic redundant strategy, RDRS)与 3 种典型冗余策略在任务执行效率和有效资源利用方面的性能表现. 这 3 种策略分别为:随机冗余策略(random redundant strategy, RRS)[2]、最近数据冗余策略(close-to-file redundant strategy, CFRS)[6]、负载均衡冗余策略(load-balancing redundant strategy, LBRS)[9]. 其中,RRS 主要作为本次实验的性能基准;CFRS 是一种广泛应用于通信密集型工作流任务的冗余策略.

为分析通信计算比 *CCR* 对各种冗余策略的影响,本次实验分 4 组进行,DAG 任务的 *CCR* 值分别为 0.2,0.5,1.0,2.0,实验所用 DAG 任务的节点数目从 50 逐步增加到 500. 为客观对比各种冗余策略的性能表现,所有随机 DAG 任务均采用 HEFT 算法[27]来生成调度方案,实验结果如图 3 所示:

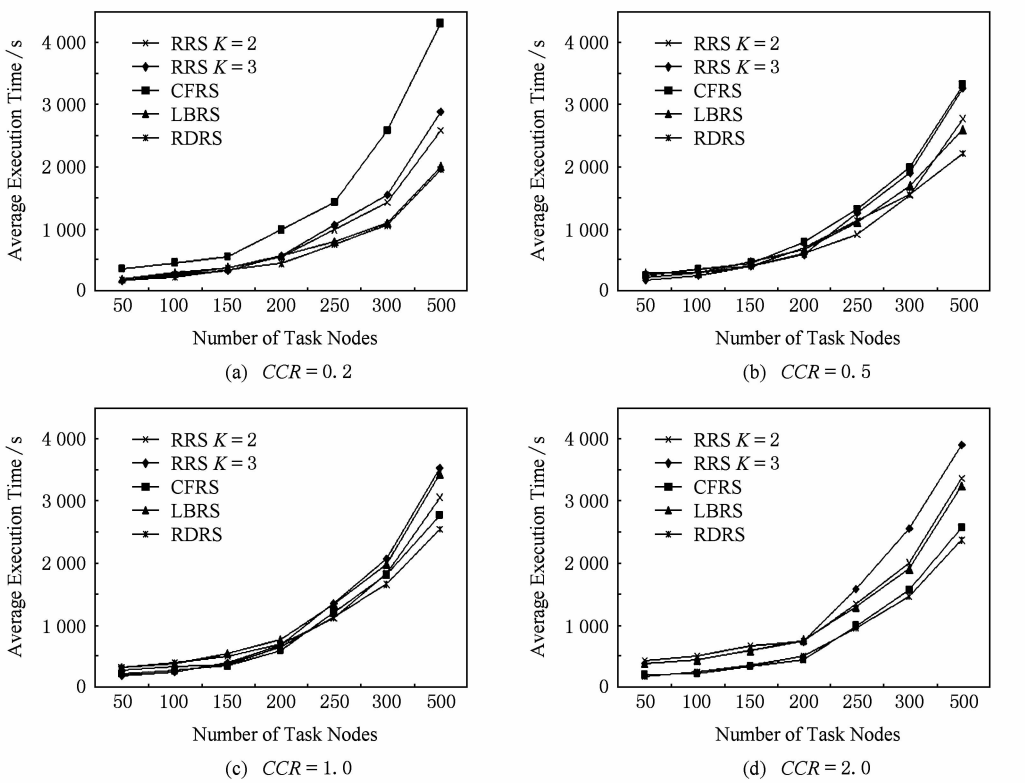


Fig. 3 Average execution time with different *CCR*.

图 3 不同 *CCR* 值的任务平均完成时间

实验结果显示,当通信计算比 *CCR* 较低时,CFRS 策略的性能表现最差;当 *CCR* 值逐步增大时,CFRS 策略的优点逐渐体现,尤其是当 $CCR=2.0$ 时,CFRS 与 RDRS 的性能差异只是在任务规模达到 300 以上时才达到 5%~8%. 对 RRS 而言,冗余度 K 对任务执行效率的影响主要体现在 DAG 任务规模超过 250 时,但这种差异却会随着任务 *CCR* 值的增加而增加. 通过分析实验过程数据,我们发现 RRS 策略与 LBRS 策略虽然实现机制完全不同,但从最终结果而言却都趋向"负载均衡"这一方向,差别仅仅在于:LBRS 在每次调度过程中都直接以负载均衡为目标,而 RRS 策略则是间接通过概率均衡的方式来实现整体的负载均衡.

在 4 种冗余策略中,RDRS 的性能表现最优,为了分析 RDRS 策略性能优越的原因,我们引入了有效资源利用率(effective resource utilization rate, *ERUR*)这一性能指标,其公式如下所示:

$$ERUR=\frac{\sum_{i=1}^{n}U_i(t)-\sum_{i=1}^{n}\sum_{j=1}^{K-1}V_i^j}{\sum_{i=1}^{n}U_i(t)}. \tag{20}$$

其中,$U_i(t)$为 t 时刻 r_i 的资源利用率,V_i^j 为无效冗余请求对 r_i 的资源占用率.

实验统计了各种冗余策略的平均有效资源利用率,如图 4 所示. 为分析 *ERUR* 的变化情况,实验记录了 *ERUR* 的实时数据,图 5 为任务规模达到 500 时的 *ERUR* 实时数据.

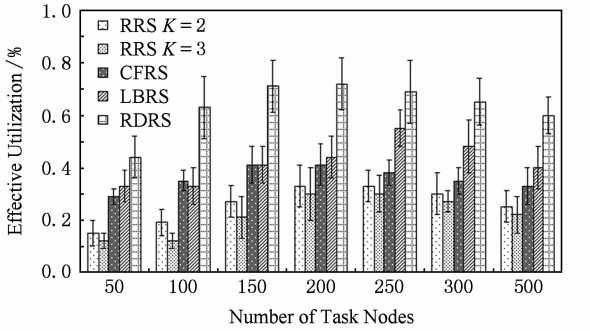


Fig. 4　Average *ERUR* with different job sizes.

图 4　不同任务规模下的平均有效资源利用率

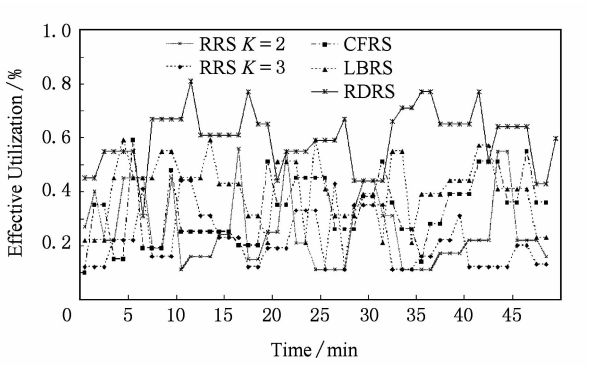


Fig. 5　Real-time *ERUR* logs.

图 5　有效资源利用率的实时统计

图 4 的实验数据显示,RDRS 策略的有效资源利用是 5 种策略中最高的,尤其是当任务规模为 100～200 时,RDRS 策略的 *ERUR* 指标相对 CFRS 策略和 LBRS 策略都高出约 73%以上. 而 RRS 策略的 *ERUR* 指标是最低的. 在 4 种策略中,只有 RRS 采用了固定冗余度的方式,这种方式虽然实现简单,但其显著缺陷是缺乏动态的自适应能力. 通过分析图 5 的实时 *ERUR* 数据可知,系统采用 RRS 策略时,资源有效利用率倾向于恒定地处于低位,这显然是因为冗余度 K 固定不变所导致的,而且 K 值越大,有效资源利用率越低. 而其余 3 种策略均采用冗余度可变的方式,其中的 CFRS 策略的冗余度主要由 DAG 任务的 *CCR* 特性决定,而 LBRS 策略的冗余度主要由系统负载分布状态决定,显然 LBRS 策略相对 CFRS 而言更具负载感知能力,因此 LBRS 的有效资源率相对 CFRS 约高出 7%～23%,这种差异在任务规模较大时表现得更明显. RDRS 策略则总是依据预留成功率来判断冗余度 K 的取值,当系统负载较重时,任务冗余预留请求的成功率将显著减低,因此冗余度 K 被自适应地调低,从而动态地降低了冗余机制对系统性能的负面影响.

本次实验设定 RDRS 中的 $W^*=0.6$,即当冗余预留请求的成功率低于 60%时,系统总是选择最小的 K 值. 本次实验的详细数据显示,RDRS 策略对 61%的任务都选择 $K=1$,对 33%的任务选择 $K=2$,仅有 6%的任务选择了 $K>2$ 的值,而且较大的 K 值都出现在系统资源较为空闲的时刻,即任务负载较轻的时刻. 正是这种自适应的动态冗余度策略,使得 RDRS 策略能够在资源空闲度较高时通过较高的冗余度来获得可靠性提高,而在系统负载较重时,则通过主动降低 K 值来减少冗余策略的负面影响.

图 5 同时显示,在某些时候 *ERUR* 值会出现跳跃式的增长,通过详细分析实验过程,我们发现这种跳跃式的增长是由于系统负载突然增加所导致的. 例如,任务负载中出现一个较大的并行分支时,系统资源在短时间内将面对一个较为集中的负载流,这种情况在大规模工作流中较为常见. 由于 *ERUR* 是 $U_i(t)$的递增函数,因此通过实时 *ERUR* 数据可以间接地了解网格系统的负载波动情况.

本次实结论如下:1)固定冗余度策略无法有效处理负载动态变化的情况,且会导致很低的有效资源利用率;2)可变冗余度策略具有动态负载感知能力;3)RDRS 策略具有动态负载感知能力,并且能够在可靠性和有效资源利用率之间实现动态平衡.

4.2　策略参数分析

本次实验采用真实工作流任务 INVMOD[28] 作为工作负载,目的是分析不同的W^*参数取值对 INVMOD 的执行效率影响. 为了真实模拟网格系统,实验引入了背景工作负载,用于分析资源竞争情况下 RDRS 策略的性能表现. 背景工作负载包括 10 000 个独立子任务,其任务平均时间间隔服从参数为 λ 的泊松分布. 实验分 4 组进行,λ 的取值分别为 0.2,0.5,1.0,2.0 时,不同独立子任务 *Nodes* 为 100～500 的实验结果如图 6 所示.

实验结果显示,当 INVMOD 规模较小时,W^*参数对任务执行效率的影响很小,而当任务规模逐步增大时,W^* 参数的影响略有增加,具体体现为:当 W^* 参数从 0.1 逐步增大时,任务完成时间相对有所降低,当到达一个低点时,任务执行效率又随 W^* 参数的增加而增加. 产生这种现象的原因是:W^* 参数的升高将降低冗余度 K 值,而 RDRS 策略具有动态负载感知能力,当系统负载较低(包括 INVMOD 负载和背景工作负载)时,较低的冗余度对任务执行效率的负面影响很小,因此任务执行效率略微降低. 如前文所述,冗余策略通过选择执行速度最快的任务作为返回结果,因此具有提升任务执行效率的正面影响,因此当冗余度进一步降低时,冗

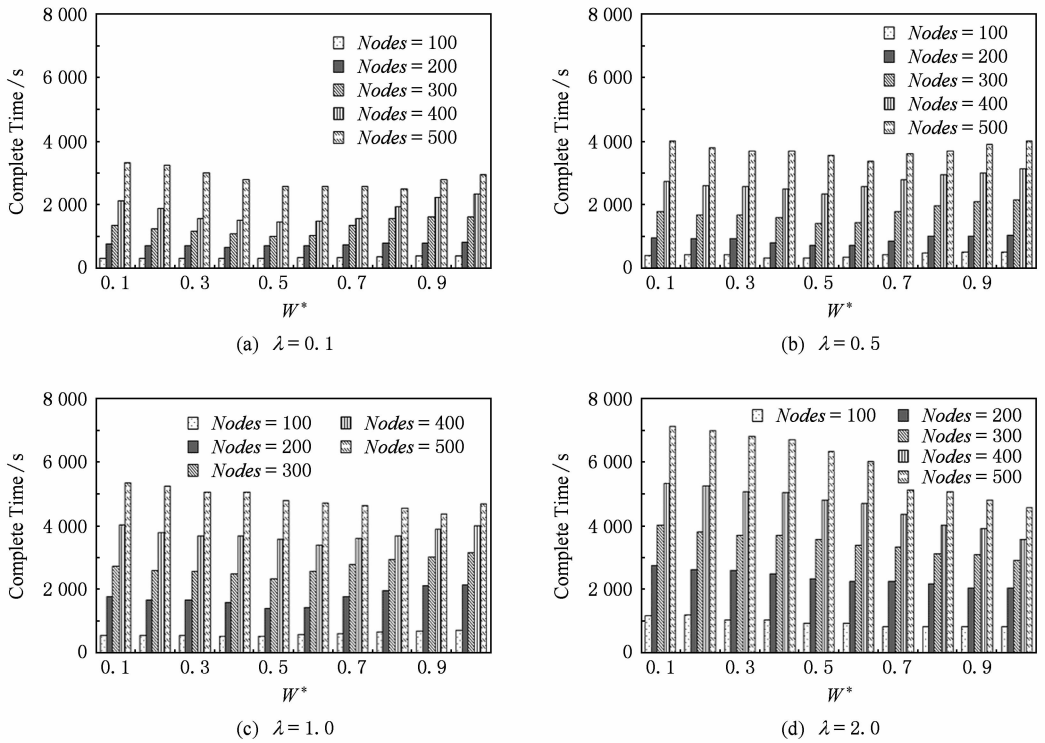


Fig. 6 Execution time with different background workloads.

图 6 不同背景负载下的任务完成时间

余策略的这种正面效应被降低,从而导致任务执行效率反而随冗余度降低而下降.

实验数据显示背景工作负载对 INVMOD 工作流的执行效率影响十分显著,尤其是当 $\lambda>1.0$ 后,INVMOD 的执行效率显著降低,而且随 W^* 参数的下降而逐步上升.产生这种现象的原因是:当背景负载很重时,其系统性能开销成为影响 RDRS 性能表现的主要因素,而冗余策略对任务执行效率的正面影响已经不是主要因素了.因此,当 W^* 参数不断上升时,冗余度一直下降,由此减少的无效资源负载提升了 INVMOD 工作流的执行效率.

本次实验结论如下:1)在系统负载较低时,提高 W^* 参数值可以提高任务执行效率,但同时会削弱冗余策略对任务执行效率的正面效应,最终结果处决于两者综合作用;2)当系统负载较重时,提高 W^* 参数值总是可以提高任务的执行效率.

5 结 论

针对网格计算环境中的冗余机制在系统可靠性和任务执行效率之间难以平衡的问题,提出一种动态冗余策略 RDRS,用于减少由于冗余机制对任务执行效率的负面影响. RDRS 依据冗余度与预留成功率之间的量化转换关系,依据负载变化情况来动态调整冗余请求.实验结果显示:RDRS 策略不仅具有动态负载感知能力,而且能够在系统可靠性和有效资源利用率之间实现动态平衡;当系统整体负载较重时,通过提高 RDRS 中的 W^* 参数值可以有效提高任务的执行效率.今后的工作将主要关注 RDRS 策略中 W^* 参数的动态优化方法,同时进一步研究冗余度与各类任务执行延迟之间关系.

参 考 文 献

[1] Foster I, Kesselman C. The Grid2: Blueprint for a New Computing Infrastructure [M]. San Francisco: Morgan Kaufmann, 2004

[2] Casanova H. Benefits and drawbacks of redundant batch requests [J]. Journal of Grid Computing, 2007, 5(2): 235-250

[3] Rood B, Lewis M J. Availability prediction based replication strategies for grid environments [C] //Proc of IEEE CCGRID'10. Piscataway, NJ: IEEE, 2010: 25-33

[4] Yang Juan, Bai Yun, Qiu Yuhui. EDCP—A duplication checking process used in duplication based resource allocation policies [J]. Journal of Computer Research and Development, 2006, 43(7): 1233-1239 (in Chinese)
(杨娟,白云,邱玉辉. EDCP—通用于任务复制资源分配算法的复制有效性检查过程[J]. 计算机研究与发展, 2006, 43(7): 1233-1239)

[5] Li K L, Tan T F, Wang F. Parallelization methods for implementation of discharge simulation along resin insulator surfaces [J]. Computers and Electrical Engineering, 2011, 37(1): 30-40

[6] Desprez F, Vernois A. Simultaneous scheduling of replication and computation for data-intensive applications on the grid [J]. Journal of Grid Computing, 2006, 4(1): 19-31

[7] Amir Y, Danilov C, Dolev D, et al. Steward: Scaling byzantine fault-tolerant replication to wide area networks [J]. IEEE Trans on Dependable and Secure Computing, 2010, 7(1): 80-93

[8] Bozdag D, Ozguner F, Catalyurek U V. Compaction of schedules and a two-stage approach for duplication-based DAG scheduling [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20(6): 857-871

[9] Dobber M, Mei R D, Koole G. Dynamic load balancing and job replication in a global-scale grid environment: A comparison [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20(2): 207-218

[10] Tang X Y, Li K L, Liao G P, et al. List scheduling with duplication for heterogeneous computing systems [J]. Journal of Parallel and Distributed Computing, 2010, 70(4): 323-329

[11] Zheng Q, Veeravalli B, Tham C K. On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs [J]. IEEE Trans on Computers, 2009, 58(3): 380-393

[12] Prodan R, Fahringer T. Overhead analysis of scientific workflows in grid environments [J]. IEEE Trans on Parallel and Distributed Systems, 2008, 19(3): 378-393

[13] Dun N, Taura K, Yonezawa A. Fine-grained profiling for data-intensive workflows [C] //Proc of IEEE CCGRID'10. Piscataway, NJ: IEEE, 2010: 571-572

[14] Shibata T, Choi S J, Taura K. File-access characteristics of data-intensive workflow applications [C] . //Proc of IEEE CCGRID'10. Piscataway, NJ: IEEE, 2010: 522-525

[15] Dai Y S, Pan Y, Zou X. A hierarchical modeling and analysis for grid service reliability [J]. IEEE Trans on Computer, 2007, 56(5): 681-691

[16] Walters J P, Chaudhary V. Replication-based fault tolerance for MPI applications [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20(7): 997-1010

[17] Chtepen M, Claeys F H A, Dhoedt B, et al. Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids [J]. IEEE Trans on Parallel and Distributed Systems, 2009, 20(2): 180-190

[18] Fu Wei, Xiao Nong, Lu Xicheng. Replica placement and update mechanism for individual QoS-restricted requirement in data grids [J]. Journal of Computer Research and Development, 2009, 46(8): 1408-1415 (in Chinese)
(付伟, 肖侬, 卢锡城. 个体 QoS 受限的数据网格副本管理与更新方法[J]. 计算机研究与发展, 2009, 46(8): 1408-1415)

[19] Tang X Y, Li K L. A novel security-driven scheduling algorithm for precedence constrained tasks in heterogeneous distributed systems [J]. IEEE Trans on Computers, 2011, 60(7): 1027-1029

[20] Shin K S, Cha M J, Jang M S, et al. Task scheduling algorithm using minimized duplications in homogeneous systems [J]. Journal of Parallel and Distributed Computing, 2008, 68(8): 1146-1156

[21] Zhang J Y, Honeyman P. Hierarchical replication control in a global file system [C] //Proc of IEEE CCGRID'07. Piscataway, NJ: IEEE, 2007: 155-162

[22] Elghirani A H, Subrata R, Zomaya A Y. A proactive non-cooperative game-theoretic framework for data replication in data grids [C] //Proc of IEEE CCGRID'08. Piscataway, NJ: IEEE, 2008: 433-440

[23] Haddad C, Slimani Y. Economic model for replicated database placement in grid [C] //Proc of IEEE CCGRID'07. Piscataway, NJ: IEEE, 2007: 283-289

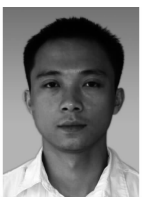
[24] Pathan A K, Buyya R. Economy-based content replication for peering content delivery networks [C] //Proc of IEEE CCGRID'07. Piscataway, NJ: IEEE, 2007: 887-895

[25] Roy A, Sander V. Advance reservation API [S]. Muncie: Global Grid Forum (GGF), 2002

[26] Meng X H, Zhu Y A, Wu X M. Improved dynamic programming algorithms for the 0-1 knapsack problem [C] //Proc of ICCIT'11. Piscataway, NJ: IEEE, 2011: 19-22

[27] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel and Distributed Systems, 2002, 13(2): 260-274

[28] Theiner D, Wieczorek M. Reduction of calibration time of distributed hydrological models by use of grid computing and nonlinear optimisation algorithms [C] //Proc of Int Conf on Hydroinformatics. Piscataway, NJ: IEEE, 2006: 1-8



Xiao Peng, born in 1979. Received his master degree in 2004, and now PhD candidate in the Central South University. His main research interests include grid computing and distributed systems.

Hu Zhigang, born in 1963. Professor and PhD supervisor of the Central South University. Senior member of China Computer Federation. His main research interests include, distributed systems, and embedded systems(zghu@mail.csu.edu.cn).