软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(10):2497−2508 [doi: 10.3724/SP.J.1001.2011.03912]

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

基于资源-预留图的动态网格资源预留机制*

高 瞻+, 罗四维

(北京交通大学 计算机与信息技术学院,北京 100044)

Dynamic Grid Resource Reservation Mechanism Based on Resource-Reservation Graph

GAO Zhan+, LUO Si-Wei

(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

+ Corresponding author: E-mail: bjtugaozhan@gmail.com

Gao Z, Luo SW. Dynamic grid resource reservation mechanism based on resource-reservation graph. *Journal of Software*, 2011,22(10):2497−2508. http://www.jos.org.cn/1000-9825/3912.htm

Abstract: Under the traditional static resource reservation mechanism (SRRM), once a user's reservation request has passed the admission test, it is scheduled for a certain resource immediately. SRRM considers neither the impact of the resource change on the schedule target nor the impact of resource error on the reservation in the book-ahead time. A dynamic resource reservation mechanism (DRRM) is presented, in which the accepted reservation requests are scheduled during the consumption the resource. The resource-reservation graph (RRG) is introduced to describe DRRM, and the modification rules of RRG have also been presented. The simulation experimental results show that though DRRM loses some admission percentage, it considerably decreases task preemption, dramatically improves the resource utilization, and has a better capacity of fault tolerance to the resource error ratio.

Key words: resource reservation; book-ahead time; resource utilization; task scheduling; task preemption

摘 要: 在传统的静态资源预留机制(static resource reservation mechanism,简称 SRRM)下,用户的预留请求一旦通过接纳测试就立即被调度到某个资源上.因此,SRRM 无法考虑在预留提前时间(book-ahead time)内作业队列的变化对调度目标产生的影响以及资源故障对预留任务的影响.提出了一种动态的网格资源预留机制(dynamic resource reservation mechanism,简称 DRRM),在该机制下,通过接纳测试的预留请求在其实际使用资源时才被调度.引入了资源-预留图(resource-reservation graph,简称 RRG)对 DRRM 进行说明,并给出了 RRG 的修改规则.DRRM 能够感知预留提前时间内资源状态的变化,根据其运行时信息动态地调度已接纳的预留请求.模拟实验结果表明,与 SRRM 相比,DRRM 损失了一定的预留请求接纳率,但大大减少了预留任务对非预留任务的抢占,显著提高了网格资源的有效利用率,并且对网格资源故障具有更好的容错效果.

关键词: 资源预留;提前预留时间;资源利用率;任务调度;任务抢占

中图法分类号: TP301 **文献标识码**: A

随着网格技术和应用的发展,许多应用领域(如实时计算、多媒体应用等)对网格服务提出了严格的 QoS(服务质量,quality of service)要求.这种 QoS 保证通常可以通过对相应资源的配置及预留来实现[1,2].另一方面,一些

* 基金项目: 国家高技术研究发展计划(863)(2006AA01A121)
收稿时间: 2009-10-28; 定稿时间: 2010-06-29

网格应用(如交互式数据分析、远程沉浸等)需要同时访问异构的网格资源,这些资源往往隶属于不同的管理域,受制于不同的管理策略,因此很难对它们进行协同分配.为了解决这个问题,资源预留的方法被采用,它可以保证在将来的某一特定时间段内所有需要的资源同时可用.因具有上述优点,资源预留被视为下一代网格资源管理系统的一个重要的功能[3],并已被一些调度系统所支持,如 COSY[3],ICENI[4].

所谓资源预留就是指在实际使用资源之前对其进行将来一段时间的预订,确保资源使用者在使用过程中获得所需的资源能力.通常,资源预留请求的内容包括资源使用的开始时间、持续时间以及与资源属性相关的参数(如 CPU 主频、CPU 个数、内存大小等).资源预留请求的提出时间通常早于它的开始时间,两者之间的时间间隔叫做预留提前时间(book-ahead time).在预留任务的开始时间到来时,如果当前剩余的资源能力无法满足它的要求,调度器通常会结束正在运行的非预留任务,从而释放出足够的资源能力以保证预留任务的顺利运行.由于预留任务限定了资源的使用时间段,可能会打断非预留任务的运行,造成资源利用时间的不连续,因此会产生大量的“资源能力碎片”.关于这种负面的影响,一些研究对此进行了评估.Sulistio[5]和 Smith[6]的研究表明,针对于他们所选择的网格作业,当其中只有 20%为预留任务时,资源利用率下降到不存在预留任务时的 66%,同时非预留作业的平均等待时间则增加了 71%.如何在支持资源预留的同时提高资源利用率,引起了人们的关注.文献[7]使用 backfilling 的方法在作业队列中寻找合适的作业来“填补”资源能力碎片;文献[8−10]采用了灵活的资源预留请求,允许推迟预留任务的开始运行时间来减少资源能力碎片的产生.本文提出了一种动态的资源预留机制(dynamic resource reservation mechanism,简称 DRRM),在该机制下,调度器根据每个资源的运行时状态灵活地进行预留资源的动态分配,能够有效减小资源浪费,并对网格环境的动态变化具有更好的“鲁棒性”.

1 相关工作

文献[11,12]描述了网格资源预留的一般过程.通常,用户提出的预留请求首先要接受资源管理系统的接纳测试,如果通过测试,则在合适的资源上创建相应的预留任务;否则,该预留请求被拒绝.GARA[1]是一个基于 Globus 的支持 CPU 计算能力和网络带宽的资源预留框架.在 GARA 中,用户向预留 Agent 提交自己的预留请求,预留 Agent 利用网格信息服务查找能够满足用户需求的候选资源,并向该资源所属管理域的 GRAM 发出 Creat Reservation 的命令.由于候选预留资源可能不只一个,因此需要从中选择最合适的资源,并在该资源上创建预留任务.对于预留资源的选择标准,GARA 并没有明确定义,但是建议使用“最早发现”或“最优发现”的策略.文献[13]则使用了基于存活时间的候选资源选择策略,即在预设存活时间内所查找到的候选资源中选择资源属性与用户需求最接近或资源性能最好的资源.PBS[14]调度系统则判断用户的预留请求是否与某个候选资源的预留队列中的预留任务以及在该资源上当前正在运行的非预留任务存在时间上的冲突,如果不存在冲突,则把预留请求调度到该资源上;否则,继续对下一个候选资源进行上述判断.在文献[15]中,网格任务调度器在得到所有符合用户资源需求的候选资源后,根据每个资源上当前运行作业的剩余运行时间以及该资源的作业队列长度预测出资源的最早可用时间,从中过滤出最早可用时间小于预留开始时间的候选预留资源.试图把预留请求调度到这些资源上,从而避免任务抢占的发生.

在上述系统中,当预留请求被提出以后,如果存在符合需求的资源,就被按照一定的策略插入到某个候选资源的预留任务队列中,在其实际使用资源之前就完成了资源分配,我们把这种机制称为静态的资源预留机制(static resource reservation mechanism,简称 SRRM).在 SRRM 下,当预留请求提出时就对其进行调度,由于此时调度器无法准确预测在预留任务实际使用资源时的作业队列状态,因而难以控制预留请求对非预留任务和资源利用率的影响.这是由于,一方面预测模型难以准确预测每个作业的期望执行时间;另一方面,在预留提前时间内作业队列可能会发生变化,比如作业优先级的改变、新作业的到来甚至新的预留任务的到来.另外,由于网格的动态性特征,在预留提前时间内可能会发生资源故障,从而导致预留任务的失败.对于这种情况,SRRM 缺乏必要的容错性.本文提出了一种动态的资源预留机制 DRRM,在该机制下,当用户提出自己的资源预留请求后,系统只对其进行接纳测试,并不进行实际的资源分配.当预留任务的资源使用时间到来时,调度器再根据每个候选资源的运行时信息,把预留请求调度到合适的资源上.与 SRRM 相比,由于考虑了在预留提前时间内资源的变化

情况, DRRM 能够做出更符合实际情况的调度,有效地缓冲网格动态性所造成的影响.另外,用户在提出自己的预留请求后能够实时得到系统的反馈,不会感受到实际资源分配的延时.

2 基于资源-预留图的资源预留机制

假设作业在资源上是顺序执行的,且资源一次只能处理一个作业.非预留任务被打断后,可以在合适的时间从断点处继续执行或者从头开始重新运行.前一种处理方式需要底层的资源管理器以及操作系统的支持,并且需要一些额外而繁琐的操作,比如任务挂起、设立检查点等.本文只考虑了后一种处理方式.假设当前网格中有 N 个资源、M 个已经被接纳的预留请求(预留任务),我们使用资源-预留图(resource-reservation graph,简称 RRG)来表示当前网格系统中的资源和预留任务的关系.

定义 1. 资源-预留图是一个三元组$\langle R,Q,E\rangle$,其中:R 代表图中所有资源节点的集合,一个资源节点对应于网格中的一个资源 r_j,$R=\{r_j|1\leqslant j\leqslant N,N$ 为网格中所有资源的个数$\}$;Q 代表图中所有预留节点的集合,一个预留节点对应于网格中的一个预留任务 q_i,$Q=\{q_i|1\leqslant i\leqslant M,M$ 为所有预留任务的个数$\}$;如果 r_j 能够满足 q_i 的资源需求,则在图中存在一条连接 q_i 和 r_j 的边 e_{ij},$E=\{e_{ij}|1\leqslant i\leqslant M,1\leqslant j\leqslant N\}$是边集合.

根据定义 1,网格中预留任务和资源个数的变化对应着在资源-预留图中节点和边的变化.我们以具有 N 个资源节点和 M 个预留节点的资源-预留图 G 为例来描述节点和边的修改规则.把一个预留任务的最早开始时间记作 et,最迟结束时间记作 lt,预留持续时间记作 dur,则从 et 到 lt 之间的时间间隔叫做预留窗口.不失一般性,假设 G 中的预留节点是按照最早开始时间非降序排列的,即 $q_{i-1}.et\leqslant q_i.et\leqslant q_{i+1}.et(1<i<M)$,并且每个资源节点具有唯一的、大小在 1 和 N 之间的编号.下面给出所需的参数定义:

- 边 e_{ij} 的值 c_{ij}:如果边 e_{ij} 存在(即资源节点 r_j 可以满足预留节点 q_i 的资源需求),则 c_{ij} 为 1,否则为 0.
- 预留节点 q_i 的度数 $deg(q_i)$:对于一个预留节点,把与其连接的边的条数称为它的度数,
$$deg(q_i)=\sum_{j=1}^{N}c_{ij}.$$
- 预留节点 q_i 的候选资源集$\lambda(q_i)$:能够满足 q_i 资源需求的资源节点的集合,$\lambda(q_i)=\{r_j|c_{ij}=1,1\leqslant j\leqslant N\}$.
- 资源节点 r_j 的候选预留集$\eta(r_j)$:资源需求能够被 r_j 满足的预留节点的集合,$\eta(r_j)=\{q_i|c_{ij}=1,1\leqslant i\leqslant M\}$.
- 预留节点 q_i 的前相交集$\alpha(q_i)$:与 q_i 在预留窗口上相交,且其节点号在 i 之前的预留节点的集合,即
$$\alpha(q_i)=\{q_j|q_j.lt\geqslant q_i.et,1\leqslant j<i\}.$$
- 预留节点 q_i 的后相交集$\beta(q_i)$:与 q_i 在预留时间窗口上相交,且其节点号在 i 之后的预留节点的集合,即$\beta(q_i)=\{q_j|q_j.et\leqslant q_i.lt,1<j\leqslant M\}$.

2.1 预留节点的添加规则

在图 G 中添加一个预留节点,意味着一个新的预留请求被系统接受.由于接受一个新的预留请求可能会影响到已有的预留任务(已被接受的预留请求),引起资源分配的冲突,因此必须对其进行一定的接纳测试.

对于一个新来的预留请求,按照最早开始时间非减的顺序把它插入到已有的预留节点序列中.假设它对应的位置为 k,则我们用预留节点 q_k 来表示它,如图 1(a)所示.对预留节点 q_k 来说,如果它的某个候选资源 r_j 也是其前相交集中某个预留节点 q_i 的候选资源,我们把 r_j 叫做冲突资源,把这种情况称为 q_k 和 q_i 关于 r_j 相冲突.由于 q_i 的开始时间较早,可能会优先占用 r_j(取决于实际的调度),从而剥夺 q_k 对该资源的使用权,因此,每发生一次预留冲突,q_k 的可用候选资源数就可能会减少 1 个.由于在 q_k 的前相交集中只有一个预留任务会实际地被调度到某个冲突资源上,因此,关于同一个冲突资源的实际最大冲突次数为 1.如图 1(b)所示,假设$\alpha(q_k)=\{q_{k-2},q_{k-1}\}$.由于在 q_{k-2} 和 q_{k-1} 之中最多只有一个能够被调度到资源 r_j 上,因此,q_k 和$\alpha(q_k)$关于 r_j 的最大冲突数为 1.同理,如果 q_k 与某个预留节点关于多个资源相冲突,则实际的最大冲突次数也为 1.如图 1(c)所示,假设$\alpha(q_k)=\{q_{k-1}\}$.由于 q_{k-1} 只能被调度到 r_j 和 r_{j-1} 其中的一个资源上,因此,q_k 与 q_{k-1} 之间的最大可能冲突数为 1.可见,q_k 关于其候选资源的实际最大冲突次数为

$$\sum_{r_j\in\lambda(q_k)}\operatorname{sgn}\left(\sum_{q_i\in\alpha(q_k)}c_{ij}\right).$$

q_k与其前相交集的实际最大冲突次数为

$$\sum_{q_i\in\alpha(q_k)}\operatorname{sgn}\left(\sum_{r_j\in\lambda(q_k)}c_{ij}\right).$$

我们定义 q_k的自由度为

$$freeDeg(q_k)=Deg(q_k)-\min\left(\sum_{r_j\in\lambda(q_k)}\operatorname{sgn}\left(\sum_{q_i\in\alpha(q_k)}c_{ij}\right),\sum_{q_i\in\alpha(q_k)}\operatorname{sgn}\left(\sum_{r_j\in\lambda(q_k)}c_{ij}\right)\right).$$

如果下面的不等式(1)成立,那么该预留请求可以被系统接纳;否则,拒绝该预留请求.

$$freeDeg(q_k)\cdot\prod_{q_m\in\beta(q_k)}freeDeg(q_m)>0 \tag{1}$$

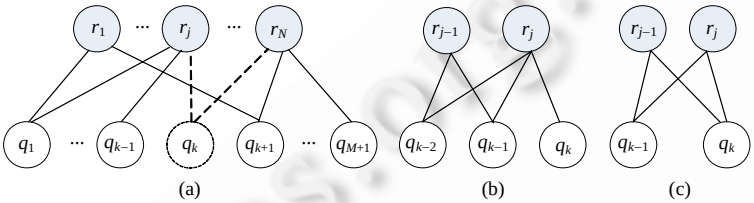

Fig.1 Resource-Reservation graph

图 1 资源-预留图

事实上,当预留请求的自由度为 0 时,它仍有可能被系统接纳.这是由于,在实际的执行过程中,与之相冲突的其他预留任务可能并不会被调度到对应的冲突资源上.通常,用户的预留任务具有一定的重要性,为了避免已被接纳的预留请求在其开始时间到来时反而无法执行的情况,我们严格限制预留请求的自由度必须大于 0.用addQNode 算法来描述预留节点的添加规则,具体过程如下:

算法 1. *addQNode*(*newQ*,*G*).

输入:新到来的用户预留请求 *newQ* 和网格系统当前的资源-预留图 *G*;

输出:true(*newQ* 被系统接纳)或者 false(*newQ* 被系统拒绝).

1. 把 *newQ* 按照开始时间由早到晚的顺序插入到 *G* 的预留任务队列 *q* 中,令 *newQ* 在 *q* 中的序号为 *k*
2. **if** q_k的自由度 $freeDeg(q_k)$=0
3. 把 q_k从预留任务队列 *q* 中删除
4. **return** false
5. **end if**
6. **for** 每一个预留任务 $q_m(q_m\in\beta(q_k))$
7. **if** $freeDeg(q_m)$=0 //接纳 q_k将使得不等式(1)不成立
8. 把 q_k从预留任务队列 *q* 中删除
9. **return** false
10. **end if**
11. **end for**
12. update ***G*** //把 q_k加入到图 *G* 中,对 *G* 的状态进行更新
13. **return** true

2.2 边的删除规则

当预留任务的开始时间到来时,它将打断某个在其候选资源上正在运行的非预留任务,产生一定的抢占代价.文献[6]使用非预留任务所占用的节点个数和已运行时间来计算抢占代价.在本文中,我们选择一种比较简单但不失代表性的方法来定义抢占代价.假设非预留任务 p_k 在资源 r_j 上的期望执行时间为 et_{kj},当它在资源 r_j 上运行了 pt_{kj} 时间长度时被某个预留任务打断,则我们定义此时产生的抢占代价为 $cost(p_k,r_j,pt_{kj})=w_j\cdot pt_{kj}$.其中,$w_j$ 为资源 r_j 的单位时间代价.比如,单位时间所需的费用反映了经济代价,单位时间可以执行的指令数则反映了计算能力代价.

当预留节点 q_i 的预留开始时间到来时,在资源节点 $r_j(r_j\in\lambda(q_i))$上找到当前正在运行的非预留任务,记作 p_k.如果 $q_i.lt-q_i.et-q_i.dur\geqslant et_{kj}-pt_{kj}$,那么我们只要把 q_i 的开始时间推迟到 $q_i.et+et_{kj}-pt_{kj}$,而不必打断 p_k 的执行,此时的抢占代价为 0;否则,如果把 q_i 调度到 r_j 上,会使得 q_i 在 $q_i.et$ 时刻打断 p_k 的执行,抢占资源 r_j,此时的抢占代价为 $cost(p_k,r_j,pt_{kj})=w_j\cdot pt_{kj}$.

根据不同的目标,可以有不同的预留任务调度策略.把预留任务调度到资源能力与其需求最匹配的资源上,有利于增加通过接纳测试的任务个数;把预留任务调度到可以产生最小抢占代价的资源上,则可以提高资源的有效利用率,减小对非预留任务执行的影响;在资源能力相同的资源中选择能够产生最小抢占代价的资源,则可以说是前面两者的折衷.本文把最小化抢占代价作为预留任务调度的目标,使用算法 deleteEdge 描述资源-预留图中边的删除规则.

算法 2. *deleteEdge*(q_i,G).

输入:开始时间到来的预留任务 q_i,网格系统当前的资源-预留图 G;

输出:*null*.

1. **for** 每一个资源 $r_j(r_j\in\lambda(q_i))$
2. 找到在该资源上当前正在运行的非预留任务,记作 p_k
3. **if** $q_i.lt-q_i.et-q_i.dur\geqslant et_{kj}-pt_{kj}$
4. 把 q_i 调度到 r_j 上
5. $q_i.et:=q_i.et+et_{kj}-pt_{kj}$
6. **return**
7. else $cost_j:=w_j\cdot pt_{kj}$
8. **end if**
9. **end for**
10. 把 q_i 调度到资源 r_m 上($m=\arg\min_j\{cost_j\}$)
11. 删除 q_i 和除 r_m 之外的候选资源节点之间的边以及资源 r_m 与$\beta(q_i)$中所有预留节点之间的边
12. **return**

2.3 预留节点的删除规则

当一个预留任务运行完毕后,从图 G 中删除它所对应的预留节点以及与该节点相关的边.

2.4 资源节点的添加规则

网格具有动态性特点,随时可能有新的资源加入到系统中.当一个新的网格资源可用时,在图 G 中生成一个对应的新的资源节点,找到资源需求能够被它满足的预留节点(不包括那些正在运行的以及已被调度但由于开始时间被推迟而尚未运行的预留节点),并连接它们之间的边.

2.5 资源节点的删除规则

当一个网格资源变得不可用时,已被系统接受的一些预留请求可能会发生预留失效,需要删除图 G 中相应的节点和边.假设资源 r_m 出现故障,r_m 的删除规则如图 2 所示.在第(4)行中,当资源出现故障后仍然能够使不等

式(1)成立的预留任务不会发生预留失效,因此应该在图 G 中保留其对应的预留节点.在第(5)行中,我们每次删除预留开始时间最早的预留节点来试图为它的后相交集中的预留节点释放出可用的候选资源.

(1) Let $S=\eta(r_m)$. Delete the resource node corresponding to r_m and all of its edges from G.
(2) Find in S the task running on r_m or scheduled to r_m (without running because of the delay of its begin time) and delete the reservation node corresponding to the task and all of its edges from G.
(3) **while** $S\neq\varnothing$
(4) Find in S all the reservation nodes satisfying the inequality (1). Keep them in G and delete them from S
(5) Find in S the reservation node with the smallest index and delete it and all of its edges from G.
(6) **end while**

Fig.2 Rule to delete resource node in case of resource error

图 2 出现资源故障时节点的删除规则

3 实验过程及结果分析

3.1 实验描述和参数定义

我们在模拟任务集合和真实任务集合上分别进行了 DRRM 和 SRRM 的对比实验.实验中考虑了 SRRM 下的两种调度策略:r_schedule 和 m_schedule.前者把新来的预留请求随机地调度到一个候选资源上;后者把新来的预留请求调度到将会导致最小抢占代价的候选资源上.模拟任务集合和真实任务集合的构成如下:

模拟任务集合:我们模拟生成一组网格资源,每个资源具有一个非预留队列和预留队列,分别存储调度器分配给它的非预留任务和预留任务.在每个非预留队列中生成足够多的运行时间在 5~50 内均匀分布的非预留任务,使得在每个预留任务开始执行时,在各个资源上都有正在运行的非预留任务(模拟在每个预留任务的 book-ahead time 内都有非预留任务到达).每个预留请求的 book-ahead time 和预留窗口长度分别在 1~20 和 5~50 内均匀分布.我们把资源能力和预留请求的资源需求随机划分为 1~5 个等级,一个预留请求的候选资源的资源能力等级必须不小于它的资源需求等级.

真实任务集合:我们使用位于 Ohio Supercomputing Center 的一个 Linux 集群的历史计算任务集合(http://www.cs.huji.ac.il/labs/parallel/workload/l_osc/OSC-Clust-2000-2.1-cln.swf.gz)作为实验样本.该集群由 57 个节点(32 个为 2CPU 节点,25 个为 4CPU 节点)组成,任务集合包含在两年之中所处理的 80 714 个任务,数据格式符合 Standard Workload Format 格式[16].我们把任务样本中的 10%作为预留任务,其余的非预留任务平均分配到各个节点上.假设预留任务的资源需求分为 3 种类型:quad_type,dual_type 和 careless_type.前两种类型的预留任务分别要求预留 CPU 个数为 2 和 4 的计算节点,最后一种类型的预留任务对节点的 CPU 个数没有要求.假设预留任务的到达服从泊松过程,并把每个预留任务的预留提前时间设置为预留时间窗口的一半.令原始任务集合中的某项任务的历史执行时间和所使用的 CPU 个数分别为 t 和 n,在我们的实验中所预留或所占用的节点的 CPU 个数为 m,那么我们把 $\frac{t\cdot n}{m}$ 作为在该节点上的预留持续时间或期望执行时间.

实验中使用到的参数如下:

1) 资源个数 *resNumber*:模拟生成的网格资源的个数.
2) 期望时间间隔 λ:我们把资源预留请求的到来看作是一个离散的泊松分布过程,即相邻两个请求之间的时间间隔服从指数分布,我们把该时间间隔的期望记做 λ.
3) 预留请求个数 *reqNumber*:我们选取前 *reqNumber* 个到来的预留请求作为实验的考察对象.
4) 预留灵活度 *flexibility*:预留的最大可推迟时间占其预留窗口的比例,即

$$flexibility=(let\text{-}est\text{-}dur)/(let\text{-}est).$$

5) 预留接纳率 AP(admission percentage):被接纳的预留请求个数占全部预留请求个数的比例.
6) 抢占率 PP(preemption percentage):导致任务抢占的预留任务个数与被接纳的预留请求个数的比值.

7) 总抢占代价 TPC(total preemption cost):我们采用在第 2.2 节中所提出的抢占代价,即 $cost(p_k,r_j,pt_{kj})=w_j \cdot pt_{kj}$.由于 w_j 是一个常量并且只与资源本身的属性相关,在实验中,我们把它的值简化为 1.一批预留任务所产生的抢占代价之和叫做这批预留任务的总抢占代价.

8) 故障时间比σ:资源出现故障的时间与最后一个预留任务的完成时间的比值(实验从 0 时间开始).

9) 预留失效率η:当资源预留请求通过接纳测试后,由于资源故障导致预留任务失败的概率.

3.2 模拟任务集合上的实验结果及分析

在实验 1 中,我们改变资源的个数来观察 DRRM 和 SRRM 对预留任务的调度效果.*reqNumber*,λ和 *flexibility* 的值分别取 200,2 和 0.1,把 *resNumber* 的值由 5 增加到 100,结果如图 3 和图 4 所示.*SRRM_r* 和 *SRRM_m* 分别表示在 SRRM 下使用 r_schedule 和 m_schedule 策略的调度结果.

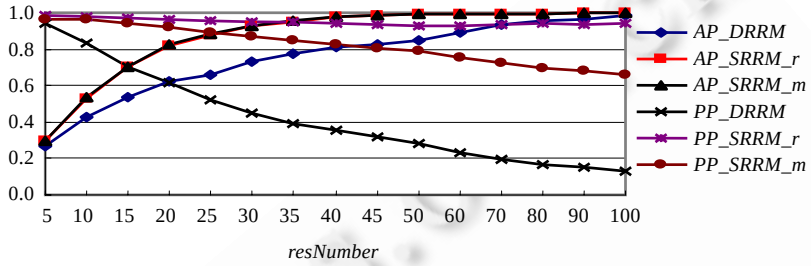

Fig.3 Impact of resource number on the admission percentage and preemption percentage

图 3 资源个数对预留接纳率和抢占率的影响

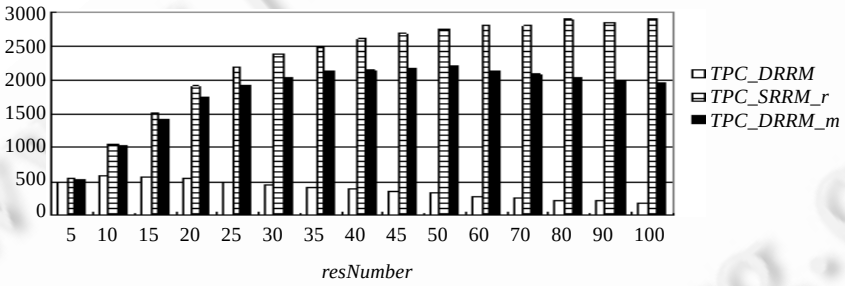

Fig.4 Impact of resource number on the total preemption cost

图 4 资源个数对总抢占代价的影响

由图 3 可以看出,随着 *resNumber* 的增加,*AP_SRRM_m*,*AP_SRRM_r* 和 *AP_DRRM* 都在增大,相同条件下,前两者几乎相等而 *AP_DRRM* 相对较小.这是由于,在 DRRM 下,预留任务的实际调度存在不确定性,在进行接纳测试时必须考虑所有可能发生的冲突,因而更加严格.当资源个数较少时,随着 *resNumber* 的增加,在 SRRM 和 DRRM 下,*AP* 的差距逐渐增大,当 *resNumber*=25 时达到最大值,此时,前者是后者的 1.33 倍;之后,随着 *resNumber* 的增加,两者逐渐接近,当 *resNumber*=100 时,两种机制下的预留请求几乎全部被接受.尽管 SRRM 的预留接纳率较高,但也会导致较高的资源抢占率.在图 3 中,*PP_SRRM_m* 和 *PP_SRRM_r* 始终大于 *PP_DRRM*,且随着资源的增加,这种差距迅速扩大.*PP_SRRM_r* 受资源个数的影响很小,在 *resNumber* 由 5 增加到 100 的过程中,波动的幅度小于 5%.这是由于,r_schedule 把预留请求随机地调度到候选资源上,是否会发生任务抢占与资源个数没有直接关系.*PP_SRRM_m* 和 *PP_DRRM* 则随着资源的增多而不断下降,这是由于,m_schedule 和 DRRM 总是把预留任务优先调度到可以避免发生抢占的资源上,资源个数越多,避免抢占的可能性越大.与 *PP_SRRM_m* 相比,*PP_DRRM* 受资源个数的影响更大,在整个过程中,前者仅下降了 31.7%,而后者则下降了高达 86.1%.图 4 对两种机制下的总抢占代价进行了比较.可以看出:随着资源个数的增加,*TPC_SRRM_r* 先增大,之后保持稳定;

TPC_SRRM_m 先增后减;*TPC_DRRM* 在 *resNumber* 由 5 增加到 10 的过程中有小幅增长,之后则不断下降.对于 *TPC_SRRM_r* 来说,*resNumber* 的增加会导致更多的预留请求被接纳,而 *PP_SRRM_r* 则没有明显的变化,因此会发生更多的资源抢占,从而产生更大的总抢占代价.当 *resNumber* 增加到一定程度后,*AP_SRRM_r* 不再增大,*TPC_SRRM_r* 也将保持稳定.在 *resNumber* 由 5 增加到 50 的过程中,*AP_SRRM_m* 的增长速度要大于 *PP_SRRM_m* 的下降速度;之后,前者保持稳定而后者继续下降.因此,*TPC_SRRM_m* 会出现先升后降的变化.对于 DRRM 来说,显然,*resNumber* 对 PP 的影响要大于其对 AP 的影响,因此,*TPC_DRRM* 整体上随着 *resNumber* 的增加而减小.在相同条件下,*TPC_SRRM_m* 要略小于 *TPC_SRRM_r*,而 *TPC_DRRM* 要远远小于前两者.值得注意的是,虽然 m_schedule 策略总是把预留任务调度到能够产生最小抢占代价的资源上,但是 *TPC_SRRM_m* 明显大于 *TPC_DRRM*.这是因为,m_schedule 在预留请求到来时把它调度到“最优”(能够产生最小的抢占代价)的资源上,这种操作可能会影响到已经分配到该资源上的尚未运行的预留任务,使得当前的调度干扰到先前的“最优”调度,缺乏一定的“后视性”.而 DRRM 则按照每个预留任务实际开始的顺序进行调度,更容易做出相对“较优”的调度.当 *resNumber*=25 时,SRRM 和 DRRM 下,AP 的差距最大,此时,前者是后者的 1.33 倍;然而,此时 *TPC_SRRM_r* 和 *TPC_SRRM_m* 却分别是 *TPC_DRRM* 的 4.46 倍和 3.9 倍.可见,与 SRRM 相比,尽管 DRRM 牺牲了一定的预留接纳率,却可以挽回更大的资源损失,并且在资源个数较多时,这种优势更加明显.

在实验 2 中,改变预留请求的到达间隔来观察 SRRM 和 DRRM 的调度效果.*reqNumber*,*resNumber* 和 *flexibility* 的值分别取 200,10 和 0.1,把λ的值以 2 为步长由 2 增加到 20,结果如图 5 和图 6 所示.

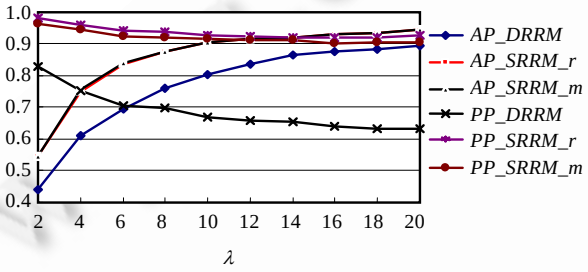

Fig.5 Impact of request interval on the admission percentage and preemption percentage

图 5 预留请求到达间隔对预留接纳率和抢占率的影响

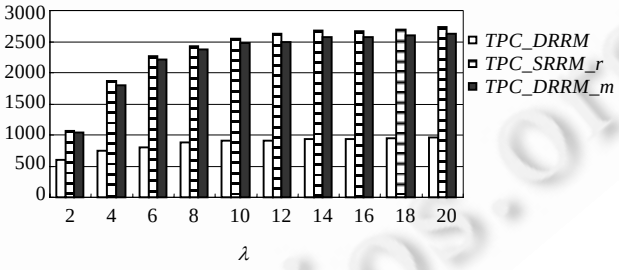

Fig.6 Impact of request interval on the total preemption cost

图 6 预留请求到达间隔对总抢占代价的影响

从图 5 可以看出,随着λ的增加,两种机制下的 AP 也在不断增大.这是由于λ的增加使得各个预留请求的预留窗口在时间上的分布更为分散,相交的概率降低,因此更容易被系统接纳.*AP_SRRM_r* 和 *AP_SRRM_m* 始终大于 *AP_DRRM*,但随着λ的增加,它们之间的差距趋于缩小.*PP_SRRM_r* 和 *PP_SRRM_m* 随着λ的增加略有减小;*PP_DRRM* 则有较大的下降.对 DRRM 来说,λ的增加会减小每个预留请求的相交集,使其有更多的可选资源,有利于避免任务抢占的发生,降低抢占率.图 6 描述了两种机制下总抢占代价的变化.可以看出,随着λ的增加,*TPC_DRRM*,*TPC_SRRM_r*,*TPC_SRRM_m* 都在增大,在相同条件下,*TPC_SRRM_r* 略大于 *TPC_SRRM_m* 而远大于 *TPC_DRRM*.对 SRRM 来说,λ的增加导致更多的预留请求被接受,而 PP 变化不大,因而会导致更大的总抢占

代价.对 DRRM 来说,一方面,λ的增加有利于降低 PP;另一方面,也会导致更多的预留请求被接纳,发生更多的任务抢占.实验结果表明,λ对后者的影响要大于前者.当λ=6 时,SRRM 和 DRRM 下,AP 之间的差距最大,前者是后者的 1.2 倍.而此时,*TPC_SRRM_r* 和 *TPC_SRRM_m* 却分别为 *TPC_DRRM* 的 2.84 倍和 2.76 倍.可见,我们可以得到与实验 1 相同的结论,即 DRRM 损失了一定的预留接纳率却减小了更大的资源损失.并且随着λ的增加,DRRM 和 SRRM 下的预留接纳率逐渐接近,而总抢占代价的差距却在不断扩大.

实验 3 中,我们观察两种机制下预留请求的灵活度对总抢占代价和抢占率的影响.我们首先生成 200 个预留请求,从中选取能够同时被 SRRM 和 DRRM 接纳的(70 个)作为观察对象.实验中,*reqNumber*,*resNumber* 和λ的值分别取 200,10 和 2,以 0.1 为步长把 *flexibility* 由 0.1 增加到 0.9,实验结果如图 7 和图 8 所示.由图 7 可以看到,随着预留灵活度的增加,两种机制下的抢占率和总抢占代价都在下降.在 *flexibility* 由 0.1 增加到 0.9 的过程中,*PP_DRRM* 下降了 66.7%,而 *PP_SRRM_r* 和 *PP_SRRM_m* 仅分别下降了 34.8%和 39.1%;*TPC_DRRM* 下降了 59.4%,而 *TPC_SRRM_r* 和 *TPC_SRRM_m* 仅分别下降了 32.8%和 33.7%.可见,在 DRRM 机制下,放宽对预留任务开始时间的限制,更加有利于减小预留任务对非预留任务和资源利用率的负面影响.

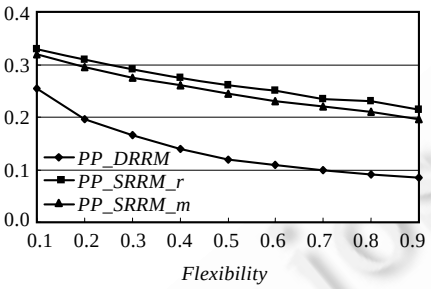

Fig.7 Impact of reservation flexibility on the preemption percentage

图 7 预留灵活度对抢占率的影响

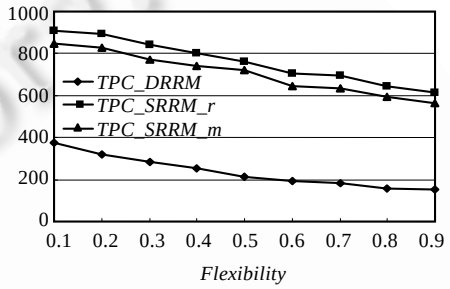

Fig.8 Impact of reservation flexibility on the total preemption cost

图 8 预留灵活度对总抢占代价的影响

3.3 真实任务集合上的实验结果及分析

第 1 组实验主要观察预留请求到达间隔对预留接纳率和总抢占代价的影响.我们把 *flexibility* 设为 0,以 60 为步长把λ由 60s 增加到 780s,结果如图 9 和图 10 所示.可以看到:随着λ的增加,DRRM 和 SRRM 下的预留请求接纳率都在增加;相同情况下,SRRM 将比 DRRM 接纳更多的预留请求.尽管 SRRM 在预留接纳率方面优于 DRRM,但也会导致更大的抢占代价.当λ=300s 时,*AP_SRRM_r* 和 *AP_DRRM* 之间的差距最大,此时,前者比后者大 27%.然而,*TPC_SRRM_r* 却比 *TPC_DRRM* 大了 136%.可见,虽然 DRRM 损失了相对较小的预留接纳率,但却可以极大地减少抢占代价,这与在模拟任务集合上的实验结论是一致的.

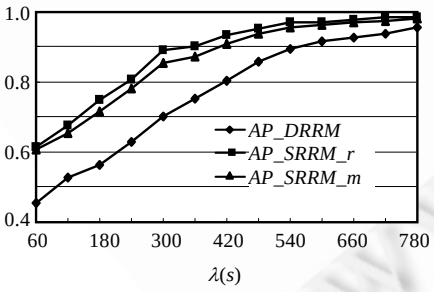

Fig.9 Impact of reservation arrival interval on the admission percentage

图 9 预留请求到达间隔对预留请求接纳率的影响

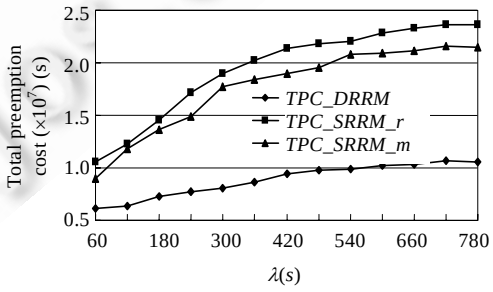

Fig.10 Impact of reservation arrival interval on the total preemption cost

图 10 预留请求到达间隔对总抢占代价的影响

第 2 组实验考察了预留灵活度在提高调度效果方面的影响.选取能够同时被 DRRM 和 SRRM 接纳的预留请求(在实验中为 1 073 个),把λ设置为 240s,以 0.1 为步长把 *flexibility* 由 0 增加到 0.9,实验结果如图 11 和图 12 所示.可以看到,预留请求的灵活度越大,所造成的任务抢占率和抢占代价越小.随着 *flexibility* 由 0 增加到 0.9, *TPC_DRRM*,*TPC_SRRM_r* 和 *TPC_SRRM_m* 分别下降了 61.3%,25.8%和 31.3%,并且 DRRM 下的抢占率比 SRRM 下的抢占率下降得更快.在整个过程中,*PP_SRRM_r* 和 *PP_SRRM_m* 分别下降了 31%和 37%,而 *PP_DRRM* 则下降了高达 67%.可见,在 DRRM 机制下,预留请求灵活度的优势能够更好地得以发挥,更有利于对资源有效利用率的改善.

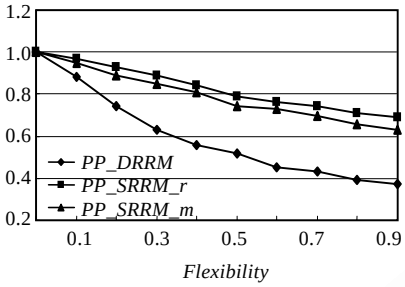

Fig.11 Impact of reservation flexibility on the preemption percentage

图 11 预留灵活度对抢占率的影响

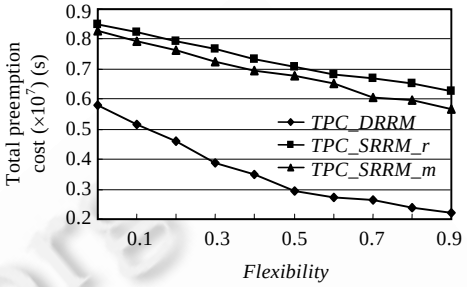

Fig.12 Impact of reservation flexibility on the total preemption cost

图 12 预留灵活度对总抢占代价的影响

3.4 预留失效分析

根据资源发生故障的时刻,可以把故障划分为两类:对于一个已经被接纳的预留请求 q 来说,如果资源故障发生在其运行之前,我们称其为排队时故障;如果故障发生在其运行的过程中,我们称其为运行时故障.假设资源 r_i 出现故障的概率为 p_i,各资源发生故障的事件之间是独立的,在 SRRM 机制下,q 将被调度到 r_i 上;在 DRRM 机制下,q 也会被调度到 r_i($r_i \in M$,M 是 r_i 的候选资源集合)上.

对 q 来说,如果 r_i 出现的故障是运行时故障,此时无论是在 SRRM 还是 DRRM 机制下,预留都会失效,即 $\eta_{DRRM}=\eta_{SRRM}=p_i$.如果 r_i 出现的故障是排队时故障,由于 SRRM 已经把 q 插入到 r_i 的预留队列中,显然,此时 q 的预留失效率 $\eta_{SRRM}=p_i$.而在 DRRM 机制下,如果$|M|=1$,即 q 只有一个候选资源 r_i,那么也会发生预留失效.在这种情况下,$\eta_{SRRM}=\eta_{DRRM}=p_i$.如果$|M|>1$,调度器会尝试把 q 调度到其他正常工作的候选资源上.由于把 q 调度到其他资源上可能会与另外的已被接纳的预留请求产生冲突,因此还需对 q 进行接纳测试,即把 q 当作一个新到来的预留请求来对待.我们把 q 未通过接纳测试的事件记为 A,把 q 通过接纳测试的事件记为 B,并且假设根据第 2.2 节所提出的调度算法,q 最终被调度到资源 r_j 上.那么,q 的预留失效率 $\eta_{DRRM}=p_i\cdot(P(A)+p_j\cdot P(B))$,其中,$P(A)$和 $P(B)$ 分别为 A 事件和 B 事件出现的概率,p_j 则是资源 r_j 出现运行时故障(相对于预留请求 q)的概率.由于 $P(A)+P(B)=1$,$p_j \leqslant 1$,因此 $\eta_{DRRM} \leqslant \eta_{SRRM}$.

值得注意的是,尽管当出现运行时故障时,两种机制都无法避免预留失效的发生,但相对于 SRRM,我们的机制具有更好的容错效果.假设在 SRRM 机制下,在故障时间到来时调度器分配给资源 r_i 的尚未运行的预留任务的集合为 R,则 R 中所有的任务都会发生预留失效.而在 DRRM 机制下,调度器只是确定了每个预留请求的候选资源集合,并没有明确地将其分配到某个具体的资源上.因此,针对于当前正在运行的预留任务 q 来说,r_i 发生了运行时故障;而对于其他把 r_i 作为候选资源的预留任务来说,r_i 只是发生了排队时故障,正如上面所讨论的,并不一定会导致资源预留的失效.可见,DRRM 对网格环境的动态变化具有更好的“鲁棒性”.

我们在模拟任务集合上通过实验来观察 DRRM 对资源故障的容错效果.从所有资源中随机选择一个作为故障资源,在 DRRM 机制下,根据图 2 中的规则处理资源故障情况,在 SRRM 机制下使用 r_schedule 策略.实验中,*reqNumber*,*resNumber*,*flexibility* 和λ的值分别取 200,10,0.1 和 2,以 0.1 为步长,把故障时间比σ的值由 0.1 增加到 0.9,实验结果如图 13 所示.可以看到,随着σ的增加,在两种预留机制下,预留失效率都在下降.在相同条件

下,η_{DRRM}总是小于η_{SRRM},在σ=0.5时,两者间的差距最小,此时,前者是后者的47.3%;在σ=0.4时,两者间的差距最大,此时,前者是后者的68.4%.

可见,针对于资源故障DRRM具有较好的容错效果,这与我们的理论分析是一致的.

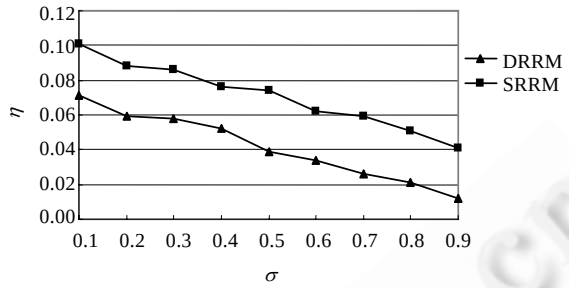

Fig.13 Impact of error time ratio on the reservation failure percentage

图13 故障时间比对预留失效率的影响

4 结束语

在传统的静态预留机制(SRRM)下,预留请求一被提出就被立即调度到某个资源上,没有考虑在预留提出时间内资源的动态变化.事实上,在预留提前的那段时间内,可能会有新的非预留任务和预留任务到达资源的作业队列中,导致对预留任务的调度与目标产生偏差;甚至会出现资源故障,导致预留任务的失败.本文提出了一种动态的资源预留机制(DRRM),在该机制下,当用户提出自己的预留请求时,仅对其进行接纳测试,等预留开始时间到来时再进行实际的资源分配.在用户看来,自己的预留请求可以马上得到响应,并不会感受到实际资源分配的推迟所带来的影响.我们定义了资源-预留图(RRG)来描述DRRM对预留请求的接纳测试、调度策略以及对资源故障的处理.与SRRM相比,尽管DRRM会在一定程度上降低预留请求的接纳率,但却可以大大减少任务抢占的次数,并且对资源故障具较好的容错效果.当预留请求开始时间的弹性较大、候选资源较多且发生的频率较低时,DRRM更具优势.

References:

[1] Foster I, Kesselman C, Lee C, Lindell B, Nahrstedt K, Roy A. A distributed resource management architecture that supports advance reservations and co-allocation. In: Dawn B, Wendy R, eds. Proc. of the Int'l Workshop on Quality of Service (IWQoS'99). London: IEEE Communications Society, 1999. 27−36. [doi: 10.1109/IWQOS.1999.766475]

[2] Hu CM, Huai JP, Wo TY, Lei L. A service oriented grid architecture with end to end quality of service. Journal of Software, 2006, 17(6):1448−1458 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1448.htm [doi: 10.1360/jos171448]

[3] Cao J, Zimmermann F. Queue scheduling and advance reservation with COSY. In: Amaral JN, ed. Proc. of the 18th IEEE Int'l Parallel and Distributed Processing Symp. Santa Fe: IEEE Computer Society, 2004. 881−888. [doi: 10.1109/IPDPS.2004.1302989]

[4] McGough S, Young L, Afzal A, Newhouse S, Darlington J. Workflow enactment in ICENI. In: Proc. of the UK e-Science All Hands Meeting. 2004. 894−900. http://pubs.doc.ic.ac.uk/ahm04-workflowenactment-iceni/

[5] Sulistio A, Buyya R. A grid simulation infrastructure supporting advance reservation. In: Gonzalez T, ed. Proc. of the IASTED Int'l Conf. on Parallel and Distributed Computing and Systems. Cambridge: ACTA Press, 2004. 1−7.

[6] Smith W, Foster I, Taylor V. Scheduling with advanced reservations. In: Keleher P, ed. Proc. of the 14th Int'l Parallel and Distributed Processing Symp. Los Alamitos: IEEE Computer Society, 2000. 127−132. [doi: 10.1109/IPDPS.2000.845974]

[7] Mu'alem AW, Feitelson DG. Utilization, predictability, workloads, and user runtime estimation in scheduling the IBM SP2 with backfilling. IEEE Trans. on Parallel and Distributed Systems, 2001,12(6):529−543. [doi: 10.1109/71.932708]

[8] Farooq U, Majumdar S, Parsons EW. Impact of laxity on scheduling with advance reservations in grids. In: Fujimoto R, Karatza H, eds. Proc. of the 13th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Los Alamitos: IEEE Computer Society, 2005. 319−324. [doi: 10.1109/MASCOT.2005.33]

[9] Hu CM, Huai JP, Wo TY. Flexible resource capacity reservation mechanism for service grid using slack time. Journal of Computer Research and Development, 2007,44(1):20−28 (in Chinese with English abstract).

[10] Naiksatam S, Figueira S. Elastic reservations for efficient bandwidth utilization in LambdaGrids. Future Generation Computer Systems, 2007,23(1):1−22. [doi: 10.1016/j.future.2006.02.013]

[11] Wu ZA, Luo JZ. Dynamic multi-resource advance reservation in grid environment. In: Li KQ, Jesshope K, Jin H, eds. Proc. of the 2007 IFIP Int'l Conf. on Network and Parallel Computing. Dalian: Springer-Verlag, 2007. 13−22. [doi: 10.1109/E-SCIENCE.2005.13]

[12] Kuo D, Mckeown M. Advance reservation and co-allocation protocol for grid computing. In: Stockinger H, Buyya R, Perrott R, eds. Proc. of the Int'l Conf. on e-Science and Grid Technologies. Los Alamitos: IEEE Computer Society, 2005. 164−171. [doi: 10.1109/E-SCIENCE.2005.13]

[13] Tangpongprasit S, Katagiri T, Kise K, Honda H, Yuba T. A time-to-live based reservation algorithm on fully decentralized resource discovery in grid computing. Parallel Computing, 2005,31(6):529−543. [doi: 10.1016/j.parco.2005.03.005]

[14] PBS professional. http://www.altair.com/software/pbspro.htm

[15] Röblitz T, Schintke F, Reinefeld A. Resource reservations with fuzzy requests. Concurrency and Computation: Practice and Experience, 2006,18(13):1681−1703. [doi: 10.1002/cpe.1023]

[16] Standard workload format. 2006. http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

附中文参考文献:

[2] 胡春明,怀进鹏,沃天宇,雷磊.一种支持端到端 QoS 的服务网格体系结构.软件学报,2006,17(6):1448−1458. http://www.jos.org.cn/1000-9825/17/1448.htm [doi: 10.1360/jos171448]

[9] 胡春明,怀进鹏,沃天宇.一种基于松弛时间的服务网格资源能力预留机制.计算机研究与发展,2007,44(1):20−28.

高瞻(1982－),男,安徽淮北人,博士生,主要研究领域为网格计算.

罗四维(1943－),男,博士,教授,博士生导师,主要研究领域为人工神经网络,分布式并行计算,网格计算.