面向网格工作流的松弛预留策略

肖鹏[1,2]，胡志刚[2]，阎朝坤[2]，屈喜龙[1]
（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，湖南 长沙 410083）

摘 要：针对网格任务倾向于高估预留时间的事实，提出一种支持叠交预留请求的松弛预留策略，用于降低预留机制对系统性能的负面影响。理论分析给出了违约风险的量化计算方法，并针对工作流的协同预留问题设计了相应的预留接纳算法。实验结果显示，松弛策略在面对较高的预留请求率时能显著降低预留机制对系统性能所造成的负面影响；基于实际工作流的实验结果显示，该策略能有效降低由于高估预留时间而导致的任务执行效率损失。

关键词：网格计算；工作流；协同预留；预留违约
中图分类号：TP393 **文献标识码**：A **文章编号**：1000-436X(2012)02-0023-07

Resource reservation based on relaxed policy for grid workflow

XIAO Peng[1,2], HU Zhi-gang[2],YAN Chao-kun[2], QU Xi-long[1]
(1. Department of Computer and Communication, Hunan Institute of Engineering, Xiangtan 411104, China;
2. School of Information Science and Engineering, Central South University, Changsha 410083, China)

Abstract: To address the problem that applications tend to overestimate reservation duration, a novel reservation technique based on overlapping relaxing policy was proposed, so as to mitigate the negative effects of stringent reservation mechanisms on system performance. The analysis of the relaxed policy was represented theoretically and a relaxed admission algorithm was designed for large-scale grid workflows. Experimental results show that relaxed policy can significantly mitigate the negative effects of classical policies when system is in presence of heavy workload. Experimental results based on a real workflow indicate that the relaxed policy is able to decreasing the efficiency loss due to the overestimation of reservation durations.

Key words: grid computing; workflow; co-reservation; reservation violation

1 引言

网格环境[1]中的预留服务主要用于保证资源分配时刻的可获得性，从而为资源协同分配提供 QoS 保证[2]。近期网格性能测评报告显示，过度使用预留机制会对性能造成严重负面影响，例如资源有效利用率下降[2~4]、系统负载不均衡[5,6]、服务请求拒绝率升高[7,8]、调度性能低下[9]。解决预留机制的负面影响成为改进网格系统性能的一个重要的课题。由于资源异构性和负载动态性，任务难以准确估计其预留数量和时间，导致的结果是：①任务倾向于高估其预留时间，以此确保足够的时间弹性[3,7,8]；②任务倾向重复提交预留请求，以此降低资源故障对任务执行的影响[2,3,10]。Dun 等人[11]对大量工作流的性能测评显示，任务实际执行效率往往因为不准确的预留估计而极不确定；Prodan 等人[12]在分析导致工作流延迟的因素时发现，预留协商延迟是影响工作流执行效率的关键因素。基于预留机制存在的问题，本文提出一种支持时空二维松弛的预留策略，允许系统在一定条件下接纳存在叠交的预留

收稿日期：2011-03-21；**修回日期**：2011-12-03
基金项目：国家自然科学基金资助项目（60673165, 60970038）；湖南省自然科学基金资助项目（10JJ6099）
Foundation Items: The National Natural Science Foundation of China (60673165, 60970038); The Natural Science Foundation of Hunan Province (10JJ6099)

请求，目标是降低传统预留机制对系统性能的负面影响。

2 相关研究

Smith 等人[2]从等待时间、预留延迟、拒绝率等方面分析了预留机制对任务调度的影响，其实验数据显示：预留机制能有效降低任务平均等待时间，但面对较高的预留请求率时，以上性能指标都会显著恶化。为降低预留机制对资源利用率的负面影响，回填技术[6]被广泛应用，例如文献[3]中提出了基于回填技术的多资源协同预留算法，其实验数据显示，回填技术能有效减轻预留机制给系统性能带来的负面影响；Sulistio 等人[4]则针对用户频繁取消预留请求的问题提出一种超额预留机制，以此降低频繁取消预留请求所导致的资源效率损失。

传统预留机制的另一个主要缺陷是容易产生大量“资源能力碎片”。针对这一问题，胡春明等人[13]提出一种基于弹性时间的预留策略，通过引入时间弹性因子，以此提高资源调度的自主性和资源综合利用率；Farooq 等人[7]则提出一种“关键任务重调度”机制，用于降低关键路径上的任务预留请求被拒绝的概率；Jawad 等人[14]则提出了一种基于“动态关键路径”的协同预留机制，其核心思想是将通信关联度强的父子节点聚合，从而重构工作流的关键节点和路径；与本文研究最为接近的是 Sulistio 等人在文献[4]中提出“超额预留”机制，但该研究只给出“超额预留”的概念，而对其将导致的违约风险缺乏量化分析和解决策略。而本文则明确给出了量化分析松弛策略的预留违约风险的方法。

3 定义与问题描述

设网格系统由 N 个资源站点组成，表示为集合 $\{R_1,R_2,\cdots,R_N\}$；工作流表示为有向无环图 $G=<T, E>$，其中，$T=[t_1, t_2, \cdots, t_n]$为工作流的子任务集合，$E=\{<t_i, t_j>|if < t_i, t_j>\in T\times T\}$ 为工作流的有向边集合；子任务 t_i 的预留请求表示为三元组$<s_i,e_i,r_i>$，s_i 为预留起始时间，e_i 为预留截止时间，r_i 为资源预留量；资源时间槽 S_k 表示为三元组$<\lambda_k, \mu_k, \pi_k>$，$\lambda_k$ 为时间槽起始时间，μ_k 为结束时间，π_k 为空闲资源量

定义 1 若 S_k 对 t_i 的预留请求$<s_i,e_i,r_i>$以概率 $P_{k,i}$ 满足$(s_i\geqslant\lambda_k)\cap(e_i\leqslant\mu_k)\cap(f_k\geqslant r_i)$，则称时间槽 S_k 松弛满足 t_i 的预留请求。

定义 2 与子任务 t_i 的预留起始时间和截止时间叠交的预留请求集合记分别记为 $S^-(t_i)$和 $S^+(t_i)$：

$$S^-(t_i)=\{t_j\mid\forall j\in[1\cdots i-1],\ s_j<t_i<e_j\} \quad (1)$$

$$S^+(t_i)=\{t_j\mid\forall j\in[1\cdots i-1],\ s_i<s_j<e_i\} \quad (2)$$

定义 3 设随机事件 E_i 表示子任务 t_i 没有发生预留违约，随机事件 E_i^- 表示集合 $S^-(t_i)$没有导致 t_i 发生预留违约，随机事件 E_i^+ 表示集合 $S^+(t_i)$没有导致 t_i 发生预留违约。

当采用松弛预留策略时，由于 $S^-(t_i)$和 $S^+(t_i)$不为空，因此有可能产生预留违约的情况。由于任务存在高估预留时间的倾向，任务实际执行时其预留资源可能提前释放，从而不会实际产生预留违约。

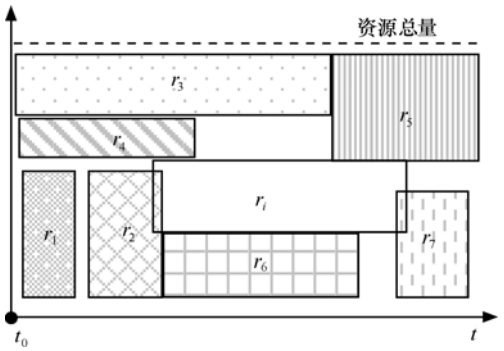


图 1 资源预留时间槽示例

以图 1 为例，当 t_i 的预留请求 r_i 到达时，时间槽上已经存在 7 个预留请求，按照传统预留的定义，则不存在能满足 r_i 的空闲时间槽。$S^-(t_i)=\{r_2, r_3, r_4\}$ 且 $S^+(t_i)=\{r_5, r_7\}$，若其中的预留请求提前释放资源，则 r_i 实际并不会发生预留违约。由定义 3 可知， t_i 不发生预留违约的概率计算公式为

$$\Pr\{E_i\}=\Pr\{E_i^-\}\Pr\{E_i^+\} \quad (3)$$

工作流 $G=<T, E>$的协同预留方案可以表示为从资源需求集合到资源站点集合的映射 $\boldsymbol{M}$:$\{r_1, r_2,\cdots, r_n\}\times\{R_1,R_2,\cdots,R_N\}\rightarrow\{0,1\}$。显然协同预留方案 $\boldsymbol{M}$ 为 $n\times N$ 矩阵，$\boldsymbol{M}_{i,j}=1$ 表示在 C_j 上预留资源给子任务 t_i。综合上述定义和分析，面向工作流的松弛预留策略可以归结为以下优化问题：

$$\begin{aligned}&\text{Min} \quad Makespan(G=<T, E>)\\&\text{s.t.} \quad \Pr\{E|\boldsymbol{M}\}\geqslant V^*\\&\boldsymbol{M}: T\times\{R_1,R_2,\cdots,R_N\}\end{aligned} \quad (4)$$

其中，$\Pr\{E|\boldsymbol{M}\}\geqslant V^*$则表示协同预留方案 $\boldsymbol{M}$ 不发生

预留违约的概率大于等于 V^*。V^*为松弛预留阈值，其数值越大则表示松弛接纳策略越保守，当 V^*=1 时松弛预留策略退化为传统预留策略。由于预留方案 $\boldsymbol{M}$ 的解空间为 $2^{n\times N}$，上述优化问题显然属于 NP-Complete。对此，第 4 节的预留接纳策略将采用进化算法[15]来求取近似最优解。

4　策略分析与求解

4.1　违约概率分析

由式(3)可知，工作流中子任务 t_i 的预留违约概率由 $\Pr\{E_i^-\}$ 和 $\Pr\{E_i^+\}$ 合成。

定理 1　设 $\Pr\{E_i^-\}$ 为集合 $S^-(t_i)$不会导致 t_i 发生预留违约的概率，则 $\Pr\{E_i^-\}$ 满足以下条件：

$$\prod_{k\in S_i^-(t_i)}\Pr\{T_k\leqslant s_i\}\leqslant\Pr\{E_i^-\}\leqslant\prod_{k\in f_j^*}\Pr\{T_k\leqslant s_i\}\tag{5}$$

其中，T_k 为随机变量表示子任务 t_k 的实际完成时间，集合 f_j^* 是 $S^-(t_i)$子集中满足如下条件的子集：

$$\begin{cases}\prod\limits_{k\in f_i^*}\Pr\{T_k\leqslant s_i\}\geqslant\prod\limits_{k\in f_i}\Pr\{T_k\leqslant t_i\}\\ \pi+\sum\limits_{k\in f_j^*}r_k\geqslant r_i\end{cases}\tag{6}$$

证明　集合 $S^-(t_i)$不为空是导致 t_i 发生预留违约的唯一条件。设资源总量为 C，依据 t_i 的预留量与 s_i 时刻空闲资源量的相对关系分 2 种情况：

①若 t_i 的资源预留量 r_i 满足

$$r_i\geqslant C-\min\{r_j\mid j\in S_i^-(t_i)\}\tag{7}$$

则 t_i 需要 $S^-(t_i)$中所有预留请求都在 s_i 之前完成，才能保证不发生预留违约，$\Pr\{E_i^-\}$ 的计算公式为

$$\Pr\{E_i^-\}=\prod_{k\in S_i^-(t_i)}\Pr\{T_k\leqslant s_i\}\tag{8}$$

②若资源预留量 r_i 满足

$$r_i\geqslant C-\min\{r_j\mid j\in S_i^-(t_i)\}\tag{9}$$

则只要 $S^-(t_i)$中的部分预留请求在 s_i 之前完成，就能保证 t_i 不发生预留起始时间违约。最理想的情况是找到一个 $S^-(t_i)$的子集，当该子集中所有预留请求都在 s_i 之前完成时，t_i 发生预留违约的概率最小。显然，该子集必须满足的条件就是式(6)给出的约束条件，此时 $\Pr\{E_i^-\}$ 的计算公式为

$$\Pr\{E_i^-\}=\prod_{k\in f_j^*}\Pr\{T_k\leqslant s_i\}\tag{10}$$

综合以上 2 种情况的分析可知，t_i 不出现预留违约的概率 $\Pr\{E_i^-\}$ 具有严格的上界和下界，其计算式分别由式(8)和式(10)给出，证毕。

由定理 1 可知，计算 $\Pr\{E_i^-\}$ 的关键在于找到满足式(6)的最优子集 f_j^*。由式(6)给出的条件可知，求取 f_j^*的问题可以归结为式(11)所示的的最优规划问题。该规划问题与经典的 0-1 背包问题相同，求解算法在文献[16]中有详细描述，本文不作赘述。

$$\begin{aligned}&\max\ \ \Pr\{E_i^-\}=\prod_{k\in f_i^*}\Pr\{T_k\leqslant s_i\}\\&\text{s.t.}\quad \pi+\sum_{k\in f_j}r_k\geqslant r_i\\&\qquad f_j^*\subseteq S_i^-(t_i)\end{aligned}\tag{11}$$

定理 2　设 $\Pr\{E_i^+\}$ 为 $S^+(t_i)$不会导致 t_i 发生预留违约的概率，若将 $S^+(t_i)$预留请求按起始时间升序排列为<$r_1,r_2,\cdots,r_m$>，则 $\Pr\{E_i^+\}$ 的计算公式为

$$\Pr\{E_i^+\}=\Pr\{T_i\leqslant s_k\}\tag{12}$$

其中，r_k 的预留起始时间满足以下条件

$$\sum_{j=1}^{k-1}r_j\leqslant C-r_i<\sum_{j=1}^{k}r_j\tag{13}$$

证明　若 t_i 的实际完成时间提前并释放其资源，则 $S^+(t_i)$中部分预留请求将不会产生预留违约。关键问题是：t_i 的实际完成时间提前到何种程度才能保证 $S^+(t_i)$的预留请求都不会产生预留违约。为此，可以将 $S^+(t_i)$中的预留请求按起始时间升序排列为<$r_1,r_2,\cdots,r_m$>，并从中找到一个满足式(13)所示条件的 r_k。显然，如果 t_i 的实际完成时间早于 r_k，$S^+(t_i)$中预留请求<$r_1,r_2,\cdots,r_k$>都不会和 t_i 的预留请求发生任何交叠；另一方面，t_i 提前释放的资源能够保证 $S^+(t_i)$中<$r_{k+1},r_{k+2},\cdots,r_m$>都不发生会由于空闲资源不够而导致的预留违约。综上所述，得证。

4.2　预留接纳算法

本文主要考虑网格工作流，由于子任务之间存在相互依赖关系，因此必须首先计算出子任务之间的关联强度。本文采用了 PMCC[17]公式来计算工作流子任务之间的关联强度。工作流中任意两子任务 t_i 和 t_j $(i\neq j)$之间的关联强度计算公式如下：

$$\psi_{i,j}=\begin{cases}1, & t_j\in S_i^-(t_i)\ \text{或}\ t_j^+\in S_i(t_i)\\ \dfrac{\sum(T_iT_j)-\overline{T}_i\overline{T}_j}{\sqrt{\left(\sum T_i^2-\dfrac{(\sum T_i)^2}{N}\right)\left(\sum T_j^2-\dfrac{(\sum T_j)^2}{N}\right)}}, & \text{其他}\end{cases}\quad(14)$$

其中，T_i 和 T_j 为随机变量，分别表示任务 t_i 和 t_j 的实际完成时间。针对网格工作流的基于松弛预留策略的协同预留接纳算法实现如下所示。

Input:

1) Lower bound of relaxed policy: V^*

2) Matrix of resource time slots: S

Output:

1) Matrix of Co-reservation Scheme: M

2) None-violation probability of M: $\Pr\{E|M\}$

Begin

1) *Makespan*:= ∞;

2) Generate a random co-reservation scheme M;

3) while evolution iteration W is not reached do

4) for each t_i in G do

5) if $S^-(t_i)$ and $S^+(t_i)$ is empty then

6) continue;

7) else

8) Using 0-1 knapsack algorithm to find f_j^*;

9) Compute upper bound of $\Pr[E_i^-]$ by formula (10);

10) Find that r_k satisfying formula (13);

11) Compute $\Pr[E_i^+]$ by formula (12);

12) for each t_j that not in $S^-(t_i)\cup S^+(t_i)$

13) Compute $\psi_{i,j}$ by formula (14);

14) end if

15) end for

16) $\Pr\{E|M\}:=\prod_{i=1}^{n}\overline{\psi}_{i,j}\cdot\Pr\{E_i^-\}\cdot\Pr\{E_i^+\}$

17) Mutate M into M^* using swap-mutation with the probability of 0.5;

18) if $\Pr\{E|M\}<V^*$ then

19) $M:=M^*$; continue;

20) Compute the makespan of M as *Makespan**

21) if *Makespan** < *Makespan* then

22) *Makespan* := *Makespan**;

23) $M:=M^*$;

24) end if

25) end while

26) return M and $\Pr\{E|M\}$;

End.

算法采用进化策略来获得调度长度最优化的协同预留方案。step 4 ~ step 15 是采用松弛策略的分析结论来计算工作流中每个子任务不发生预留违约的概率，step 16 则求取整个协同预留方案不出现预留违约的概率。算法的整体复杂度为 $O(Wn^2N)$，W 为进化迭代数目，n 为工作流中的任务节点数，N 为网格系统的资源站点数目。

5　实验与性能分析

实验构建了一个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，各集群节点的性能指标参考 DAS-2[18]的配置。实验首先以随机 DAG 作为工作负载，然后采用 WIEN2K[19]工作流进行实验分析。

5.1　随机 DAG 实验分析

REXR 是本文提出的松弛预留策略；CONR[2]为传统的严格预留策略，主要作为实验分析的基线；CBFR[6]为保守回填预留策略；MALR[20]为可塑预留策略；OVBR[4]为超额预留策略。实验分 5 组进行，预留请求率从 5%逐步增加到 25%。实验设定 OVBR 的超额接纳率为 20%，REXR 的接纳阈值 V^*为 0.8。图 2 为不同预留请求率的资源平均利用率。实验数据显示，当预留请求率较低时(5%和 10%)，CBFR、MALR、OVBR 和 REXR 4 种策略的资源利用率差别很小，且相对 CONR 策略而言大约能提高 13%；当预留请求率超过 15%时，这 4 种策略的性能表现出现明显分化，其中 OVBR 策略的性能下降最为显著，而 CBFR 策略和 MALR 的性能差异只在预留请求率到达 20%以上时才比较明显。为了分析各种策略之间的性能差异，统计了各个资源站点的平均资源率，如图 3 所示。

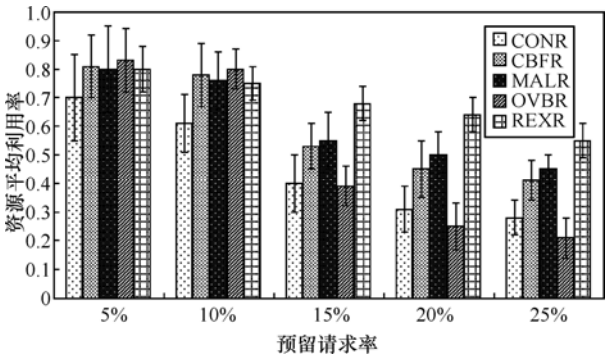


图 2　不同预留请求率下的资源平均利用率

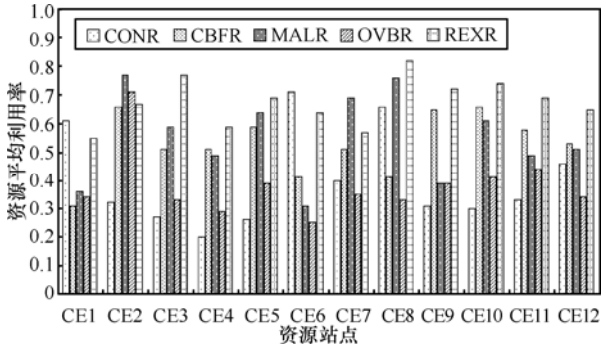


图 3　各个资源站点的平均利用率

CONR 策略偏向于将预留请求均衡地分配到各个站点，由于资源站点的静态性能和动态负载压力差异很大，因此 CONR 策略在面对较高的预留请求率时其性能表现显著恶化；CBFR 和 MALR 则偏向于向静态性能指标较高的站点预留资源，因而导致这些站点面临更高负载；OVBR 在实现层面上类似 CONR，由于实验设定了 20%的超额接纳率，在预留请求率不高时，20%的超额预留没有对系统造成任何负面影响，因此 OVBR 策略的表现是 5 种策略中最优的，但随着预留请求率的上升，OVBR 不仅不能提高资源利用率，反而频繁导致预留违约，从而使得有效资源利用率显著下降。REXR 策略的主要特点是随预留请求率的增大而相对保持平稳，例如当预留请求率从 5%增大到 25%时，REXR 只损失了约 24%的资源利用率。

图 4 为 5 种策略在拒绝率方面的性能比较。实验数据显示：当预留请求率较低时，5 种策略的拒绝率差别不大，原因是为整个网格系统处于低负载状态，此时预留机制的负面效果尚为突出；当预留请求率升高时，CBFR 的拒绝率增加最快，其次是 CONR 策略和 MALR 策略，而 OVBR 策略的拒绝率几乎和 REXR 相同。OVBR 策略和 REXR 策略的拒绝率相似的原因是两者都放宽了预留接纳的标准。为进一步分析这 2 种策略在不同参数下的性能表现，用不同的参数 V^*和 O^*重复本次实验，其中 V^*为 REXR 策略的松弛接纳阈值，而 O^*为 OVBR 的超额接纳率，实验结果如图 5 所示。

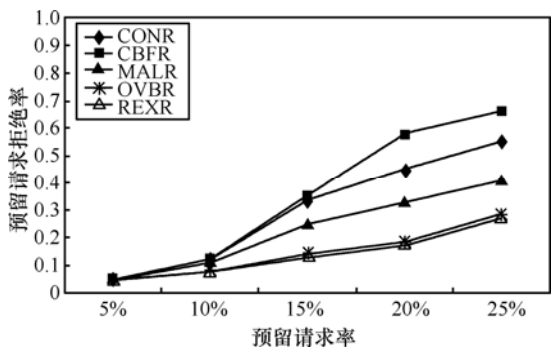


图 4　不同预留请求率下的预留请求拒绝率

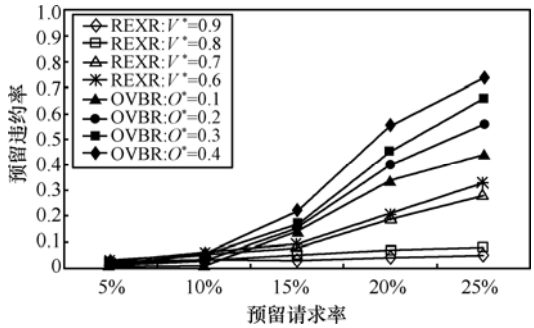


图 5　不同参数下的预留违约率比较

实验结果显示，OVBR 在较低的预留请求率时几乎没有任何预留违约，而当预留请求率达到并超过 15%时，其预留违约率显著增加。当 OVBR 策略采用 40%的超额预留率时，其违约率高达 74%。即使预留请求率为 20%，OVBR 的违约率也高达 55%相对而言，REXR 的 V^*参数也会对其违约率产生一定影响，具体表现为：V^*的值越小，违约率随之增加得越快，但总体幅度明显小于 OVBR，而且只要 V^*值不小于 0.8，其预留违约率就能控制在 7%以内。

REXR 算法的时间开销由进化策略的迭代参数 W 和任务规模决定，本次实验统计了 REXR 算法不同参数取值下的算法的平均执行时间，实验结果如表 1 所示。任务规模对算法执行效率的影响明显大于参数 W 的影响。当任务规模较小时，REXR 算法收敛速度很快，此时的 W 参数几乎不影响算法执行效率；当任务规模达到 200 以上时，进化迭代参数至少要达到 500 以上，算法才能获得有效收敛。

表 1　　不同参数下的算法执行时间/s

任务规模	W 参数				
	200	500	1 000	2 000	4 000
50~100	9.45	9.67	10.10	9.67	9.51
100~150	14.39	19.26	24.35	23.19	23.34
150~200	27.17	41.23	45.14	47.68	51.34
200~250	39.55	69.14	89.34	87.41	85.37

综合以上实验数据和分析，本次实验的主要结论是：①当预留请求率低于 10%时，CBFR 策略和 OVBR 策略能有效提高资源利用率；②当预留请求率大于 10%时，REXR 策略能显著降低过渡预留导

致的资源效率损失；面对较低的预留请求率，松弛接纳阈值 V^*可以设置得较低，但面对较高的预留请求率时，V^*取值不应过小。

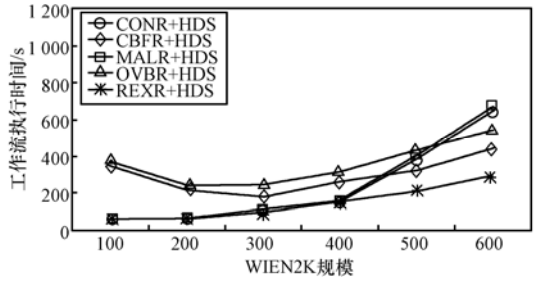
5.2　实际工作流分析

WIEN2K 工作流[19]包括 2 个并行分支，其问题规模由这 2 个并行分支决定，实验设定问题规模从 100 逐步增加到 600。实验过程中向网格系统注入了一定的背景工作负载，从而与 WIEN2K 共同竞争系统资源。为分析任务高估预留时间对其执行效率的影响，本次实验将 WIEN2K 中各个任务节点的预留时间高估了β倍，实验分 4 组进行，β值分别为 10%、20%、30%和 40%，实验结果如图 6 所示。

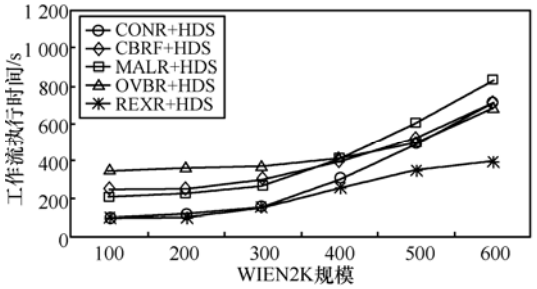
实验结果显示，当任务预留时间高估率为 10%且任务规模为 100~400 时，CONR、MALR 和 REXR 3 种策略对应的 WIEN2K 执行时间差别很小，但当任务规模达到 500 以上时，REXR 策略的性能优势就比较明显了，其对应的工作流执行时间大约只有前 2 种策略的 51%；随着任务高估率β值的增加，所有策略所对应的执行时间都显著增加，不过相对增幅出现了明显差异，其中 MALR 策略的相对增幅最明显，而 REXR 策略的执行时间随β值而增加的幅度则远远小于其他 4 种策略。

背景负载对 WIEN2K 执行效率的影响十分明显。当任务规模小于 400 时，OVBR 策略在本次实验中的性能表现是最差的，OVBR 策略表现不佳的主要原因是，背景负载在实验开始阶段的到达速度较快，系统在面对较高额外负载时，其资源可用量显著降低，而超额预留策略则更加减少了系统未来的可预留资源量，因此 WIEN2K 第 2 个并行分支中的许多子任务出现了执行延迟，从而影响任务整体的执行效率。但这种情况在β值较高时得到了一定程度的缓解，原因刚好是因为较高的β值意味着并行分支中的预留时间被高估，很多预留请求实际被提前释放，其效果等价于取消预留请求，从而令 OVBR 策略的性能表现得到一定程度的提升。

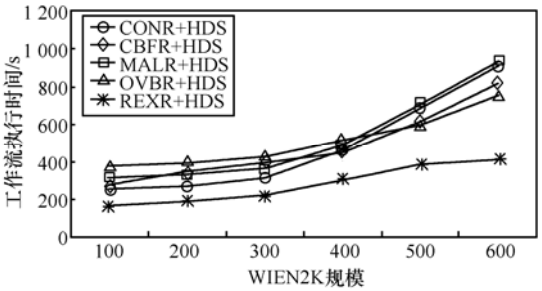
MALR 策略在任务规模超过 400 后，其性能表现明显低于其他策略，产生这种现象的原因与 WIEN2K 工作流的结构特征有关。因为 WIEN2K 的主要执行时间集中在 2 个大的并行分支部分，其并行子任务类似一组独立子任务。因此 WIEN2K 的最终执行时间是由这些子任务集合的最迟完成时间决定。而 MALR 策略的优越性主要体现在子任务差异度很大的情况下，此时 MALR 策略能依据子任务的计算量和资源需求量之间的线性关系来规划一个总体完成时间较优的预留方案，而本实验所分析的 WIEN2K 则不具备这种特许。因此 MALR 策略的平均执行时间大约是 REXR 策略的一倍左右。



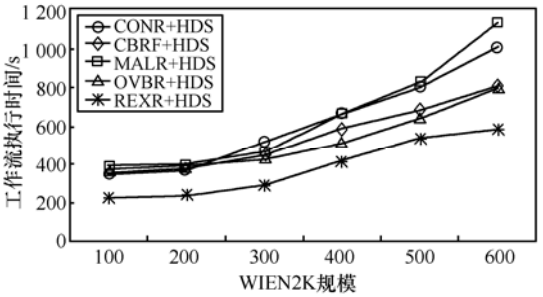
(a) β=10%



(b) β=20%



(c) β=30%



(d) β=40%

图 6　不同β值下 WIEN2K 执行时间对比

本次实验的主要结论是：①高估预留时间将对工作流执行效率产生较大负面影响，REXR 策略能够有效降低这种负面影响；②OVBR 和 CBFR 难以

适应突发负载增高的情况，当系统整体负载较重时，2 种策略对任务执行效率的负面影响显著增大；③MALR 的性能表现受工作流结构特征影响较大。

6　结束语

提出一种支持时空二维松弛的预留策略，用于解决预留机制对系统性能的负面影响，并采用进化算法与松弛预留策略相结合的方法设计了一个协同资源预留接纳算法。实验结果显示，松弛预留策略不仅能有效降低预留服务对系统性能的负面影响，而且在能显著降低由高估预留时间而导致电工作流执行效率损失。今后的主要工作是在松弛预留模型中引入资源失效情况下的分析和处理技术。

参考文献：

[1] FOSTER I, KESSELMAN C. The Grid 2 [M]. San Francisco: Morgan Kaufmann, 2004.

[2] SMITH W, FOSTER I, TAYLOR V. Scheduling with advanced reservations[A]. IPDPS'00[C]. Cancun, Mexico, 2000.127-132.

[3] CAO J W, ZIMMERMANN F. Queue scheduling and advance reservations with COSY[A]. IPDPS'04[C]. Santa Fe, New Mexico, USA, 2004.63-70.

[4] SULISTIO A, KIM K H, BUYYA R. Managing cancellations and no-shows of reservations with overbooking to increase resource revenue[A]. CCGRID'08[C]. Lyon, France, 2008.267-276.

[5] AIDA K, CASANOVA H. Scheduling mixed-parallel applications with advance reservations[J]. Cluster Computing, 2009, 12(2):205-220.

[6] CASTILLO C, ROUSKAS G N, HARFOUSH K. Efficient resource management using advance reservations for heterogeneous grids[A]. IPDPS'08[C]. Miami, Florida, USA, 2008.1-12.

[7] FAROOQ U, MAJUMDAR S, PARSONS E W. Impact of laxity on scheduling with advance reservations in Grids[A]. MASCOTS'05[C]. Atlanta, Georgia, USA, 2005.319-322.

[8] LI B, SHEN B. Slack-based advance reservation for grid jobs[A]. ICACTE'10[C]. Chengdu, China, 2010.418-423.

[9] SODAN A C, DOSHI C, BARSANTI L, *et al*. Gang scheduling and adaptive resource allocation to mitigate advance reservation impact[A]. CCGRID'06[C]. Singapore, 2006.649-653.

[10] BURCHARD L O. Analysis of data structures for admission control of advance reservation requests[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(3):413-424.

[11] DUN N, TAURA K, YONEZAWA A. Fine-grained profiling for data-intensive workflows[A]. CCGRID'10[C]. Melbourne, Australia, 2010.571-572.

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

[13] 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[J]. 计算机研究与发展, 2007,44(1): 20-28.
HU C M, HUAI J P, WO T Y. Flexible resource capacity reservation mechanism for service grid using slack time[J]. Journal of Computer Research and Development, 2007,44(1):20-28.

[14] JAWAD A, THOMAS E. A new resource mapping technique for grid workflows in advance reservation environments[A]. HPCS'10[C]. Normandy, France, 2010.63-70.

[15] 公茂果, 焦李成, 杨咚咚等. 进化多目标优化算法研究[J]. 软件学报, 2009, 20(2):271-289.
GONG M G, JIAO L C, YANG D D, *et al*. Evolutionary multi-objective optimization algorithms[J]. Journal of Software, 2009, 20(2): 271-289.

[16] MENG X H, ZHU Y A, WU X M. Improved dynamic programming algorithms for the 0-1 knapsack problem[A]. ICCSIT'11[C]. Chendu, China, 2011.19-22.

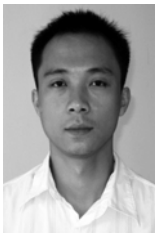
[17] SNEDECOR G W, COCHRAN W G. Statistical Methods[M]. Ames: Iowa State Press, 1980.

[18] BAL H, BHOEDJANG R R, HOFMAN R, *et al*. The distributed ASCI supercomputer project[J]. ACM Operating Systems Review, 2000, 34(4):76-96.

[19] TOPCUOGLU H, HARIRI S, WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(2): 260-274.

[20] WU L, WU C, CUI J, *et al*. An adaptive advance reservation mechanism for grid computing[A]. PDCAT'05[C]. Dalian, China, 2005.400-403.

作者简介：



肖鹏（1979-），男，湖南湘潭人，中南大学博士生，湖南工程学院计算机系讲师，主要研究方向为高性能网格计算和可信计算。

胡志刚（1963-），男，山西孝义人，博士后，中南大学信息科学与工程学院教授、博士生导师，主要研究方向为分布式系统和嵌入式系统。

阎朝坤（1978-），男，河南开封人，中南大学博士生，主要研究方向为网格计算和云计算。

屈喜龙（1978-），男，湖南新邵人，博士，湖南工程学院副教授，主要研究方向为服务计算和网络集成系统。