文章编号: 1001-0920(2008)04-0403-06

服务网格中基于请求生命期的资源预留方法和策略

梁 泉, 杨 扬, 王元卓
(北京科技大学 信息工程学院, 北京 100083)

摘 要: 基于面向服务的网格环境,针对网格服务实例所指向的资源,提出了资源预留树的预留方法. 利用在任务请求生命期内发现的所有合适资源构建资源预留树,当预留资源失效时,可直接在资源预留树内选取一个替代资源,保证资源预留的可靠性. 针对不同的用户服务质量,设计了 3 种不同的预留策略,在此基础上,提出了基于不同策略的资源预留树算法 TTL-RTA. 相关的性能分析和仿真实验表明,该算法具有较好的效率和可靠性.

关键词: 资源预留; 预留容错; 资源预留树
中图分类号: TP393 **文献标识码**: A

Methods and strategies of request time-to-live based on resource reservation in service-oriented grid environment

LIANG Quan, YANG Yang, WANG Yuan-zhuo
(School of Information Engineering, University of Science and Technology Beijing, Beijing 100083, China. Correspondent: LIANG Quan, E-mail: liangquan_lq@163.com)

Abstract: Based on the service-oriented grid environment, a method of resource reservation is proposed, which constructs a resource reservation tree(RRT). By using the resources found during the time restricted by time-to-livel TTL) of task request, RRT can be constructed. If the resource reserved before becomes invalid, the substitute can be found directly through RRT. The method can avoid a failure of resource reservation and guarantee its reliability. Furthermore, three strategies of reservation are designed according to different QoS requirements of users, based on which, an algorithm of TTL-RTA based on RRT is presented. Related performance analysis and simulation show the better efficiency and reliability of the algorithm.

Key words: Resource reservation; Reservation fault-tolerance; RRT

1 引 言

在网格计算环境里,资源预留和自适应仍然是进行服务质量(QoS)控制的两种基本技术[1],但存在比以往更大的困难. 首先,预留和分配不再局限于单个资源,而是潜在的复杂资源集合;其次,资源的类型更为广泛、异构,分属于不同的独立管理域,遵循不同的安全和使用策略,这使得预留的技术操作变得困难[2,3]. 除此之外,目前的预留技术和算法本身需要完善,预留有失败的可能. 预留时机、持续时间、资源失效等问题都会造成预留失败[3,4],而预留失败会影响任务完成,导致资源浪费,最终无法保障 QoS. 因此,必须重视预留容错,需要有更有效、更稳妥的方法来处理.

本文基于面向服务的网格系统环境,针对目前的预留方法中存在的资源失效问题,提出了基于任务请求生命期(TTL),建立资源预留树(RRT)的方法及其相应的预留策略,来避免资源预留失败,进而保障用户 QoS.

2 资源预留相关研究

文献[5,6]讨论了资源预留的类型,依据资源预留发起时间和持续时间分为静态预留和动态预留. 将立即开始的动态预留称为立即预留,将延迟开始的动态预留称为提前预留. 从系统的全局考虑,预留不局限于某一种类型. 文献[7]在处理不同的预留类型方面,采用了基于策略的资源分配方式,保持预留的灵活性,提高资源利用率. 但提前预留最具普遍性,应用范围也最广泛,因此本文将主要考虑提前预留.

收稿日期: 2006-12-28; **修回日期**: 2007-04-15.
基金项目: 国家自然科学基金重点项目(90412012,60673160).
作者简介: 梁泉(1972—),男,湖南洞口人,博士生,从事网格环境下的资源组织、管理与服务质量控制的研究;
杨扬(1958—),男,河北隆化人,教授,博士生导师,从事图像处理与模式识别、网格计算等研究.

在体系结构方面,计算机网络和实时/多媒体系统领域,有资源预留协议 RSVP 和采用 RSVP 的综合业务 IntServ 体系结构[1],能较好地实现网络资源预留. 在网格系统环境里,GARA[2]是第一个试图通过资源预留来保障网格 QoS 的体系结构,它向用户和应用程序提供了对各种资源进行预留操作的 API. G-QoSM[8-10]是在 OGSA 中支持 QoS 管理的系统,通过各模块的交互,实现 QoS 协商和资源预留的功能. 文献[11]给出了一个基于市场的非集中式计算资源预留框架,提出了结合服务价格协商的资源预留方法.

在预留的具体策略和方法方面,各领域都有不少适应自身特点的研究成果[4,12-15]. 文献[16]提出了一个在网格计算环境中基于 TTL 进行资源发现和预留的算法,能根据不同的策略自动预留资源,不需要用户干预. 文献[17]在分布式、自适应的网格体系结构中,基于网络节点依次转发用户请求,并同时在所辖的管理域内寻找匹配资源,直到找到符合要求的资源或 TTL 值变为 0 为止.

3 基于 TTL 的网格资源预留树

3.1 资源网格服务

在面向服务的网格环境里,所有的资源都可以封装成网格服务,并为之定义标准接口机制和统一的服务语义,这样可以在 VO 内基于更低级的资源组合进行资源管理,并允许跨异种平台对资源进行一致的访问. 此时,资源可描述成更为具体的、有质和量区分的网格服务.

定义 1 资源网格服务(RGS)对应某类型网格资源的描述,功能接口描述了访问机制,参数描述包括了资源量、QoS 需求等;资源提供者通过 RGS 发布自己的资源. 资源网格服务实例(RGSI)因用户请求而创建,是相应 RGS 的一个实例,对应 RGS 所描述的具体物理资源量的部分或全部.

在具体使用时,VO 关联 RGSI,RGSI 映射部分或全部物理资源. 这样,就把 RGS 与具体物理资源统一起来,遵循该资源相同的使用策略. 要特别强调的是,为描述方便,本文所指 RGS 都只关联单类型资源;所讨论的资源预留,都是对 RGSI 所映射物理资源的预留.

3.2 资源预留树

若用户请求提前预留某个 RGSI 所映射的物理资源,设为 s,在 s 被预留时刻起,到 s 开始被使用的时刻止,这段时间称为 s 的阻塞时间. s 在阻塞时间里状态有可能发生变化,若 s 失效,则用户必须重新启动资源发现过程,协商、预留其他的资源. 另一种情况是,当用户请求在首次进行资源发现的过程中,若把满足需求的多个资源记录下来,构造 RRT,当发生资源无效时,可立即选用其他节点的资源,避免任务中断,也不必重新启动资源发现和协商过程.

定义 2 RRT 是二叉查找树,每个节点对应一个 RGSI,用有限节点集合 T 表示,满足下列性质之一:

1) 集合为空,$T = \varnothing$;

2) 集合由一个根节点 r_0 及两棵不同的二叉查找树 T_L 和 T_R 组成,即 $T = \{r_0, T_L, T_R\}$,T_L 为 T 的左子树,T_R 为 T 的右子树. T 的节点集合 $r = \{r_i \mid i \in Z^+\}$,$\forall r_i \in r, r_i = (k_i, s_i)$,$s_i$ 为节点 r_i 关联的 RGSI,k_i 是由 s_i 某种属性对应节点 r_i 的关键值.

3.3 资源预留树构造算法

为了区分 RRT 树中节点的关键值 k_i,作如下定义.

定义 3 RRT 节点关系集 $R = A \cup B$,其中 $A = \langle \leqslant, < \rangle$,低于,更接近于,性能优于,时间早于〉,若关键值 k_i 和 k_j 的性质属于 A 中的某种关系,则记为 $k_i \prec k_j$;关系集合 $B = \langle > \rangle$,高于,更远离于,性能差于,时间晚于〉,若关键值 k_i 和 k_j 的性质属于 B 中的某种关系,则记为 $k_i \succ k_j$.

由定义 3,规定 RRT 中,$\forall k \in T_L : k \prec r_0$,$\forall k \in T_R : k \succ r_0$. 现在给出 RRT 构造算法如下:

算法 1 RRT 构造算法

CreateReserveTree(T, s, α, rl);

输入:s 是某个 RGSI;

α 为 s 某项属性的合理表示;

$rl \in R$;

结果:T

Step1: 参数准备,设置为算法接受的形式;

Step2: 若 $Tr = \varnothing$,则创建一个树节点 p,令 $pk = \alpha$,p 指向 s,转 Step6;

Step3: 访问节点 p,比较节点 p 关键值 k 和 α;

Step4: 若 $\alpha \prec k$,则在该节点左子树 T_L 中创建节点,即 CreateReserveTree(T_L, s, α, rl);

Step5: 若 $\alpha \succ k$,则在该节点右子树 T_R 中创建节点,即 CreateReserveTree(T_R, s, α, rl);

Step6: 返回 T 树,退出算法.

算法 CreateReserveTree(T, s, α, rl) 的构造受具体预留策略约束,预留策略确定 s 的某个属性特征,并赋给变量 α,同时预留策略预先选定关系集 R 中的某种关系 rl,通过判断树中已有节点的关键值 k 与 α 是否符合关系 rl,来确定创建 T 的左子树还是右子树. 通过变量 α 和关系 rl,预留策略可以在 RRT 的构造中施加影响.

4　基于 TTL 的网格资源预留策略和算法

4.1　预留登记

RRT 虽然为用户请求预备了多个网格资源,但不能独占.考虑到任务的发起要创建相应的 VO,可以在 VO 中为多个成员设置统一的预留列表,记录预留的资源信息,不妨记为 VOReservList.确定预留某个资源时,把它登记到 VOReservList 中,同时标记该资源为预留状态,当该资源失效时,可直接从 RRT 选取备用资源.

4.2　预留策略

在资源策略方面,目前主要是进行值或者值范围的匹配,更多地关注目标资源与用户需求的"匹配",对其他因素考虑不多[2,4,10].本文根据具体任务的特点,从系统总体性能的角度出发,不仅关注资源的"匹配",而且重视查找资源的效率,以提高预留的综合性能.为此在构造 RRT 时,根据不同需求,设计不同的预留策略(RSG).

(1) RSG1

FFFS(first found first server) 策略,系统在服务发现过程中,把第一个满足用户需求的 RGS 返回给请求,通知用户预留或立即使用其对应资源,该资源的性能可能正好与用户需求相同,或者远远高于用户需求.此时,根据发现资源的时间早晚来创建 RRT 树,该策略可描述如下:

前提:通过服务发现,得到满足用户需求的资源 s.

决策过程:

Step1: 以"发现时刻"作为 s 在 RRT 节点中的关键值,赋值给 α

Step2: 取 A 中"时间早于"元素,即适用比较操作"$\prec$".

Step3: 查询 VOReservList,

1) 若尚无对应预留资源登记,则把 s 登记到 VOReservList,置 s 为预留状态.

2) 若已有预留资源 x 登记,则比较 s 和 x 的 α 值:$s \prec x$,s 替换 x 登记到 VOReservList,置 s 为预留状态;$s \succ x$,不替换,置 s 为预留状态.

Step4: 根据 s,α,rl = "$\prec$"的已知条件,用 CreateReserveTree(T,s,α,rl) 创建 RT 树.

(2) RSG2

CUR(closest to users' requirement) 策略,最接近用户需求的策略.在发现的 RGS 中,把各属性值最接近用户需求的资源作为优先选择.譬如,若用户的需求为"CPU = 1.0 G",有两个匹配的资源 A 和 B,对应属性分别为 A:CPU = 1.2 G;B:CPU = 1.5 G,则把 A 作为优先选择,原因是 A 比 B 更接近用户的性能需求.此时根据"属性值"来创建 RRT 树,则只要取关系集 A 中"更接近于"元素,适用比较操作"$\prec$",就可以把 RSG1 改写为 RSG2.

(3) RSG3

HUR(higest to users' requirement) 策略,最高于用户需求的策略.与 CUR 预留策略不同的是,把属性值最高于用户需求的资源作为优先选择,以满足用户追求最好性能的需求.这时,只要把 RSG2 中的比较关系改为"性能优于",就可以得到 HUR 预留策略.

4.3　预留算法 TTL-RTA

用户请求提交以后,系统启动资源发现和协商过程,选择满足需求的 RGS.这是一个较为复杂的过程,RGS 数量巨大,找到最合适的资源并非易事;还要考虑系统的整体性能和用户忍耐的时间极限,资源发现和选择过程不能无休止地进行下去.为此,为每个用户请求设置 TTL,规定请求存续的时间.当 TTL 值为 0 时,如果仍没找到合适的资源,则该请求被终止或需重新提交,在请求重新提交后,设置新的 TTL 值.

至此,提出本文完整的资源预留算法,并命名为基于 TTL 的预留树算法(TTL-RTA).TTL-RTA 算法包含两个主要过程:RRT 构造和 RRT 使用.其中 RRT 构造包括:前向路径访问(FPA)和后向路径访问(BPA).FPA 基于 TTL 构造 RRT;BPA 对 RRT 进行处理,通过回溯,除登记到 VOReservList 上的资源之外,释放所有其他资源的预留,避免独占.具体算法描述如下:

算法 2　网格资源预留算法

TTL-RTA 算法包含两个主要过程:RRT 构造和 RRT 使用.

RRT 构造,前向路径访问 FPA:

Step1: 提交请求,系统创建相应的 VO,设定请求 TTL 值.

Step2: 本地搜索,若发现符合需求的资源,则转 Step4.

Step3: 全局搜索.

Step4: 针对返回的 RGS 列表,从预留策略集{RSG1,RSG2,RSG3}中选择某个策略构造 RRT.

Step5: TTL = TTL - 1,若 TTL = 0,则 FPA 完成,转到 Step6;TTL 不为 0 时,若是本地搜索,则转 Step2,否则转 Step3.

后向路径访问 BPA:

Step6: 检查刚刚构造的 RRT,若预留数目为 0,则此次预留失败,可重新提交请求,转 Step11.

Step7: 回溯 RRT,返回上一个节点.

Step8: 检查该节点所对应的资源是否被登记到 VOReservList 中,若是,则转 Step10.

Step9: 释放该资源的预留.

Step10: 若 RRT 回溯未完成,则转 Step7.

Step11: 退出此次 RRT 构造.

RRT 使用:

Step1: 若预留的资源被正常使用,则转 Step4.

Step2: 若预留的资源失效,则采用中序遍历 RRT,找到首个有效资源,立即使用;同时把失效的资源从 VOReservList 中删除.

Step3: 若 RRT 中找不到有效资源,则 RRT 失效,需要重新构造,转 Step4.

Step4: 释放资源预留,销毁 RRT.

5 性能分析和仿真

5.1 TTL-RTA 算法的复杂性分析

不管采用哪种预留策略,假定资源数为 n,递归定义的算法本身会有 $O(2^n)$ 的时间复杂度,结合循环创建过程则有 $O(n\times 2^n)$;在 BPA 阶段,可以近似认为时间复杂度为 $O(n)$. 因此,TTL-RTA 在构造 RRT 时,时间复杂度近似 $O(n+n\times 2^n)\approx O(n\times 2^n)$,$n$ 通常是非常有限的. 因为在实际预留时,预留的资源数目不会也无需太大,所以算法的时间复杂度不会影响性能.

当预留的资源失效时,TTL-RTA 需要从 RRT 搜索替代资源. 根据定义 2 和算法 1,若只考虑等概率情况,则一次成功地搜索所访问节点的平均数为 $\bar{d}+1$,其中 $\bar{d}$ 为目标在树中的深度,$\bar{d}$ 可求解. 设 RRT 节点数目为 n,$I(n)$ 表示当 $n>1$ 时,RRT 平均内部路径长度,则有

$$I(n)=\frac{1}{n}\sum_{i=1}^{n-1}(I(i)+I(n-i-1)+n-1)=\frac{2}{n}\sum_{i=1}^{n-1}I(i)+n-1,\quad n>1. \tag{1}$$

递归求解式(1),可得到

$$I(n)=\begin{cases}0, & n=1;\\ 1, & n=2;\\ ((n+1)I(n-1)+2n-2)/n, & n>2.\end{cases} \tag{2}$$

进一步求解式(2)得

$$I(n)\approx 2(n+1)/(\ln n+\gamma)-4n, \tag{3}$$

其中 $\gamma\approx 0.577\,215$ 为欧拉常数. 最终可以得到

$$\bar{d}=I(n)/n\approx 2\left(\frac{n+1}{n}\right)(\ln n+\gamma)-4=O(\log n). \tag{4}$$

式(4)表明,在有 n 个资源的 RRT 中,成功搜索一个替代资源,所耗费的时间比构建 RRT 要低,因为 $O(\log n)$ 与 $O(n\times 2^n)$ 相比,复杂度要低很多. 因此,若在构建 RRT 时不影响系统性能,那么在使用时更不会影响.

5.2 预留的失败概率分析

若不考虑所预留资源的阻塞时间,则资源可以看成是对状态而言的、独立同分布的离散型随机变量 X. 因为只关心预留的资源对用户请求而言是否可用,其状态只有"可用"或"失效"两种,故 $X\sim B(1,p)$,其概率分布为

$$P\{X=k\}=p^{1-k}(1-p)^k,\quad k=0,1, \tag{5}$$

其中 $0<p<1$ 是资源处于失效状态时的比率. 对多个资源则有

$$P\{X_i=k\mid i=1,2,\cdots,n\}=\prod_{i=1}^{n}P\{X_i=k\}=\prod_{i=1}^{n}[p^{1-k}(1-p)^k]=p^{n(1-k)}(1-p)^{nk},\quad k=0,1. \tag{6}$$

式(5)和(6)分别代表预留单个资源和多个资源时的资源状态概率分布,取 $k=0$ 表示失效状态,则资源失效概率分别为

$$P_1=P\{X=0\}=p,$$

$$P_2=P\{X_i=0\mid i=1,2,\cdots,n\}=p^n.$$

考虑到资源失效意味着预留的失败,因此 P_1 和 P_2 也分别代表了失败概率,易知 $P_2<P_1$.

若考虑资源的阻塞时间,由于在阻塞时间内,网格系统中非常微小的某种独立的随机因素都有可能导致资源失效,从理论上来说其概率接近于正态分布. 本文近似用标准正态分布来描述,此时,对单个资源而言,其失效概率分布为

$$P_1=F(t)=\frac{1}{\sqrt{2\pi}}\int_0^t e^{-\frac{z^2}{2}}\,dz,\quad 0<t<+\infty.$$

对多个资源意味着相应概率 $P_2\approx P_1^n$. 当资源数目大于 1 时,为 TTL-RTA 预留方法,其失败概率随阻塞时间而增长,但低于单个资源方法,且资源数目增加,失败概率显著减小.

5.3 仿真及结果分析

本文利用 GridSim[18] 作为仿真工具,它提供了丰富的函数库以支持模拟网格环境中的异构资源、用户、应用程序、用户代理和调度器,而且 GridSim 能够支持基于提前预留的资源管理和调度体系,用户可以调用相关库函数收集模拟的统计数据. 仿真实验模拟扁平的分布网格环境,只考虑计算资源类型,每个节点不少于 10 个资源,资源能力在(500

MHz,2 GHz) 之间. 请求的产生服从泊松分布,资源需求平均值为1 GHz,任务平均大小为1 GHz,请求的 TTL 值为(0.20).

为测试 TTL-RTA 的性能及因不同预留策略而呈现的差异,可使用如下两个目标参数:1) 系统接纳量,指能找到需求资源而被系统接纳的请求比率;2) 平均响应时间,指请求发出到执行结果返回的时间长度.

测试请求的 TTL 值对目标参数的影响. 图1为 TTL 值和系统接纳量. 图1表明,当 TTL 值较小时(< 8),系统接纳量随 TTL 值增加而激增,当 TTL 值足够大时(> 8)将趋于平缓,这是因为 TTL 值与寻找资源的时间长短有关;另一方面,在 TTL 值较小时,3种预留策略区别不大,而当 TTL 值足够大时,RSG1对应的系统接纳量要高于RSG2和RSG3,因为 RSG1对资源发现约束较小;RSG2和RSG3的系统接纳量几乎相同,因为要找到性能最接近或最高的资源,其运行复杂度是非常接近的. 图2为 TTL 值和平均响应时间. 图2表明,RSG2的响应时间最长,且随着 TTL 值的增大有变大的趋势,这是因为越接近需求的性能参数,资源的性能相对越差,执行时间越长;相反,RSG3响应时间最短,且随着 TTL 值增大有变小的趋势,因为有机会找到更好性能的资源,执行时间相应就短;RSG1的响应时间有波动,它所发现的资源性能具有随机性.

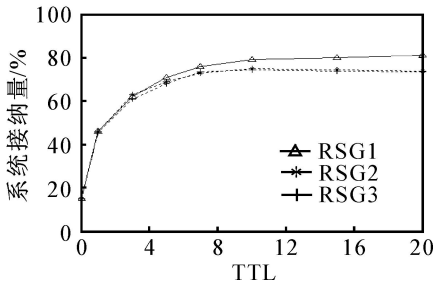

图1　TTL 值和系统接纳量

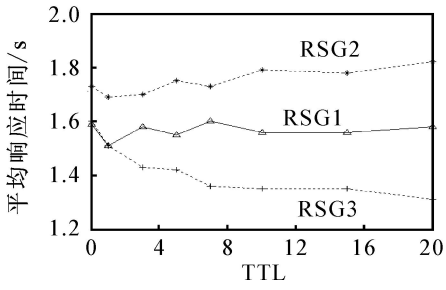

图2　TTL 值和平均响应时间

测试请求的发生次数对目标参数的影响. 设定 TTL = 6,在(0.20 s)时间内,请求发生服从泊松分布,并立即向系统提交,提取系统处理请求的结果数据. 图3为请求发生和系统接纳量. 图3显示,TTL-RTA 3种预留策略的系统接纳量非常接近,并且都随着请求发生的时间呈上升趋势. 这是因为请求发生的次数下降,使资源竞争减少,从而能获得更高的系统接纳量. 图4为请求发生和平均响应时间. 图4显示,在请求发生次数较高的时间里,RSG1,RSG2和RSG3的平均响应时间比较接近;RSG1有些波动,仍然表明找到匹配资源的随机性;RSG2在整个过程中,平均响应时间比较平稳,由于受其策略影响,相对最高;RSG3在请求发生次数增加阶段,平均响应时间呈下降趋势,次数较少时则变化不大,因为固定的 TTL 值限制了发现更高性能资源的机会.

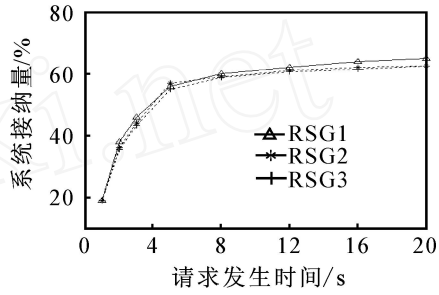

图3　请求发生和系统接纳量

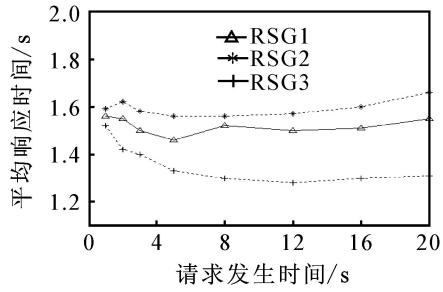

图4　请求发生和平均响应时间

6　结　　论

本文针对预留中存在的资源失效问题,提出构造资源预留树来增强预留的可靠性,防止预留失败. 根据不同的用户需求,构建资源预留树时采用不同的预留策略,进一步提高了预留的准确性和效率. 随后,对有关算法做了性能分析和仿真. 另外,资源预留的反馈控制可以有效地对预留进行实时调节,增强预留的适应性,这是未来值得研究的课题.

参考文献(References)

[1] 梁泉,杨扬,梁开健,等. 网格系统的服务质量(QoS)保障与控制综述[J]. 控制与决策,2007,22(2):121-126.
(Liang Quan, Yang Yang, Liang Kai-jian, et al. Guarantee and control of quanlity of service on grid system: A survey[J]. Control and Decision,2007, 22(2): 121-126.)

[2] Foster I, Roy A, Sander V. A quality of service architecture that combines resource reservation and application adaptation[C]. Proc of the 8th Int Workshop on Quality of Service. Los Alamitos: IEEE Computer Society Press, 2000: 181-188.

[3] Schröder J, Götzer M, Müller R. Resource management in next generation networks[J]. Int J of Electronics and Communications, 2006, 60(2): 116-124.

[4] Chakrabarti G, Kulkarni S. Load balancing and resource reservation in mobile[J]. Ad Hoc Networks, 2006, 4(2): 186-203.

[5] Wolf L C, Steinmetz R. Concepts for resource reservation in advance [J]. Multimedia Tools and Applications, 1997, 4(3): 255-278.

[6] MacLaren J. Advance reservation: State of art. GGF GRAAP-WG [EB/OL]. http://www.fz-juelich.de/zam/RD/coop/ggf/fraap/graap-wg.html, 2003-02/2006-08.

[7] Avery P, Cavanaugh R. Policy based scheduling for simple quality of service in grid computing[C]. Proc of the 18th Int Parallel and Distributed Processing Symposium. Los Alamitos: IEEE Computer Society Press, 2004: 315-324.

[8] Al-Ali R, Laszewski G, Amin K, et al. QoS support for high-performance scientific applications[C]. Proc of the IEEE/ACM 4th Int Symposium on Cluster Computing and the Grid. Los Alamitos: IEEE Computer Society Press, 2004: 134-143.

[9] Al-Ali R, Amin K, Laszewski G, et al. An OGSA-based quality of service framewok[C]. Proc of the 2th Int Workshop on Grid and Cooperative Computing. Berlin: Springer-Verlag, 2003: 225-233.

[10] Al-Ali R, Hafid A, Rana O, et al. An approach for QoS adaptation in service-oriented grids [J]. Concurrency and Computation: Practice and Experience J, 2004, 16(5): 401-412.

[11] Stiller B, Smirnow M, Karsten M, et al. From QoS provisioning to QoS charging[C]. Lecture Notes in Computer Science 2511. Berlin: Springer-Verlag, 2002: 303-314.

[12] Salamah M, Lababidi H. Dynamically adaptive channel reservation scheme for cellular networks[J]. Computer Networks, 2005, 49(6): 787-796.

[13] Kim S, Varshney P K. Adaptive online bandwidth allocation and reservation for QoS sensitive multimedia networks[J]. Computer Communications, 2005, 28(17): 1959-1969.

[14] Helali A, Soudani A, Nasri S, et al. An approach for end-to-end QoS and network resources management [J]. Computer Standards and Interfaces, 2005, 28(1): 93-108.

[15] Curti C, Ferrari T, Gommans L, et al. On advance reservation of heterogeneous network paths[J]. Future Generation Computer Systems, 2005, 21(4): 525-538.

[16] Sanya Tangpongprasit, Takahiro Katagiri, Kenji Kise, et al. A time-to-live based reservation algorithm on fully decentralized resource discovery in grid computing [J]. Parallel Computing, 2005, 31(6): 529-543.

[17] Iamnitchi A, Foster I. On fully decentralized resource discovery in grid environments[C]. Proc of 2th Int Workshop on Grid Computing. Berlin: Springer-Verlag, 2001: 51-62.

[18] Sulistio A, Yeo C S, Buyya R. Visual modeler for grid modelling and simulation (GridSim) toolkit, innovative grid computing workshop[C]. Proc of the 3rd Int Conf on Computational Science. Berlin: Springer-Verlag Publications, 2003: 1123-1132.

下 期 要 目