计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
44(1):20~28, 2007

一种基于松弛时间的服务网格资源能力预留机制

胡春明　怀进鹏　沃天宇
(北京航空航天大学计算机学院　北京　100083)
(hucm@act.buaa.edu.cn)

Flexible Resource Capacity Reservation Mechanism for Service Grid Using Slack Time

Hu Chunming, Huai Jinpeng, and Wo Tianyu
(*School of Computer Science and Engineering, Beihang University, Beijing* 100083)

Abstract　Resource reservation and allocation is commonly used to provide QoS guarantee for grid services. However, the reservation request with fixed parameters may introduce resource capacity fragment problem caused by the peaks of resource utilization. In this paper, a flexible capacity reservation mechanism is proposed, in which slack time is added into the reservation scenario to get more flexibility of resource reservation. Beside this, an algorithm is introduced to support the reservation and admission control with slack time. Four strategies are proposed according to the existing scheduling algorithms. The algorithm is implemented in CROWN service grid middleware. Evaluation result shows that better performance can be achieved by using slack-enabled algorithm, and the min-min-based strategy performs better.

Key words　distributed computing; service grid; resource reservation; resource capacity management; quality of service; admission control; slack time; CROWN

摘　要　通过资源能力预留为网格服务提供确定的 QoS 保证是实现服务网格 QoS 管理的基础和关键. 针对确定性资源能力预留的"资源能力碎片"问题,提出了一种支持松弛时间的灵活资源能力预留机制,并设计了支持松弛时间的资源预留请求接纳控制算法. 在 CROWN 的节点服务器上实现了该机制,并通过仿真实验进行性能评价. 结果表明,和已有的确定型预留机制相比,支持松弛时间的资源能力预留机制,使资源能力调度具有更多的自主性,可显著地提高网格资源的综合利用效率.

关键词　分布式计算;服务网格;资源预留;资源能力管理;服务质量;接纳控制;松弛时间;CROWN

中图法分类号　TP393

随着网格技术和应用的不断深入,网格核心功能和传统的非功能特性共同成为研究应用的焦点,一些关键领域或业务应用场景(如实时计算、可视化等)对网格提出了严格的服务质量要求,缺乏服务质量保证逐渐成为制约网格应用的瓶颈,并引起学术界和工业界共同关注. 在服务网格环境中,作业最终通过一组对特定类型服务的调用而得到执行,其服务执行过程中的性能等主要 QoS 参数将直接受到所分配资源能力数量的影响,可通过对资源能力进行建模,利用资源能力预留和分配,为服务的执行提供确定的 QoS 保证[1].

资源预留是指在实际使用资源之前进行必要的接纳测试和预留,使资源使用者在使用过程中获得所需的资源能力,并对资源的使用过程提供过载保

收稿日期:2006-07-04;修回日期:2006-08-29
基金项目:国家自然科学基金项目(90412011);国家杰出青年科学基金项目(60525209);国家"九七三"重点基础研究发展规划基金项目(2005CB321803);国家"八六三"高技术研究发展计划基金项目(2005AA119010)

护.我们可以将资源预留划分为确定性预留和灵活预留两类.对确定性预留,由于资源能力需求已经确定地对一个时间段提出了一定数量的资源能力要求,在分配资源时可选余地非常有限,容易产生大量的"资源能力碎片",无法有效地利用资源能力.针对这一问题,本文提出了一种新的灵活资源能力预留机制,它通过向资源能力请求加入松弛时间,提出支持松弛时间的资源能力请求接纳控制和资源能力分配算法,以提高资源能力的综合利用效率.

1 相关工作

在网格的服务质量管理方面,GARA[1]和G-QoSM[2]是两个重要的研究工作.

GARA(Globus advance reservation architecture)[1]是一个基于Globus、面向高端应用的端到端服务质量应用框架,它针对计算网格中作业的执行集成了对CPU计算能力与网络带宽两类资源的接纳控制和资源预留功能,支持对这两类资源的提前预留和立即预留,提高计算网格作业执行的服务质量.但GARA只支持确定性的预留.

G-QoSM[2]是一个面向服务网格QoS框架,它提供了一组OGSA/OGSI兼容的网格服务,通过对CPU计算能力与网络带宽等资源的预留保证网格服务的QoS,并通过扩展服务注册表UDDI提供QoS感知的服务发现能力.但在提供网格服务的QoS保证时,G-QoSM也只提供了对确定性预留的支持.

在灵活资源预留方面,文献[3]提出了一种针对网络带宽的预留机制(flexible advance reservation),允许资源预留请求的参数(特别是对资源能力的需求数量)在运行时依据资源状态进行调整,在一定程度上解决资源能力碎片问题,提高预留请求接纳率.但是,其提前预留机制要求所分配的资源能力在运行时动态调整,限制了灵活预留机制的应用范围,并可能导致额外的开销,在一定程度上使此类灵活预留机制变得不可行.

针对作业调度问题,已经出现了大量的研究工作,并取得了重要的结果,提出了许多有趣的作业调度算法,如OLB,Fast Greedy,Min-min,Max-min,Greedy等[4].除这些启发式的调度算法之外,还有一些研究工作将进化算法(如遗传算法、模拟退火算法、禁忌搜索等)应用到作业调度问题上,提出基于GA,SA,GSA和Tabu等算法的作业调度方案.文献[4]通过仿真实验比较了这些作业调度算法的性能,并认为Min-min算法可以在作业长度有限的情况下取得较好的调度性能.这些研究工作定义了一个较为理想的作业模型,对作业的开始时间没有确定的要求.而在网格应用环境中,作业的执行往往具有确定的底线,在底线后完成作业将变得没有意义.因此,这些已有的调度算法需要经过适当约束来满足网格环境中的作业调度需求.

此外,还有一些针对特定资源类型的能力分配与控制的研究工作,如支持CPU能力分配的DSRT[5]以及支持网络带宽预留和分配的其他工作[6-8].这些工作对具体资源的预留提供了实现机制,是本文工作开展的基础.

2 资源能力管理

2.1 资源能力及其度量

资源能力是对网格资源功能性指标的一种抽象,不同类型的资源可具有相互独立的多种资源能力.从共享方式上,资源能力可以分为共享型与互斥型两种.共享型资源能力是指一定量的资源能力可以共享给多个资源能力的消费者(如服务或应用程序),可分配的资源能力总量并不因分配给某个消费者而降低,如广播网络中的带宽资源、各类信息资源等;相应地,互斥型资源能力则指那些不可由多个资源能力消费者共享,一旦分配就导致能力总量下降的资源能力,如CPU计算力、网络带宽以及存储容量等,这类资源能力的多少直接对网格服务请求执行过程的性能影响很大,是我们进行网格服务QoS保证所重点关注的一类资源能力.

对于特定的资源而言,互斥型资源能力的总量是一定的,可以通过绝对或相对的形式进行度量,如CPU能力可以通过百分比、单位时间内执行浮点运算个数(以MFlops或TFlops为单位)进行定性的刻画.

以计算能力为例,当一台计算机上同时部署两个应用程序,且它们都被封装成网格服务时,其计算能力就需要在多个资源能力消费者之间进行共享.我们需要一种机制来定义每一资源能力消费者(应用程序或网格服务)对该资源的各类资源能力的需求,以明确地把握在某一特定时刻每种资源的互斥型资源能力的余量及是否可以满足资源能力消费者的需求.

图1给出了一个资源能力管理的基本框架,其中虚线以上部分是通用实现,而虚线以下则需要

依赖特定于资源及其能力的类型而定制实现. 资源能力管理的目的是在有限资源能力的前提下为资源能力需求合理地分配各类资源能力,其中,资源需求管理模块用于接受并管理资源需求,资源分配及其优化模块首先调用接纳控制模块进行接纳测试,如果通过测试,则计算当前的最优资源分配方案,并通过资源能力管理接口调用不同的资源能力适配器对各类资源能力进行分配.

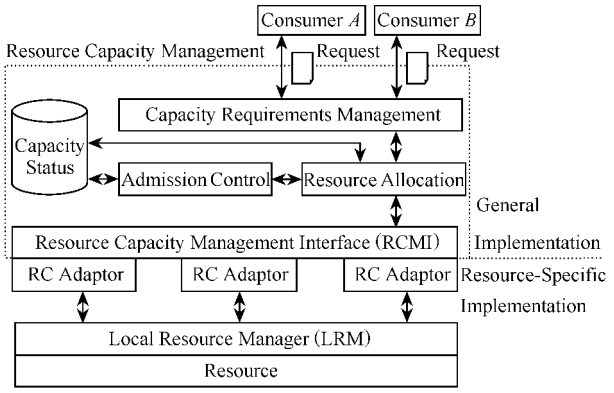

Fig. 1 A framework for resource capacity management.

图 1 资源能力管理的基本框架

2.2 基于时间槽的资源能力管理

目前,时间槽(time slot)是进行此类资源能力管理的一种有效手段[1,3,6,9],其基本思想是通过一个 x-y 坐标系($x\geqslant 0, y\geqslant 0$)表示资源能力. x 轴的有效范围为 $[t_0, +\infty)$,定义资源在某一时刻 $t\in[t_0, +\infty)$ 时,可用资源能力的最大值为一函数 $Capacity(t)$,则 $\int_0^{+\infty} Capacity(t)$ 在 x-y 坐标系中所覆盖的面积就代表所有的可用资源能力. 在网格应用环境中,最常见的 $Capacity(t)$ 可用一个常量函数表示:

$$Capacity(t) = C_{\mathrm{MAX}}, \tag{1}$$

或者用一个分段函数表示:

$$Capacity(t) = \begin{cases} C_{\mathrm{MAX}_1}, & t\in[t_0, t_1), \\ C_{\mathrm{MAX}_2}, & t\in[t_1, t_2), \\ \square & \\ C_{\mathrm{MAX}_n}, & t\in[t_{n-1}, t_n). \end{cases} \tag{2}$$

为了简化计算,我们可以将时间轴划分为若干个宽度为 Δt 的小区间,其中第 k 个区间的开始时间为 $t_0+(k-1)\times\Delta t$,结束时间为 $t_0+k\times\Delta t$. 我们用高度为 $\min(Capacity(t_0+(k-1)\times\Delta t), Capacity(t_0+k\times\Delta t))$ 的小矩形表示在时间区间 $[t_0+(k-1)\times\Delta t, t_0+k\times\Delta t)$ 之间的资源能力,称每一个矩形区域为时间槽. 同样地,时间区间 $[t_{\mathrm{start}}, t_{\mathrm{end}})$ 上的函数 $CapacityDemand(t)$ 表示一个资源能力需求,将其划分成若干时间槽,可直观地表示资源的可用能力和已分配的能力,并基于这些信息判断一个新的资源需求是否可以被满足.

有很多种数据结构可以用来表示时间槽. 最简单的方式是直接记录每个被接纳的资源能力请求,并将其储存为一个数组. 这样,当新的资源能力请求被接纳时,向数组该数组中增加一个记录,当已有的资源能力请求被释放时直接删除对应记录. 这种存储方法的缺点是资源分配时需遍历所有记录,并可能造成"资源能力碎片",影响资源能力分配的效率.

另一种存储方案是按照时间组织数据结构. 如果可用资源能力函数是一常数,在初始状态下,$ResourceCapacity$ 只包含一个 $TimeSlot$ 结构,其开始时间($startTime$)是当前时间 t_0,结束时间是 $+\infty$. 系统收到并接纳一个资源请求后,将 $[t_1, t_2)$ 上的资源量 C_1 分配出去,只要将原来的一个 $TimeSlot$ 结构改变为 3 个 $TimeSlot$ 结构,分别描述从当前时间 t_0 到 t_1,t_1 到 t_2 以及 t_2 到 $+\infty$ 的资源能力状态(如图 2 所示). 其中,$TimeSlot$ 1 和 $TimeSlot$ 3 的已分配能力 $allocatedCapacity=0$,而 $TimeSlot$ 2 的 $allocatedCapacity=C_1$. 此后,每接纳一个资源能力需求,就需要将该需求的开始和结束时间所对应的 $TimeSlot$ 分割成两个 $TimeSlot$,并修改所有相关 $TimeSlot$ 的已分配资源能力值. 如果可用资源能力函数是如式(2)所示的分段函数,则在初始状态下 $ResourceCapacity$ 包含若干个 $TimSlot$,每个 $TimeSlot$ 对应分段函数中的一个分段.

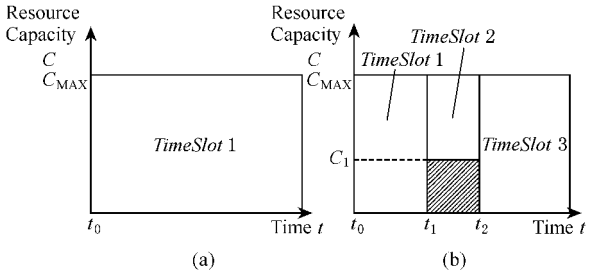

Fig. 2 Reservation using $TimeSlot$ objects. (a) Before reservation and (b) After reservation.

图 2 用 $TimeSlot$ 对象表示资源能力分配. (a) 分配前; (b) 分配后

上述时间槽可以表示一个资源的某一个类型的资源能力,而具有多种能力的资源可通过多个此类数据结构分别刻画. 如果资源能力的请求者同时需要对多种资源能力提出要求,则该请求仅在每一资源能力均得到满足时被接纳.

2.3 基本接纳控制算法

接纳控制是根据所剩余的资源能力决定是否可以接纳一个新的请求的机制,它可保证系统的负载被控制在一个合理的范围内,并保护已接纳的资源能力需求. 在尝试为某个资源能力需求分配资源能力前,首先需要进行接纳测试,检查是否有足够多的剩余资源能力可供分配.

基于上述分析,我们提出了一种对新资源能力请求的接纳控制和资源分配算法 BasicAllocation,其核心思想是不再调整已分配的资源能力需求,只根据新资源能力需求创建新的 *TimeSlot*,并按时间顺序依次比较新的资源能力请求所影响到的 *TimeSlot*,确认其是否有足够的剩余资源能力.

算法 1. BasicAllocation.

输入:

ResourceCapacityRequest *newReq*;

ResourceCapacity *resourceCapacity*.

输出: *Boolean*.

```
① {
② n = get_number_of_timeslots (resourceCapacity);
③ for(i = 1;i≤n;i ++) {
④   TimeSlot timeSlot = resourceCapacity.timeSlots[i];
⑤   if(((newReq.startTime≥timeSlot.startTime)
⑥     &&(newReq.startTime<timeSlot.endTime))
⑦     ‖((newReq.startTime≤timeSlot.startTime)
⑧     &&(newReq.endTime≥timeSlot.endTime))
⑨     ‖((newReq.startTime≥timeSlot.startTime)
⑩     &&(newReq.endTime≤timeSlot.endTime))
⑪     ‖((newReq.endTime>timeSlot.startTime)
⑫     &&(newReq.endTime≤timeSlot.endTime))) {
⑬       if(timeSlot.remainingCapacity<newReq.
⑭         requiredCapacity) return FALSE;
⑮       }
⑯   }
⑰   return TRUE;
⑱ }
```

在算法中,输入是记录某种资源能力的 *resourceCapacity* 对象以及新接收到的资源能力请求 *newReq*,输出是布尔型的接纳测试结果. 返回 TRUE 表示有足够多的剩余资源能力,可接纳该资源能力请求. 其中,第②行中 n 记录了当前所有 *TimeSlot* 块的总数. ③~⑯行循环检查每一个 *TimeSlot*,如果 *newReq* 涉及到某个 *TimeSlot*,则比较该 *TimeSlot* 的剩余能力是否可以满足 *newReq* 的要求(第⑬,⑭行).

3 支持松弛时间的资源能力预留与接纳控制

第 2 节中的确定性资源预留方式及基本接纳控制算法灵活性较差,这是因为资源能力需求明确对一个时间段提出资源能力要求,使得资源管理系统在进行资源能力的预留和分配时可选择余地非常有限,而且其资源分配容易产生许多"资源能力碎片",难以有效地利用资源能力. 本节提出了一种灵活资源能力分配机制,通过向资源能力请求中加入松弛时间,并设计支持松弛时间的资源能力,请求接纳控制和资源能力分配算法,提高资源能力的综合利用效率.

3.1 资源能力预留中的松弛时间

网格中许多作业的执行通常并不要求作业在尽可能短的时间内被执行完毕,只要在一个确定的时间点前得到执行即可. 作业在执行中可以有一定的松弛时间. 因此,我们修改资源能力需求定义,增加了松弛时间 *SlackTime*,而带有松弛时间的资源能力需求意味着对资源能力的需求可以从[*StartTime*, *StartTime* + *SlackTime*]这个区间内开始,并持续 *Duration* 所规定的时间. 这样,资源能力管理器在分配资源能力时就可以根据松弛时间合理的调整资源能力的分配,以便更好地利用资源. 资源预留请求可表示为

```
structure ResourceCapacityRequestWithSlack {
  ResourceCapacity capacity; /* 所描述的资源能力
  DateTime startTime; /* 该能力需求的开始时间 */
  TimeSpan duration; /* 该能力需求的持续时间 */
  TimeSpan slackTime; /* 该能力需求的松弛时间 */
  double requiredCapacity; /* 对资源能力的需求量 */
}
```

和传统的资源分配问题[10]相比,带松弛时间的资源能力预留和分配问题增加了开始时间和有限松弛时间两项约束. 对于带松弛时间的资源能力预留和分配问题而言,只有开始时间介于区间[T_{start}, $T_{start}+T_{slack}$]的分配方案才是可行解. 这样,可以将传统资源分配问题转化为带松弛时间的资源能力预留和分配问题在 $T_{slack}=+\infty$ 时的立即预留问题,在设计相关算法时,可与 Min-min[4,11-12],Max-min[4] 以及 Suffrage[13] 等传统资源分配中的启发式算法进行结合.

3.2 支持松弛时间的资源能力预留与接纳控制

我们将一个时间段内接收到的资源能力请求

分为3类:①尚未分配的资源能力请求,记为集合 U;②已分配但尚未开始的资源能力请求,记为集合 A;③分配且当前时间已超过所分配的开始时间(即已分配且已开始)的资源能力请求,它们在后续的资源分配中被锁定,记为集合 L. 当提出新的资源能力请求 *newReq* 时,在不影响 L 的前提下,如果能找到一个资源分配方案,使所有 A 与 L 中的任何一个资源能力请求都能够得到满足,则认为该新请求 *newReq* 可被接纳.

我们给出带松弛时间的接纳控制和资源能力分配算法 AdmissionControlWithSlack.

算法2. AdmissionControlWithSlack.

输入:

ResourceCapacityRequest *newReq*;
ResourceCapacityRequestSet U, A, L;
ResourceCapacity *resourceCapacity*.

输出:

Boolean *admissionResult*;
ResourceCapacity *scheduleSchema*.

```
① {
② scheduleSchema = new ResourceCapacity();
③ Rebuild scheduleSchema by L;
④ U = A ∪ {newReq}; A = ∅
⑤ While (U <> ∅) {
⑥   ResourceCapacityRequest currentReq =
    pickOut(U);
⑦   If (BasicAllocation (scheduleSchema,
    currentReq)) == FALSE) {
⑧     scheduleSchema = NULL;
⑨     admissionResult = FALSE; return;
⑩   } else {
⑪     U = U - {currentReq};
⑫     A = A ∪ {currentReq};
⑬   }
⑭ }
⑮ admissionResult = TRUE; return;
⑯ }
```

在进行接纳测试时,第③行根据已分配且已开始的资源能力请求集合 L 重新构造 *TimeSlot* 表;第④行将 *newReq* 以及所有在集合 A 中的资源能力请求都加入 U 中,并将 A 设置为空集;第⑥行按照一定的策略从 U 中选择一个请求 *currentReq*,第⑦~⑬行将其分配到 *scheduleSchema* 所记录的当前资源分配状态下并更新 *scheduleSchema*,如果不存在一个可能的分配方案,则返回 FALSE,表示新的资源能力请求无法被接纳. 如果找到了一个可能的分配方案,则将已分配的资源从 U 集合移到 A 集合中(第⑪,⑫行),并继续执行算法,直到 U 中的所有请求均得到分配.

基于已有的静态资源分配算法,在上述算法选择下一个待分配资源能力请求(第⑥行 *pickOut*(U))时可有许多不同的策略. 下面我们选择和比较5种优先选择策略:FIFO策略、Min-Slack策略、Min-min-based策略、Max-min-based策略和 Suffrage-based策略.

1) FIFO策略. 这是最简单的请求选取策略. 每次从 U 中选择到来最早的请求作为下一个资源能力分配的对象,不考虑资源能力请求的其他属性.

2) Min-Slack策略. 选择 U 中松弛时间最小的请求作为下一个资源能力分配的对象.

3) Min-min-based策略. 采用类似 Min-min 算法的思想,依次尝试在当前情况下将资源能力分配给 U 中每个未分配的资源请求,并选择分配后资源能力最早得到释放的请求作为下一个资源能力分配的对象.

4) Max-min-based策略. 采用类似 Max-min 算法的思想,依次尝试在当前情况下将资源能力分配给 U 中每个未分配的资源请求,并选择分配后资源能力最晚得到释放的请求作为下一个资源能力分配的对象.

5) Suffrage策略. 采用类似 Suffrage 算法的思想,依次尝试在当前情况下将资源能力分配给 U 中未分配的资源请求,记录资源释放的时间 t_1;然后依次尝试先分配其他请求,再分配该请求的资源释放时间 t_2(一般情况下 $t_2 \geqslant t_1$),并将 $t_2 - t_1$ 作为该资源请求的 Suffrage 值. 每次选择 Suffrage 值最大的资源能力请求作为下一个资源能力分配的对象.

3.3 在CROWN节点服务器中的实现

服务网格系统 CROWN 是由北京航空航天大学研制的一组服务网格中间件系统,它支持面向服务的网格应用开发和运行,包括节点服务器(node server)、资源定位与描述服务(resource location and description service, RLDS)、网格开发工具等11类基本构造模块,其中,节点服务器是网格服务的基本运行环境,对底层异构资源进行服务化封装.

在实现上,CROWN 节点服务器基于 GT 4.0 进行了功能扩展,其基本结构是一种可扩展的链式结构,是目前网格服务容器实现的主要结构. 其基

本思想是将每个服务请求的处理过程看做一个请求处理链、服务处理过程和响应处理链,并将请求链划分为传输协议的处理、全局处理和特定于服务的处理3个阶段. 所有对服务请求和响应的预处理工作按照功能和所处的阶段封装为若干个链节点,链节点之间的数据共享通过访问一个称为"服务上下文(service context)"的数据结构来实现.

为了实现带松弛时间的资源能力预留机制,我们在CROWN节点服务器的链式结构中扩展了服务预约链节点(*ServiceReservationHandler*)和SLA管理链节点(*SLAManagementHandler*),其中,服务预约链节点主要用来截获并处理服务预留请求,充当资源能力消费者的角色,根据用户选择的服务级以及服务配置创建资源能力需求并进行接纳测试,并将测试结果放入服务上下文. 资源封装服务可通过将其配置为特定于服务的链节点获得相应的资源预留能力.

4 性能评价

4.1 请求选择策略的参数和评价指标

通过仿真,我们对不同请求选择策略进行分析评价:首先根据指定的参数产生一组资源能力请求序列作为测试数据集合,并将该序列作为接纳控制算法的输入;接纳控制算法每次从序列中按时间顺序取出最早的请求进行接纳测试和资源能力分配,直到请求序列中的每个请求均被处理完毕为止.

资源能力请求序列的产生受到以下几个参数的影响:

1) 平均请求间隔$\overline{T_{\text{interval}}}$与请求速率$\mu$:一般认为,资源能力请求到来是一个柏松分布的离散过程,这样任两个连续请求的到达时间间隔即服从负指数分布. 平均请求间隔$\overline{T_{\text{interval}}}$是指该时间间隔的均值. 我们定义平均请求间隔$\overline{T_{\text{interval}}}$的倒数为请求速率,即$\mu = 1/\overline{T_{\text{interval}}}$.

2) 系统负载δ:指单位时间到来的所有资源能力请求对资源的需求总量之和与单位时间内可提供的资源能力总量的比值. 其计算公式如下:

$$\delta = \frac{\sum_{i=1}^{n} T_{\text{duration},i} \times C_i}{\int_{t_0}^{t_0+\Delta t} Capacity(t)}, \tag{2}$$

其中,n为在时间段$[t_0, t_0+\Delta t]$中到来的资源能力请求个数;C_i表示n个请求中第i个请求的资源能力需求量;T_{duration,i_i}表示第i个请求的资源能力需求持续时间;t_0表示n个请求的最早开始时间;$t_0+\Delta t$表示n个请求的最晚结束时间;δ的取值范围为$[0.0, 1.0]$,可通过调节平均请求间隔T_{interval}和平均资源能力需求数量$\overline{C}$来调节.

3) 平均持续时间$\overline{T_{\text{duration}}}$:指所产生的一组资源能力请求的平均持续时间均值.

4) 平均资源能力需求数量$\overline{C}$:指所产生的一组资源能力请求在单位时间的平均资源需求量.

5) 松弛因子λ:在产生资源能力请求时,每个请求带有一定程度的松弛时间. 我们将松弛时间同资源能力请求的持续时间T_{duration}的比值定义为松弛因子,即

$$\lambda = \frac{\overline{T_{\text{slack}}}}{T_{\text{duration}}}. \tag{3}$$

6) 立即预留所占的比例$P_{\text{immediate}}$:指请求中立即预留的比例. $P_{\text{immediate}} + P_{\text{advancee}} = 100\%$.

我们选择接纳率R_A和有效资源利用率U_E参数来评价上述4种不同的请求选择策略以及基于这些策略的接纳控制算法.

7) 接纳率R_A:指在一个确定的时间段内到达的所有请求中被接纳的请求个数与总个数的比值:

$$R_A = \frac{N_{\text{accept}}}{n}. \tag{4}$$

8) 有效资源利用率U_E:指所有被接纳请求的资源能力需求总量与所有到达请求的资源能力需求总量的比值:

$$U_E = \frac{\sum_{i=1}^{n} AdmissionTest(i) \times C_i \times T_{\text{duration},i}}{\sum_{i=1}^{n} C_i \times T_{\text{duration},i}}, \tag{5}$$

其中,$AdmissionTest(i)$是接纳测试函数:

$$AdmissionTest(i) = \begin{cases} 1, & 1 \leqslant i \leqslant n \text{ 且第 } i \text{ 个请求被接纳} \\ 0, & 1 \leqslant i \leqslant n \text{ 且第 } i \text{ 个请求被拒绝} \end{cases}. \tag{6}$$

有效满足率是相对于绝对资源利用率U而言的,U是所有被接纳的请求的资源能力需求总量与该时间段内可用资源能力总量的比值. 由于U受到系统负载的直接影响,$U \leqslant \delta$,因此不能直观地区分接纳控制算法的效果.

$$U = \frac{\sum_{i=1}^{n} AdmissionTest(i) \times C_i \times T_{\text{duration},i}}{\int_{t_0}^{t_0+\Delta t} Capacity(t)}. \tag{7}$$

4.2 结果分析

下面,我们通过仿真评价上述 4 种不同策略的资源分配效果,主要方法是通过 4 组实验,分别对系统负载、资源能力需求量、松弛时间及立即预留请求比例在采用不同策略时的接纳控制算法效果进行评价. 仿真中,资源能力函数 $Capacity(t)=C_{MAX}=1.0$.

第 1 组实验是为了考察不同的请求选取策略在不同系统负载下的效果. 我们设置立即预留比例 $P_{immediate}=0.2$,松弛因子 λ 在区间[2.0,2.5]均匀分布,平均持续时间 $\overline{T_{duration}}=25$,平均资源能力需求数量 $\overline{C}$ 在区间[0.2,0.8]. 通过调节平均请求间隔让系统负载 δ 在区间[0.1,1.5]区间变动,并比较采用不同策略的接纳控制算法的效果. 结果如图 3 和图 4 所示. 显然,随着系统负载的上升,请求的接纳率和有效资源利用率均呈下降趋势. 在系统满载($\delta=1.0$)时,可以获得 55% ~ 70% 的接纳率,这主要是因为请求在时间上分布不均造成的. 在系统轻载时,各种请求选取策略差别不显著,随着系统负载的上升,不同请求选取策略的接纳率和有效资源利用率表现出差异性,Min-min-based 请求选取策略可以持续获得较高的接纳率和有效资源利用率.

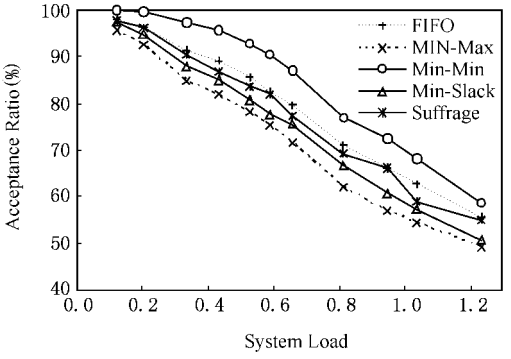

Fig. 3 System load vs. R_A.

图 3 系统负载对 R_A 的影响

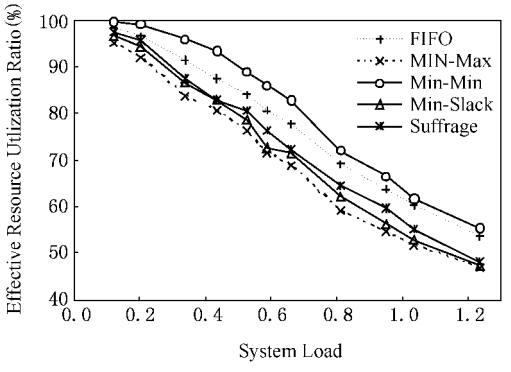

Fig. 4 System load vs. U_E.

图 4 系统负载对 U_E 的影响

第 2 组实验主要考察在资源能力预留中引入松弛时间后对接纳控制和资源分配效率的影响. 首先,我们设置立即预留比例 $P_{immediate}=0.2$,$\overline{T_{duration}}=25$,平均资源能力需求数量 $\overline{C}$ 在区间[0.2,0.8]内平均分布,通过调节平均请求间隔 $\overline{T_{int\ erval}}$ 让系统负载稳定在 50% ±0.5% 的范围内,并让松弛因子 λ 在区间[0.0,3.0]之间变化,考察松弛因子 λ 对接纳控制算法效果的影响(如图 5 和图 6 所示).

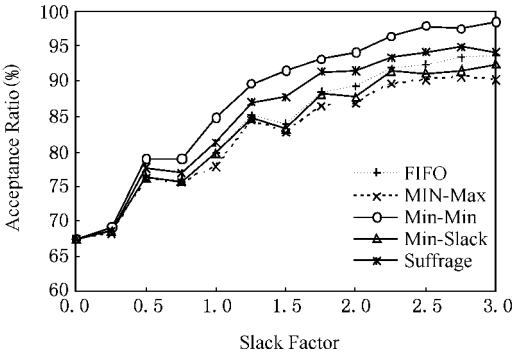

Fig. 5 Slack ratio vs. R_A.

图 5 松弛因子 λ 对 R_A 的影响

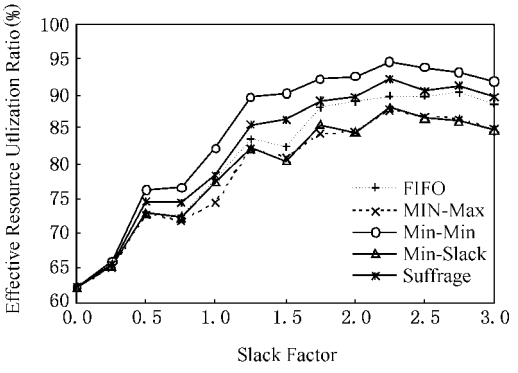

Fig. 6 Slack ratio vs. U_E.

图 6 松弛因子 λ 对 U_E 的影响

结果显示,增加松弛因子 λ 可以有效地改善资源能力预留时的接纳率和资源使用效率,但当 $\lambda>2.0$ 后,提高松弛因子对增加接纳率和有效资源利用率的改进效果不很明显. 当 λ 值比较小时,不同请求选取策略的差别很小,但随 λ 的增大,请求选取策略之间的差别开始显现. Min-min-based 的请求选取策略在最好情况下比其他请求选取策略提高 4% ~ 7% 的接纳率和 4% ~ 11% 的有效资源利用率.

另一组实验选择 Min-min-based 的策略为代表,分别考察不带松弛时间($\lambda=0$)和带松弛时间($\lambda=2.25$)时接纳控制算法的效果随系统负载变化的效果并进行对比分析,如图 7 和图 8 所示. 图 9 直

观地显示了引入松弛时间对接纳率和资源利用效率带来的优化率. 上述结果表明,引入松弛时间后,资源能力调度具有更多的自主性,从而使资源能力的使用更加有效,且随着系统负载的上升优化效果越明显. 例如,在系统负载达到 1.25 时接纳率的优化率达到 59.3%,有效资源利用率的优化率达到 52.4%.

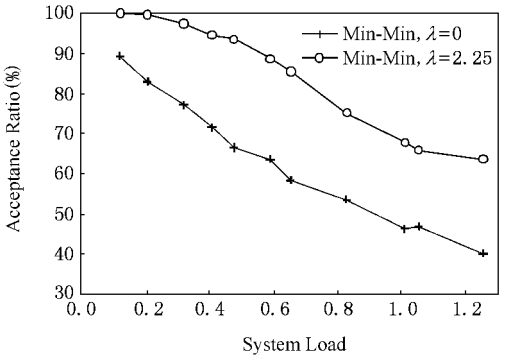

Fig. 7 Benefit of slack time: R_A.

图 7 松弛时间对 R_A 的优化

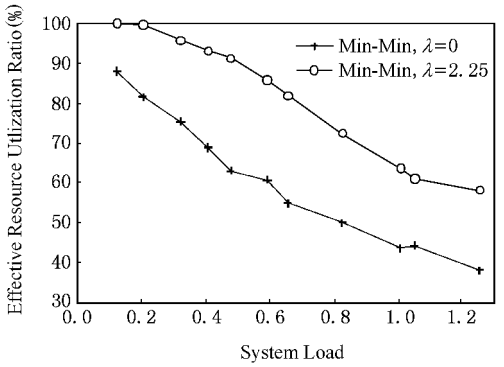

Fig. 8 Benefit of slack time: U_E.

图 8 松弛时间对 U_E 的优化

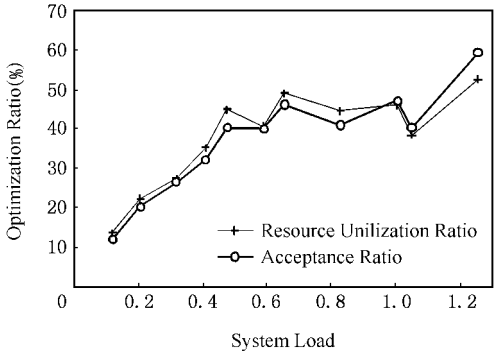

Fig. 9 The optimization ratio.

图 9 松弛时间对算法的影响

最后一组实验比较算法对立即预留和提前预留这两种情况的适应情况. 我们设置 $\overline{T_{\text{duration}}}=25$,平均资源能力需求数量 $\bar{C}$ 在区间[0.2,0.8]内平均分布,通过调节平均请求间隔让系统负载 δ 稳定在 50%±0.5% 的范围内,松弛因子 λ 在区间[2.0,2.5]均匀分布,并让立即预留所占的比例 $P_{\text{immediate}}$ 在区间[0.0,1.0]中变化. 实验结果如图 10 和图 11 所示. 结果表明,在提前预留为主的应用环境中,Min-min 算法具有较好的接纳率和有效资源利用率,随着立即预留比例的提高,不同请求选取策略之间的差别逐步缩小,接纳率和有效资源利用率略有上升. 其主要原因是相对于立即预留,提前预留增加了 T_{start} 的约束条件而使资源能力分配受到一定限制,立即预留比例的提高会适当增加灵活性.

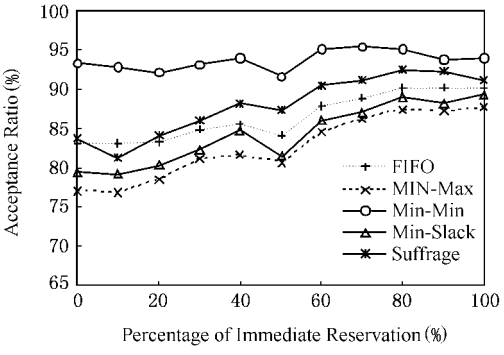

Fig. 10 $P_{\text{immediate}}$ vs. R_A.

图 10 立即预留对 R_A 的影响

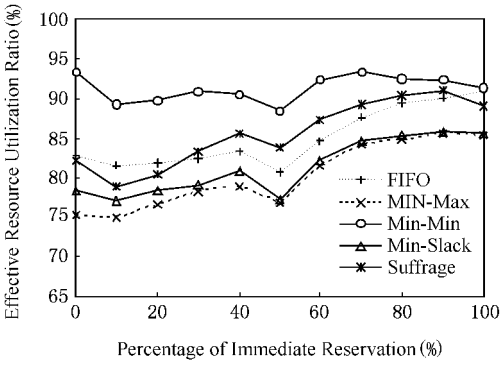

Fig. 11 $P_{\text{immediate}}$ vs. U_E.

图 11 立即预留对 U_E 的影响

通过上面的一组实验,我们可以得到以下结论:①引入松弛时间后,资源能力调度具有更多的自主性,使资源能力的使用更加有效,且随着系统负载的上升优化效果越明显;②适当增加松弛因子 λ 可以有效地改善资源能力预留时的接纳率和资源使用效率;③无论采用哪种请求选取策略,随着系统负载的上升,请求的接纳率和有效资源利用率均呈下降趋势,但相对其他请求选取策略中,Min-min-based 的策略可以使接纳控制和资源分配算法获得相对好的效果;④适当提高立即预留比例可使接纳率和有效

资源利用率略有上升. 上述实验结果对于在特定应用环境中适当选取算法参数具有直接的指导作用.

5 结束语

本文研究了如何在服务网格环境下通过资源能力的预留和分配有效地利用网格资源,并为网格服务的执行提供 QoS 保证. 针对已有资源预留机制存在的'资源能力碎片'问题,本文提出了一种支持松弛时间的资源能力预留机制,给出了支持松弛时间的资源预留请求接纳控制算法. 仿真实验结果表明,和已有的确定型预留机制相比,支持松弛时间的资源能力预留机制使资源能力调度具有更多的自主性,可显著地提高网格资源的综合利用效率.

参 考 文 献

[1] I Foster, C Kesselman, C Lee, *et al*. A distributed resource management architecture that supports advance reservations and co-allocation [C]. The Int'l Workshop on Quality of Service (IWQoS '99), London, UK, 1999

[2] R J Al-Ali, O Rana, D Walker, *et al*. G-QoSM: Grid service discovery using QoS properties [J]. Computing and Informatics Journal, 2002, 21(4): 363-382

[3] J B Xing, C Wu, M L Tao, *et al*. Flexible advance reservation for grid computing [C]. The 2nd Int'l Workshop on Grid and Cooperative Computing (GCC 2003), Shanghai, 2003

[4] T D Braun, H J Siegel, N Beck, *et al*. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems [C]. The 8th IEEE Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, 1999

[5] K Nahrstedt, H Chu, S Narayan. QoS-aware resource management for distributed multimedia applications [J]. Journal on High-Speed Networking, 1998, 7(3): 227-255

[6] L O Burchard. On the performance of computer networks with advanced reservation mechanisms [C]. The 11th IEEE Int'l Conf on Network (ICON '03), Sydney, Australia, 2003

[7] G Hoo, W Johnston, I Foster, *et al*. QoS as middleware: Bandwidth reservation system design [C]. The 8th IEEE Symp on High Performance Distributed Computing, Redondo Beach, CA, USA, 1999

[8] P Nanda, A Simmonds. Providing end-to-end guaranteed quality of service over the internet: A survey on bandwidth broker architecture for differentiated service network [C]. The 4th Int'l Conf on IT (CIT '01), Berhampur, India, 2001

[9] D Ferrari, A Gupta, G Ventre. Distributed advanced reservation of real-time connections [J]. Journal on Multimedia Systems, 1997, 5(3): 187-198

[10] N Katoh, T Ibaraki. Resource Allocation Problems: Handbook of Combinatorial Optimization [G]. In: Z Du, P M Pardalos, eds. Boston, MA: Kluwer Academic Publishers, 1998. 159-260

[11] X S He, X H Sun, G Laszewski. QoS guided min-min heuristic for grid task scheduling [J]. Journal of Computer Science and Technology, 2003, 18(4): 442-451

[12] X H Sun, W Ming. GHS: A performance prediction and task scheduling system for grid computing [C]. The 2003 IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France. 2003

[13] H Casanova, A Legrand, D Zagorodnov, *et al*. Heuristics for scheduling parameter sweep applications in grid environment [C]. The 9th Heterogeneous Computing Workshop (HCW '2000), Cancun, Mexico. 2000

Hu Chunming, born in 1977. Received his Ph. D. degree in computer science in Beihang University, Beijing, China. His current research interests are distributed computing, middleware and grid computing.
胡春明,1977 年生,博士,主要研究方向为分布式计算、中间件和网格技术.

Huai Jinpeng, born in 1962. Professor and Ph. D. supervisor of Beihang University. Senior member of China Computer Federation. His main research interests are computer software and theory, network information security and grid computing.
怀进鹏,1962 年生,教授,博士生导师,中国计算机学会高级会员,主要研究方向为计算机软件与理论、网络信息安全、网格技术.

Wo Tianyu, born in 1978. Ph. D. candidate in computer science of Beihang University. His main research interests are distributed computing and grid computing.
沃天宇,1978 年生,博士研究生,主要研究方向为分布式计算、网格技术.

Research Background

Resource resesvation and allocation is commonly used to provide QoS guarantee for grid services. However, the reservation request with fixed parameters may introduce resource capacity fragment problem caused by the peaks of resource utilization. In this paper, a flexible capacity reservation mechanism is proposed, in which slack time is added into the reservation scenario to get more flexibility of resource reservation. Beside this, an algorithm is introduced to support the reservation and admission control with slack time. Four strategies are proposed according to the existing scheduling algorithms. We implemented the algorithm in CROWN service grid middleware. Evaluation result shows that better performance can be achieved by using slack-enabled algorithm, and the min-min-based strategy performs better.