ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.1, January 2009, pp.152−163
doi: 10.3724/SP.J.1001.2009.03167

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

一种自适应的网格计算资源组织与发现机制*

孙海龙+, 怀进鹏, 富公为

(北京航空航天大学 计算机科学与工程学院,北京 100191)

Self-Adaptive Mechanism for Computational Resource Organization and Discovery in Grids

SUN Hai-Long+, HUAI Jin-Peng, FU Gong-Wei

(School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

+ Corresponding author: E-mail: sunhl@act.buaa.edu.cn

Sun HL, Huai JP, Fu GW. Self-Adaptive mechanism for computational resource organization and discovery in grids. *Journal of Software*, 2009,20(1):152−163. http://www.jos.org.cn/1000-9825/3167.htm

Abstract: Resource discovery is one of the important research issues in grid computing. As computational resources are fundamental resources supporting various grid applications, it is of great importance to study computational resource organization and discovery. However, existing approaches are limited in terms of efficiency, scalability, adaptability and the ability to support a wide range of query mechanisms. After analyzing the requirement of application resources, this paper introduces the primary attribute (PA), and describes a computational resource. Then it proposes resource category tree (RCT) to organize and discover the computational resources. This paper presents the mechanism of organizing resources based on RCT, including the basic concepts and rationale, the self-organizing design to support resource dynamics, the load-aware self adaptation and primary-backup based fault tolerance. It also designs four searching algorithms to support comprehensive query types in the framework of resource organization by using RCT, and a theoretical complexity analysis is presented as well. In the end, the paper evaluates the performance of RCT through simulations.

Key words: resource discovery; resource category; load-aware; self-organizing; self-adaptive; CROWN (China research and development over wide-area network)

摘 要: 资源发现是网格计算中一个重要的研究问题.计算资源作为支撑网格应用的基础资源,其组织与发现机制尤为重要,但现有的技术和方法在效率、可伸缩性、自适应的动态演化以及对查询方式的支持方面仍有较大的局限性.基于网格应用对计算资源需求特征的深入分析,通过引入计算资源的主属性概念,按照平衡二叉排序树对计算资源进行分类组织,提出基于资源分类树(resource category tree,简称 RCT)的资源组织与发现机制.首先,讨论了基于 RCT 对计算资源的组织机制,包括 RCT 的基本概念和原理、支持资源动态加入和退出以及资源状态动态变化的自组织机制、负载感知的自适应演化机制和基于备份节点的容错机制;然后,在基于 RCT 的资源组织结构下,设计了支持 4 种查询方式的搜索算法,并对算法的复杂度进行了分析;最后,通过多组仿真实验对 RCT 的性能进行了评估.

* Supported by the National Natural Science Foundation of China under Grant No.90412011 (国家自然科学基金); the National Natural Science Funds for Distinguished Young Scholar of China under Grant No.60525209 (国家杰出青年科学基金); the National Basic Research Program of China under Grant No.2005CB321803 (国家重点基础研究发展计划(973))

Received 2007-01-19; Accepted 2007-09-13

关键词: 资源发现;资源分类;负载感知;自组织;自适应;CROWN
中图法分类号: TP393 文献标识码: A

网格计算[1]的主要目标是建立基于网络(如互联网)的跨自治域资源共享和协同问题求解环境.通常,人们把网格计算环境中涉及到的计算机、存储、网络、仪器设备、数据和软件等统称为网格资源,其中,将支撑各类网格应用运行的硬件资源(如集群、存储、超级计算机和 PC 机等)称为计算资源.网格计算资源在传统的计算网格中是最基本的一类资源,所有的作业都需要调度到计算资源上进行处理.此外,即使在基于开放网格服务体系结构 OGSA(open grid service architecture)的服务网格中,计算资源也是支撑网格系统与应用服务运行的关键资源,构成了整个网格运行的最基础支撑环境.通过研究分析服务网格[2]中计算资源和网格服务的特点,在我们以前的研究工作[3,4]中提出了将两者分离设计的技术方法,实现了可信的服务热部署、服务发现和服务到资源的动态映射等技术,并提出将计算资源作为网格中一类重要的资源,研究其高效的发现、调度分配与共享协同机制,本文重点研究计算资源的组织与发现问题.

网格资源的发现面临很多技术挑战,近年来,人们提出网格信息服务(grid information service,简称 GIS)[5]专门用于解决网格资源发现的问题,出现了一些具有代表性的 GIS 系统,如 Globus MDS[6]和 Condor 的群匹配机制[7]等.同时,P2P[8]和语义[9]等技术也被引入到网格中以解决资源发现的问题.然而,在研究领域中提到的很多资源发现方法很少在实际的网格系统中得到部署,目前,网格中的资源发现大多基于 Globus MDS 等,下面我们分析导致这种情况的主要原因.第一,为了保证系统的可伸缩性,避免采用集中式结构易出现的单点失效和性能瓶颈问题,大多数 GIS 系统采用某种分布式的拓扑结构,资源的描述性元信息被注册到各个 GIS 节点上,并提出了资源搜索算法来处理用户的查询请求.但是,这些系统不区分资源的特点,资源的元信息被随机地注册到各个 GIS 节点上,而为了获得满足用户查询条件的所有资源,搜索算法就必须对所有 GIS 节点进行遍历(即盲目搜索,blind search),影响了资源发现的效率.因此,如果能够按照某些特性对资源进行分类组织和管理,就可以有效地缩小搜索的范围,减少资源发现的代价,提高资源发现效率.第二,网格资源的规模巨大,不仅资源的加入和退出具有高度的动态性,而且计算资源自身的状态也在动态变化,这也要求建立一种具有自组织和自适应的动态演化特性的资源组织和发现机制.

从网格应用的角度看,在我们的 CROWN 项目[10]研发过程中,基于 CROWN 中间件[11]开发了大量的网格应用,这些应用对计算资源的需求方面呈现出不同的特点.例如,大整数分解应用,需要完成对 128 位以上大整数的质因数分解,需要大量 CPU 资源(属于计算密集型应用);在数字巡天图 DSS(digital sky survey)应用中,从空间望远镜中获取海量的观测数据,然后通过图形化界面提供用户查询指定区域内的星空图,需要至少 60GB 的空间来存储数据,而用户查询却不需要很强的计算能力(属于数据密集型应用).因此,计算资源需要根据应用需求的特性进行组织(例如,对拥有强大 CPU 能力的资源进行聚类,支持计算密集型服务),可以避免在各网格节点间的盲目搜索,从而有效地提高资源发现的效率.进一步地,我们观察到网格应用对计算资源的需求大部分可以通过计算资源的数值化属性来描述.例如,磁盘空间的大小和 CPU 的负载等.在对计算资源进行分类组织时,若能考虑到按照数值化属性进行预排序,则能够进一步提高资源发现的效率,并降低系统开销.

一般地,计算资源可通过一组基于〈名,值〉对的属性进行描述,我们将最能反映资源特点的属性定义为主属性(primary attribute,简称 PA),并基于 PA 取值将具有相同主属性的计算资源组织为一个平衡的二叉排序树.以平衡二叉排序树作为计算资源的基本组织单位,对每个网格虚拟组织(virtual organization,简称 VO)中的计算资源进行组织,同时兼顾多个 VO 的资源信息共享与发现,在提高资源发现效率的同时,为大规模网格系统中的计算资源发现提供了支持.

基于上述思路,通过引入计算资源的主属性概念,按照平衡二叉排序树对计算资源进行分类组织,本文提出了基于资源分类树(resource category tree,简称 RCT)的资源组织与发现机制;针对网格环境中存在不同虚拟组织,提出了基于 RCT 的跨越虚拟组织的资源组织体系结构;针对资源动态的加入/退出以及资源状态的动态性,设计了 RCT 的自组织机制.由于用户对资源的需求是不可预知的,因此导致了 RCT 各个节点上所承担的资源查

询和管理负载的不均衡,重载的节点会引发性能瓶颈,过于轻载节点的存在会增加维护和搜索开销,我们提出并设计了负载感知的自适应演化机制,使 RCT 能够自主地均衡各个节点之间的负载.同时,针对 RCT 节点可能突然失效的问题,设计了基于备份节点的容错机制;在基于 RCT 的组织结构基础上,设计了支持包括单点-单属性、单点-多属性、范围-单属性和范围-多属性的 4 种查询方式的搜索算法,并对算法的复杂度进行了理论分析.最后,通过仿真实验进行性能评估.实际上,本文提出的对计算资源进行分类组织的方法,其思路同样适用于其他网格资源的组织与发现问题.

本文第 1 节介绍相关研究工作.第 2 节提出 RCT 的定义并基于 RCT 的资源组织结构.第 3 节讨论关键技术和算法的设计.第 4 节介绍性能测试的方法和结果.第 5 节对研究工作进行总结.

1 相关工作

在已有的网格资源发现系统中,一般按照集中式、层次式、P2P 以及混合结构等对资源进行组织,并基于各自的组织结构设计相应的资源发现算法.(1) 集中式的资源组织在小规模环境下具有较高的效率,但其可伸缩性较差,并且存在单点失效和性能瓶颈等潜在问题,不适合大规模资源共享与协同的网格环境.Condor[7]采用了集中式的资源组织与发现机制,通过声明式的 ClassAd 描述用户查询请求和资源属性,采用 gang-matchmaking 的方法将用户的查询匹配到合适的资源.(2) 在层次式的组织结构中,位于较高层次的节点往往需要维护比低层次节点更多的信息,因此,随着层次的增多,高层节点的负载增加,不利于系统的扩展.Globus MDS[6]定义了两个协议(GRIP 和 GRRP)和两个组件(GIIS 和 GRIS)用以构建层次化的网格信息服务,文献[12]对 MDS 和 Condor 进行了全面的性能测量与分析.(3) P2P 是另一种重要的资源组织方式,一般可分为结构化和无结构的 P2P,前者多采用分布式哈希表(distributed hash table,简称 DHT)技术,建立描述资源的属性键值和资源位置的映射关系,典型的工作包括 Chord[13]和 CAN[14]等,能够对资源进行快速的定位,但难以支持多属性查询和范围查询,且容易因出现查询热点而导致负载分布不均衡等问题;无结构的 P2P 系统(如 Gnutella[15])对资源及其存储的位置关系不作强制性的限制,降低了维护的成本,但因资源搜索多采用洪泛(flooding)机制,因而增加了网络开销.(4) 混合结构综合考虑其他结构的优缺点,针对具体需求,对资源进行组织.上述这些方法主要解决如何将用户的查询请求路由到目标节点的问题,但没有考虑应用对资源需求的特点,需要遍历所有的信息服务节点进行盲目搜索,降低了资源发现的效率.

针对网格中计算资源的发现,文献[16]讨论了一种基于 DHT 的 P2P 方法,其中,资源属性的静态与动态部分和 Resource ID 绑定,作为基于 Pastry 的系统中的一个键值,资源被表示为 Pastry 环中层叠的弧,弧的起点代表静态属性集合,弧长代表动态属性的取值范围.但是,这种方法没有提供一种负载均衡的机制用以解决热点查询(hot-spot)可能导致的瓶颈问题.文献[17]提出了一种 SOG(self-organizing group)的层次与 P2P 的混合结构,基于具体资源特征的相似度组织资源,其中特征相似的节点组成一个组,并通过 gossip 协议选举出一个领导者.SOG 中的“组”类似于 RCT,但其区别在于,RCT 通过基于主属性值的排序二叉树组织资源,且 RCT 在定义时考虑了应用对资源的需求.

在范围查询和多属性查询方面,也有很多颇具代表性的研究工作[18−20].文献[18]提出了一种逻辑树 RST (range search tree),支持 DHT 结构上的范围查询.RST 的优势在于不需要动态维护树的结构,因为一个属性的值域已提前分成了 2^n 个子区间,每个节点可以在本地推导出树结构.RST 能够根据查询和注册负载的变化进行动态结构调整,但 RST 要求资源同时向许多节点进行注册,由于在动态网格系统中资源频繁更新状态信息,这种方式会产生巨大的资源注册开销.Mercury[19]通过创建路由 hub,并将路由 hub 组织成一个环形的层叠网来支持多属性和范围查询.所谓的路由 hub 是指负责管理某一个资源属性的节点,即每个属性对应于一个路由 hub,一个资源必须向所有的路由 hub 进行注册,可见,当属性的数量很多时,路由 hub 的维护开销以及资源的查询开销很大.与 Mercury 不同,RCT 选择少数有代表性的主属性进行资源的组织,其开销较小.DPTree[20]采用基于跳图(skip graph)的层叠网对资源进行组织,资源的数据首先映射到一个逻辑上的 R-Tree 上,然后将 R-Tree 映射到跳图层叠网结构.基于跳图的路由算法,DPTree 设计了处理多维属性区间查询的算法.RCT 与 R-Tree 相比,所采用

的是平衡的二叉树作为资源组织的层叠网结构,重点考虑了基于资源的动态性的自适应调整机制.

2 计算资源组织机制

对资源进行有效的组织是解决资源高效发现的前提,本节介绍基于 RCT 的计算资源组织机制.

2.1 RCT的定义与性质

网格系统中广泛地采用基于属性的方法对资源进行描述,这种方式简单而有效.本文通过一组基于〈名,值〉对的属性来描述每个计算资源的元信息.例如,可以通过属性集合 S={CPU=2.5 GHz,Mem=1.0 GB,Disk=80GB,…,CPU load=30%}来描述一台计算机.实际系统运行中,资源在真实计算环境下的动态属性(例如 CPU 负载和可用内存空间等)更为重要,因为动态信息代表了一个计算资源的实际可用能力,是系统调度器和资源代理正确调度或资源选取决策的基础.

如前所述,我们将最能反映资源能力特征的属性称为主属性,如果主属性值为非数值型,则可通过哈希函数等数学手段进行数值化处理.例如,一台存储服务器的主属性可定义为可用磁盘空间,反映了存储服务器的主要能力特征.实际中,资源主属性可以根据应用需求灵活定义.为了更加准确地对 RCT 进行描述,我们有:

定义 **1**. 设 D=[L,U]是某个主属性的值域,L 和 U 是 D 的下界和上界,D_i=[L_i,U_i]和 D_j=[L_j,U_j]为 D 的两个子区间,且 $D_i \cap D_j=\varnothing$.如果 $U_i \le L_j$,则定义 $D_i<D_j$,或者 $D_j>D_i$.例如,[1,2]<[3,4].

定义 **2**. 资源分类树 RCT.设选定主属性 A 的值域为 D,将 D 划分为 n 个互不相交的子区间 D_i(i=1..n),且 D_i 由节点 N_i 进行管理,将 N_i 按照 D_i 的大小组织为一个平衡的二叉树(也称为 AVL 树),称该平衡二叉树为资源分类树(resource category tree,简称 RCT).

对任意一个计算资源 R,如果 R 满足:(1) R 具有属性 A(记为 A^R),A^R 的取值范围为[L^R,U^R]($L^R \le U^R$),A^R 当前的取值为 v;(2) [L^R,U^R]⊆[L,U],则将资源 R 注册到负责 D_i($v \in D_i$)的节点 N_i.

与传统 AVL 树相比,(1) RCT 是分布式的结构,每个节点由一个真实的物理节点代表;(2) 每个 RCT 节点负责一个取值区间,而不是单个数值,使每个 RCT 节点能够组织管理所有主属性值位于该区间内的计算资源.例如,如果主属性是可用磁盘空间,某个 RCT 节点负责的区间为[30,50],那么具有大于 30GB 而小于 50GB 可用磁盘空间的计算资源都将注册到该节点,由该节点进行管理.

定义 **3**. 管理节点 HR(head of a sub range).所有主属性取值属于 D_n 的资源在节点 n 注册,每个 RCT 节点负责管理某个计算资源的集合,我们将 RCT 节点简称为管理节点(head of a sub range,简称 HR).

每个节点只需维护与直接子节点和父节点之间的连接关系,诸如注册、更新和查询等操作可以从任意一个节点发起.上层节点不需要比下层节点维护更多的信息或承担更多的负载,因此 RCT 的节点之间是相互对等的关系.图 1 是一个 RCT 的示例,主属性为可用磁盘空间,其中阴影圈代表 HR 节点,HR 节点下面的方块代表注册到 HR 的计算资源.

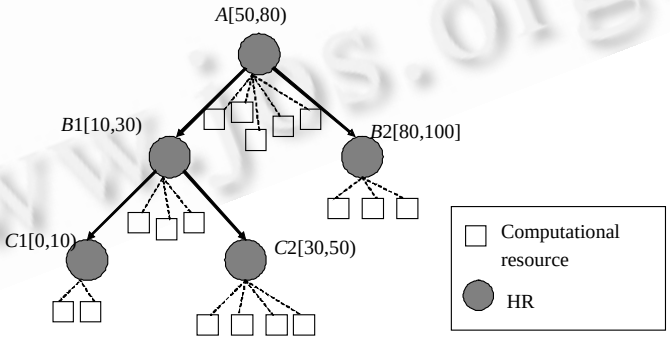

Fig.1 An example of RCT

图 1 RCT 示例

由于 RCT 是一棵平衡的二叉排序树,因此满足:(1) 若左子树不空,则左子树上所有节点值均小于根节点的值;(2) 若右子树不空,则右子树上所有节点的值均大于根节点的值;(3) 左子树和右子树的深度之差不大于 1;(4) 左右子树分别为平衡的二叉排序树.此外,假定 N 是 RCT 中所有 HR 的数量,$lc(n)$与 $rc(n)$分别代表 n 的左直接子节点和右直接子节点,则 RCT 具有如下 3 个性质(其中 D_i 表示 HR i 所负责的区间):

$$\bigcup_{i=1}^{N} D_i = D \tag{1}$$

$$D_i \cap D_j = \varnothing,\ \forall i, j \in [1, N] \tag{2}$$

$$D_{lc(i)} < D_i < D_{rc(i)},\ \forall i \in [1, N] \tag{3}$$

针对两个 HR 节点之间的邻接关系,我们有如下定义:

定义 **4**. 如果 HR i 和 HR j 所负责的值域 D_i 和 D_j 相邻,则 i 就是 j 的一个邻居.如果 i 是 j 的一个邻居而且 $D_i<D_j$,则 i 称为 j 的左邻居,表示为 i=*L-neighbor*(j);j 称为 i 的右邻居,表示为 j=*R-neighbor*(i).

由以上定义和性质可知,最左 HR 没有左邻居,而最右 HR 没有右邻居.如图 1 所示,*C*2 和 *B*2 是 *A* 的邻居,*C*2 是 *A* 的左邻居,*B*2 是 *A* 的右邻居.其中 *C*2 没有左邻居,*B*2 没有右邻居.

2.2 基于**RCT**的资源组织结构

在网格环境下,为了协同问题求解,在一定的共享和协同规则下形成了多个虚拟组织 VO,每个 VO 拥有和管理着一定数量和种类的资源,但求解复杂问题经常需要多个 VO 资源的共享与协同,需要建立 VO 之间的资源发现机制.对此,我们提出了一种基于 RCT 来组织资源的两层结构(如图 2 所示),首先,每个 VO 定义了一组最能够描述其内部资源特征的主属性,基于指定的主属性,VO 内资源组织为多棵 RCT 树;其次,有些复杂的作业可能需要动态地访问其他 VO 的资源,因此,为了支持 VO 之间的计算资源发现,每个 VO 部署一个 RCT 索引服务(RCT index service,简称 RIS),它是一个用来存储 VO 的主属性信息和 RCT 的访问入口点的基础服务(可实现为 Web service 或 Grid service).当一个查询请求不能在 VO 内部得到满足时,RIS 通过其他 VO 的 RIS,为用户提供透明的跨越虚拟组织的资源发现.特别需要指出的是,如第 2.1 节所述,在 VO 内部的树形结构和传统树形组织结构的含义有所不同,RCT 节点之间的层次关系反映了各节点所负责的取值区间的大小关系,而各节点在组织管理上是相互对等的关系.此外,各个 VO 的 RIS 可以根据实际需求采用成熟的 P2P 等组织结构,在 RIS 之间实现用户请求的高效路由和转发,本文对此暂不作深入探讨.

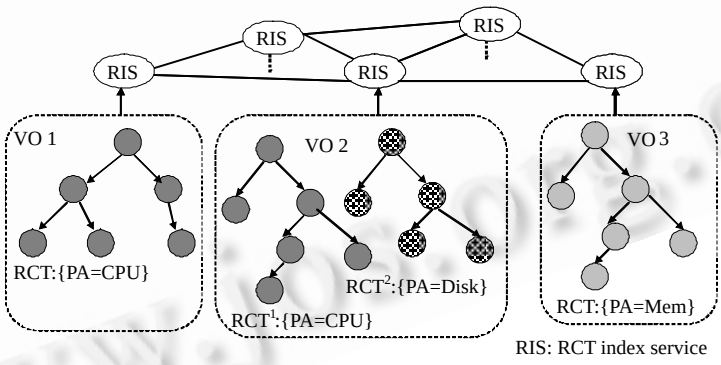

Fig.2 Resource organization with RCT

图 2 基于 RCT 的资源组织结构

2.3 **RCT**的自组织机制

为了使 RCT 能够适应网格的动态性,保证系统的较好扩展性,我们将自组织特性引入到 RCT 的设计中.通过计算资源的自组织机制来完成 RCT 的建立和维护过程,具体包括 RCT 的初始化、负载感知的自主演化、计算资源动态性维护以及容错机制 4 个方面.

2.3.1 初始化

RCT 由若干个 HR 组成,其初始化过程是产生第 1 个 HR 的过程.HR 负责管理多个计算资源的加入、退出和迁移,必须保证 HR 具有较强的资源能力和可用性.我们通过计算能力 c(CPU 主频)来衡量处理能力,拥有较强计算能力的 HR 降低了因过载而出现性能瓶颈的风险,能够更好地完成对计算资源的管理和维护;资源的可用性一般通过 $MTTF/(MTTF+MTTR)$来衡量,其中 MTTF(mean time to failure)为平均发生故障时间,而 MTTR(mean time to recover)为平均修复时间,这里我们用资源的平均在线时间 t_{on} 作为衡量 MTTF 的指标,t_{on} 越长意味着更好的可用性,用平均离线时间 t_{off} 作为衡量 MTTR 的指标.由于 RCT 的自组织特征,HR 需要根据处理能力和可用性两个指标从 VO 中的计算资源中选出,无须管理员的预先配置.

我们在每个 VO 中部署一个 RCT 索引服务(RIS),VO 的管理员通过 RIS 配置了 VO 内计算资源的主属性和值域等信息.当计算资源加入网格之后,首先访问 RIS 服务,根据加入的计算资源的可用性和处理能力以及 VO 的主属性配置信息,RIS 服务选出第 1 个 HR,完成 RCT 的初始化,具体过程如下:

(1) 计算资源 R 向 RIS 查询 RCT 的配置信息;

(2) 资源 R 检查自身是否满足加入 RCT 条件(即主属性的值域),如果满足,转至(3);否则,终止;

(3) 资源 R 向 RIS 提交 HR 申请,其中包括自身可用性和能力信息;

(4) RIS 存储 R 提交的相关信息.当一个 RCT 的候选者达到一定数量时,RIS 将比较它们的可用性和能力,选举出第 1 个最合适的 HR,然后,RIS 通知其他候选资源向选定的 HR 注册,RCT 的初始化完成.

第 1 个 HR 确定之后,后加入的计算资源通过 RIS 获取 HR 的访问入口点,直接加入到 RCT 中,不再提交 HR 申请.因此,RIS 只负责选择 RCT 的第 1 个 HR,其他 HR 的加入和选择,由 RCT 自主完成.

2.3.2 负载感知的自主演化

当 RCT 完成初始化后,RCT 中只有一个 HR 节点,负责主属性的整个值域,后面加入的计算资源都会注册到该 HR 节点中;并且,所有资源的查询请求也都由其进行处理.如果没有相应的处理机制,注册和查询负载会使 HR 成为系统的性能瓶颈,甚至导致系统瘫痪.为了防止这种情况的发生,系统必须能够选出更多的 HR 来分别负责主属性值域上的不同区间,这些 HR 节点按照 RCT 结构的性质建立连接关系.

另一方面,当一个 HR 处于轻载状态,它管理和维护少数的计算资源,同时,它的存在也会增加 RCT 的深度,从而增加了平均搜索长度.当轻载 HR 的存在所增加的搜索开销大于其管理资源的效用时,为了减少平均搜索长度,降低平均搜索开销,应将轻载 HR 负责的值域区间与其邻居加以合并,以减少轻载 HR 节点的存在.

综上分析,为了保证 RCT 的自组织性和高效率,我们提出了负载感知的自主演化机制.在讨论具体算法之前,需要考虑如下几个问题:

首先,如何定义衡量一个 HR 负载 l 的指标?由于描述资源的信息是一些〈名,值〉对,管理计算资源对存储空间的需求不大,其中的开销主要体现为计算能力的使用,因此,本文用 CPU 负载表示 HR 的负载.定义 l_{light} 和 l_{over} 分别为轻载和过载的阈值:当 $l\le l_{light}$ 时,HR 轻载;当 $l\ge l_{over}$ 时,HR 过载.另外,定义一个警戒阈值 $l_{warning}$ ($l_{light}<l_{warning}<l_{over}$)以避免一个 HR 节点由于接收负载而很快变成过载,引起整个系统振荡而趋于不稳定.当 HR 的负载位于 $l_{warning}$ 和 l_{over} 之间时,意味着 HR 接近过载,不能接收其他 HR 转移的负载.例如,当 l 大于 90% (l_{over}=90%)时,HR 被视为过载;当 l 小于 10%(l_{light}=10%)时,HR 被视为轻载;当 l 在 10%和 80%($l_{warning}$=80%)之间时,HR 可以帮助其他 HR 均衡负载.

第二,HR 所负责的取值区间如何进行分裂?假定 D_n 是 HR n 负责的值域区间.根据上述分析,当 n 过载时,D_n 会分裂,同时选出新的 HR 节点来均衡负载.我们设计了两种机制分裂一个区间:平均分裂(average split,简称 AS)和基于方差分析的分裂(analysis of variance-based split,简称 AVS).在 AS 中,待分裂的 HR 所负责的区间被平均分为 2 个或 3 个子区间;AVS 通过方差分析,根据 D_n 上的负载分布情况进行分裂.例如,假定 HR 所负责的区间为[10,100],负载主要集中在[80,100].AS 会把区间平均分为[10,55]和[55,100];而 AVS 的分裂结果可能是[10,90]和[90,100].

第三,如何确保 HR 能够有充分的资源能力用于对资源的管理?HR 从参与网格计算的资源中自动选出,其

本身也会承担其他作业的处理.为了保证用户作业处理不受影响,通过资源预留机制预留出部分处理能力用于执行 HR 的功能.例如,可以为一个作为 HR 的资源预留 20%的 CPU 时间.

基于对以上 3 个问题的讨论,我们提出了 LS(load splitting)和 LM(load merging)算法,实现对 RCT 中重载 HR 和轻载 HR 的自主调整.在这两种算法的作用下,随着 HR 节点的分裂与合并,RCT 不断进行动态的演化和调整,从而保证整个系统的高效运行.

算法 **1**. HR 重载情况下的调整算法 LS.

1: ln=L-neighbor(n);
2: rn=R-neighbor(n);
3: $load_{ln}$=*getLoad*(ln);
4: $load_{rn}$=*getLoad*(rn);
5: if $load_{ln}\le l_{warning}$ and $load_{rn}\le l_{warning}$ then
6: split D_n into 3 sub ranges D_1, D_2 and D_3($D_1<D_2<D_3$), according to AVS policy;
7: transfer resources in D_1 from n to ln;
8: transfer resources in D_3 from n to rn;
9: $D_n=D_2$; $D_{ln}=D_{ln}+D_1$; $D_{rn}=D_{rn}+D_3$;
10: else if $load_{ln}\le l_{warning}$ then
11: split D_n into 2 sub ranges D_1 and D_2($D_1<D_2$), according to AVS policy
12: transfer resources in D_1 from n to ln;
13: $D_{ln}=D_{ln}+D_1$; $D_n=D_2$;
14: else if $load_{rn}\le l_{warning}$ then
15: split D_n into 2 sub ranges D_1 and D_2($D_1<D_2$), according to AVS policy;
16: transfer resources in D_2 from n to rn;
17: $D_n=D_1$; $D_{rn}=D_{rn}+D_2$;
18: else
19: split D_n into 2 sub ranges D_1 and D_2($D_1<D_2$), according to AVS policy;
20: select an HR n' from resources in D_1;
21: $D_{n'}=D_1$; $D_n=D_2$;
22: Insert HR n' into current RCT;
23: balanceTree();
24: end if

如算法 1 所示,LS 算法描述了当一个 HR 处于重载状态时,重载 HR 自主调整的过程.首先,分别获取左、右两个邻居节点(邻居节点的定义见第 2.1 节中的定义 4)的负载(第 1 行~第 4 行),其中 *getLoad* 方法是每个 HR 节点必须实现的方法,用于其他节点查询其负载的情况;根据两个邻居节点的负载状态分裂区间(为了能够有效均衡负载,区间的分裂策略采用 AVS),将负载转移给相应的邻居(第 5 行~第 22 行),注意,为了保证 RCT 的二叉排序树的性质,只能向相邻的两个节点进行转移负载;如果两个邻居都不能接收负载,需要选举新的 HR 平衡负载(第 20 行),由于增加了新的节点,会导致 RCT 某个分支的高度增加,因此需要对 RCT 进行树的平衡化调整(第 23 行).AVL 树的平衡化算法见文献[21],此处不再赘述.

同时,我们设计了 LM 算法,因受篇幅所限,省略了其伪代码的表示.当一个 HR 处于轻载状态时,LM 算法给出了轻载 HR 进行自主调整的过程.轻载 HR 尝试与邻居节点进行合并,合并之后能够减小平均搜索路径的长度,从而提高资源发现的效率.首先,分别获取两个邻居的负载;如果两个邻居的负载都在警戒阈值以上,即不适合接收新的负载,因此等待下一个周期重新尝试;然后,根据两个邻居节点的负责情况,负载被转移到左邻居或右邻居,其中,当左、右邻居都可以接收负载时,需要将 HR 的区间等分为两部分,由于当前 HR 负责的区间均为

轻载状态,则分裂策略采用 AS,即按照区间长度进行平均分割;然后,节点 n 退出,不再承担 HR 的管理任务,同时删除与其他 HR 的连接关系;最后,由于节点 n 的退出,必然导致 RCT 某一分支的高度减小,因而需要进行平衡化处理.

实际上,基于 LS 算法和 LM 算法,RCT 中的 HR 节点之间自主地实现了负载的均衡,避免了因查询热点造成的性能瓶颈.由算法描述可知,每次节点的调整,最多涉及到 3 个节点,即当前 HR 节点及其左、右两个邻居节点,因此,执行 LS 和 LM 算法的开销不会影响到太多的节点.就 LS 和 LM 算法本身来说,所涉及的操作包括本地执行的区间分割和节点之间的负载转移,前者的开销可以基本忽略,后者的开销主要来自于节点之间的通信.在转移负载的过程中,本质上是把 HR 节点所负责管理的资源的元信息传输到接收负载的节点,而元信息是基于名-值的字符串描述,其数据量一般不超过 1KB.

2.3.3 计算资源的动态性维护

由于网格环境下的计算资源不仅动态加入和退出,而且其状态也会因所处理的作业而不断发生变化,需要提供相应的机制来维护资源的动态性.当一个计算资源连入网格系统,通过索引服务完成注册到一个 RCT,资源完成向某个 HR 的注册之后,将向该 HR 周期地更新它的状态信息.

当一个资源突然退出时,管理该资源的 HR 需要在尽可能短的时间内感知到资源的离开,以避免向用户提供不可用的资源信息.为此,资源在向 HR 注册之后,需要周期性地向 HR 发送状态更新消息.如果资源状态没有变化,更新信息的内容为空.另一方面,如果 HR 长时间没有收到资源的更新信息,就认为该资源已经离开,并清除相关信息.

此外,由于 HR 只管理主属性值属于其负责区间范围的资源,如果资源的主属性值由于动态变化而超出 HR 负责的区间范围,那么 HR 遍历 RCT 搜索合适的 HR,并将资源转移给相应的 HR 进行管理,并将该资源的信息从本地清除.

2.3.4 容错机制

HR 负责管理主属性值处于某一区间的所有资源,尽管 HR 的选取标准保证了 HR 具有较强的可用性,但仍然无法避免因网络等各种突发原因导致 HR 突然失效的问题.针对这一问题,我们提出基于备份节点的容错解决方案,即在 RCT 自组织的过程中,每当选出新的 HR 时,同时选出一个 HR 的备份节点.主 HR 和备份 HR 保持数据的同步,当主 HR 失效时,立即切换到备份 HR,同时选出新的备份节点.通过这种机制,可以有效地提高 RCT 的容错能力.

3 基于 RCT 的计算资源发现算法与性能分析

3.1 资源发现算法

根据查询请求中属性的个数和查询条件的范围,可将查询分为 4 类:单点-单属性查询($Q1$)、范围-单属性查询($Q2$)、单点-多属性查询($Q3$)和范围-多属性查询($Q4$).例如,$Q1$:“可用存储=90GB”;$Q2$:“CPU load<50%”;$Q3$:“可用存储=100GB AND OS=Windows XP”;$Q4$:“Available memory>256 MB AND CPU load<80%”.这里,我们将针对 4 种查询而需要 RCT 执行的搜索过程分别称为 $Q1$-*search*,$Q2$-*search*,$Q3$-*search* 和 $Q4$-*search*.

RCT 支持全部 4 种查询方式,算法 3 给出了 $Q2$-*search* 的搜索算法,基于 $Q2$-*search* 算法可以很容易地得到支持其他 3 种查询的搜索算法,这里不再赘述.需要说明的是,为了在 RCT 各个节点之间均衡查询负载,算法支持从任意一个节点发起查询,这不同于传统 AVL 树只能从根节点开始查询的方式.

算法 **2**. $Q2$-search 算法.

1: func search (D_q,n):ResultSet
2: ResultSet: $rs1$, $rs2$, $rs3$, $rs4$;
3: $D_0=D_q\cap D_n$;
4: split D_q-D_0 into D_1 and D_2($D_1<D_2$);
5: if D_0!=∅ then $rs1$=search(D_0,n);

6: if D_1!=∅ then $rs2$=search(D_1,$lc(n)$);
7: if D_2!=∅ then D'=D_2∩$D_{r\text{-}childT(n)}$;
8: if D'!=∅ then $rs3$=search(D',$rc(n)$);
9: if D_2–D'!=∅ and $parent(n)$!=null then
10: $rs4$=search (D_2–D',$parent(n)$);
11: return $rs1$∪$rs2$∪$rs3$∪$rs4$
12: end if
13: end func

在算法 2 中,D_q 是用户指定的主属性值的查询范围,$D_{l\text{-}childT(n)}$和 $D_{r\text{-}childT(n)}$分别表示节点 n 的左子树和右子树所覆盖的区间范围.该算法是一个递归函数,查询结果汇集到 *ResultSet* 结果集中,为此定义了 4 个用于存放中间结果的临时结果集 $rs1$,$rs2$,$rs3$ 和 $rs4$(第 2 行).首先将 D_q 分为 3 个子区间 D_0,D_1 和 D_2(第 3 行~第 4 行),D_0 是 D_n 与 D_q 的交集,D_1 和 D_2 是分别与 D_0 左、右邻接的两个子区间.如果 D_0 非空,对区间 D_0 的搜索在当前 HR 执行(第 5 行),并将结果存储到 $rs1$ 中;如果 D_1 非空,搜索左子树(即 $lc(n)$),并将结果存储到 $rs2$ 中(第 6 行);如果 D_2 非空,根据 D_2 和 $D_{r\text{-}childT(n)}$的关系来确定搜索右子树(即 $rc(n)$)或者父节点(第 7 行~第 10 行),并将结果分别存储在 $rs3$ 和 $rs4$ 中.最后将 $rs1$~$rs4$ 的结果汇合,并返回(第 11 行).例如,针对图 1 所示的 RCT,如果要搜索可用磁盘空间位于[8G,40G]的资源,搜索入口点为 $B1$.根据算法 2,先将 D_q=[8,40]划分为 D_0=[10,30],D_1=[8,10]和 D_2=[30,40],然后分别在 $B1$,$C1$ 和 $C2$ 这 3 个节点上执行对 D_0,D_1 和 D_2 的搜索,最后由 $B1$ 汇集所有结果并返回给用户.

此外,与在单一 RCT 内部的搜索不同,为了支持跨越 VO 的资源发现,需要在不同 VO 的 RCT 之间进行资源搜索.在这种情况下,每个 VO 的 RIS 服务作为本 VO 的搜索代理,它接收来自其他 VO 的搜索请求并代理用户在本 VO 内执行搜索,最终将查询结果返回.

3.2 性能分析

与传统二叉排序树的搜索机制不同,RCT 主要特点是:(1) RCT 本身是分布式结构,用户可以从任何一个节点开始执行搜索,而传统的二叉树只能从根节点开始执行搜索;(2) RCT 中每一个节点维护一组数值,搜索分为两步,先定位区间,然后在区间内进行本地搜索,而传统二叉树每个节点只维护单一数值点.在处理查询请求的过程中,节点执行本地搜索的开销相对较小,主要的开销发生在 RCT 节点之间的网络通信上,因此这里采用平均搜索长度(average search length,简称 ASL)对 RCT 的性能进行分析.所谓 ASL 是指处理一个资源查询请求需要搜索的 RCT 节点的平均个数.为了便于分析,这里我们假定:

(1) 每个计算资源被用户搜索的概率是相等的;
(2) 每个 HR 节点被用户选为查询入口的概率是相等的.

文献[22]给出了详细的推导过程,对于一个深度为 h(h>2)的满二叉树,其平均搜索长度为

$$ASL=\frac{2^h\times(h-4)+1}{2^h-1}+\frac{2^h\times[(h-1)\times 2^h+1]}{(2^h-1)^2}+\frac{h\times 2^{h+1}}{(2^h-1)^2}\approx 2h-5.$$

当 h≤2 时,ASL=1.因为 RCT 不一定是满二叉树,因此其平均搜索长度为

$$ASL=\begin{cases}l, l>0, l\in[2h-7,2h-5], h\ge 3\\ 1, h\le 2\end{cases}.$$

从以上分析可以看出,RCT 的平均搜索长度与树的深度相关,深度越大,平均搜索长度越大.值得注意的是,在查询负载总量一定的情况下,当树的深度较大时,意味着 HR 节点较多,平均每个 HR 节点承担的搜索负载较小;反之,当树的深度较小时,表明 HR 节点较少,平均每个 HR 节点承担的搜索负载较大,容易导致系统的性能瓶颈.因此,需要平衡每个 HR 节点的平均负载和平均搜索长度,通过第 2 节提出的基于负载感知的自主演化机制可进行优化调整.

4 性能测试

通过事件驱动的方式,我们仿真了一个具有 100 个 HR 节点的 RCT 结构,每个 HR 平均管理 20 个计算资源,主属性的值域区间 D 为[0,100],D 在 100 个 HR 中均匀分割,即每个 HR 的子区间长度为 1.客户端查询请求的发生服从泊松随机过程.采用的测量指标如下:

(1) 每个 HR 节点处理的查询请求数 QN(query number):用于评估 HR 在不同情形下的平均查询负载.

(2) 平均搜索长度 ASL:表明了在查询处理中,搜索的 HR 节点的平均数量.综合指标(1)和指标(2),可以评估资源发现的效率.

(3) 查询负载的标准差.由于 RCT 具有负载感知的自主演化能力,该指标用于评估查询负载在各 HR 之间的均衡分布程度,其值越小意味着负载均衡效果越好.

在第 1 个实验中,我们考察指标(1)和指标(2),通过改变查询请求的平均到达速率评估 RCT 的性能,并且与基于不支持信息复制的层次结构的查询机制(即高层节点中并不存储低层节点所管理的资源信息,在处理查询请求时,需要遍历所有的节点)进行了对比,图 3 和图 4 给出了实验结果,我们观察到,与层次结构相比,RCT 给每个节点带来的查询开销更小,并且具有更短的平均搜索长度.需要指出的是,为了更好地显示结果,图 3 中基于层次结构的查询请求数已经进行了缩小 50 倍的预处理.

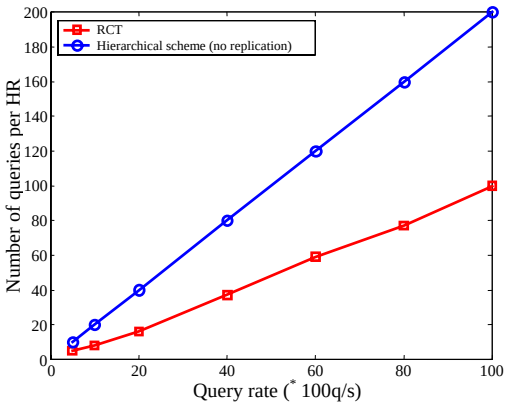

Fig.3 Query rate vs. query load per HR
图 3 查询速度和每个 HR 的查询负载

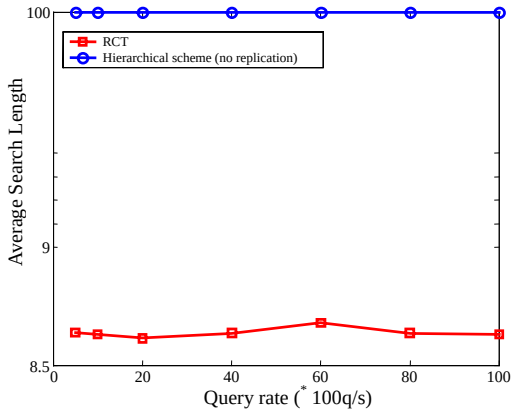

Fig.4 Query rate vs. ASL
图 4 查询速度和平均搜索长度

RCT 可以根据 HR 的负载状态进行自主的演化,第 2 个实验用于评估 HR 在过载和轻载情况下 RCT 的演化过程.第 2 节中定义的 3 个阈值 l_{over},l_{light} 和 $l_{warning}$ 可用每秒处理的查询请求数来表示,分别设为 50,25 和 40.一方面,使 90 个 HR 以每秒处理 40 个查询请求的速率处于正常工作状态;另一方面,其余的 10 个 HR 的负载从 1 变化到 120,经历了从轻载、正常到过载的过程.图 5 显示了 RCT 的自适应负载均衡机制可以在轻载和过载情况下有效均衡查询负载,从而极大地降低了系统瓶颈出现的风险.图 6 描绘了在 10 个 HR 节点负载变化过程中,RCT 中 HR 节点数目的变化情况.当 10 个 HR 处于轻载状态,它们将与其他 HR 合并,HR 总数减少;当 10 个 HR 处于过载状态,会产生新的 HR 分担负载,使得 HR 总数增加.

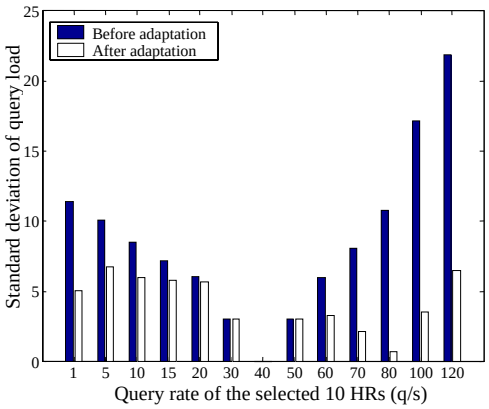

Fig.5 Standard deviation of query load
图 5 查询负载的标准差

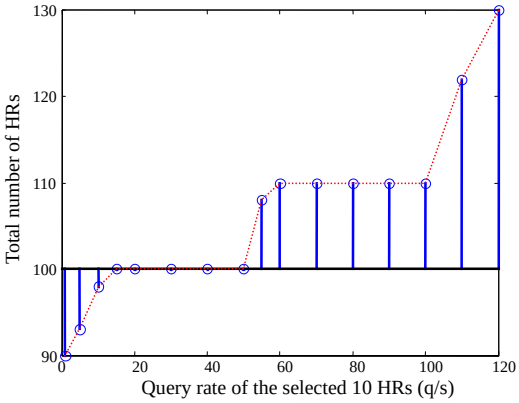

Fig.6 Number of HRs during adaptation
图 6 RCT 自适应调整过程中的 HR 数量

5 结束语

计算资源作为一类重要的网格资源,研究其发现机制具有重要的意义.本文根据网格应用对计算资源的需求特征以及计算资源能够通过数值化的属性进行描述的特点,提出了基于 RCT 的计算资源组织和发现机制,其中,RCT 采用平衡的二叉排序树作为基本的拓扑结构,但与传统的层次结构不同,节点之间是有结构的对等关系,对资源的组织过程以及自身的维护具有自组织和自演化的特点,并提供了一定的容错机制.RCT 支持常见的 4 种查询:单点-单属性查询、范围-单属性查询、单点-多属性查询和范围-多属性查询.实验结果表明,RCT 提高了计算资源发现的效率,同时具有很好的负载感知的自演化特性,保证了系统的可伸缩性.

RCT 为网格中计算资源的组织与发现提出了一种有效的解决方案,基于本文提出的技术思路,目前我们正在进行系统实现,逐步解决 RCT 在实际网格环境部署中可能遇到的各种问题,并计划在系统部署之后与其他网格信息服务系统进行性能对比分析.此外,我们还将继续研究将资源分类组织的思想用于解决网格服务等其他网格资源的组织与发现问题,以及如何在基于不同网格中间件基础设施的网格系统之间进行资源发现.

References:

[1] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organization. The Int'l Journal of Supercomputer Applications, 2001,15(3):200−222.

[2] Foster I, Kesselman C, Nick JM, Tuecke S. Grid services for distributed system integration. IEEE Computer, 2002,35(6):37−46.

[3] Sun H, Zhong L, Huai J, Liu Y. OpenSPACE: An open service provisioning and consuming environment for grid computing. In: Stokinger H, Buyya R, Perrott R, eds. Proc. of the 1st IEEE Int'l Conf. on e-Science and Grid Computing. Los Alamitos: IEEE Computer Society, 2005. 288−295.

[4] Sun H, Zhu Y, Hu C, Huai J, Liu Y, Li J. Early experience of remote & hot service deployment with trustworthiness in CROWN grid. In: Cao J, Nejdl W, Xu M, eds. Advanced Parallel Process Technologies. LNCS 3756, Berlin: Springer-Verlag, 2005. 301−312.

[5] Plale B, Dinda P, Laszewski G. v.Key Concepts and Services of a Grid Information Service. 2002. http://www.cs.indiana.edu/dde/papers/plalePDCS2002.pdf

[6] Czajkowski K, Fitzgerald S, Foster I, Kesselman C. Grid information services for distributed resource sharing. In: Williams AD, ed. Proc. of the 10th IEEE Int'l Symp. on High Performance Distributed Computing. Los Alamitos: IEEE Computer Society, 2001. 181−194.

[7] Raman R. Matchmaking frameworks for distributed resource management [Ph.D. Thesis]. Madison: University of Wisconsin-Madison, 2001.

[8] Iamnitchi A, Foster I, Nurmi DC. A peer-to-peer approach to resource location in grid environments. In: Jacobs A, ed. Proc. of the 11th IEEE Int'l Symp. on High Performance Distributed Computing. Los Alamitos: IEEE Computer Society, 2002. 419.

[9] Heine F, Hovestadt M, Kao O. Towards ontology-driven P2P grid resource discovery. In: Buyya R, ed. Proc. of the 5th IEEE/ACM Int'l Workshop on Grid Computing. Los Alamitos: IEEE Computer Society, 2004. 76–83.

[10] CROWN project. 2008. http://www.crown.org.cn/

[11] Huai JP, Hu CM, Li JX, Sun HL, Wo TY. CROWN: A service grid middleware with trust management mechanism. Science in China (Series E), 2006,36(10):1127–1155 (in Chinese with English abstract).

[12] Zhang X, Freschl JL, Schopf JM. A performance study of monitoring and information service for distributed systems. In: Azada D, ed. Proc. of the 12th IEEE Int'l Symp. on High Performance Distributed Computing. Los Alamitos: IEEE Computer Society, 2003. 270–281.

[13] Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. In: Govindan R, ed. Proc. of the 2001 SIGCOMM Conf. New York: ACM Press, 2001. 149–160.

[14] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Govindan R, ed. Proc. of the 2001 SIGCOMM Conf. New York: ACM Press, 2001. 161–172.

[15] Gnutella. 2008. http://rfc-gnutella.sourceforge.net/index.html

[16] Cheema AS, Muhammad M, Gupta I. Peer-to-Peer discovery of computational resources for grid applications. In: Jacob JC, ed. Proc. of the 6th IEEE/ACM Int'l Workshop on Grid Computing. Los Alamitos: IEEE Computer Society, 2005. 179–185.

[17] Padmanabhan A, Wang S, Ghosh S, Briggs R. A self-organized grouping (SOG) method for efficient grid resource discovery. In: Jacob JC, ed. Proc. of the 6th IEEE/ACM Int'l Workshop on Grid Computing. Los Alamitos: IEEE Computer Society, 2005. 312–317.

[18] Gao J, Steenkiste P. An adaptive protocol for efficient support of range queries in DHT-based systems. In: Koenig H, ed. Proc. of the 12th IEEE Int'l Conf. on Network Protocols. Los Alamitos: IEEE Computer Society, 2004. 239–250.

[19] Bharambe AR, Agrawal M, Seshan S. Mercury: Supporting scalable multi-attribute range queries. In: Yavatkar R, ed. Proc. of the 2004 SIGCOMM Conf. New York: ACM Press, 2004. 353–366.

[20] Li M, Lee W, Sivasubramaniam A. DPTree: A balanced tree based indexing framework for peer-to-peer systems. In: Almeroth KC, ed. Proc. of the 14th IEEE Int'l Conf. on Network Protocols. Los Alamitos: IEEE Computer Society, 2006. 12–21.

[21] Yan W, Wu W. Data Structure. 2nd ed., Beijing: Tsinghua University Press, 1992. 237–240 (in Chinese).

[22] Fu GW, Sun HL. Complexity analysis of random search with AVL tree. Technical Report, TR-BUAA-ACT-06-01, 2008. http://www.crown.org.cn/downloads/AVL-Complexity.pdf

附中文参考文献:

[11] 怀进鹏,胡春明,李建欣,孙海龙,沃天宇.CROWN:面向服务的网格中间件系统与信任管理.中国科学(E 辑),2006,36(10):1127–1155.

[21] 严蔚敏,吴伟民.数据结构.第 2 版,北京:清华大学出版社,1992.237–240.

孙海龙(1979－),男,河北承德人,博士,主要研究领域为网格,Web 服务,P2P 等分布式计算技术.

富公为(1982－),男,硕士,主要研究领域为网格计算.

怀进鹏(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为计算机软件与理论,中间件与服务网格,网络信息安全.