计算网格环境下基于多址协同的作业级任务调度算法 *

张伟哲 [1**] 方滨兴 [1,2] 胡铭曾 [1] 刘欣然 [2,3] 张宏莉 [1] 高 雷 [1]

(1. 哈尔滨工业大学计算机科学与技术学院,哈尔滨 150001; 2. 国家计算机网络应急技术处理协调中心, 北京 100029; 3. 北京邮电大学计算机科学与技术学院, 北京 100876)

摘要 计算网格下多管理域机群互连为作业级任务协同调度创造了机遇, 同时在协同性、异构适应性、网络适应性和算法可扩展性方面对传统的作业调度模型与算法提出了新的挑战. 通过引入网格环境下作业级多址任务调度模型与性能模型, 提出多址任务协同调度算法框架. 以最优和贪心资源选择策略为核心, 提出两种作业级多址协同调度算法. 同 Sabin 与 Yahyapour 等人提出的单址与多址协同算法进行实验对比, 验证了调度模型与算法的有效性与先进性.

关键词 计算网格 并行作业调度 多址协同 资源选择 资源预约 作业回填

1 引言

随着网络与通信水平的不断提高, 并行计算系统由封闭的超级计算机向开放的网络计算系统发展. 网络计算系统(network computing system)是由互联的异构机群通过共享本地资源构成的虚拟计算环境. 网格(the Grid)作为大规模的网络计算系统, 由通讯链路连接多组织和多管理域的计算资源, 构成了Internet级的网络计算环境 [1].

网格系统按照其设计目的广义上可以分为计算网格(computing Grid)、数据网格(data Grid)和服务网格(service Grid)三类. 其中计算网格通过聚合不同组织与机构高性能机群的计算能力, 协同解决孤立机群无法在合理时间内完成的大规模科学计算和工程应用问题 [2]. 虽然计算网格具备处理挑战性应用的基础设施和

收稿日期: 2005-12-25; 接受日期: 2006-06-21

* 国家重点基础研究发展计划(批准号: G2005CB321806)和国家自然科学基金(批准号: 90412001)资助项目

** E-mail: wzzhang@hit.edu.cn

潜力, 但是获得较低的作业平均响应时间和较高的机群利用率必须借助于有效的资源管理, 尤其是多个机群协同工作的任务调度机制.

近年来, 随着Globus, Legion, Condor-G和UNICORE等保障作业可以远程部署到多个异构机群的网格基础设施不断完善, 计算网格环境的多址任务调度模型和协同调度算法研究得到广泛关注. 目前, 国内外计算网格环境下的任务调度策略研究根据研究对象不同可以分为两类 [3]: (1) 应用级任务调度: 由传统机群环境下基于任务图(DAG)的调度问题演化而来. 通过将计算密集型网格应用抽象为粗粒度约束任务图, 采用经济模型和数学规划策略将其映射到网格计算资源, 目的是优化网格环境下某个或某类具体应用的运行性能; (2) 作业级任务调度: 其研究对象是高性能机群上独立作业集的性能优化问题, 是高性能机群作业调度研究在网格环境下的延伸. 本文主要着眼于作业级任务调度, 研究在异构、多机群真实网格环境下如何进行协同作业调度, 以保证较好的平均响应时间和机群利用率.

即便在封闭的超级计算机中, 并行作业调度依然是一个极富挑战性的问题. 而网格计算系统的"多址性、多样性、自治性、成长性"等自然特征对作业调度提出了新的挑战. 因此, 我们有必要指出网格环境下作业调度系统应具备的特征:

(1) 协同性. 计算网格通常由多个组织的机群构成, 例如TeraGrid[1], E-science[2] 和DAS[3]等等. 作业调度系统为避免本地机群对作业请求节点数的限制, 应允许并行作业跨多个机群并发执行. 对于宽度较大的网格作业, 支持多机群运行可以提高作业调度的公平性和系统资源利用率. 同时, 调度系统应具备在多机群间自适应资源选择和映射的能力.

(2) 异构适应性. 网格有别于传统的串行与同构并行机群, 其组成资源多种多样, 多个机群间软硬件资源性能差异显著. 适合网格环境的调度系统必须考虑到机群间的异构性, 并制定适应其变化的调度策略.

(3) 网络适应性. 为支持异构适应性和多机群协同运行, 需要将作业从用户提交的站点传送到其他被调度到的站点. 用户作业(程序和数据)在不同网络环境下传输时间对调度结果有重要影响, 调度系统应具备适应不同网络环境的能力.

(4) 规模可扩展性. 封闭的并行机群计算资源数目有限且固定, 而网格的成长性使资源规模不断扩大, 调度算法自身决策开销无法忽略. 因此算法设计时需要更加慎重地考虑其时间复杂度问题.

本文针对计算网格自然特性引发或加剧的协同、异构、网络性能差异、算法

1) The TeraGrid project. http://www.teragrid.org/, 2001
2) The Europe Data Grid. http://egee-intranet.web.cern.ch/egee-intranet/gateway.html, 2000
3) The Distributed ASCI Supercomputing's Site. http://www.cs.vu.nl/das, 2001

复杂度等作业调度新旧问题, 着眼于真实世界中计算网格系统(机群内部同构高速局域网络互连, 而机群间性能异构低速 Internet 互连), 深入研究多机群任务调度模型与协同调度算法.

本文的主要贡献有两点: 第 1, 针对计算网格环境下并行非抢先空间共享作业的协同调度问题, 提出了多机群协同调度算法框架和两种新的作业级多机群协同算法(最优与贪心多址协同算法). 算法框架由资源选择、资源预约和作业回填三个部分构成, 每个部分相对独立, 易于集成其他相关工作, 进一步提高算法性能. 最优与贪心多址协同算法均满足协同性、异构适应性、网络适应性, 而贪心多址协同算法还具备良好的规模可扩展性; 第 2, 基于一个真实计算网格的多机群与网络性能数据, 将传统机群上的作业模型和工作日志扩展为网格环境负载工作流, 在平均加权响应时间、平均加权等待时间和吞吐率等多个测度与相关工作进行详细对比, 实验结果表明本文提出的算法具有很好的综合性能.

2 相关工作

本节首先详细阐述作业级任务调度的相关研究, 而后进一步指出当前算法的局限性和作业级多机群协同调度深入研究的意义.

2.1 作业级计算网格任务调度

近十年间并行系统中作业调度问题得到了广泛的研究, 历年的并行作业调度策略专题工作组(Workshop on Job Scheduling Strategies for Parallel Processing)[1)] 与异构计算专题工作组 (International Heterogeneous Computing Workshop)[2)]中收录了许多此方面的重要工作. 其中, Feitelson等人综述了面向封闭的超级计算机作业调度的研究状况和存在的问题 [4,5], 并进一步指出面向分布式工作站集群和计算网格的并行作业调度研究是未来的发展趋势[6]. 计算网格环境下并行作业协同调度算法研究大致可分为单址协同与多址协同算法两类.

单址协同算法允许作业在计算网格内选择不同机群运行, 但是不允许任何作业同时占用来自不同机群的资源并发运行(即作业运行不能跨越机群边界). Abawajy 与 Dandamudi 提出了一个动态在线作业调度策略, 通过多机群间作业流动态分配, 提高作业响应时间与系统利用率 [7]. 同时, Sabin等人提出多机群并发请求策略, 将作业请求并发提交到多个独立机群, 一旦开始运行后立即取消冗余请求, 可以有效地降低作业平均响应时间 [8]. 最近, Ernemann等人研究发现,

1) Workshops on Job Scheduling Strategies for Parallel Processing. http://www.cs.huji.ac.il/~feit/parsched/, 2005

2) The International Heterogeneous Computing Workshop. http://www.cs.umass.edu/~rsnbrg/hcw2005/hcw05_prev_workshops.html, 2005

利用分布在不同时区的全球网格机群协同, 采用单址协同和作业回填调度算法, 所有提交作业的平均响应时间较处于一个时区的国家网格可以降低 30%[9]. 此外的一些研究尝试采用作业迁移来解决单址协同中资源选择问题 [10], 引入遗传算法进一步提高网格调度的质量 [11,12].

多址协同算法取消了作业运行不能跨越机群边界的限制, 允许一个并行作业的不同部分并发运行于多个机群中. 多址算法有助于降低宽作业的响应时间和突破封闭机群资源数目的限制, 但是由于当前机群间网络连接具有高延迟、低带宽的特点, 运行时间会有所下降. 然而, Yahyapour 等人的模拟结果表明即使当附加通讯开销达到原程序运行开销的25%时, 同构机群间多址协同算法仍然优于单址协同算法 [13,14]. 而后, 进一步对多址协同算法针对作业分片限制进行优化 [15], 实验证明了在所有机群中计算资源总数恒定的前提下机群数目和每个机群内所含资源数目不会影响多址协同算法的性能 [16]. Epema等人同样在同构网格下, 设计了动态多址协同调度服务框架 [17~19], 实现了基本的调度模块 [20~23], 并评估了多址协同算法中作业结构和大小、机群大小、机群内部与机群间网络通讯比等因素变化对作业平均响应时间的影响 [24~26]. 其他的一些研究工作要求作业具备可塑性(modulability), 允许作业在运行过程中中断与迁移, 便于在多址协同算法引入预约与迁移策略 [27,28].

2.2　当前作业级协同算法局限

当前作业级协同算法的研究存在局限性, 多数算法仅仅满足计算网格自然特性(协同性、异构适应性、网络适应性和规模可扩展性)中的部分特征. 其中, 单址协同算法 [7~12]多数由传统的封闭管理域异构计算系统作业调度算法扩展而来, 忽视了多址协同对降低作业响应时间和提高资源使用率的影响, 不具备完备的协同性. 虽然许多研究工作提出了多址协同问题和调度算法 [13~28], 然而这些工作假设所有高性能机群均为同构, 与真实的网格计算环境有较大的差距, 在异构适应性方面有所欠缺. 此外, 部分工作 [11,12]忽略了作业传输开销和算法复杂性, 不具备网络适应性和算法规模的可扩展性, 因此难以在大规模网格调度环境中得到实际应用.

针对上述缺陷, 本文研究着眼于以下两点: 其一, 依托正在建设的“973 国家虚拟计算环境试验床”中机群和网络性能信息, 面向跨管理域运行的并行网格作业, 建立计算网格模型、作业调度系统模型和多址执行性能模型. 其二, 设计具备协同、异构适应、网络适应和规模可扩展等网格自然特征的作业级协同调度算法, 在广域网互连的异构计算网格情景下, 研究并行作业多址协同算法的性能.

3 网格作业调度模型

3.1 基本概念

定义 1 节点(node): 亦作资源. 为具有独立处理器和存储器的主机, 是调度算法资源管理的最小单位.

定义 2 站点(site): 亦作址. 为紧耦合的同构节点集合, 通常为高性能计算机群.

定义 3 作业(job): 亦作任务. 是用户提交的资源请求. 作业独立且作业间不存在约束关系. 每个作业需要 $N(N\geqslant 1)$个节点并发执行. 作业请求节点数称为作业的宽度, 作业运行时间称为作业的长度.

定义 4 协同(co-allocation): 作业不受网格用户提交位置局限, 在计算网格内部不同站点或站点集合上并发执行. 单址协同即作业可在计算网格内选择本地站点或远程站点运行, 但是不允许任何作业同时占用来自不同站点的节点资源, 选择本地站点执行称为本地单址, 而选择远程站点执行称为异地单址. 多址协同取消了作业运行不能跨越站点边界的限制, 允许一个并行作业的不同部分并发运行于多个站点中. 若作业在 m 个站点上并发执行, 称该作业为 m-址运行.

定义 5 异构因子(heterogeneous factor): 定义 h_i 描述不同站点计算能力与负载差异对作业执行时间的影响程度. 其中 i 表示站点. 异构因子受站点的计算能力、体系结构和存储方式等多个因素影响, 难以通过理论计算确定站点异构因子. 然而, 可以通过提取标准测试程序的代码“骨骼”, 在不同站点上运行, 获得相对某站点的异构因子.

定义 6 多址因子(multi-site factor): 定义 $p_{1,2,\cdots,n}$ 描述用户作业多址运行时, 站点间由于通讯同步等额外开销对执行时间的影响程度, 其中 n 表示站点. 影响多址因子的因素主要包括站点间带宽、延迟, 不同站点的拓扑连接情况. 多址因子同样可以采用标准测试程序在真实环境下运行的方法获得.

3.2 作业级多址协同调度模型

通过引入网格作业元调度器和作业多址协同调度机制, 网格环境下的作业级多址协同调度模型如图 1 所示, 主要元素包括网格元调度器、本地调度器、站点、作业和网格用户.

网格元调度器采用集中式架构, 掌握所管理站点及其内部节点的运行状态. 用户作业请求通过全局作业管理队列提交到元调度器中. 元调度器中协同调度算法负责将作业映射到不同站点. 事实上, 此模型中的元调度器和全局队列是性能瓶颈, 具有易失效性, 可以采用汇集作业请求的分布式信息服务器和协作决策的分布式网格调度器替代. 目前绝大多数相关工作采用集中式架构元调度器评

估网格作业调度算法性能 [7~26].

本地调度器分布在各站点, 作用为传输和执行作业, 并不影响全局调度策略.

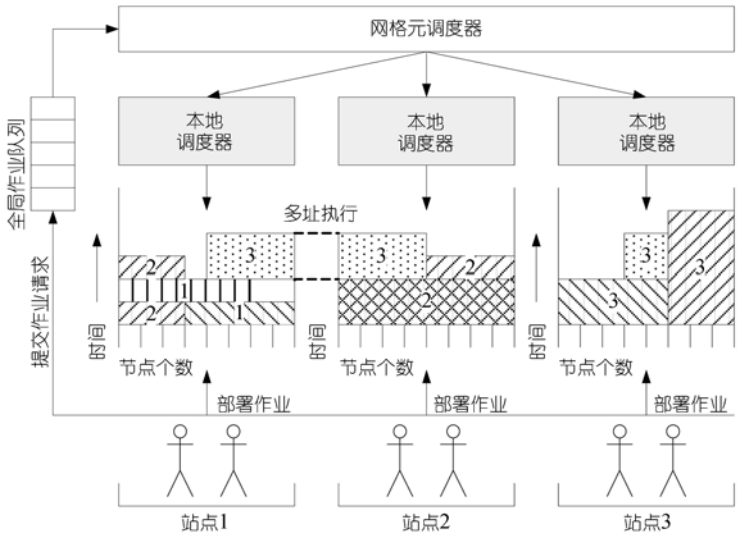

图 1　作业级多址协同调度模型

网格站点通常是高性能机群. 站点内部节点同构, 由高速总线互连, 连接速率多为每秒千兆字节以上. 站点间由于节点性能和负载不同, 整体性能存在差异, 且通过广域网互连, 传输低速, 不稳定. 站点间可以交换用户提交的任务, 共享计算资源. 文献 [9,13~26]中假设所有站点的节点均为同构, 本文网格站点模型着眼真实的计算网格环境, 强调站点内部节点同构而站点间节点异构.

网格作业是调度的原子对象, 不能分解为子作业. 作业宽度固定, 作业在运行过程中不允许被抢先(低带宽站点间作业迁移开销较大). 作业对节点的使用是独占式的, 运行过程中作业不具备可塑性. 在站点数量满足作业宽度的前提下, 作业可以在任意站点的任意子集上执行, 可以通过站点内部与站点间网络传递信息, 但并行作业对所分配节点必须同时占用与释放.

网格用户以在线作业流方式提交作业, 调度器对未提交的作业无先验知识. 用户提交后即不再干预, 直至作业运行结束或超时被取消.

作业级多址协同调度的基本流程为:首先由用户在某一个或多个站点部署作业. 部署完毕后, 用户向网格元调度器发出作业提交请求. 元调度器将作业请求维持在等待作业队列中, 并根据当前资源占用情况, 决定该作业单址协同或多址协同执行, 确定被调度的作业放入执行队列中. 如果该作业的调度结果在远程单址或多址执行, 则需将作业从提交站点传输到目的站点. 作业的传输和执行由站

点本地分布式调度器控制, 并将执行结果反馈给元调度器, 最终通知用户运行结果.

3.3 作业多址运行性能模型

保证在异构、多址的情况下作业运行时间的准确预估是调度算法设计与评估的基础, 用户可以采用Sodhi和Subhlok[29]提出的基于代码"骨骼"的作业运行时间预估方法获取程序在某标准机群上运行时间, 此方面研究超出了本文的探讨范畴. 本节给出作业在本地单址、异地单址和多址协同情况下性能预估.

设作业 j 预估运行时间为 T_{origin}(用户给出的作业预估运行时间统一为面向IBM SP 等某标准超级计算机的运行时间). 由于站点的异构性, 作业在单个站点 i 上的运行时间受异构因子 h_i 影响, 本地单址情况下最终预估执行时间 T_i 为

$$T_i=h_i\times T_{\text{origin}}. \tag{1}$$

当本地作业迁至其他单址运行时, 需考虑作业传输开销. 设作业的大小为 s, 在站点 i 上提交, 最终调度至站点 j 执行. 站点 i 和 j 的网络带宽与延迟为 $b_{i,j}$ 和 $l_{i,j}$, 则作业从站点 i 传输至站点 j 的网络开销为 $t_{\text{trans}(ij)}=s\times b_{i,j}+l_{i,j}$, 其中 $b_{i,j}$ 为网络带宽的倒数. 若 $i=j$ 则 $t_{\text{trans}(ij)}=0$. 因此, 作业在站点 i 提交, 在站点 j 异地单址执行的最终预估执行时间 $T_{i->j}$ 为

$$T_{i->j}= T_i\times h_j/h_i + t_{\text{trans}(ij)}. \tag{2}$$

将本地作业迁至多址运行时, 除传输开销外还要增加由于不同站点作业间通信与同步引入的额外运行开销, 由多址因子 $p_{j,k\cdots m}$ 表示. 同时, 由于大多数多址作业并发时需保持同步, 所以其运行时间应不短于性能最低的站点独立运行该作业的时间. 由此, 若作业 j 在站点 i 被提交, 并被调度到站点 $j,k\cdots m$ 上, 多址执行最终预估执行时间 $T_{i->(j,\,k\cdots m)}$为

$$T_{i->(j,\,k\cdots m)}= T_i\times p_{(j,\,k\cdots m)}\times\max(h_j, h_k,\cdots, h_m)/h_i + \max(t_{\text{trans}(i,\,j)}, t_{\text{trans}(i,\,k)},\cdots, t_{\text{trans}(i,\,m)}). \tag{3}$$

4 作业级多址协同调度算法

基于作业级多址协同调度模型和性能模型, 本节首先提出了作业级多址协同调度算法的基本框架. 然后, 分别阐述了框架中资源选择、资源预约与作业回填策略, 提出了最优与贪心多址协同调度算法. 最后, 给出了作业级多址协同调度算法族和时间复杂性.

4.1 基调度策略

由于先来先服务(FCFS)在线调度策略可保证一定程度的公平性、易于实现, 同时具有较低的时间复杂度 [30], 被多数相关工作作为作业级协同调度的基算法. 因此, 本文对全局队列中采用FCFS调度策略. 本地用户部署作业之后, 按照作业

请求的提交时间依次放入全局作业队列. 元调度器根据作业提交时间, 从队列头开始依次为队列中的作业分配资源. 但是, 当用户提交宽度较大的作业时, 严格遵循FCFS会导致队列中窄作业阻塞, 空闲节点增多, 降低系统的吞吐率 [31]. 回填(backfilling)策略通过对当前宽作业进行资源预约, 并移动队列后宽度较小的作业至这些空闲节点, 在不影响当前作业启动时间的基础上有效提高系统吞吐率 [32, 33]. 因此, 基调度策略由backfilling增强的FCFS构成.

基调度策略可分为资源选择、资源预约与作业回填策略 3 个部分. (1) 资源选择: 是多址协同调度的核心, 不仅在 FCFS 的基调度策略中得到应用, 还应用到资源预约以及作业回填策略中, 是本文研究的重点. 资源选择的目的是确定在不同资源组合情况下作业的运行时间, 包括立即单址、立即多址、预约单址和预约多址执行. (2) 资源预约: 在当前资源无法满足作业宽度或作业虽可立即运行但其执行时间大于预约执行时间的情况下, 需采用预约策略对该作业所需节点资源进行预约. 当部分资源可用时, 不再分配给其他的任务, 直到该作业所需资源得到满足后投入运行. 预约节点集选取的依据是根据各节点正在运行作业的预估运行时间和节点的最早可用时间. (3) 作业回填: 在被预约节点当前运行作业结束后, 预约作业运行之前的空隙时间影响了系统利用率. 为既不影响已经预约了节点的大作业运行又能提高系统的利用率, 一方面通过预约节点选取策略减少这些空隙的存在, 另一方面采用回填思想, 从作业队列中选取估计运行时间小于空隙时间的作业投入运行. 作业级多址协同调度算法具体描述如下:

算法 1　作业级多址协同调度算法框架.

输入:　(1)作业等待队列, (2)站点集合

输出:　(1)映射结果

中间变量:　(1) Δt, 调度间隔; (2)*Currentjob*, 作业队列中未调度首作业; (3) *status*, 表示两种资源选择结果: 立即执行与预约执行.

步骤 1　初始化. 若作业队列为空, 等待 Δt 调度间隔, 重新检测作业队列状态; 若队列非空, 从作业队列与本地调度器中分别收集作业请求信息与站点状态信息.

步骤 2　映射. 针对当前队列中所有未分配资源的作业请求, 将当前作业队列首作业赋予 *Currentjob*;

(a) 资源选择. 调用资源选择算法 ResourceSelection(*CurrentJob*), 根据选择结果(立即执行与预约执行)对 *status* 赋值;

(b) 资源预约与回填. 若 *CurrentJob* 的 *status* 为可以立即在单址或多址运行, 则通知本地调度器传输并运行作业; 若 *status* 为预约执行, 则调用资源预约算法 Reserve (*CurrentJob*)和作业回填算法 Backfilling(*CurrentJob*);

(c) 更新作业队列与站点状态.

步骤 3 返回映射结果并从步骤 1 重新执行.

4.2 资源选择策略

本节给出两种资源选择策略——最优多址资源选择算法和贪心多址资源选择算法. 资源选择基于多址作业性能模型与站点信息确定当前作业立即执行或预约执行, 同时返回资源映射集合.

最优资源选择策略试图考察所有的资源组合包括(立即单址, 立即多址和预约单址、预约多址). 其中预约指“即使该作业能够获得足够的节点得以执行, 仍然保持在作业队列中以等待具有更高性能的节点得到释放”的策略, 从而确定该作业最早完成时间. 描述见算法 2. 最优资源选择策略不具备规模可扩展性, 为此我们提出了一种具有多项式时间复杂度的贪心资源选择策略. 该策略考虑网格环境站点异构性, 首先根据站点的异构因子 h 对所有站点进行排序, 然后选择出异构因子最小的前 m 个站点计算 m-址执行时间. 描述见算法 3.

算法 2 最优多址资源选择算法 .

输入: (1) *Currentjob*, 作业队列中未调度首作业; (2) 站点集合

输出: (1) *status*, (2) 资源映射集合

中间变量: (1) $T_{i->j}$, 作业在站点 i 提交运行于站点 j 的执行时间预估;

(2) $T_{i->(j,\,k\cdots m)}$, 作业在站点 i 提交运行于站点 $j,k\cdots m$ 的执行时间预估;

(3) $T'_{i->j}$, 作业在站点 i 提交预约运行于站点 j 的执行时间预估;

(4) $T'_{i->(j,\,k\cdots m)}$, 作业在站点 i 提交预约运行于站点 $j, k\cdots m$ 的执行时间预估;

(5) $T_{i,\,\mathrm{available}}$, 站点 i 的最早可用时间.

步骤 1 资源限制检查. 若 *Currentjob* 请求的节点数目超过各站点节点数目的总和, 则丢弃此作业, 程序返回; 若 *Currentjob* 请求的节点数目超过当前所有空闲节点数目总和, 置 *status* 状态为预约标志, 程序返回; 若不满足上述两种情况, 则执行步骤 2.

步骤 2 计算立即执行时间. 枚举单址与多址执行 *Currentjob* 的资源映射组合.

(a) 对所有空闲节点大于 *Currentjob* 作业请求的单个站点, 计算 $T_{i->j}$;

(b) 对所有空闲节点大于 *Currentjob* 作业请求的多址组合, 计算 $T_{i->(j,\,k\cdots m)}$.

步骤 3 计算预约执行时间. 枚举单址与多址预约执行 *Currentjob* 的资源映射组合.

(a) 对所有站点, 计算 $T_{i,\,\mathrm{available}}$;

(b) 对节点总数大于 *Currentjob* 作业请求的全部单个站点, 计算 $T'_{i->j}\leftarrow T_{i->j}+T_{j,\,\mathrm{available}}$;

(c) 对节点总数大于 *Currentjob* 作业请求的全部多址组合, 计算 $T'_{i->(j,\,k\cdots m)}$

$\leftarrow T_{i->(j,\,k\cdots m)} + \max(T_{j,\,\text{available}}, T_{k,\,\text{available}}, T_{m,\,\text{available}})$.

步骤 4　若最小作业完成为 $T_{i->j}$ 或 $T_{i->(j,\,k\cdots m)}$, 则置 *status* 状态为立即执行标志, 并返回 *Currentjob* 映射资源集合; 否则, 置 *status* 状态为预约执行标志, 并返回 *Currentjob* 映射资源集合.

算法 3　贪心多址资源选择算法.

输入:　(1) *Currentjob*, 作业队列中未调度首作业; (2) 站点集合

输出:　(1) *status*, (2) 资源映射集合

中间变量:　(1) $T_{i->j}$, 作业在站点 i 提交运行于站点 j 的执行时间预估;

(2) $T_{i->(j,\,k\cdots m)}$, 作业在站点 i 提交运行于站点 $j, k\cdots m$ 的执行时间预估;

(3)　$T'_{i->j}$, 作业在站点 i 提交预约运行于站点 j 的执行时间预估;

(4)　$T'_{i->(j,\,k\cdots m)}$, 作业在站点 i 提交预约运行于站点 $j, k\cdots m$ 的执行时间预估;

(5)　$T_{i,\,\text{available}}$, 站点 i 的最早可用时间.

步骤 1　资源限制检查. 若 *Currentjob* 请求的节点数目超过各站点节点数目的总和, 则丢弃此作业, 程序返回; 若 *Currentjob* 请求的节点数目超过当前所有空闲节点数目总和, 置 *status* 状态为预约标志, 程序返回; 若不满足上述两种情况, 则执行步骤 2.

步骤 2　计算立即执行时间. 贪心选择单址与多址立即执行 *Currentjob* 的资源映射组合.

(a) 按照异构因子 h 升序构成长度为 N 的站点队列;

(b) 从排序后的站点队列中选择前 l 个站点(l 初值为 1), 若此 l 个站点的空闲节点总数小于 *Currentjob* 作业请求节点数, 删除空闲节点数目最小的站点, 加入队列中下一个站点. 当空闲节点数满足 *Currentjob* 作业请求时, 计算 $T_{i->j}$ 或 $T_{i->(j,\,k\cdots m)}$; $l=l+1$, 重复(b)至 $l>N$.

步骤 3　计算预约执行时间. 贪心选择单址与多址预约执行 *Currentjob* 的资源映射组合.

(a) 对所有站点, 计算 $T_{i,\,\text{available}}$;

(b) 从排序后的站点队列中选择前 l 个站点(l 初值为 1), 若此 l 个站点的节点总数小于 *Currentjob* 作业请求节点数, 删除节点数目最小的站点, 加入队列中下一个站点. 当节点数满足 *Currentjob* 作业请求时, 计算 $T'_{i->j} \leftarrow T_{i->j} + T_{j,\,\text{available}}$ 或 $T'_{i->(j,\,k\cdots m)} \leftarrow T_{i->(j,\,k\cdots m)} + \max(T_{j,\,\text{available}}, T_{k,\,\text{available}}, T_{m,\,\text{available}})$; $l=l+1$, 重复(b)至 $l>N$.

步骤 4　若最小作业完成为 $T_{i->j}$ 或 $T_{i->(j,\,k\cdots m)}$, 则置 *status* 状态为立即执行标志, 并返回 *Currentjob* 映射资源集合; 否则, 置 *status* 状态为预约执行标志, 并返回 *Currentjob* 映射资源集合.

4.3 资源预约策略

与资源选择策略相同, 资源预约也分为最优资源预约与贪心资源预约, 这两种预约算法分别对应最优与贪心资源选择的步骤 3 部分. 确定预约策略后, 元调度器通知本地调度器进行资源保持.

事实上, 计算网格环境下的多址预约与传统封闭超级计算机群预约有较大的差异. 由于跨越多个站点和管理域, 缺乏不同管理域调度策略、优先级、工作负载的协同信息, 因此本地调度器间、本地调度器与网格元调度器间难以真正协同预约. 采用Snell等人提出的高级资源预约方法可以规避此类问题 [28].

4.4 作业回填策略

回填算法对等待作业队列中被预约节点作业后面的作业进行调度. 为保障调度系统的公平性, 基调度算法采用 First-Fit, 即调度系统依次考察队列中已预留资源作业后面的作业. 若该作业可以回填, 则回填执行; 若该作业无法回填, 则继续等待. 回填策略也可以采用最优与自适应贪心回填策略确定回填目标节点集. 回填分为 Easy 和 Conservative 两种, 描述如算法 4 所示.

算法 4 回填策略

输入: (1) *Currentjob*, 作业队列中未调度首作业; (2) 站点集合

输出: (1) 回填作业集合, (2) 回填作业分配的映射资源集

中间变量: $T_{i,\,\mathrm{available}}^{j}$ 站点 i 中节点 j 的最早可用时间

步骤 1 初始化. 计算所有站点 i 中的空闲节点 j 的最早可用时间 $T_{i,\,\mathrm{available}}^{j}$, 包括已经被其他作业预约的节点. 根据 $T_{i,\,\mathrm{available}}^{j}$ 对节点排序.

步骤 2 Conservative. 若调度器采用 Conservative 作业回填策略, 对 *Currentjob* 之后的所有作业调用 4.3 节的资源预约策略, 只有不影响任何已经预约和正在执行作业执行时间的后续作业, 才会被加入回填作业集合, 并更新 $T_{i,\,\mathrm{available}}^{j}$, 执行步骤 4.

步骤 3 Easy. 若调度器采用 Easy 作业回填策略, 对 *Currentjob* 之后的所有作业调用 4.3 节的资源预约策略, 只要不影响 *Currentjob* 执行时间的后续作业, 均会被加入回填作业集合, 并更新 $T_{i,\,\mathrm{available}}^{j}$, 执行步骤 4.

步骤 4 返回回填作业集合和映射资源集.

4.5 作业级多址协同调度算法族

4.1 节算法框架分为资源选择、资源预约与作业回填 3 个部分, 每个部分均可以采用最优与贪心多址资源选择算法, 而且作业回填部分还可以选择不同的

backfilling 策略. 因此, 多址协同调度算法框架可以衍生出一族多址作业协同调度算法. 本文中对算法名称作如下约定:若作业级多址协同调度算法框架中资源选择、资源预约和作业回填均采用最优资源选择策略, 则称此算法为作业级最优多址协同调度算法, 算法复杂度为 $O(2^N)$; 若采用贪心资源选择策略则称为作业级贪心多址协同调度算法, 算法复杂度为 $O(N^2)$, N 为站点数目.

虽然基于 FCFS 和 backfilling 的基调度算法在传统的超级计算机和作业级计算网格调度算法中得到广泛应用, 然而本文提出的基于最优与贪心资源选择的多址协同算法具有 2 个优点: (1) 允许作业多址执行, 有助于提高系统利用率, 降低宽作业的响应与执行时间. (2) 同构假设下协同算法由于作业传输和同步惩罚, 要求遵循“本地单址执行>异地单址执行>多址执行”进行资源选择. 异构环境下由于站点性能差异, 此偏序规则不再适用, 本文的算法自适应选择较好的资源组合以满足异构适应性, 兼顾网络适应性, 同时作业级贪心多址协同调度算法具备规模可扩展性.

5　仿真实验结果及性能分析

5.1　仿真实验环境及算法设置

计算网格实验所模拟的硬件环境以国家重点基础研究发展计划支持的“虚拟计算环境试验床”的基础设施为蓝图. “虚拟计算环境试验床”是由国家计算机网络应急技术处理协调中心(CNCERT/CC)和哈尔滨工业大学(HIT)协作建设, 以国家计算机网络应急技术处理协调中心遍布全国 31 个省份的网络基础设施及计算资源为基础, 对分布自治资源进行集成和综合利用, 构建起一个开放、安全、动态、可控的大规模虚拟计算环境实验平台, 研究并验证虚拟计算环境聚合与协同机理. 当前正在建设的 3 个站点主机总数共计 224 台, 资源配置情况如表 1 所示. 仿真实验以“虚拟计算环境试验床”站点与节点规模为基础, 测试不同异构因子、多址因子与网络性能下调度算法的效果.

表 1　“虚拟计算环境试验床”配置

站点	地理位置	机型	节点数	每节点处理器	主存	址内网络性能		址间网络性能	
						带宽/Mb・s^{-1}	延迟/ms	带宽	延迟
CNCERT/CC	北京	曙光 4000L	128	2*Xeon 2.4 GHz	2GB	950~960	0.2	平均带宽 50~150 Kb/s;平均延迟 550~600ms	
HIT	哈尔滨	曙光服务器	32	2*Xeon 2.4 GHz	2GB	900~950	0.2		
CNCERT/CC	上海	Beowulf cluster	64	2*AMD Athlon 1.5 GHz	2GB	90~95	0.15		

基于虚拟计算环境试验床的体系结构与性能信息, 我们开发了离散事件仿真工具包研究多址协同算法. 设模拟环境中站点数目为 3 个, 共 224 个节点, 分别为 CNCERT/北京(128 节点)、CNCERT/上海(64 节点)、HIT/哈尔滨(32 节点). 用两组异构因子 h_i(北京 1.0, 上海 1.0, 哈尔滨 1.0), (北京 1.4, 上海 1.0, 哈尔滨 0.6) 分别表示站点异构差异较小与较大情况. 异构差异较大时, 最快站点为最慢站点性能两倍. 址内带宽 1000 Mbps, 址间带宽 10 Kbps~10 Mbps. 多址因子 $p_{j,k\cdots m}$ 从 1.0 变化至 1.6. 实现作业级最优与贪心多址协同调度算法时, 设置调度间隔 Δt 为 1 s.

5.2 计算网格环境作业工作流

准确地测试和验证调度算法依赖于选取有代表性的负载工作流. 然而, 由于目前缺乏真实网格环境下的负载工作日志和负载模型, 多数网格调度研究者通过将传统机群上的作业模型和工作日志加以扩展作为网格环境负载工作流. 本文采用Feitelson等人收集的Cornell Theory Center高性能计算机群IBM RS/600 SP 共 11 个月并行负载工作日志 [1)], 对其进行改造扩展以适应计算网格模型.

从 CTC 负载日志中提取 10000 条输入作业加以改造, 包含的原始属性为: 作业提交时间(SubmitTime), 作业预估执行时间(EstimateTime), 作业实际执行时间(FinishTime). 这三个时间属性的单位是秒, 其中 SubmitTime 的数值是调度器开始运行到当前时间的秒数.

(1) SubmitTime: 作业提交时间. 这个属性是用户将作业在本地站点上部署之后, 向中央调度器提交作业请求的时间戳.

(2) EstimateTime: 作业预估执行时间. 这个属性由用户给出, 是用户对自己提交作业运行时间的估计值, 是用户作业实际运行时间的上限. 调度器将根据这个属性对用户作业进行调度.

(3) FinishTime: 作业实际执行时间. 在模拟实验中, 模拟器根据这个属性判断用户作业是否已经执行完毕. 为保证公平性, 若实际执行时间超出了预估执行时间的上限, 未完成的用户作业将被站点本地调度系统强制结束.

需要加以改造的 CTC 工作负载日志的属性包括:

(1) RequestResource: 作业需要节点数, 即作业宽度. 原始 CTC 机群环境共有 512 个节点, 而在虚拟计算环境试验床中规模最小的 HIT 站点为 32 个节点. 为满足此计算网格环境中作业在 3 个站点间调度的需要, 易于单址与多址调度算法比较, 将最大宽度作业设置为 32.

需要扩展的属性包括:

1) Feitelson D. Parallel Workloads Archive. URL: http://www.cs.huji.ac.il/labs/parallel/workload/, 2002

(1) SubmitSite: 作业提交站点. 在多址作业协同模型中, 用户首先将其作业部署在本地站点上. 因此, 调度器需要知道该用户作业部署的站点才能加以调度. 构造的作业负载中, 这个属性的数值采用随机函数在 3 个站点中产生.

(2) Size: 作业大小. 在多址作业协同模型中, 部署在一个站点上的作业可能需要传输至其他站点执行. 在站点之间通过 Internet 传输作业, 调度系统需要作业大小和网络带宽来确定作业传输时间. 构造的作业负载中, 这个属性值在 1~100 MB 间随机产生. 作业宽度低于 16 的作业称为窄作业, 宽度在 17~32 之间的作业称为宽作业;将作业长度低于 1 h 的作业称为短作业, 长度在 1 h 以上的作业称为长作业.

除以上与调度算法性能相关的属性之外, 作业工作负载中还包括一些附加属性, 它们是用户对运行作业的节点的约束. 例如, 节点硬件系统结构、操作系统、处理器主频、主存储器及外存储器容量等. 由于本文主要针对作业级的调度系统性能进行研究, 因此这些附加属性不在本文考虑范围之内.

下面是经过扩展的 CTC 作业工作负载日志片断:

表 2　计算网格环境作业负载工作日志片断

提交时间	预估执行时间/s	实际执行时间/s	请求节点数	作业大小	提交站点
406862	600	647	4	51	CNCERT/北京
406983	64800	228	7	1	CNCERT/北京
408355	600	417	16	38	HIT/哈尔滨
410065	23500	18796	24	42	CNCERT/上海

同时, 我们还在改造后 CTC 日志基础上提高宽作业所占比重, 以检测调度算法在各种条件下性能. 作业负载描述如下:

Job Workload 1: 改造后的 CTC 工作负载;

Job Workload 2: CTC 90%的作业宽度在 11~32 之间.

5.3　性能评价指标

定义 7　作业平均加权响应时间(AWRT):

$$AWRT = \left(\sum_{j \in \text{Jobs}} Cost_j \times (T_{j,\text{end}} - T_{j,\text{submit}}) \right) \Big/ \sum_{j \in \text{Jobs}} Cost_j .$$

定义 8　作业平均加权等待时间(AWWT):

$$AWWT = \left(\sum_{j \in \text{Jobs}} Cost_j \times (T_{j,\text{start}} - T_{j,\text{submit}}) \right) \Big/ \sum_{j \in \text{Jobs}} Cost_j .$$

定义 9　系统平均利用率(AU):

$$AU = \sum (N_{\text{used}} / N_{\text{total}}) \Big/ Num .$$

其中 $T_{j,\text{end}}$, $T_{j,\text{start}}$ 和 $T_{j,\text{submit}}$ 分别表示作业 j 的结束时间、等待时间和提交时间. 作业开销 $Cost_j$ 为作业长宽之积, 即 $W_j\times(T_{j,\text{end}}-T_{j,\text{start}})$. W_j 为作业宽度, 即作业请求资源的数目. N_{used} 表示每秒有作业运行的节点数目, N_{total} 表示节点总数, Num 为检测总次数.

两个重要的经典在线调度测度为作业平均相应时间(ART)和作业平均等待时间(AWT). 但是, 这两个测度在本文的多址作业调度模型中将小作业与大作业同等考虑, 缺乏对资源的消耗情况的度量. 因此通过对ART与AWT进行加权得到作业平均加权响应时间(AWRT)和作业平均加权等待时间(AWWT), 这两个测度在作业级网格任务调度得到广泛应用 [9,13~16,27]. 其中, AWRT与调度系统的作业吞吐率成反比, 从调度系统的角度反映调度算法性能. AWWT描述用户作业响应时间, 从用户角度来反映对调度系统的满意程度. 因此, 仿真实验将主要采用这两个测度, 同时考察系统平均利用率(AU), 从系统与用户角度验证算法性能.

5.4 实验结果及分析

本节首先将最优与贪心多址协同算法与Yahyapour和Sabin等人提出的两种单址协同算法 [8,9]进行性能比较, 而后与Yahyapour等人提出的多址协同算法 [13~16]对比, 评测本文算法的有效性. 另外, 我们还考察了不同的网络性能与回填算法对多址协同算法的影响.

5.4.1 多址与单址协同算法

多址协同运行可以充分利用资源碎片, 从而获得较高的资源利用率和较高的总体调度性能; 然而由于多址作业跨站点进程间通信带来的额外时间开销加大了系统整体的 AWRT 值. 支持多址运行究竟给调度系统的总体性能带来的是正面影响还是负面影响是本节需要研究的问题.

首先考察贪心多址协同算法和 Yahyapour 单址协同算法. 作业日志为 JobWorkload1 和 JobWorkload2, 带宽因子为 10.0, 多址因子从 1.0 变化到 1.6. 分两种情况: (1) 当站点异构差异较小时, 两者 AWRT 如图 2(a)和(b)所示. 由于单址协同算法中作业并不受多址因子的影响, 所以 AWRT 值保持水平. 多址协同中作业受到多址因子的影响, 其 AWRT 值随多址因子的变大呈上升趋势. JobWorkload1 仅当多址运行额外时间开销不超过原作业时间的 10%时, 多址协同的性能优于单址协同. 而对 JobWorkload2 当多址运行额外开销小于原作业执行时间的 40%时, 多址协同性能都要优于单址协同. 原因在于 JobWorkload1 作业的宽度分布在 1~32 之间且窄作业居多, 单址协同使资源碎片不能被跨站点作业利用, 但可被单址运行的窄小作业利用. 而 JobWorkload2 中宽作业居多, 没有足够多的窄小作业来填补大作业留下的资源空隙, 多址算法显示出较大的优势. 同时,

如图 2(c)所示, 采用 JobWorkload1 两种算法的系统利用率均为 95%左右, 而采用 JobWorkload2 多址协同算法的系统利用率不变而单址算法降低到 83%. (2) 当系统资源异构性比较大时(北京 1.4, 上海 1.0, 哈尔滨 0.6), 两种算法的 AWRT 如图 3 所示. 多址协同算法性能相对单址协同有所下降, JobWorkload2 在多址因子为 1.0 时, AWRT 性能仅高出 4%; 当多址因子超过 1.2 时, 多址调度性能开始低于单址协同. 其主要原因是多址作业同步开销导致多址运行作业的执行时间应不短于

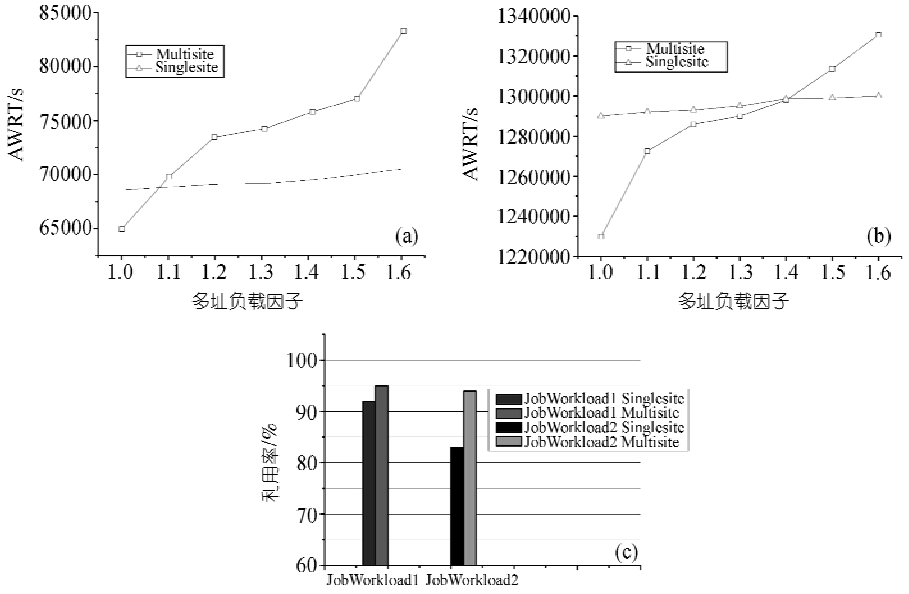

图 2　异构差异较小时多址协同算法和 Yahyapour 单址协同算法性能比较

(a) JobWorkload1 AWRT; (b) JobWorkload2 AWRT; (c) JobWorkload1/JobWorkload2 AU

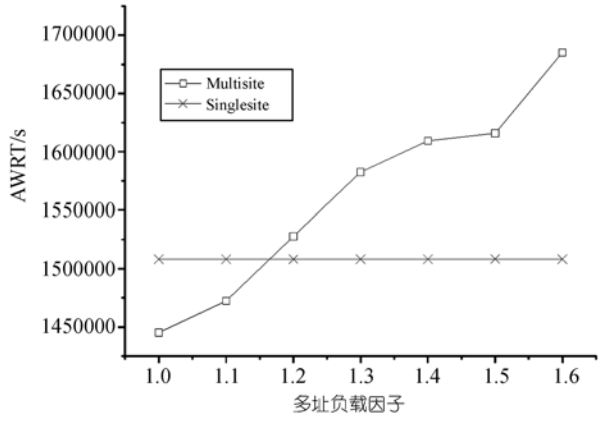

图 3　异构差异较大时多址协同算法和 Yahyapour 单址协同算法性能比较

该作业在性能最低的站点上单址运行的时间. 资源异构性较大的情况下, 多址调度的作业实际上是将性能高的资源“降低”到性能最低资源的标准. 资源异构性越大, 多址调度使系统总体性能降低的幅度越大.

其次考察贪心多址协同算法和 Sabin 单址协同算法. Sabin 等研究者于 2003 年进行的工作中, 考虑到组成网格环境的各站点间的异构性以及当前负载, 设计了一个适用于异构网格环境的调度系统. Sabin 的调度算法的基本思想是采用单址资源选择策略, 作业等待队列是分布式的. 每一个站点的本地调度器维持着一个本地作业等待队列. 网格用户在本地部署作业后, 调度系统会将作业传输到网格内的所有站点上, 并在各站点提交作业请求. 当某站点的本地调度器首先发现本地作业队列中某一作业可以运行时, 该调度器会通知其他站点的调度器将它们本地相同作业从等待队列中清除.

采用本文提出的贪心多址协同与 Sabin 算法进行比较. 在网络带宽分别为 1 MBps, 100 和 50 KBps 的状况下, 采用 JobWorkload1 作业工作负载, 分别考察异构因素较小和较大两种情况, 两种算法的运行结果如图 4 所示. 贪心多址协同性能与 Sabin 算法接近. 但 Sabin 算法处理作业宽度的能力不如多址协同调度算法. 多址协同所允许作业的最大宽度为所有站点数的节点数加和. 然而由于 Sabin 模型中无法支持多址调度, 其所允许作业的最大宽度为单个站点中最大的节点数. 随着网格规模的扩大, 加入到网格环境的站点越多, Sabin 模型允许的最大作业宽度与网格规模无关, 只跟具有最大节点数的站点有关, 成为系统瓶颈.

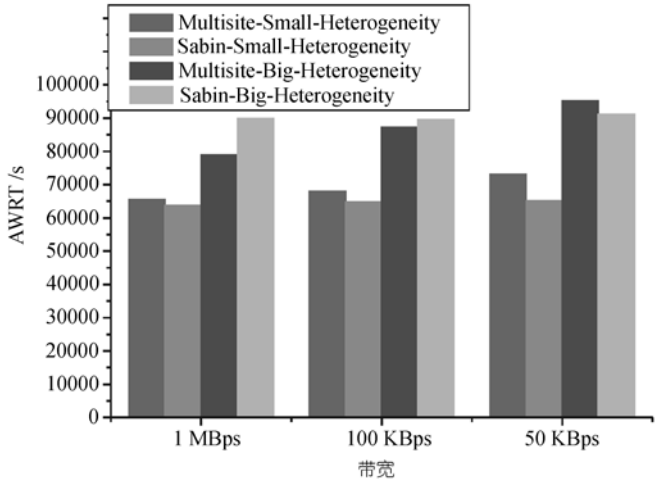

图 4 多址协同与 Sabin 单址协同算法性能比较

综上所述, 多址协同算法的性能受到作业负载类型、资源异构性、队列特征等诸多因素的影响. 相对单址算法, 多址算法性能随作业宽度增加、平均加权响

应时间降低而系统利用率升高, 但是随站点间异构差距加大多址协同算法性能会有所下降. 此外, 单址调度算法作业处理宽度明显低于多址算法, 宽度高于任何单址节点总数的作业会被全部丢弃, 难以满足网格用户需求.

5.4.2　最优、贪心多址与 Yahyapour 多址协同算法

本节通过仿真实验对比最优、贪心多址协同算法和 Yahyapour 等人提出的多址协同算法性能差异. 首先考察 JobWorkload1 在带宽因子为 10.0(100 KBps), 站点异构差异较小(均为 1.0), 多址因子从 1.0 变化到 1.6 过程中, 三种算法的 AWRT 与 AWWT 值如图 5 所示. 其次, 当站点异构差异较大时(北京 1.4, 上海 1.0, 哈尔滨 0.6), 算法的 AWRT 与 AWWT 值如图 6 所示. 算法回填方式均为 Conservative.

如图 5 所示, 站点异构差异较小时, 最优与贪心多址协同算法性能显著优于 Yahyapour 的多址算法, 且随着多址因子的增大优势更加明显. 在异构因子达到 1.6 时, 相对 Yahyapour 多址协同算法, 最优多址协同算法的 AWRT 降低近 100%, 而 AWWT 性能则要降低 2 倍. 这是由于随着多址因子增加, 多址运行开销变大, 采用本文算法中自适应资源选择可以避免一部分作业多址运行而“节省”下来较多的时间. 此外, 贪心多址协同算法的资源选择策略是采用启发式策略优先选取

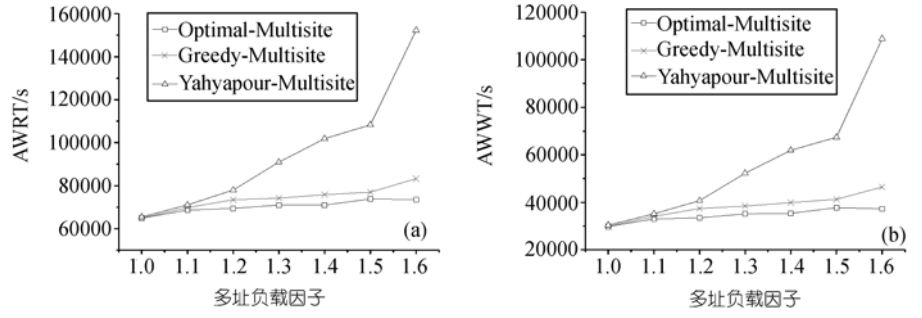

图 5　异构差异较小时最优、贪心多址和 Yahyapour 协同算法 AWRT 与 AWWT 性能比较

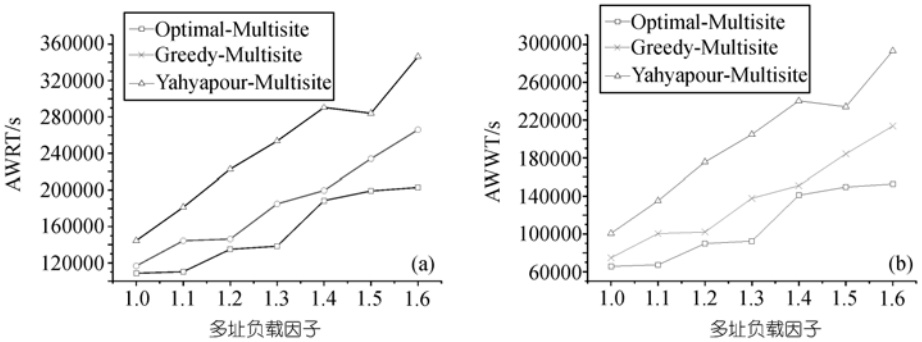

图 6　异构差异较大时最优、贪心多址和 Yahyapour 协同算法 AWRT 与 AWWT 性能比较

性能比较高的资源, 因此在性能上比最优多址协同算法略有降低, 但仍高于非自适应算法性能. 在异构因子达到 1.6 时, 贪心多址协同算法 AWRT 性能高于非自适应调度算法性能 82%, 而 AWWT 性能则要高出 134%.

如图 6 所示, 在站点资源性能差异较大的网格环境中, 异构因子 1.6 时, 最优算法和贪心算法的 AWRT 性能仍然高于 Yahyapour 多址协同算法 30%和 70%.

综上所述, 无论站点间异构性如何, 采用自适应资源选择的最优与贪心多址协同算法, 其 AWRT 与 AWWT 性能均显著优于 Yahyapour 的多址协同算法.

5.4.3 网络带宽对算法性能影响

本节考察网络带宽因子对协同算法性能的影响. 考察自贪心多址协同算法, 网络带宽因子为 10.0, 多址因子从 1.0 变化到 1.6. 回填方式为 Easy.

如图 7 所示, 多址协同算法总体性能随网络带宽的降低而下降. 当网络带宽小于 20 KBps 时, 算法性能有一个非常大幅度的下降, 20 和 10 KBps 性能上的差异达到 128%. 因此整合带宽过低的站点会极大地影响网格系统的整体性能.

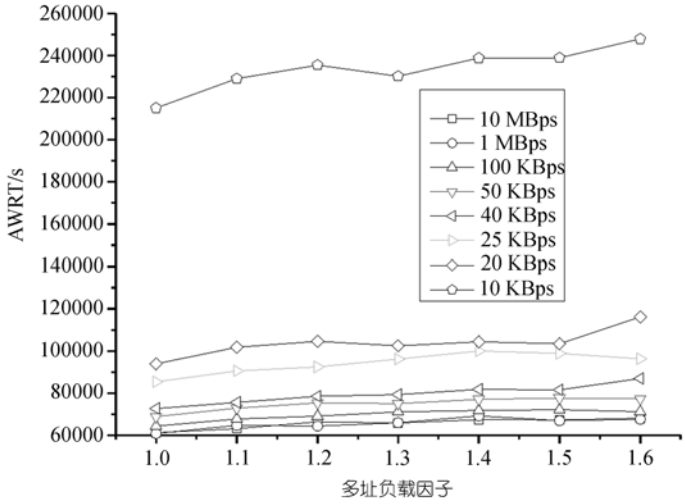

图 7 网络带宽因子对多址协同算法性能的影响

5.4.4 Easy 与 Conservative 回填算法

本节研究 Conservative 与 Easy 回填方式对调度算法性能的影响. 考察贪心多址协同算法, 网络带宽因子为 10.0, 多址因子从 1.0 变化到 1.6. 首先观测资源异构性较小时, 两种回填方式的性能. 图 8 中的(a), (b)分别为两种回填方式在不同作业负载下性能.

如图 8 所示, 无论对于何种作业负载调度算法, 采用 Easy 回填方式的性能总

是优于 Conservative 方式, 根据作业负载的不同, 性能优势幅度在 2%~17%间波动. 原因是Easy回填策略仅仅考虑等待队列中第一个作业的预约时间, 因此后面的作业比 Conservative 方式具有更大的机会进行回填, 提高了资源利用率和系统

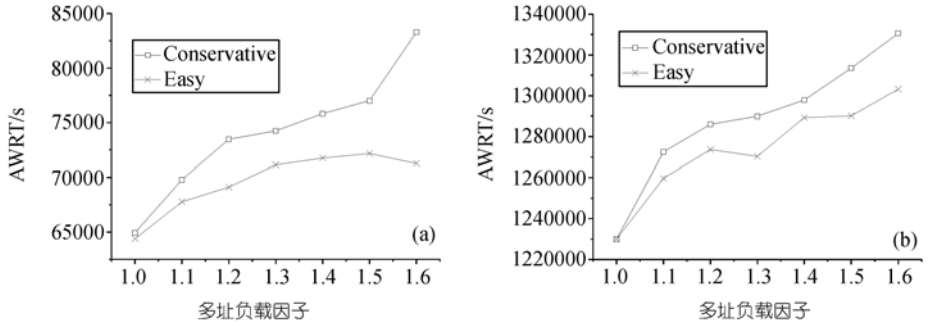

图 8　异构差异较小时 Conservative 和 Easy 回填方式的性能比较

(a) JobWorkload1; (b) JobWorkload2

性能.

如图 9 所示, 在资源异构性比较大的环境中, Easy 回填方式 AWRT 性能比 Conservative 回填方式高出 22%~76%. 异构性的加大, 增加了作业队列中等待调度的作业数量, 从而更凸现出 Easy 和 Conservative 回填方式在回填作业数量上的差异.

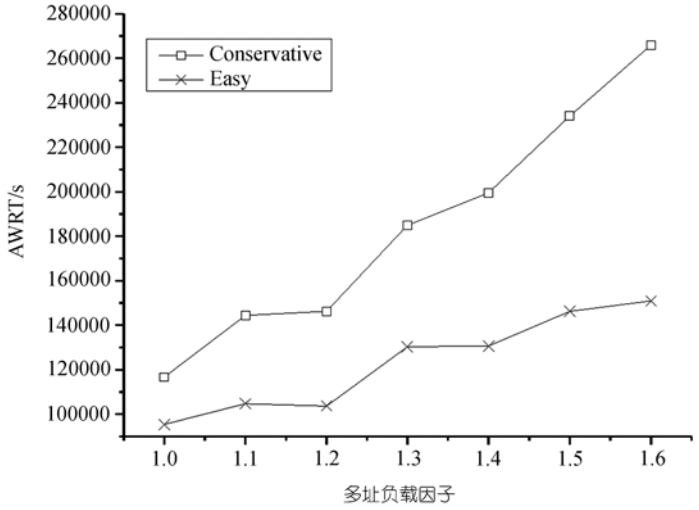

图 9　异构性差异较大时 Conservative 和 Easy 回填方式性能比较

综上所述, 无论在资源异构性较小或较大的环境中, 对于不同类型的作业负载, Easy 回填方式总是比 Conservative 回填方式具有更高的性能, 但 Easy 回填方

式并不具备最完美的公平性.

6 结束语

本文针对网格环境下作业级多址协同调度模型与调度算法进行深入研究, 主要贡献如下: (1) 通过分析计算网格的自然特征, 提出作业级网格任务调度应满足协同性、异构适应性、网络适应性和规模可扩展性; (2) 提出计算网格环境下作业级多址协同调度模型和性能模型; (3) 引入多址作业协同调度算法框架的同时, 提出最优与贪心多址协同调度算法; (4) 实验验证表明, 本文提出的多址协同算法在平均加权响应时间、平均加权等待时间和系统利用率等方面显著优于经典的单址与多址协同算法, 考察了网络性能对多址算法影响, 提出 Easy 回填方式较 Conservative 回填方式更为适合网格环境下的作业回填.

本文在网格调度方面做了一些初步的工作, 我们未来将在如下的几个方面开展工作: (1) 本文调度算法未考虑系统失效情况, 而网格作为开放虚拟计算环境资源失效不可避免, 应进一步在调度模型中引入失效检测与迁移策略, 增强作业生存性; (2) 多管理域网格系统具有不同的安全需求, 调度算法应进一步整合不同网格系统的信任策略, 形成信任驱动的网格任务调度算法; (3) 网格环境中的任务调度需要考虑不同用户的多种 QoS 需求, 调度算法应能协调多种请求.

参 考 文 献

1 Foster I. The Grid: A new infrastructure for 21st century science. Phys Tod, 2002, 55(2): 42—47
2 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. 2nd ed. San Fransisco: Morgan Kaufmann, 1999. 279—290
3 Sabin G, Kettimuthu R, Rajan A, et al. Scheduling of parallel jobs in a heterogeneous multisite environment. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol. 2862. Berlin: Springer-Verlag, 2003. 87—104
4 Feitelson D G. A Survey of Scheduling in Multiprogrammed Parallel Systems. IBM Watson T J Research Center Technical Report RC 19790(87657), 1994
5 Feitelson D G, Rudolph L. Parallel job scheduling: Issues and approaches. In: Feitelson D G, Rudolph L, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 949, Berlin: Springer-Verlag, 1995. 1—18
6 Feitelson D G, Rudolph L, Schwiegelshohn U. Parallel job scheduling——a status report. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 3277, Berlin: Springer-Verlag, 2004. 1—16
7 Abawajy J H, Dandamudi S P. Parallel job scheduling on multicluster computing systems. In: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'03). Oakland: IEEE Computer Press, 2003. 11—18
8 Sabin G, Kettimuthu R, Rajan A, et al. Scheduling of parallel jobs in a heterogeneous multisite environment. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Berlin: Springer-Verlag, 2003. 87—104
9 Ernemann C, Hamscher V, Yahyapour R. Benefits of global grid computing for job scheduling. In:

Proceedings of 5th IEEE/ACM International Workshop on Grid Computing in Conjunction with SuperComputing 2004. Oakland: IEEE Computer Press, 2004. 374—379

10 England D, Weissman J B. Costs and benefits of load sharing in the computational grid. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 3277, Berlin: Springer-Verlag, 2004. 160—175

11 Martino V D, Mililotti M. Sub optimal scheduling in a grid using genetic algorithms. Paral Comp, 2004, 30(5-6): 553—565[DOI]

12 Gao Y, Huang J Z, Rong H. Adaptive Grid Job Scheduling with Genetic Algorithm. Future Gener Comp Sys, London: Elsevier Press, 2005, 21: 151—161

13 Hamscher V, Schwiegelshohn U, Streit A, et al. Evaluation of job-Scheduling strategies. In: Proceedings of Grid Computing (Grid 2000) at 7th International Conference on High Performance Computing (HiPC-2000). Lect Notes in Comput Sci, Vol 1971, Berlin: Springer-Verlag, 2004. 191—202

14 Ernemann C. Hamscher V, Schwiegelshohn U, et al. On advantages of grid computing for parallel job scheduling. In: Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002). Oakland: IEEE Computer Press, 2002. 39—47

15 Ernemann C, Hamscher V, Streit A, et al. Enhanced algorithms for multisite scheduling. In: Proc 3rd IEEE/ACM International Workshop on Grid Computing (Grid 2002) at Supercomputing 2002. Lect Notes in Comput Sci, Vol 2536, Berlin: Springer-Verlag, 2002. 219—231

16 Ernemann C, Hamscher V, Streit A, et al. On effects of machine configurations on parallel job scheduling in computational grids. In: International Conference on Architecture of Computing Systems, Karlsruhe, 2002. 169—179

17 Banen S, Bucur A, Epema D. A measurement-based simulation study of processor co-allocation in multicluster systems. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 2862, Berlin: Springer-Verlag, 2003. 105—128

18 Sinaga J, Mohamed H, Epema D. A dynamic co-allocation service in multicluster systems. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 3277, Berlin: Springer-Verlag, 2005. 194—209

19 Mohamed H, Epema D. The design and implementation of the KOALA co-allocating grid scheduler. In: Proc, European Grid Conference. Lect Notes in Comput Sci, Vol 3470, Berlin: Springer-Verlag, 2005. 640—650

20 Bucur A, Epema D. Priorities among multiple queues for processor co-allocation in multicluster systems. In: Proc 36th Annual Simulation Symp, Orlando. Oakland: IEEE Computer Press, 2003. 15—27

21 Bucur A, Epema D. The maximal utilization of processor co-allocation in multicluster systems. In: Proc 17th International Parallel and Distributed Processing Symposium (IPDPS 2003). Oakland: IEEE Computer Press, 2003. 60—69

22 Bucur A, Epema D. The performance of processor co-allocation in multicluster systems. In: Proc 3rd IEEE/ACM Int'l Symp on Cluster Computing and the Grid (CCGrid2003). Oakland: IEEE Computer Press, 2003. 302—309

23 Bucur A, Epema D. Trace-based simulations of processor co-allocation policies in multiclusters. In: Proc 12th IEEE Int'l Symp on High Performance Distributed Computing (HPDC-12). Oakland: IEEE Computer Press, 2003. 70—79

24 Bucur A, Epema D. The influence of the structure and sizes of jobs on the performance of co-allocation. In: Proc 6th Workshop on Job Scheduling Strategies for Parallel Processing, Lect Notes in Comput Sci, Vol 1911, Berlin: Springer-Verlag, 2000. 154—173

25 Bucur A, Epema D. The influence of communication on the performance of co-allocation. In: Feitelson D, Rudolph L, eds. Proc. 7th Workshop on Job Scheduling Strategies for Parallel Processing. Lect Notes in

Comput Sci, Vol 2221, Berlin: Springer-Verlag, 2001. 66—86

26 Bucur A, Epema D. Local versus global queues with processor co-allocation in multicluster systems. In: Feitelson D, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing, Lect Notes in Comput Sci, Vol 2537, Berlin: Springer-Verlag, 2002. 184—204

27 Goldman A, Queiroz C. A model for parallel job scheduling on dynamical computer grids. Concurr Comput-Prac Exp, 2004, 16(5): 461—468[DOI]

28 Snell Q, Clement M, Jackson D, et al. The performance impact of advance reservation meta-scheduling. In: Feitelson D G, Rudolph L, Schwiegelshohn U, eds. Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 1911, Berlin: Springer-Verlag, 2000. 137—153

29 Sodhi S, Subhlok J. Skeleton based performance prediction on shared networks. In: Proceedings of the 4th IEEE Symposium on Cluster Computing and the Grid (CCGrid'04). Washington: IEEE Computer Press, 2004. 723—730

30 Schwiegelshohn U, Yahyapour R. Fairness in parallel job scheduling. J Scheduling, 2000, 3(5): 297—320[DOI]

31 Jones J, Nitzberg B. Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In: Proc. 5th Workshop on Job Scheduling Strategies for Parallel Processing, Lect Notes in Comput Sci, Vol 1659, Berlin. Springer-Verlag, 1999. 1—16

32 Skovira J, Chan W, Zhou H, et al. The EASY-loadleveller api project. In: Proc 2nd Workshop on Job Scheduling Strategies for Parallel Processing. Lect Notes in Comput Sci, Vol 1162, Berlin: Springer-Verlag, 1999. 41—47

33 Feitelson D, Rudolph L, Schweigelshohn U, et al. Theory and practice in parallel job scheduling. In: Proc 3rd Workshop on Job Scheduling Strategies for Parallel Processing, Lect Notes in Comput Sci, Vol 1291, Berlin: Springer-Verlag, 1999. 1—34