基于滑动窗口的蚁群在线调度算法

孙立斌 邓 蓉
(同济大学计算机科学与技术系 上海 201804)
(同济大学嵌入式系统与服务计算教育部重点实验室 上海 200092)

摘 要 任务调度是网格计算领域的一个核心问题。目前,国内外有大量针对网格离线调度问题的研究,对在线调度问题却研究得较少。基于滑动窗口的蚁群算法SWbAC(Silde Window based Ant Colony)采用两类窗口分别记录最近一段时间到达的任务信息以及资源的负载情况,并通过这些信息对未来进行预测。为了验证算法的有效性,首先对GridSim平台进行了扩展,模拟了资源负载实时变化的情况。对比实验的结果表明,SWbAC算法的平均任务周转时间要比Online Min-Min算法短3%~10%。

关键词 网格仿真器,在线调度,滑动窗口,蚁群
中图法分类号 TP301 **文献标识码** A

Slide Window Based Ant Colony Algorithm for Online Grid Scheduling

SUN Li-bin DENG Rong
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)
(The Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 200092, China)

Abstract Task scheduling is a key problem of grid computing. Currently, a large number of researches focus on offline grid scheduling. Research on online scheduling is inadequate. Here a slide window based ant colony algorithm (SWbAC) for online grid scheduling was proposed. Two kinds of slide windows are used respectively to record recently received gridlet and the dynamic statuses of grid resources. These information are used to predict the future incoming gridlets and resources' statuses. The experiment platform is based on an extension for Gridsim, simulating the dynamic changes of grid resources' status. Compared with Online Min-Min algorithm, our algorithm performs 3%~10% better in view of average makespan of gridlets.

Keywords Grid simulator, Online scheduling, Slide window, Ant colony

1 引言

网格任务调度[1]是网格计算领域的一个核心问题,也是一个NP完全问题[2]。国内外有许多关于网格调度的研究论文,但其中绝大部分都是针对离线的情况。在离线调度中,被调度任务集中的任务达到一定的数量或者到了一定时间间隔,才会开始调度。对于任务集合中先来的任务来说,其需要等待较长的时间才能够得到调度。此外,离线算法的复杂度往往较高,因此扩展性也较差。

蚁群算法是一种生物智能算法,1992由意大利学者Marco Dorigo首先提出[3],其成功解决了旅行商问题。该算法模仿了蚂蚁的觅食过程,采用信息素及启发式信息进行解的构造。经过十余年的发展,蚁群算法已被应用到包括网格调度在内的诸多领域,并取得了较好的效果。目前,国内外有许多基于蚁群算法的网格离线任务调度的研究,文献[5]利用蚁群算法来平衡网格的负载;文献[6]中提出了基于蚁群算法缩短任务的总超时时间的算法;文献[7]将蚁群算法应用到了多QoS工作流调度中。然而,这些算法针对的都是离线的情况。

本文提出了一种基于滑动窗口的蚁群在线调度算法SWbAC(Silde Window based Ant Colony)。在网格环境中,网格的资源是自治的、动态的,即使没有任务在资源上执行,资源也会有本地的负载,并不断地动态变化。为了适应网格这种动态的环境,将任务调度到最合适的资源上,这需要了解最近一段时间网格资源的负载情况。有两种方法可以获得网格资源的负载情况,第一种方式是每隔一段时间向资源发出查询请求,这种方式的优点是实时性高,但是仅凭查询得到的CPU速率并不能精确反映任务在指定资源上的运行时间。第二种方式是根据最近从资源上返回的任务的实际执行时间来判断。这种方式的优点是更真实地反映了资源的情况,但缺点是如果最近没有任务从资源返回,就无法得知资源最新的实时情况。本文综合上述两种方式的优点,用查询获得的

到稿日期:2012-05-26 返修日期:2012-09-01 本文受国家科技支撑计划项目(2012BAH15F03)资助。
孙立斌(1989-),男,硕士,主要研究方向为操作系统、分布式计算等,E-mail:justdoit-cool@163.com;邓 蓉(1973-),女,讲师,主要研究方向为操作系统、分布式计算。

信息作为启发式信息、真实反馈的信息作为信息素，以辅助蚂蚁判断资源的优劣。

此外，本算法还考虑了是否为将来可能到达的长任务预留优质资源。网格资源虽然是动态变化的，但也具有一定的惯性，本算法采取滑动窗口记录最近提交的任务的时间间隔和大小，并根据这些信息及当前被调度任务的大小、当前资源的状况等来综合考虑是否将优质资源预留给后续任务。

算法的实验环境基于 GridSim[8-11] 的扩展网格仿真平台。本文对 GridSim 仿真平台进行了二次开发，使之能够有效模拟网格资源负载动态变化的情况。实验结果表明，使用基于滑动窗口的蚁群在线调度算法 SWbAC 进行调度，任务的平均周转时间比 Online Min-Min 好 3%～10%。

本文第 2 节对网格问题调度问题进行形式化描述；第 3 节介绍系统架构和算法的详细步骤；第 4 节是 SWbAC 算法和 Online Min-Min 算法的对比实验结果；最后总结全文。

2 问题描述及解决思路

真实网格中，任务到达的时间是随机的。离线的调度方式使得先到达的任务需要等待，这段时间里，即使有空闲的处理器，任务也不能够被映射执行。假设系统调度的时间间隔为 T，则任务平均需要等待 $T/2$ 的时间。对于短任务来说，消耗在等待上的时间会对其周转时间造成很大的影响。

表 1 符号和定义

符号	定义
TS	当前待调度的任务集合
T_i	编号为 i 的任务
T_c	当前被调度的任务
Length(i)	T_i 的大小，以 MI 为单位
ArrivalTime(i)	T_i 的到达时间
ReturnTime(i)	T_i 的返回时间
MakeSpan(i)	T_i 的周转时间 ReturnTime(i)－ArrivalTime(i)
AverSpan	任务的平均周转时间
AverLength	所有到达任务的平均长度
AverInter	任务到达的平均时间间隔
RII	查询资源动态速率间隔
TW_{size}	任务信息滑动窗口大小
m	资源的数目
R_j	编号为 j 的资源
U	空闲的资源集合
StaticSpeed(j)	R_j 的静态速度
$DynSpeed_a(j)$	由查询计算出的 R_j 动态速度
$DynSpeed_b(j)$	由实际反馈任务计算出的 R_j 动态速度
DynSpeed(j)	R_j 预计的动态速度
RW_{size}	资源速率滑动窗口大小
NumPE(j)	R_j 上处理单元个数
FreePE(j)	R_j 上空闲的处理单元个数
ETE(i,j)	T_i 在 R_j 上预计执行的时间
Score(j)	资源 j 的评分
Level	ETE(i,j)/AverInter * AverLength/Length(i)
Threshold	一个 Level 的阈值
α	启发式信息的重要程度
β	信息素的重要程度
P	选择最优资源的概率

表 1 中定义了描述 SWbAC 算法中用到的符号。TS 是当前待调度的任务集合，当所有资源都忙碌时，新接收的任务在待调度集合中等待。Length(i)、ArrivalTime(i)、ReturnTime(i) 分别代表任务 i 的大小、到达时间和返回时间。MakeSpan(i) 和 AverSpan 分别为任务 i 的周转时间以及所有任务的平均周转时间。我们的调度目标是使所有任务的 AverSpan 最小。

算法 SWbAC 用到了两类重要的滑动窗口：任务滑动窗口 TW 和资源速率滑动窗口 RW。RW 用于预测未来任务，该滑动窗口记录最近一段时间到达的任务信息。统计得出的任务平均长度和到达平均时间间隔，是决定是否需要资源预留的重要依据。另一类滑动窗口 RW 用于记录资源最近一段时间的动态速度。调度时，SWbAC 算法根据过去一段时间内各资源的动态速度来预测接下来一段时间的资源表现。

在考虑在线调度的时候还有一个问题，即如果当前有多个空闲资源存在，则需要考虑是为当前被调度的任务 T_c 安排最优的资源，还是将最优资源预留给接下来可能到达的任务。本文的算法考虑了两个因素：平均任务大小和当前任务大小的比值 R_1，以及当前任务预期执行时间和任务平均到达时间间隔的比值 R_2。R_1 越大，表示当前任务较平均大小越小。R_2 越大，表示在 T_c 执行的时间内到达一个新任务的概率越大。这里采用了一个阈值 $Level$，若 $R_1<1$ 或 $R_1 * R_2<Level$，则为其选择最优资源，不然则降低选择最优资源的可能，更多地考虑其它的资源。通过以上这些方法，可以有效地缩短任务的平均周转时间。

3 算法描述

SWbAC 利用滑动窗口记录了最近一段时间网格资源和任务到达的动态情况，这些信息与当前被调度的任务信息共同构成了调度的依据。同时，由于在线任务调度具有更大的不确定性，因此我们以 P 的概率选择最优的资源，在 $1-P$ 的概率下根据资源动态速率的快慢来概率性地选择资源。这样就有机会试探性地将任务提交到性能略低的任务上，以便于获取资源的反馈信息。

下面是系统架构以及算法的详细步骤。

3.1 系统的架构

使用算法 SWbAC 进行调度的网格拥有 4 类主要的实体：用户、调度器、网格资源以及网格信息服务器。系统架构和交互时序如图 1、图 2 所示。

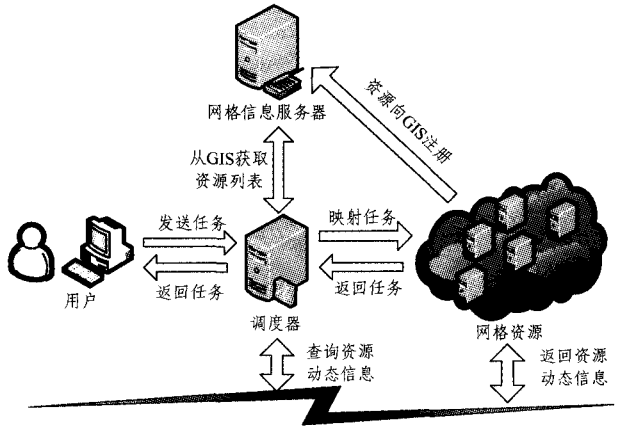

图 1 系统的架构图

系统初始化时所有的网格资源向 GIS 注册，之后调度器查询 GIS 获取可用的资源列表。运行时，用户向调度器提交(submit)任务，后者运行 SWbAC 算法，为任务在线寻找合适资源(mapped resource)。

实体交互图中 4、5、6 步骤对应于调度器和 mapped resource 之间的任务提交、运行和结果反馈过程。此外，调度器还会定时向资源查询其动态负载，由于查询的时序非常简单，就不另画出时序图。

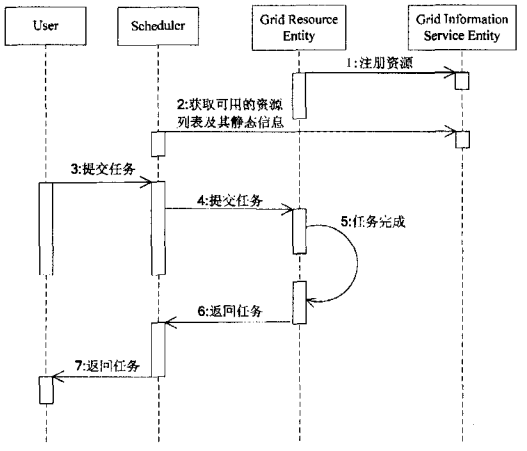

图 2　系统交互时序图

3.2　算法的具体步骤

在介绍了系统的架构之后，再来看算法的具体步骤。算法包含了初始化、信息更新和任务调度。

3.2.1　初始化工作

在调度器开始工作前，先从 GIS 获取资源列表及资源静态属性，并初始化任务滑动窗口和资源动态速率滑动窗口。

系统刚开始运行时，还没有到达和反馈回来的任务，因此将任务滑动窗口置空。将资源速率滑动窗口的第一个及资源最近反馈任务的速率都置为其静态速率。这样，在系统初启时，资源的预计动态速率就等于其静态速率。

3.2.2　信息更新

信息更新有 3 个时机：新任务到达时、任务完成返回时，以及每隔 RII 时间间隔。

(1)当新任务到达时，将任务长度及其与前一个到达任务的时间间隔(如果不是第一个任务)放入任务滑动窗口。任务滑动窗口的最大大小为 TW_{size}，如果当新任务到达时，滑动窗口内已有 TW_{size} 个任务信息，则滑动窗口需要向右滑动，即删除最早到达的任务信息。

(2)任务从 R_j 返回时，需要更新 $DynSpeed_b(j)$，即根据 R_j 最近返回的任务计算出资源动态信息，$DynSpeed_b(j)$在蚁群算法中扮演着信息素的角色。

(3)每隔 RII 时间间隔，调度器向资源查询资源的动态速率，并将其存放入资源动态速率滑动窗口中，资源滑动窗口的最大大小是 RW_{size}。与任务滑动窗口一样，当窗口中记录的速率个数大于 RW_{size} 时，窗口需要向右滑动。

3.2.3　任务调度

当 TS 非空且存在空闲资源(若 $FreePE(j)>0$，则 R_j 空闲)时，使用 SWbAC 算法为任务映射资源，任务映射的具体步骤如下：

Step1　若此时 TS 中等待调度的任务数大于 1，则从中选取长度最小的任务 T_k，为其映射资源：

$$T_k=\min\{T_j\mid Length(j)\},T_j\in TS \tag{1}$$

Step2　估计每个资源的动态速率，首先根据资源滑动窗口计算最近一段时间查询到的资源平均速率 $DynSpeed_a(j)$，已知 $DynSpeed_b(j)$为最近从 R_j 返回的任务的执行速率，则 R_j 的预计速率可通过式(2)计算得出：

$$DynSpeed(j)=DynSpeed_a(j)^{\alpha}DynSpeed_b(j)^{\beta} \tag{2}$$

Step3　先计算 T_i 在 R_j 上预计执行的时间 $ETE(i,j)$，再根据其计算资源 j 的评分：

$$ETE(i,j)=\frac{Length(k)}{DynSpeed(j)} \tag{3}$$

$$Score(j)=\begin{cases}\dfrac{1}{ETE(i,j)}, & R_j\in U\\ 0, & R_j\notin U\end{cases} \tag{4}$$

Step4　现在考虑是否将优质资源预留给后续任务，若 $Length(i)<AverLength$，设目前得分最高的资源为 R_k，根据式(5)来计算 $Level$：

$$Level=\frac{ETE(i,k)}{AverInter}*\frac{AverLength}{Length(i)} \tag{5}$$

若 $Level>Threshold$，则要修正最优资源的评分：

$$Score(k)=Score(k)*\left(\frac{Threshold}{Level}\right)^2 \tag{6}$$

$Level$ 越大，优质资源被预留的可能就越大。

Step5　生成一个随机数 q，P 是蚁群算法中的概率参数。若 $q<P$，则选择修正后的 Socre 最大的 R_n，将 T_i 映射到 R_n；否则，根据 Socre 概率选择一个资源：

$$p_{ij}=\begin{cases}\dfrac{Score(j)}{\sum_{R_k\in U}Score(k)}, & R_j\in U\\ 0, & R_j\notin U\end{cases} \tag{7}$$

式中，p_{ij} 为 T_i 选择 R_j 的概率。

4　实验结果及分析

为了验证本文中的 SWbAC 算法的有效性，将其与 Online Min-Min 算法进行对比。实验的环境是基于 GridSim 仿真平台，对 GridSim 进行了扩展，使其支持资源本地负载实时波动的情况。

表 2 列出了网格资源的属性，其中资源 CPU 速率的单位是 MIPs。表 3 列出了蚁群算法的相关参数，其中 $Threshold$ 是 $Level$ 的阈值；α 和 β 都是 0.5，表明启发式信息和信息素重要性相同；P 是蚁群算法选取得分最高资源的概率；RRI 是查询资源负载的时间间隔，单位是 s，TW_{size} 和 RW_{size} 分别为任务和资源的滑动窗口的大小。

表 2　网格资源属性

	R_1	R_2	R_3	R_4	R_5	R_6
CPU 速率	300	400	500	300	400	500
PE 数目	2	2	2	2	2	2

表 3　蚁群算法参数

参数	取值
Threshold	10
α	0.5
β	0.5
P	0.95
RRI	20
TW_{size}	20
RW_{size}	20

表 4 列出了两组用户任务属性，任务长度单位是 MI。各用户平均的任务数量服从泊松分布，任务的大小服从均匀分布，任务到达的时间间隔服从指数分布。第一组任务的用户平均任务数及任务平均长度较大，对应了负载较高的情况；第二组对应轻负载的情况。表 5、表 6 列出了这两组任务在 SWbAC 及 Online-Min Min 算法下的实验结果。

表 4　用户任务属性

用户数	平均任务数	最小长度	最大长度
8	25	20000	70000
8	16	15000	55000

表 5　重负载下 SWbAC 算法与 Online-Min Min 算法对比

任务数	任务平均长度	SWbAC 平均周转时间	Online-Min Min 平均周转时间
189	45742.68	137.20	141.50
187	42705.31	120.73	128.90
181	44869.65	124.84	136.76
210	46943.06	140.83	147.76
212	44039.89	131.20	136.21
205	44027.38	125.13	130.77
185	42913.11	124.58	130.13

表 6　轻负载下 SWbAC 算法与 Online-Min Min 算法对比

任务数	任务平均长度	SWbAC 平均周转时间	Online-Min Min 平均周转时间
124	35318.61	98.26	102.29
150	34511.62	98.95	103.50
134	34854.69	98.98	107.70
143	36258.11	108.52	113.55
122	32417.70	86.47	96.19
136	34123.60	93.25	97.97
126	35692.29	97.55	104.54

根据表 5、表 6 的实验结果可知，在重负载和轻负载条件下，SWbAC 算法与 Online-Min Min 算法相比，其任务平均周转时间分别要好 3%～8.7%和 4.4%～10%。

结束语　本文提出的 SWbAC 算法用以解决在线独立任务调度的问题。滑动窗口中记录了历史任务信息及资源动态速率，可利用过去一段时间内的任务到达情况和资源负载来对将来进行预测。

算法还采用了启发式信息与信息素结合，考虑了是否将当前的优质资源预留给后续任务，使用了 *Level* 值，若 *Level* 值大于阈值，则倾向于预留优质资源。这些措施使得 SWbAC 算法能够适应资源负载的动态变化，为任务分配合适的资源，更符合在线调度的特点。

参 考 文 献

[1] Foster I, Kesselman C. The Grid: Blueprint for a new computing infrastructure [M]. San Francisco: Morgan Kaufmann Publishers, 1998: 1-20

[2] UllMan J D. NP-complete scheduling problems [J]. J. of Computer and Systems Sciences, 1975, 10: 384-393

[3] Dorigo M. Optimization, Learning and Natural Algorithms [D]. Milan: DEI, Polytecnico di Milano, 1992

[4] Marco D, Mauro B, Thomas S. Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique [J]. IEEE Computational Intelligence Magazine, 2006, 1(4): 28-39

[5] Chang R S, Chang J S, Lin P S. An ant algorithm for balanced job scheduling in grids [J]. Future Generation Computer Systems, 2009, 25(1): 20-27

[6] Lorpunmanee S, Sap M N, Abdullah A H, et al. An ant colony optimization for dynamic job scheduling in grid environment [J]. International Journal of Computer and Information Engineering, 2007, 1(4): 207-214

[7] Chen Wei-neng, Zhang Jun. An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem with Various QoS Requirements [J]. IEEE Transactions on Systems, Man, and Cybernetics, ——Part C: Applications and Reviews, 2009, 39(1): 29-43

[8] Buyya R, Murshed M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing [J]. The Journal of Concurrency and Computation: Practice and Experience (CCPE), 2002, 14(13-15): 1175-1220

[9] 卢鹏，金海，谢夏，等. 关于网格模拟器的研究[J]. 高性能计算技术，2005，173(2)：5-9

[10] 刘宴兵，杨茜慧，王文斌. 基于 GridSim ToolKits 的网格仿真环境设计与实现[J]. 计算机科学，2008，35 (6)：83-85

[11] 李炯，卢显良，董仕. 基于 GridSim 模拟器的网格资源调度算法研究[J]. 计算机科学，2008，35(8)：95-97

[12] 范国昌. 网格计算的 Online-Min Min 任务调度算法研究[D]. 北京：北京邮电大学，2010

[13] Bajaj R, Agrawal D P. Improving Scheduling of Tasks in A Heterogeneous Environment [J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(2): 107-118

[14] Topcuoglu H, Hariri S, Wu Min-you. Performance effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274