一种基于虚拟截止时间制导的改进的 Min-Min 元任务调度算法

杨疆湖　高传善　黄昌来　李　明
（复旦大学计算机科学与工程系　上海 200433）

摘　要　在网格环境下，资源状况和用户行为相当复杂，是一个异构计算环境，元任务(meta-task)调度比传统并行调度更为复杂。如何映射一组任务到一组机器上被证明是 NP 问题，其目的一般是最小化任务完成时间(makespan)。为解决这一问题，已经提出一些启发式任务调度算法，例如具有代表性的 Min-Min 元任务调度算法。本文在 Min-Min 元任务调度算法的基础上，通过虚拟截止时间制导的方法来改进 Min-Min 算法。实验结果表明，本文提出的算法具有更短的任务完成时间。

关键词　元任务(meta-task)，虚拟截止时间，调度，算法，Min-Min

A Virtual Deadline Guided Min-Min Meta-task Scheduling Heuristic

YANG Jiang-Hu　GAO Chuan-Shan　HUANG Chang-Lai　LI Ming
(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

Abstract　In Grid environment which is a Heterogeneous Computing(HC) environment, resource status and user behavior are very complicated, so scheduling heuristics for meta-task are more complicated than traditional parallel scheduling heuristics. How to map a set of tasks on a set of machines is known to be NP-hard. The goal of those scheduling heuristics is minimize the makespan of the meta-tasks. Some heuristics are introduced to solve such scheduling problem, including Min-Min and other heuristics. In this paper, based on traditional Min-Min scheduling heuristic, a virtual deadline guided meta-task scheduling heuristic is proposed. The simulation results show that the proposed heuristic has less makespan than traditional Min-Min scheduling heuristic.

Keywords　Meta-task, Virtual deadline, Scheduling, Heuristic, Min-Min

1　引言

元任务(meta-task)调度问题是指如何调度一组相互独立的任务。元任务由一组原子的(不可分解的)和相互独立的(任务之间没有通信)任务组成。传统的并行调度算法主要是在一台并行计算机上调度应用程序的各个子任务，以实现减少总体任务完成时间。然而在网格环境下，资源状况和用户行为都相当复杂，这是一个异构计算环境[7]。异构计算环境是由一个具有不同计算能力的机器池和一个具有不同计算要求的任务池组成[1,2]。一个任务在不同的机器上有不同的计算时间。同样，不同的任务在同一台机器上有不同的计算时间。按异构程度的高低，异构计算环境分为 4 种：1) 高机器异构和高任务异构；2) 低机器异构和高任务异构；3) 高机器异构和低任务异构；4) 低机器异构和低任务异构。

如何映射一组任务到一组机器上被证明是 NP 问题[3,10]，其目的一般是最小化任务完成时间(makespan)[8]。为解决这一问题，已经提出一些启发式任务调度算法，例如 Min-Min、Max-Min、Sufferage 等[4~6]。同样，经过一些调整和变通，这些算法也能处理 QoS 问题，如某个任务需要大的存储空间或高的网络带宽等。一般的做法是将任务分为两组：具有高 QoS 要求的任务为一组，首先被映射；具有低 QoS 要求的任务分为另一组，安排在后面映射。本文提出的元任务调度算法将主要针对低 QoS 要求的任务组，即高 QoS 要求的任务组采用原来的 Min-Min 元任务调度算法映射，低 QoS 要求的任务组采用本文提出的调度算法来映射。为简单起见，在算法描述上我们只讨论低 QoS 要求的任务组的调度问题。

本文提出的元任务调度算法是对 Min-Min 元任务调度算法的改进，其思想是引入虚拟截止时间(deadline)。虚拟截止时间并非是任务组本身所具有的，而是假定的、不存在的。如果在本文的元任务调度算法下，任务组都在虚拟截止时间内完成，那么虚拟截止时间可以被拿去，该调度是有效的。如果在本文的元任务调度算法下，任务组在虚拟截止时间内不能全部完成，那么该调度是无效的，需要重新设定虚拟截止时间，重新调用本文的算法，以使任务组在新的虚拟截止时间内完成。如果虚拟截止时间足够大，本文提出的元任务调度算法等同于 Min-Min 算法。因此虚拟截止时间的选择是关键。我们先运行一遍 Min-Min 算法，得出任务组的 Makespan，然后将虚拟截止时间设为 1～1.5 倍的 Makespan，再接着运行本文提出的算法，得出优化的 Makespan。

本文第 2 节给出问题的定义；第 3 节描述 ETC 矩阵产生模型；第 4 节给出基于虚拟截止时间制导的 Min-Min 元任务调度算法；第 5 节描述实验结果和讨论；最后是文章的结语。

2　问题定义

异构计算环境由 m 个异构机器和 n 个异构的独立任务

杨疆湖　博士研究生，主要研究方向为计算机网络协议及网格计算；**高传善**　教授、博士生导师，主要研究方向为数据通信、计算机网络与分布式系统；**黄昌来**　博士研究生，主要研究方向为计算机网络协议及网络安全；**李　明**　博士研究生，主要研究方向为计算机网络协议及网络多媒体。

组成。m 个异构机器表示为 $M=\{m_1, m_2, \cdots, m_m\}$，$n$ 个异构的独立任务表示为 $T=\{t_1, t_2, \cdots, t_n\}$。为简化起见，假定满足以下 3 个条件：1) 每个任务是原子的和独立的；2) 每个机器是独占的，即当一个任务分配给一个机器时，该任务占有该机器直到运行完毕；3)静态运行时间，即在分配任务之前任务在每个机器上的期望运行时间(ETC, Expected Time to Compute)是预先知道的。

任务的期望运行时间可以由 ETC 矩阵来描述。ETC 矩阵产生模型有多种，我们将选择其中最简单的一种来作为我们实验的数据来源。

当一个任务 t_i 被分配到一个机器 m_j 时，期望完成时间 CT_{ij} 定义为机器 m_j 完成任务 t_i 的时钟时间，即

$$CT_{ij}=d_j+ET_{ij}$$

其中，d_j 表示任务 t_i 的开始时间，即上一个分配给机器 m_j 的任务 t_x 的完成时间。

虚拟截止时间定义为 S，对每一个任务适用。如果一个任务的期望完成时间 $CT_{ij}>S$，则设定 $CT_{ij}=\infty$，此任务 t_i 将不能被映射到机器 m_j 上。

CT_i 定义为任务 t_i 被最终分配到机器 m_j 上运行得出的任务完成时间，则任务完成时间 Makespan 可以计算如下：

$$\text{Makespan}=\max(CT_i),\ t_i\in T$$

传统的 Min-Min 元任务调度算法(如图 1 所示)是一个两阶段的调度算法。第一阶段算出每个未分配任务 t_i 在所有机器 m_j 上($j=1..m$)的最小完成时间 $\text{MCT}_i=\min\{CT_{ij}\}$，$m_j\in M$；第二阶段算出最小的最小完成时间 $\text{MMCT}=\min\{\text{MCT}_i\}$，$t_i\in T^*$(未分配任务组)。假设满足 MMCT 的任务为 t^*，然后将任务 t^* 分配给产生 MMCT 的机器 m^*，并将任务 t^* 移出未分配任务组 T^*。重复此算法，直至所有的任务都被分配到某个机器。最后算出任务完成时间 Makespan。

```
(1)  for each machine m_j in M
(2)      d_j = 0
(3)  end for
(4)  repeat
(5)      for each task t_i ∈ T
(6)          for each machine m_j ∈ M
(7)              compute CT_ij = d_j + ET_ij
(8)          end for
(9)          MCT_i = min { CT_ij }, m_j ∈ M, and save task-machine
     pair in set TM
(10)     end for
(11)     Find task-machine pair (t*, m*) which produces minimum
     MCT
(12)     Assign t* to m* and remove t* from T
(13)     d_m* = CT_t*
(14) until T = ∅
```

图 1　Min-Min 元任务调度算法

我们定义传统的 Min-Min 元任务调度算法得出的 Makespan 为 MS。此 MS 将作为产生虚拟截止时间 S 的基础。

3　ETC 矩阵产生模型

ETC 矩阵产生方法有 Range Based ETC Matrix Generation 和 Coefficient-of-Variation Based ETC Matrix Generation[9]。前者更简单，所以我们采用它来生成 ETC 矩阵。

设 m 为异构计算环境中的机器数，n 为任务数。设 $U(a, b)$ 为 a、b 之间平均分布的一个采样。设 R_{task} 为任务异构度，R_{mach} 为机器异构度。R_{task} 值越高，表示任务的异构度越高；R_{mach} 值越高，表示机器异构度越高。在现实异构环境中，任务的变化一般比机器的变化要大，因此我们假设高异构度的任务比高异构度的机器要求更高的异构度取值(例如 $R_{\text{task}}>>R_{\text{mach}}$)。$R_{\text{task}}$ 和 R_{mach} 的参考取值见表 1。

表 1　现实环境下 R_{task} 和 R_{mach} 的高异构和低异构取值

	高异构	低异构
任务	10^5	10^1
机器	10^2	10^1

设 $e[i, j]$ 为任务 t_i 在机器 m_j 上的期望运行时间，$e[1\cdots n-1, 1\cdots m]$ 即为 ETC 矩阵，其产生方法如图 2 所示。第 1 行对所有任务从 1 到 n 循环。第 2 行对任务 t_i 从 1 到 R_{task} 范围内随机产生一个平均分布的采样值作为该任务的异构值。第 3 行对所有机器从 1 到 m 循环。第 4 行将任务的异构值和机器的异构值相乘，得出任务 t_i 在机器 m_j 上的期望运行时间 $e[i, j]$。

```
(1)  for i from 1 to n
(2)      τ[i] = U(1, R_task)
(3)      for j from 1 to m
(4)          e[i, j] = τ[i] * U(1, R_mach)
(5)      end for
(6) end for
```

图 2　Range Based ETC Matrix Generation 算法

4　基于虚拟截止时间制导的 Min-Min 元任务调度算法

基于虚拟截止时间制导的 Min-Min 元任务调度算法首先运行一遍传统的 Min-Min 元任务调度算法，得出 MS，然后将虚拟截止时间 S 分别设为 1.0 * MS 和 1.5 * MS。接着按两个 S 值运行本文提出的算法。如果本文提出的算法得出的 Makespan<MS，那么我们将得到更好的 Makespan。如果本文提出的算法得出的 Makespan>=MS，那么我们采用原来的 Makespan。本文将通过实验测出本文提出的算法运行的平均次数、有效百分比和提高百分比。

在本文提出的算法中，所有任务被分成 3 个级别：T''，T' 和 T。T'' 表示那些只能在一台机器运行、在其他机器上都会超出虚拟截止时间的任务；T' 表示那些至少在一台机器上运行会超出虚拟截止时间的任务；T 表示那些还未被分配的任务。3 个级别的任务组中，T'' 的优先级最高，然后是 T'，再次是 T。与传统的 Min-Min 元任务调度算法相似，本文提出的算法也分为两阶段：第一阶段算出所有任务 t_i 最小完成时间，同时算出任务 t_i 的有效机器集合 V_i，并根据 V_i 的长度决定将任务 t_i 划入哪一个任务组。第二阶段按 T''，T' 和 T 的优先级顺序分配任务组中具有最小的最小完成时间的一个任务到一台机器上。如果任务组 T'' 不空，则只在 T'' 中分配任务；如果 T'' 为空，则只在 T'' 中分配任务；如果 T'' 和 T' 都为空，才能在 T 中分配任务。分配过的任务将从任务组 T 中移走，然后清空 T''，T'。重复以上的算法，直至 T 中所有的任务都别分配到某一台机器。为了防止在算法循环过程中 T' 分类的失效，如果一台机器对所有的任务失效，那么该机器将从机器集合 M 中移出。

```
(1)  for each machine m_j in M
(2)     d_j = 0
(3)  end for
(4)  repeat
(5)     T'' = ∅
(6)     T' = ∅
(7)     for each task t_i ∈ T
(8)        V_i = ∅
(9)        for each machine m_j ∈ M
(10)          compute CT_ij = d_j + ET_ij
(11)          if CT_ij ≤ S then
(12)             V_i = V_i + { m_j }
(13)          else CT_ij = ∞
(14)          endif
(15)       endfor
(16)       MCT_i = min { CT_ij }, m_j ∈ M, and save task-machine
           pair in set TM
(17)       if | V_i | = 0 then
(18)          remove t_i from T
(19)       else if | V_i | = 1 then
(20)          add t_i to T''
(21)       else if | V_i | < |M| then
(22)          add t_i to T'
(23)       end if
(24)    end for
(25)    for each m_j, m_j ∈ M
(26)       if m_j not in all V_i then
(27)          remove m_j from M
(28)       end if
(29)    end for
(30)    if T'' ≠ ∅ then
(31)       Find task-machine pair (t*, m*) which produces
           minimum MCT in T''
(32)       Assign t* to m* and remove t* from T
(33)       d_m* = CT_t*
(34)    else if T' ≠ ∅ then
(35)       Find task-machine pair (t*, m*) which produces
           minimum MCT in T'
(36)       Assign t* to m* and remove t* from T
(37)       d_m* = CT_t*
(38)    else
(39)       Find task-machine pair (t*, m*) which produces
           minimum MCT in T
(40)       Assign t* to m* and remove t* from T
(41)       d_m* = CT_t*
(42)    end if
(43) until T = ∅
```

图 3　基于虚拟截止时间制导的 Min-Min 元任务调度算法

具体算法见图 3。第 1 行到第 3 行对所有的机器 m_j 初始化 $d_j=0$。第 4 行到第 43 行重复算法主体直至任务集合 T 为空。首先，第 5～6 行在每个循环开始时初始化 T''，T' 为空。然后，第 7～24 行计算所有任务 t_i 的最小完成时间。第 8 行初始化任务 t_i 的有效机器集合 V_i 为空，第 9～15 计算任务 t_i 在每个机器 m_j 上的完成时间 $CT_{ij}=d_j+ET_{ij}$，检查 CT_{ij} 是否超出虚拟截止时间 S，如果是，则将 CT_{ij} 设为无穷大；否则，将机器 m_j 加到有效机器集合 V_i 中。第 16 行计算任务 t_i 的最小完成时间 $MCT_i=\min\{CT_{ij}\}$，$m_j\in M$。第 17～23 行根据有效任务集合长度 $|V_i|$ 作出处理。如果 $|V_i|=0$，意味着任务 t_i 没有一个有效机器，那么将任务 t_i 移出任务集合 T；如果 $|V_i|=1$，意味着任务 t_i 只有一个有效机器，那么将任务 t_i 加到任务集合 T''；如果 $|V_i|<|M|$，意味着任务 t_i 至少有一个无效机器，那么将任务 t_i 加到任务集合 T'。接着，第 25～29 行是防止在算法循环过程中 T' 分类失效的处理。如果一台机器 m_j 对所有的任务失效，那么该机器 m_j 将从机器集合 M 中移出。最后，第 30～42 行分别按优先级对任务组 T''，T' 和 T 中的任务进行分配，而且只对其中一个任务进行分配。如果 T'' 不为空，在 T'' 找出具有最小的最小完成时间的任务，设为 t^*，连同所在的机器 m^* 存入映射列表中，并将 t^* 从任务组 T 中移走（注意是 T 而不是 T''），重新计算 $d_{m^*}=CT_{t^*}$。对 T' 和 T 的处理类似 T''。最后，算法的 $\text{Makespan}=\max(d_j)$，$m_j\in M$。

下面我们将用一个实例来说明基于虚拟截止时间制导的 Min-Min 元任务调度算法。ETC 矩阵如表 2 所示。

表 2　一个异构计算环境实例：7 个任务、3 台机器

	m_1	m_2	m_3
t_1	22	21	6
t_2	7	46	5
t_3	64	83	45
t_4	53	56	26
t_5	11	12	14
t_6	33	31	46
t_7	24	11	17

我们首先运行传统的 Min-Min 元任务调度算法。

第 1 次循环：t_2 有最小的最小完成时间，将任务 t_2 分配到机器 m_3，$t_2\rightarrow m_3$。

第 2 次循环：t_1 有最小的最小完成时间，将任务 t_1 分配到机器 m_3，$t_1\rightarrow m_3$。

第 3 次循环：t_5 有最小的最小完成时间，将任务 t_5 分配到机器 m_1，$t_5\rightarrow m_1$。

第 4 次循环：t_7 有最小的最小完成时间，将任务 t_7 分配到机器 m_2，$t_7\rightarrow m_2$。

第 5 次循环：t_4 有最小的最小完成时间，将任务 t_4 分配到机器 m_3，$t_4\rightarrow m_3$。

第 6 次循环：t_6 有最小的最小完成时间，将任务 t_6 分配到机器 m_2，$t_6\rightarrow m_2$。

第 7 次循环：将任务 t_3 分配到机器 m_1，$t_3\rightarrow m_1$。

该算法得出的 Makespan=75，即 MS=75（如图 4 所示）。

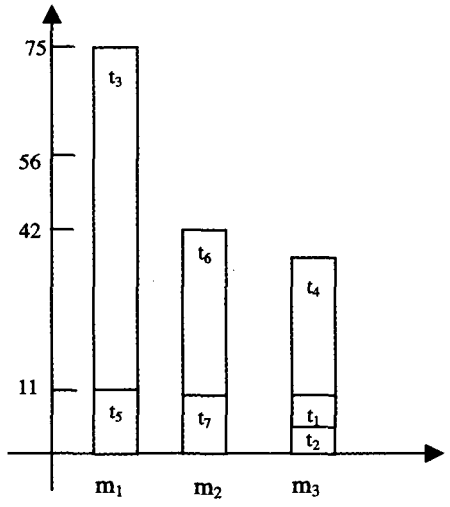

图 4　传统的 Min-Min 元任务调度算法的任务分配结果

然后我们设 S=MS=75，运行本文提出的算法。

第 1 次循环：$T''=\varnothing$，$T'=\{t_3\}$，$T=\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$。将任务 t_3 分配到机器 m_3，$t_3\rightarrow m_3$。

第 2 次循环：$T''=\varnothing$，$T'=\{t_6\}$，$T=\{t_1,t_2,t_4,t_5,t_6,t_7\}$。将任务 t_6 分配到机器 m_2，$t_6\rightarrow m_2$。

第 3 次循环：$T''=\varnothing$，$T'=\{t_2,t_4\}$，$T=\{t_1,t_2,t_4,t_5,$

$t_7\}$。将任务 t_2 分配到机器 m_1，$t_2 \rightarrow m_1$。

第 4 次循环：$T''=\varnothing$，$T'=\{t_4\}$，$T=\{t_1, t_4, t_5, t_7\}$。将任务 $t4$ 分配到机器 m_1，$t_4 \rightarrow m_1$。

第 5 次循环：$T''=\varnothing$，$T'=\{t_1, t_7\}$，$T=\{t_1, t_5, t_7\}$。将任务 t_1 分配到机器 m_3，$t_1 \rightarrow m_3$。

第 6 次循环：$T''=\varnothing$，$T'=\{t_7\}$，$T=\{t_5, t_7\}$。将任务 t_7 分配到机器 m_2，$t_7 \rightarrow m_2$。

第 7 次循环：$T''=\varnothing$，$T'=\varnothing$，$T=\{t_5\}$。将任务 t_5 分配到机器 m_2，$t_5 \rightarrow m_2$。

本文提出的算法得出的 Makespan＝60＜MS(如图 5 所示)，并且所有的任务都被成功分配。我们可以看到本文提出的算法优于 Min-Min 元任务调度算法，提高了 20%。

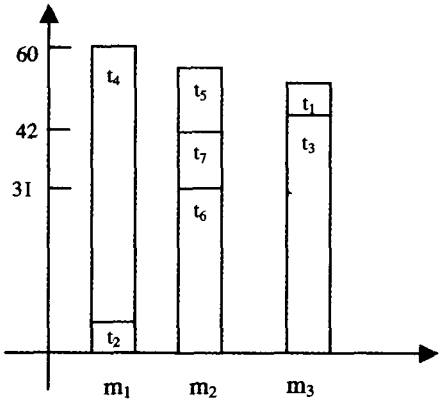

图 5　基于虚拟截止时间制导的 Min-Min 元任务调度算法的任务分配结果

5　实验结果

本文采用基于范围的 ETC 矩阵产生方法(Range Based ETC Matrix Generation)，作为实验的数据来源。实验中假定有 50 个任务、25 台机器。根据表 1，我们选择 4 组 R_{task} 和 R_{mach} 取值：第一组是低任务异构和高机器异构，即 $R_{task}=10$ 和 $R_{mach}=100$；第二组是低任务异构和低机器异构，即 $R_{task}=10$ 和 $R_{mach}=10$；第三组是高任务异构和高机器异构，即 $R_{task}=100000$ 和 $R_{mach}=100$；第四组是高任务异构和低机器异构，即 $R_{task}=100000$ 和 $R_{mach}=10$。我们开发了一套程序来实现 ETC 矩阵产生方法，对每一组 R_{task} 和 R_{mach} 取值，分别生成 100 个 ETC 矩阵，作为 100 个实验用例。我们将利用这些数据测试本文提出的算法运行的平均次数、有效百分比和提高百分比。

表 3　4 个实验组和 2 种算法的实验结果

实验组	Min-Min 的 Makespan	本文算法的 Makespan	提高百分比	有效百分比	本文算法运行次数
1	142.67	130.74	8.36%	100%	1.84
2	22.22	20.18	9.18%	100%	1.55
3	1484946	1353756	8.83%	100%	1.59
4	233573	211027	9.65%	100%	1.29

实验结果表明(如表 3 所示)，本文提出的基于虚拟截止时间制导的 Min-Min 元任务调度算法比传统的 Min-Min 元任务调度算法提高 8%～10%，而且没有出现因设置虚拟截止时间导致任务不能分配的情况，本文提出的算法是普遍有效的。对于低任务异构和高机器异构的第一组提高了 8.36%，低任务异构和低机器异构的第二组提高了 9.18%，高任务异构和高机器异构的第三组提高了 8.83%，高任务异构和低机器异构的第四组提高了 9.65%，4 个实验组提高效果相差不大。

我们还可以从表 3 中看出，本文算法运行次数从 1.29 次到 1.84 次。在第四组情况下，设置虚拟截止时间 S＝MS 可以提高大部分的异构环境任务完成时间(Makespan)。其它三组的算法，运行次数都超过了 1.5 次，意味着设置虚拟截止时间 $S=1.5*M$ 可以提高大部分的异构环境任务完成时间(Makespan)。

结论　本文提出的元任务调度算法是对 Min-Min 元任务调度算法的改进，其思想是引入虚拟截止时间(deadline)。虚拟截止时间是假定的、不存在的。如果在本文的元任务调度算法下，任务组都在虚拟截止时间内完成，那么虚拟截止时间可以被拿去，该调度是有效的。在本算法中，任务被分成 3 个级别：T''，T' 和 T。T'' 表示那些只能在一台机器运行、在其他机器上都会超出虚拟截止时间的任务；T' 表示那些至少在一台机器上运行会超出虚拟截止时间的任务。T 表示那些还未被分配的任务。3 个级别的任务组中，T'' 的优先级最高，然后是 T'，再次是 T。

本算法将虚拟截止时间设为 1～1.5 倍传统的 Min-Min 元任务调度算法 Makespan，再接着运行本文提出的算法，可以得出优化的 Makespan。实验采用基于范围的 ETC 矩阵产生方法(Range Based ETC Matrix Generation)，测试了所有 4 组任务异构度和机器异构度不同组合。实验结果表明，本算法比传统的 Min-Min 元任务调度算法提高 8%～10%。

参 考 文 献

1　Braun T D, Siegel H J, Beck N, et al. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In: IEEE Workshop on Advances in Parallel and Distributed Systems, West Lafayette, IN, Oct. 1998. 330～335

2　Maheswaran M, Ali S, Siegel H J, et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In: 8th IEEE Heterogeneous Computing Workshop(HCW '99), San Juan, Puerto Rico, Apr. 1999. 30～44

3　Eshagian M M. Heterogeneous Computing. Artech House, 1996

4　Freund R F, Gherrity M, Ambrosius S, et al. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In: Proc. The 7th IEEE Heterogeneous Computing Workshop (HCW '98), Orlando, Florida, USA, Mar. 1998. 184～199

5　Kim J-K, et al. Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines. Heterogeneous Computing Workshop, Nice, France, Apr. 2003

6　Ding Q, Chen G. A Benefit Function Mapping Heuristic for a Class of Meta-tasks in Grid Environments. In: CCGRID Workshop on Scheduling and Load Balancing on Clusters, Brisbane, Australia, May 2001. 654～659

7　Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputer Applications, 2001

8　Pinedo M. Scheduling: Theory, Algorithms, and Systems. Englewood Cliffs, NJ: Prentice Hall, 1995

9　Ali S, Siegel H J, Maheswaran M, et al. Task Execution Time Modeling for Heterogeneous Computing. In: IPDPS Workshop on Heterogeneous Computing, Cancun, Mexico, May 2000. 185～199

10　Fernández-Baca D. Allocating modules to processors in a distributed system. IEEE Transactions on Software Engineering, 1989, 15(11): 1427～1436