Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid

Noriyuki Fujimoto
Graduate School of Information Science and Technology,
Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531, Japan
fujimoto@ist.osaka-u.ac.jp

Kenichi Hagihara
Graduate School of Information Science and Technology,
Osaka University
1-3, Machikaneyama, Toyonaka, Osaka, 560-8531, Japan
hagihara@ist.osaka-u.ac.jp

Abstract

The most common objective function of task scheduling problems is makespan. However, on a computational grid, the 2nd optimal makespan may be much longer than the optimal makespan because the speed of each processor of a grid varies over time. So, if the performance measure is makespan, there is no approximation algorithm in general for scheduling onto a grid. In contrast, recently the authors proposed the computing power consumed by a schedule as a criterion of the schedule. For the criterion, this paper gives a $(1 + \frac{L_{cp}(n) \cdot m(\log_e(m-1)+1)}{n})$*-approximation algorithm for scheduling precedence constrained coarse-grained tasks with the same length onto a grid where* n *is the number of tasks,* m *is the number of processors, and* $L_{cp}(n)$ *is the length of the critical path of the task graph. The proposed algorithm does not use any prediction information on the performance of underlying resources.* $L_{cp}(n)$ *is usually a sublinear function of* n*. So, the above performance guarantee converges to one as* n *grows. This result implies a non-trivial result that the computing power consumed by an application on a grid can be limited within* $(1 + \frac{L_{cp}(n) \cdot m(\log_e(m-1)+1)}{n})$ *times that required by an optimal schedule in such a case.*

1 Introduction

Public-resource computing [3], such that the project SETI@home [3] has been carrying out, is the computing which is performed with donated computer cycles from computers in homes and offices in order to perform large scale computation faster. Public-resource computing is one form of grid computing. In public-resource computing, the original users also use their computers for their own purpose. So, their use may dramatically impact the performance of each grid resource. In the following, this paper refers to a set of computers distributed on the Internet and participated in public-resource computing as a **computational grid** (or simply a **grid**). In this paper, a grid is modeled as a heterogeneous parallel processors such that the speed of each processor varies over time.

This paper addresses task scheduling of a single application onto a computational grid. In a grid, communication delay is large. So, for achieving high performance, it is necessary that the granularity of an application on a grid is coarse-grained[1]. Hence, we assume that an application consists of coarse-grained tasks. For example, in the project SETI@home, each task takes 3.9 trillion floating-point operations, or about 10 hours on a 500MHz Pentium II, yet involves only a 350KB download and 1KB upload [3]. Therefore, for the purpose of scheduling a single application, a computational grid can be modeled as a heterogeneous parallel machine such that processor speed unpredictably varies over time and communication delays are negligible.

[1] In this paper, **coarse-grain** means that the execution time of every task is much longer than the communication delay required by the task. So, in a coarse-grained task graph, communication delay is negligible in spite of large communication delay on the Internet.

The most common objective function of task scheduling problems (both for a grid and for a non-grid parallel machine) is makespan. However, on a grid, makespan of a non-optimal schedule may be much longer than the optimal makespan because the computing power of a grid varies over time. For example, consider an optimal schedule with makespan OPT. If a grid is suddenly slowed down at time OPT and the slow speed situation continues for a long period, then the makespan of the second optimal schedule is far from OPT. So, if the criterion of a schedule is makespan, there is no approximation algorithm in general for scheduling onto a grid.

In contrast, recently the authors proposed the computing power consumed by a schedule as a criterion of the schedule and, for the criterion, gave an approximation algorithm for scheduling independent coarse-grained tasks with the same length onto a grid [7]. The proposed criterion is called total processor cycle consumption (**TPCC**, for short), which is the total number of instructions the grid could compute from the starting time of executing the schedule to the completion time. TPCC represents the total computing power consumed by an application.

For the criterion, this paper gives a $(1 + \frac{L_{cp}(n) \cdot m(\log_e(m-1)+1)}{n})$-approximation algorithm, called LR (Level-wise scheduling with Replication), for scheduling precedence constrained coarse-grained tasks with the same length onto a grid where n is the number of tasks, m is the number of processors, and $L_{cp}(n)$ is the length of the critical path of the task graph. LR does not use any prediction information on the performance of underlying resources. $L_{cp}(n)$ is usually a sublinear function of n. So, the above performance guarantee converges to one as n grows. The purpose of grid computing is to accomplish very large scale scientific computation. So, typically n is very large. This result implies that, regardless how the speed of each processor varies over time, the consumed computing power can be limited within $(1 + \frac{L_{cp}(n) \cdot m(\log_e(m-1)+1)}{n})$ times the optimal one. This is not trivial because makespan cannot be limited even in the case.

The remainder of this paper is organized as follows. First, Section 2 defines the grid scheduling model used in this paper. Next, Section 3 describes the contrast between the proposed algorithm and the algorithms in related works. Then, Section 4 shows the proposed algorithm LR. Last, Section 5 proves the performance guarantee of LR.

2 A Grid Scheduling Model

2.1 A Performance Model

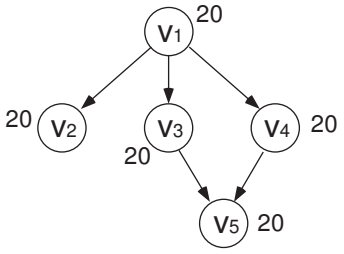
The **length** of a task is the number of instructions in the task. The **speed** of a processor is the number of instructions computed per unit time. A grid is heterogeneous, so processors in a grid have various speed by nature. In addition, the speed of each processor varies over time due to the load by the original users in public-resource computing. That is, the speed of each processor is the excess computing power of the processor which is not used by the original users and is dedicated to a grid. Let $s_{p,t}$ be the speed of processor p during time interval $[t, t+1)$ where t is a non-negative integer. Without loss of generality, we assume that the speed of each processor does not vary during time interval $[t, t+1)$ for every t by adopting enough short time as the unit time. We also assume that we cannot know the value of any $s_{p,t}$ in advance. $s_{p,t}$ may be zero if the load by the original users is very heavy or the processor is powered off. For simplicity, processor addition, processor deletion, and any failure are not considered in this paper. Fig. 1(b) shows an example of processor speed distribution. Note that processor P_3 has speed zero during time interval $[1, 3)$.

2.2 A Schedule

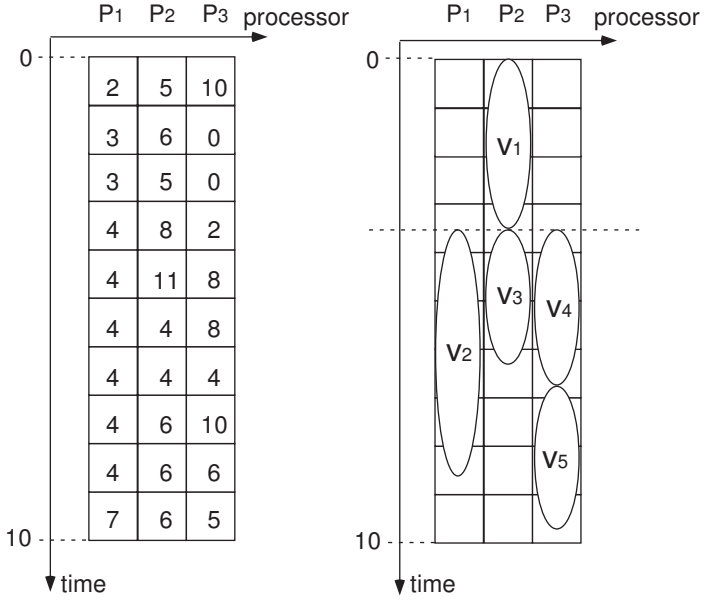
A **task graph** is represented by a directed acyclic graph $G = (V, E)$ with every node weight L, where V is a set of nodes, E is a set of directed edges, and L is a positive integer. We write a directed edge from node u to node v as (u, v). Fig. 1(a) shows an example of a task graph. The numerical value close to node v in Fig. 1(a) denotes the weight of v. This paper assumes that every node weight is the same as L. An application program on a grid is modeled as a task graph. A node in a task graph represents a task in the application. An edge (u, v) means that the computation of v needs the result of the computation of u. If $(u, v) \in E$, task u is called an **immediate predecessor** of task v. A task that has no immediate predecessor is called an **entry task**. The **length** of a path in a task graph is the number of tasks on the path. The **level** $level(v)$ of task v is the length of the longest path from v to an entry task. The longest path in a task graph is called the **critical path** in the task graph.

Let m be a number of processors in a computational grid. Let $G = (V, E)$ be a task graph with every task length L. We define a schedule of G as follows. A **schedule** S of G onto a grid with m processors is a finite set of triples $\langle v, p, t \rangle$ which satisfies the following rules R1, R2, and R3, where $v \in V$, p $(1 \leq p \leq m)$ is the index of a processor, and t is the **starting time** of task v. A triple $\langle v, p, t \rangle \in S$ means that the processor p computes the task v between time t and time $t + d$ where d is defined so that the number of instructions computed by the processor p during the time interval $[t, t + d)$ is exactly L. We call $t + d$ the **completion time** of the task v. Note that starting time and completion time of a task are not necessarily integral.

R1 For each $v \in V$, there is at least one triple $\langle v, p, t \rangle \in S$.



(a) task graph G with every task length 20



(b) processor speed s_{pt} (c) a schedule of G

Figure 1. The proposed grid scheduling model

R2 There are no two triples $\langle v, p, t\rangle, \langle v', p, t'\rangle \in S$ with $t \leq t' < t + d$ where $t + d$ is the completion time of v.

R3 If $(u, v) \in E$ and $\langle v, p, t\rangle \in S$, then there exists a triple $\langle u, p', t'\rangle \in S$ with $t' + d' \leq t$ where $t' + d'$ is the completion time of u.

Informally, the above rules can be stated as follows. The rule $R1$ enforces each task v to be executed at least once. The rule $R2$ says that a processor can execute at most one task at any given time. The rule $R3$ states that any task must receive the required data (if it exists) before its starting time. A triple $\langle v, p, t\rangle \in S$ is called the **task instance** of v. Note that $R1$ permits a task to be assigned onto more than one processor. Such a task has more than one task instances. To assign a task onto more than one processor is called **task replication**. The **makespan** of S is the maximum completion time of all the task instances in S. For example, Fig. 1(c) shows a schedule of G, i.e., $\{\langle v_1, P_2, 0\rangle, \langle v_2, P_1, 7/2\rangle, \langle v_3, P_2, 7/2\rangle, \langle v_4, P_3, 7/2\rangle, \langle v_5, P_3, 27/4\rangle\}$. The makespan of the schedule is $48/5$.

2.3 A Criterion of a Schedule

Let $G = (V, E)$ be a task graph with every task length L. Let S be a schedule of G onto a grid with m processors. Let M be the makespan of S. Let $s_{p,t}$ be the speed of processor p during the time interval $[t, t + 1)$. Then, the **total processor cycle consumption** (TPCC, for short) of S is defined as $\sum_{p=1}^{m} \sum_{t=0}^{\lfloor M \rfloor - 1} s_{p,t} + \sum_{p=1}^{m} (M - \lfloor M \rfloor) s_{p,\lfloor M \rfloor}$. For example, TPCC of the schedule in Fig. 1(c) is $32 + 55 + 48 + 7 \times \frac{3}{5} + 6 \times \frac{3}{5} + 5 \times \frac{3}{5} = 145.8$.

The criterion means the total computing power dedicated

to the application. The longer makespan is, the larger TPCC is. Conversely, the larger TPCC is, the longer makespan is. That is, every schedule with good TPCC is a schedule also with good makespan. The goodness of the makespan seems to be reasonable for the dedicated computing power, i.e., the corresponding TPCC . In this sense, the criterion is meaningful.

2.4 A Grid Scheduling Problem

This paper addresses the following grid scheduling problem:

- Instance: A task graph G of n tasks with the same length L, a number m of processors, unpredictable speed $s_{p,t}$ of processor p during the time interval $[t, t+1)$ for each p and t
- Solution: A schedule S of G onto a grid with m processors
- Measure: The TPCC $\sum_{p=1}^{m} \sum_{t=0}^{\lfloor M \rfloor - 1} s_{p,t} + \sum_{p=1}^{m} (M - \lfloor M \rfloor) s_{p,\lfloor M \rfloor}$ of S where M is the makespan of S

A **makespan optimal schedule** is a schedule with the smallest makespan among all the schedules. An **optimal schedule** is a schedule with the smallest TPCC among all the schedules. Note that the set of makespan optimal schedules is the same as the set of optimal schedules.

3 Related Works

This section summarizes known results on complexity, heuristic algorithms, optimal algorithms, and approximation algorithms for scheduling onto a homogeneous parallel machine, a heterogeneous parallel machine, or a grid. First, Section 3.1 introduces a brief notation to describe various scheduling problems. Next, Section 3.2 summarizes known results in the case of invariable processor speed. In this case, all the known results are for some simple task graphs (i.e., a set of independent tasks, tree, or chains). No results are known for a general task graph. Then, Section 3.3 summarizes several known results in the case of variable processor speed. In this case, all the known results are heuristic algorithms, i.e., algorithms without performance guarantee.

Thus, for grid scheduling (i.e., scheduling for $Q; s_{jk}$), all the known algorithms are heuristic algorithms, i.e., algorithms without performance guarantee. In contrast to this, the proposed algorithm is an approximation algorithm, i.e., an algorithm with performance guarantee. Even in the case of invariable processor speed (i.e., scheduling for Q), neither optimal algorithm nor approximation algorithm is known for a general task graph (i.e., prec). The proposed algorithm is the first approximation algorithm for grid scheduling.

3.1 Problem Description

For describing problems, the shorthand notation slightly extended from one in the literature [10] is used. Problems are described by three fields (e.g., $P|p_j = 1|C_{max}$ where $|$ is a delimiter): the left field represents the machine environment, the middle field describes constraints, and the right field describes the objective function criterion.

Possible machine environments in the left field include 'P', 'P_m', 'Q', 'Q_m', and '$Q; s_{jk}$'. Notation 'P' indicates that m identical parallel processors are available, where m is part of the problem instance. Notation 'P_m' indicates that m identical parallel processors are available. Notation 'Q' indicates m parallel processors with speed ratios, where m is part of the problem instance. Notation 'Q_m' indicates m parallel processors with speed ratios. Notation '$Q; s_{jk}$' indicates Q such that processor speed unpredictably varies over time.

Possible constraints in the middle field include ones on the topology of the task graph, task processing times, and communication delays. Notation 'prec' denotes that a precedence relation is imposed on the tasks. Notation 'chains' and so on denotes the precedence relation imposed on the tasks. If any topology is not specified, then it means that a set of independent tasks is specified. Notation '$p_j = 1$' indicates all the tasks have unit processing times. Notation 'unpredictable p_j' indicates every task processing time cannot be known in advance. Notation '$c_{jk} = 1$' indicates all the communication delays are unit times.

Possible criteria in the right field include C_{max} and $\sum s_{jk}$. Notation 'C_{max}' indicates that the criterion is makespan of a schedule. Notation '$\sum s_{jk}$' indicates that the criterion is TPCC of a schedule.

Using this notation, the problem tackled in this paper is represented as $Q; s_{jk}|\text{prec}; p_j = L| \sum s_{jk}$ (where L is a given constant).

3.2 The Case of Invariable Processor Speed

Let n be the number of tasks. Let m be the number of processors. Let PTAS be the class of problems that admit a polynomial-time approximation scheme [24]. Let FPTAS be the class of problems that admit a fully polynomial-time approximation scheme [24]. Then, the following results are known:

- $P||C_{max}$ is strongly $\mathcal{NP}$-hard [9].
- $P_2||C_{max}$ is $\mathcal{NP}$-hard [18].

- $P|\text{prec}; p_j = 1|C_{max}$ is $\mathcal{NP}$-hard [23].
- $Q|\text{chains}; p_j = 1|C_{max}$ is $\mathcal{NP}$-hard [14].
- $Q|\text{chains}; p_j = 1|C_{max}$ is strongly $\mathcal{NP}$-hard [16].
- $Q|\text{chains}; p_j = 1; c_{jk} = 1|C_{max}$ is strongly $\mathcal{NP}$-hard [16].
- $Q_2|\text{chains}; p_j = 1|C_{max}$ can be solved in polynomial time [5].
- $Q_2|\text{tree}; p_j = 1|C_{max}$ can be solved in polynomial time if one processor is a times faster than the other, where a is an integer [15].
- $Q_2|\text{tree}; p_j = 1|C_{max}$ can be solved in polynomial time if one processor is $(1 + 1/k)$ times faster than the other, where k is an integer [8].
- $Q|p_j = 1|C_{max}$ can be solved in $O(n^2)$ time [10].
- $Q|p_j = 1|C_{max}$ can be solved in $O(n \log m)$ time [17].
- $Q_2||C_{max}$ is approximable within a factor of $(1 + \epsilon)$ in $O(n^2/\epsilon)$ time for any $\epsilon > 0$ [13]. ($Q_2||C_{max}$ is in FPTAS.)
- $Q||C_{max}$ is approximable within a factor of $(1 + \epsilon)$ in $O((\log m + \log(3/\epsilon))(n + 8/\epsilon^2)m \cdot n^{3+40/\epsilon^2})$ time for any $0 < \epsilon \leq 1$ [12]. ($Q||C_{max}$ is in PTAS.)
- $Q_2|\text{complete intree}; p_j = 1; c_{jk} = 1|C_{max}$ can be solved in $O(n)$ time [4].
- $Q_2|\text{chains}; p_j = 1; c_{jk} = 1|C_{max}$ can be solved in $O(n)$ time [16]

3.3 The Case of Variable Processor Speed

- For $Q; s_{jk}||C_{max}$, some heuristic algorithms without task replication are known [1, 6, 11, 20, 25].
- For $Q; s_{jk}||C_{max}$, some heuristic algorithms with task replication are known [3, 19].
- For $Q; s_{jk}||C_{max}$, the authors gave a $(1 + \frac{m(\log_e(m-1)+1)}{n})$-approximation algorithm [7].
- For $Q; s_{jk}|\text{prec}|C_{max}$, some heuristic algorithms without task replication are known [2, 21, 22].

Algorithm LR
Input: A task graph G with the same length tasks, a number m of processors
Output: A dynamic schedule of G onto m processors
begin
  Compute the level of every task in G.
  Let $L_{cp}(n)$ be the maximum level of the levels of all the tasks in G.
  for i:=1 **to** $L_{cp}(n)$ **do begin**
    Let V be the set of the tasks with level i in G.
    /* Note that every task in V is independent of */
    /* the other tasks in V. */
    /* Hence, V can be scheduled by RR. */
    Schedule V onto the m processors
    using sub-algorithm RR.
  end
end

Figure 2. A Pseudo Code of LR

4 The Proposed Algorithm LR

For minimum TPCC scheduling of independent coarse-grained tasks with the same length, recently the authors presented $(1 + \frac{m(\log_e(m-1)+1)}{n})$-approximation algorithm RR (list scheduling with Round-robin order Replication) where n is the number of tasks and m is the number of processors in a grid [7]. RR does not use any prediction information on the underlying resources. RR achieves the above approximation ratio regardless how the speed of each processor unpredictably varies over time. RR uses task replication.

LR uses RR as a sub-algorithm. Notice that, in a task graph, the tasks with the same level are independent each other. LR schedules precedence constrained tasks level by level using RR. First, LR schedules the tasks with level 1 (i.e., the entry tasks) using RR. After completion of all the tasks with level 1, LR schedules the tasks with level 2 using RR. After completion of all the tasks with level 2, LR schedules the tasks with level 3 using RR. LR repeats this process until all the tasks are scheduled. Fig. 2 shows a pseudo code of LR.

5 The Performance Guarantee of the Proposed Algorithm LR

On the properties of RR, the authors proved the following lemmata in [7]:

Lemma 1 *From the starting time of a schedule to the completion time, RR never makes processors idle.*

Lemma 2 *The total number of replicas in a schedule generated by RR is at most* $m \log_e(m-1) + m$.

These lemmata can be used to prove the approximation ratio of LR as follows.

Theorem 1 *The TPCC of a schedule generated by LR is at most* $(1 + (L_{cp}(n) \cdot m(\log_e(m-1)+1))/n)$ *times the optimal TPCC .*

Proof: Consider scheduling precedence constrained n tasks such that every task has L instructions. Let OPT be the optimal makespan. Let $L_{cp}(n)$ be the length of the critical path of the task graph. Let n_l be the number of the tasks with level l ($1 \leq l \leq L_{cp}(n)$). From Lemma 1 and Lemma 2, LR computes the n_l tasks for at most $(n_l + m(\log_e(m-1)+1))L$ TPCC using RR as a subroutine. So, the total TPCC required by LR is $\sum_{l=1}^{L_{cp}(n)}(n_l + m(\log_e(m-1)+1))L = (\sum_{l=1}^{L_{cp}(n)} n_l)L + (L_{cp}(n) \cdot m(\log_e(m-1)+1))L = nL + (L_{cp}(n) \cdot m(\log_e(m-1)+1))L$. On the other hand, the optimal TPCC is at least nL because the grid must perform nL instructions until time OPT. So, the theorem follows. ∎

Theorem 1 implies the following things:

- $L_{cp}(n)$ is usually a sublinear function of n. So, if m is fixed, then the approximation ratio of LR decreases with an increase in n. Thus, the approximation ratio of LR converges to one as n grows. The convergence speed depends on how small $L_{cp}(n)$ is relative to n.

- If n is fixed, then the approximation ratio of LR increases gradually with an increase in m.

Therefore, it turns out that LR generates almost optimal schedules if the number of tasks is enough larger than the number of processors. Table 1 through Table 3 show some values of approximation ratios of the proposed algorithm LR in the case that $L_{cp}(n) = \log_2 n$, $L_{cp}(n) = \sqrt[3]{n}$, and $L_{cp}(n) = \sqrt{n}$. As shown also in Table 1 through Table 3, small $L_{cp}(n)$ is desirable. Small $L_{cp}(n)$ means high parallelism of the corresponding task graph. In conclusion, it can be stated that the proposed algorithm achieves good approximation ratios if the given task graph has high parallelism, a large number of tasks relative to the number of processors, and coarse granularity.

6 Conclusion

This paper has shown that minimum processor cycle scheduling is approximable within a factor of $(1 + \frac{L_{cp}(n) \cdot m(\log_e(m-1)+1}{n})$ without using any kind of prediction information in the case of precedence constrained coarse-grained tasks with the same length. This result implies a non-trivial result that, in such a case, wasteful use of dedicated computing power can be limited without any prediction information on the performance of underlying resources, regardless how the speed of each processor varies over time.

References

[1] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid? In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 520–528, 2000.

[2] A. Alhusaini, V. Prasanna, and C. Raghavendra. A unified resource scheduling framework for heterogeneous computing environments. In *8th Heterogeneous Computing Workshop (HCW)*, pages 156–165, 1999.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[4] J. Blazewicz, F. Guinand, B. Penz, and D. Trystram. Scheduling complete trees on two uniform processors with integer speed ratios and communication delays. *Parallel Processing Letters*, 10(4):267–277, 2000.

[5] P. Brucker, J. Hurink, and W. Kubiak. Scheduling identical jobs with chain precedence constraints on two uniform machines. *Mathematical Methods of Operations Research*, 49(2):211–219, 1999.

[6] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *9th Heterogeneous Computing Workshop (HCW)*, pages 349–363, 2000.

[7] N. Fujimoto and K. Hagihara. Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid. In *32nd Annual International Conference on Parallel Processing (ICPP-03)*, pages 391–398, 2003.

[8] H. N. Gabow. An almost-linear algorithm for two-processor scheduling. *Journal of the ACM*, 29:766–780, 1982.

[9] M. R. Garey and D. S. Johnson. Strong NP-completeness results: motivation, examples, and implications. *Journal of the ACM*, 25(3):499–508, 1978.

[10] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[11] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In *1st IEEE/ACM International Workshop on Grid Computing (GRID)*, pages 214–227, 2000.

[12] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[13] E. Horowitz and S. K. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23:317–327, 1976.

Table 1. Approximation ratios of the proposed algorithm LR in case that $L_{cp}(n) = \log_2 n$

		# of processors					
		32	64	128	256	512	1024
# of tasks	100000	1.00581438	1.013898031	1.032282797	1.073488173	1.164771062	1.365081332
	200000	1.003072587	1.007348122	1.017074676	1.038879237	1.087191507	1.193222454
	300000	1.002112892	1.00505439	1.011747072	1.026752147	1.060001881	1.1329806
	400000	1.001618992	1.003873614	1.009003977	1.020507194	1.045998741	1.101952121
	500000	1.001316492	1.003150285	1.00732336	1.016680701	1.037417864	1.082937204
	600000	1.001111579	1.00266023	1.006184628	1.01408779	1.031602933	1.070050896
	700000	1.000963291	1.002305557	1.005360411	1.012210918	1.027393603	1.060722354
	800000	1.000850845	1.002036584	1.004735308	1.010787385	1.024200865	1.053646508
	900000	1.000762553	1.001825367	1.004244404	1.009669412	1.021693359	1.048089141
	1000000	1.000691326	1.001654964	1.003848336	1.008767381	1.019670127	1.04360496
	1100000	1.000632613	1.00151449	1.00352182	1.008023726	1.018002086	1.039907924
	1200000	1.000583355	1.001396633	1.00324786	1.007399754	1.016602462	1.036805746
	1300000	1.00054142	1.00129629	1.003014605	1.006868474	1.015410731	1.034164305
	1400000	1.000505274	1.001209794	1.002813531	1.00641048	1.01438337	1.031887147
	1500000	1.000473784	1.001134437	1.002638348	1.006011452	1.01348826	1.029903099
	1600000	1.000446097	1.00106818	1.002484313	1.005660586	1.012701179	1.028158477
	1700000	1.000421558	1.001009453	1.002347781	1.005349581	1.012003503	1.026612009
	1800000	1.000399654	1.000957029	1.002225899	1.005071945	1.011380677	1.025231436
	1900000	1.000379977	1.000909936	1.00211641	1.004822532	1.010821154	1.023991171
	2000000	1.000362203	1.000867393	1.002017496	1.004597205	1.010315661	1.022870659

[14] W. Kubiak. Exact and approximate algorithms for scheduling unit time tasks with tree-like precedence constraints. In *Abstracts EURO IX - TIMS XXVIII Paris*, page 195, 1988.

[15] W. Kubiak. Optimal scheduling of unit-time tasks on two uniform processors under tree-like precedence constraints. *Zeitschrifts fur Operations Research*, 33:423–437, 1989.

[16] W. Kubiak, B. Penz, and D. Trystram. Scheduling chains on uniform processors with communication delays. *Journal of Scheduling*, 5(6):459–476, 2002.

[17] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *Hand Book in Operations Research and Management Science*, volume 4: Logistics of Production and Inventory, pages 445–522. North-Holland, 2002.

[18] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Machines*, 1:343–362, 1977.

[19] D. Paranhos, W. Cirne, and F. Brasileiro. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *International Conference on Parallel and Distributed Computing (Euro-Par), Lecture Notes in Computer Science*, volume 2790, pages 169–180, 2003.

[20] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *9th Heterogeneous Computing Workshop (HCW)*, pages 3–16, 2000.

[21] M. Sirbu and D. Marinescu. A scheduling expert advisor for heterogeneous environments. In *6th Heterogeneous Computing Workshop (HCW)*, pages 74–87, 1997.

[22] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye. The software architecture of a virtual distributed computing environment. In *High-Performance Distributed Computing Conf.*, pages 40–49, 1997.

[23] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Science*, 10:384–393, 1975.

[24] V. V. Vazirani. *Approximation Algorithms*. Springer Verlag, 2001.

[25] M. Yarrow, K. McCann, R. Biswas, and R. Van der Wijngaart. An advanced user interface approach for complex parameter study process specification on the information power grid. In *1st IEEE/ACM International Workshop on Grid Computing (GRID)*, pages 146–157, 2000.

Table 2. Approximation ratios of the proposed algorithm LR in case that $L_{cp}(n) = \sqrt[3]{n}\}$

		# of processors					
		32	64	128	256	512	1024
# of tasks	100000	1.051325301	1.123716658	1.289084476	1.660998835	2.487189169	4.304295129
	200000	1.032291326	1.077853436	1.181945459	1.416070474	1.936205072	3.080244698
	300000	1.024627458	1.059382419	1.138788627	1.317397709	1.714211207	2.587029865
	400000	1.020321467	1.049003004	1.114535282	1.261941625	1.589439539	2.309806644
	500000	1.017507568	1.042219625	1.098683746	1.225694914	1.507884526	2.128599007
	600000	1.015500464	1.037380855	1.087375907	1.199837128	1.449703068	1.999322568
	700000	1.013984233	1.033725311	1.07883288	1.180301169	1.40574519	1.901648682
	800000	1.012791325	1.030849165	1.072111119	1.164929709	1.371157292	1.824793782
	900000	1.011823922	1.028516641	1.066659715	1.152463108	1.343105345	1.762461198
	1000000	1.011020759	1.026580062	1.062133595	1.142112347	1.319814123	1.710706664
	1100000	1.010341343	1.024941822	1.058304675	1.133355896	1.300110211	1.666923002
	1200000	1.009757749	1.023534601	1.055015647	1.125834052	1.283184281	1.62931197
	1300000	1.009250041	1.02231034	1.052154204	1.119290006	1.268458511	1.596589699
	1400000	1.008803574	1.021233733	1.049637838	1.11353509	1.255508396	1.567812963
	1500000	1.008407326	1.020278209	1.047404458	1.10842732	1.244014459	1.542271902
	1600000	1.008052832	1.019423359	1.045406365	1.103857619	1.233731268	1.519421175
	1700000	1.00773347	1.01865322	1.043606256	1.099740686	1.224466892	1.498834312
	1800000	1.007443984	1.017955116	1.041974506	1.096008773	1.21606889	1.480172591
	1900000	1.007180138	1.01731884	1.04048726	1.09260733	1.208414516	1.46316327
	2000000	1.006938484	1.016736072	1.039125077	1.089491899	1.201403733	1.447584064

Table 3. Approximation ratios of the proposed algorithm LR in case that $L_{cp}(n) = \sqrt{n}\}$

		# of processors					
		32	64	128	256	512	1024
# of tasks	100000	1.347815073	1.839151513	2.962069069	5.488451563	11.1023397	23.45237634
	200000	1.245876122	1.593237177	2.387127247	4.173284344	8.142372524	16.87410679
	300000	1.20073304	1.484328181	2.132488764	3.590784035	6.831339181	13.96038679
	400000	1.173827536	1.419415756	1.980714535	3.243585781	6.049889852	12.22362817
	500000	1.155468521	1.375121749	1.87714753	3.006663695	5.516637929	11.03847648
	600000	1.141917603	1.342427558	1.800702119	2.831784228	5.123026016	10.16367046
	700000	1.131386506	1.317018991	1.741291464	2.695873355	4.817121753	9.483793093
	800000	1.122898061	1.296538588	1.693403624	2.586322172	4.570546262	8.935773396
	900000	1.115867246	1.279574949	1.653738579	2.495581632	4.36630879	8.481849892
	1000000	1.109919591	1.265224622	1.620183947	2.418819468	4.19353323	8.09785064
	1100000	1.104802798	1.252878916	1.591316512	2.352779924	4.044891744	7.767489613
	1200000	1.100339854	1.242110757	1.566137715	2.295178684	3.915242924	7.479340063
	1300000	1.096402408	1.232610481	1.543923502	2.244359388	3.800858722	7.225116409
	1400000	1.092894822	1.224147343	1.52413435	2.199087807	3.698961192	6.998644659
	1500000	1.089744175	1.216545408	1.506358894	2.158422888	3.607432251	6.795217354
	1600000	1.086893768	1.209667878	1.490277267	2.121632891	3.524624926	6.611174086
	1700000	1.084298758	1.203406561	1.475636497	2.088139102	3.449236669	6.443619918
	1800000	1.081923152	1.197674615	1.462233527	2.057477004	3.380221953	6.290231153
	1900000	1.079737679	1.192401418	1.449903232	2.029268842	3.316730542	6.149118126
	2000000	1.07771826	1.187528874	1.438509765	2.003203847	3.258062964	6.018726242