Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors

Hidehiro Kanemitsu, *Member, IEEE*, Masaki Hanada, *Member, IEEE*, and Hidenori Nakazato, *Member, IEEE*

Abstract—Parallelization paradigms for effective execution in a Directed Acyclic Graph (DAG) application have been widely studied in the area of task scheduling. Schedule length can be varied depending on task assignment policies, scheduling policies, and heterogeneity in terms of each processor and each communication bandwidth in a heterogeneous system. One disadvantage of existing task scheduling algorithms is that the schedule length cannot be reduced for a data intensive application. In this paper, we propose a clustering-based task scheduling algorithm called Clustering for Minimizing the Worst Schedule Length (CMWSL) to minimize the schedule length in a large number of heterogeneous processors. First, the proposed method derives the lower bound of the total execution time for each processor by taking both the system and application characteristics into account. As a result, the number of processors used for actual execution is regulated to minimize the Worst Schedule Length (WSL). Then, the actual task assignment and task clustering are performed to minimize the schedule length until the total execution time in a task cluster exceeds the lower bound. Experimental results indicate that CMWSL outperforms both existing list-based and clustering-based task scheduling algorithms in terms of the schedule length and efficiency, especially in data-intensive applications.

Index Terms—Task scheduling, DAG scheduling, task clustering, heterogeneous systems

1 Introduction

Scheduling tasks in an application for parallel execution is a fundamental issue of high-performance computing systems. The objective for task scheduling depends on system requirements, e.g., maximizing the throughput, minimizing the economic cost, finishing execution by a user-specified deadline, and response time minimization. In a real-world system such as cloud computing or grid computing systems, multiple objectives must be defined in order to satisfy client requirements [1], [2], [3], [4], [5], [6]. In particular, minimizing the response time (hereafter, referred to as schedule length) is one major factor in determining system performance. The method used to find the optimal schedule varies based on the application model such as an independent task scheduling model [7], scheduling tasks with dependencies, and so on. An application with data dependencies, namely, a Directed Acyclic Graph (DAG) application, is an important representation for executing a complex, large-scale application requiring communication among tasks. DAG scheduling is known as NP-complete[8], and many heuristics for DAG scheduling in both homogeneous systems [9], [10], [11], [12], [13] and heterogeneous systems [19], [20], [21], [22], [23], [24], [25], [27], [28], [29], [31] have been proposed. In a real-world system with various requirements, it is conceivable for a specific subset of a large number of processors to be chosen for parallel execution of a DAG application, e.g., executing multiple work-flow type applications in a computer cluster and choosing a subset of computers based on the characteristics of each work-flow-type application.

List-based task scheduling heuristics for DAG scheduling algorithms in heterogeneous systems such as Heterogeneous Earliest Finish Time (HEFT) [20], Predict Earliest Finish Time (PEFT) [21], Constrained Earliest Finish Time (CEFT) [22], Minimizing Schedule Length (MSL) [24], and Heterogeneous Selection Value (HSV) [25] are high performance, low complexity algorithms. In these algorithms, scheduling priorities are decided by the average processing time and the average communication time; that is, scheduling priorities vary depending on the variation in each processing speed and each communication bandwidth. Moreover, these heuristics use only a subset of processors for execution because of the task assignment policy; thus, it follows that each scheduling priority includes unassigned processor information as well as information from the assigned processors. As a result, the schedule length cannot be reduced if there is a large variation in the task execution time and communication time or if the number of given processors is large. In particular, for a data intensive application, if a task is not assigned to a processor appropriately because of such variations, unnecessary data communications occur, and the start time of its successor tasks is delayed.

Unlike list-based scheduling heuristics, task clustering heuristics in heterogeneous systems are few [27], [28], [29], [31]. Although task clustering requires the clustering priority

- *H. Kanemitsu is with the Global Education Center, Waseda University, 1-6-1, Nishiwaseda, Shinjuku, Tokyo 169-8050, Japan. E-mail: kanemih@ruri.waseda.jp.*
- *M. Hanada is with the Department of Information Systems, Tokyo University of Information Sciences, 4-1, Onaridai, Wakaba-ku, Chiba 265-8501, Japan. E-mail: mhanada@rsch.tuis.ac.jp.*
- *H. Nakazato is with the Department of Communications and Computer Engineering, Waseda University, 3-14-9 Ohkubo, Shinjuku-ku 169-0072, Japan. E-mail: nakazato@waseda.jp.*

Manuscript received 12 July 2015; revised 29 Jan. 2016; accepted 2 Feb. 2016. Date of publication 8 Feb. 2016; date of current version 12 Oct. 2016. Recommended for acceptance by A.R. Butt.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2526682

to be assigned for each task, the priority cannot be determined until each task is assigned to a processor. In existing works, clustering priorities are decided by assuming a homogeneous system such as in Clustering for Heterogeneous Processors (CHP) [29], [30] and by using only the task and data sizes such as in Resource Aware Clustering (RAC) [27], Flexible Clustering and Scheduling (FCS) [28], and Triplet [31]; that is, existing task clustering heuristics for heterogeneous systems do not consider the actual processing time and communication time when deriving clustering priorities. In CHP and Triplet, the schedule length cannot be made much smaller because the large amount of data among the two tasks cannot be localized, even if the data is assigned to processors with a small communication bandwidth. As for RAC, the lower bound of the total execution time in a task cluster is determined by considering all the processing speeds; therefore, the number of processors used for actual execution depends on the given set of processors.

In light of the disadvantages of the list-based and task clustering heuristics, when a system has a large number of processors and a subset of processors must be chosen for execution, a task scheduling model must satisfy the following requirements: (i) a scheduling priority must be derived by the information of actual assigned processors, and (ii) the size of the workload assigned to a processor should not be affected by the given number of processors. In this paper, we propose a clustering-based task scheduling called Clustering for Minimizing the Worst Schedule Length (CMWSL) that satisfies these two requirements. CMWSL consists of four phases: (a) deriving the lower bound of the total execution time for each processor using the Worst Schedule Length (WSL) at the current clustered DAG, (b) the processor that minimizes the WSL is chosen for the cluster assignment target, (c) task clustering is performed, and the generated task cluster is assigned to the processor determined in phase (b), and (d) list-based task scheduling is performed. Experimental results show that CMWSL outperforms both existing list-based and clustering-based task scheduling algorithms in terms of schedule length and efficiency, especially in data intensive applications.

The reminder of this paper is organized as follows. Section 2 presents the assumed model and problem definition. Section 3 discusses existing research on DAG scheduling, and Section 4 presents the details of the proposed method. Section 5 details and discusses the experimental results, and finally, we conclude in Section 6.

2 Assumptions

2.1 System Model

Assume that a job can be expressed as a DAG, which is also known as a work-flow-type job. Let $G^s = (V, E, V^s_{cls})$ be the DAG, where V is the set of tasks, E is the set of edges (data communications among tasks), and V^s_{cls} is the set of task clusters including one or more tasks by s task clustering steps. This means that G^s has $(|V| - s)$ unclustered tasks and $|V| = |V^s_{cls}|$ for $s = 0$. The ith task is denoted by n_i. Furthermore, let $w(n_i)$ be the size of n_i, i.e., $w(n_i)$ is the sum of the unit time for processing by the reference processor. We define the data dependency and direction of data transfer from n_i to n_j by $e_{i,j}$, and $c(e_{i,j})$ is the sum of the unit time for transferring the data from n_i to n_j over the reference communication link. One constraint imposed by a DAG is that a task cannot begin execution until all data from its predecessor tasks arrive. We define $pred(n_i)$ to be the set of immediate predecessors of n_i, and $suc(n_i)$ to be the set of immediate successors of n_i. If $pred(n_i) = \emptyset$, n_i is called the START task, and if $suc(n_i) = \emptyset$, n_i is called the END task. If there are one or more paths from n_i to n_j, we denote such a relation by $n_i \prec n_j$.

We assume that each processor is completely connected to other processors over the network and have heterogeneous processing speeds and communication bandwidths. The set of processors is denoted by $P = \{p_1, p_2, \ldots p_n\}$, and the processing speed of p_i is denoted by α_i when the processing speed of the reference processor is set to 1. The execution time when n_k is processed on p_i is given by $t_p(n_k, \alpha_i) = w(n_k)/\alpha_i$. Let the set of communication bandwidths be $\beta = \{\beta_1, \beta_2, \ldots \beta_n\}$ when the communication bandwidth of the reference communication link is set to 1. In general, the communication setup time for the preparation, denoted by O_k occurs before p_k sends the data. If $c(e_{i,j})$ is sent from p_k to p_l, the communication time is defined by O_k and the communication speed $L_{k,l}$; in particular, $L_{k,l} = \min\{\beta_k, \beta_l\}$ and the communication time is given by

$$t_c(e_{i,j}, L_{k,l}) = O_k + c(e_{i,j})/L_{k,l}. \tag{1}$$

We assume the communication setup time is negligible, i.e., $O_k = 0$ for $\forall p_k \in P$.

2.2 Task Clustering

A task cluster is a set of tasks explicitly grouped together before every task is scheduled. As a result, every task in a task cluster is assigned to the same processor. Let the ith task cluster in V^s_{cls} be $cls_s(i)$. If n_k is included in $cls_s(i)$ by the $(s+1)$st task cluster, it is expressed by $cls_{s+1}(i) \leftarrow cls_s(i) \cup \{n_k\}$. If any two tasks, e.g., n_i and n_j, are included in the same task cluster, they are assigned to the same processor. In this case, the communication time between n_i and n_j is zero. The total task size in a task cluster is called the task cluster size. The task cluster size divided by the processing time is called the task cluster processing time and is denoted by $T(cls_s(i), p_p)$.

2.3 Schedule Length

In a DAG application, a task can begin executing if all data from its immediate predecessors have arrived. Let the start time of n_i on p_p be $t_s(n_i, p_p)$, and let the completion time of n_j on p_p be $t_f(n_j, p_p)$. Then $t_f(n_j, p_p)$ is defined as follows:

$$t_f(n_j, p_p) = t_s(n_j, p_p) + t_p(w(n_j), \alpha_p). \tag{2}$$

When all of the data from $pred(n_j)$ has arrived at n_j, n_j can begin executing immediately. However, even if all of the data from $pred(n_j)$ has arrived at n_j, n_j cannot begin until the execution of another task in the same processor is finished. The time when all of the data from every immediate predecessor task has arrived at n_j is called the Data Ready Time (DRT) [17]. If the DRT of n_j on p_q is defined when every task in $pred(n_j)$ is scheduled, $t_{dr}(n_j, p_q)$ is derived as follows:

$$t_{dr}(n_j, p_q) = \max_{\substack{n_i \in pred(n_j), \\ p_p \in P}} \{t_f(n_i, p_p) + t_c(c(e_{i,j}), L_{p,q})\}. \quad (3)$$

In (3), $t_{dr}(n_j)$ is derived from the completion time of tasks in $pred(n_j)$ assigned to the same processor and the data arrival time from tasks in $pred(n_j)$ assigned to other processors. In the former case, $p = q$ and $t_c(c(e_{i,j}), L_{p,q}) = 0$. On the other hand, the latter case requires the data transfer time. The start time of n_j, i.e., $t_s(n_j, p_q)$, is derived using the DRT as follows:

$$t_s(n_j, p_q) = \max\{t_f(n_i, p_q), t_{dr}(n_j, p_q)\}, \quad (4)$$

where n_i has been scheduled. In (4), the choice of $t_f(n_i, p_q)$ affects the completion time of n_j. If $t_s(n_j, p_q)$ is derived by $t_f(n_i, p_q)$, there must be an idle time slot that can accommodate $t_p(n_j, \alpha_q)$ and start at $t_f(n_i, p_q)$ in the case of an insertion-based policy. On the other hand, if $t_s(n_j, p_q)$ is derived by $t_{dr}(n_j, p_q)$, there is a data waiting time for data arriving from other processors. The data waiting time, denoted by $t_w(n_j, p_p)$, is given by

$$t_w(n_j, p_q) = \begin{cases} 0, \ if \ t_f(n_i, p_q) \geq t_{dr}(n_j, p_q), \\ t_{dr}(n_j, p_q) - t_f(n_i, p_q), \ otherwise. \end{cases} \quad (5)$$

During the time slot derived in (5), some tasks are inserted by an insertion-based policy in order to minimize their completion time. The schedule length is expressed by the completion time of the END task as follows:

$$Schedule \ Length = \max_{suc(n_i)=\emptyset, p_p \in P} \{t_f(n_i, p_p)\}, \quad (6)$$

where n_i is the END task, and n_i is assigned to p_p.

3 Related Work

The objective of the proposed algorithm is to find the set of processors and schedule that minimize the schedule length. To this end, our previous works [14] and [15] focus on determining the lower bound of each cluster processing time in order to find a subset of given processors. In particular, the WSL was defined and its effect on the schedule length was proved, and the lower bound of the assignment unit size for a processor was derived. In the following section, we describe the problem in conventional task scheduling algorithms, then we briefly describe these two main contributions provided by [14] and [15]. For more details, please refer to Appendix A in the supplemental file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2016.2526682.

3.1 Problem in Conventional Approaches

As we mentioned in Section 1, list-based task scheduling heuristics [20], [21], [22], [24], [25] cannot derive an accurate scheduling priority due to a performance variation for each processor. If the number of processors is large enough to assign a task to an idle processor to minimize the finish time, its finish time can be made smaller to some extent despite the existence of the performance variation. On the other hand, if the number of processors is not enough, each task must be assigned to a non-idle processor. In such a case, the accuracy of the scheduling priority is a critical issue. Deriving an inaccurate scheduling priority can raise a problem that a "critical task", which actually contributes to the reduction of the schedule length, is chosen after an non-critical task. That is, the start time of the critical task can be delayed. In case of task clustering heuristics [27], [28], [29], [31], each task cluster size becomes larger and the degree of parallelism is degraded because they have no criteria to maintain the degree of parallelism with an appropriate task cluster size. As a result, in conventional task scheduling algorithms, the schedule length cannot be made smaller in case that the number of processors is not enough for allocating tasks to idle processors.

3.2 Definition of WSL

The WSL is the maximum execution path length when each task is executed as late as possible in a processor, provided that there is no data waiting time for each task once the processor starts executing. Since the WSL derivation requires the assignment and clustering state $M(G^s)$, first, we define $M(G^s)$. At the initial clustering state G^0, each task belongs to a task cluster. Suppose that it is assigned to the virtual processor $p_i^{vt} \in P^{vt}$ having the maximum processing speed and the maximum communication bandwidth, i.e.,

$$\alpha_i^{vt} = \max_{p_k \in P}\{\alpha_k\}, \beta_i^{vt} = \max_{p_k \in P}\{\beta_k\}, L_{i,j}^{vt} = \beta_i^{vt}, \ \forall i \neq j. \quad (7)$$

Let $M : G^s \rightarrow P$ be the assignment state after a processor assignment is performed by G^s. $M(G^0)$ corresponds to $\{(cls_0(1), p_1^{vt}), (cls_0(2), p_2^{vt}), \ldots, (cls_0(|V|), p_{|V|}^{vt})\}$, where $|V| = |V_{cls}^0|$. From the initial assignment state, each virtual processor is replaced by an actual processor, and the number of task clusters is reduced by a processor assignment and task cluster, i.e., $|M(G^s)| \leq |M(G^0)|$ for $s > 0$.

For a task cluster $cls_s(i)$, $top(cls_s(i))$ is the set of tasks that begin executing first in $cls_s(i)$, $in(cls_s(i))$ is the set of tasks with incoming edges from other task clusters, and $out(cls_s(i))$ is the set of tasks with outgoing edges to other task clusters; $btm(cls_s(i))$ is the set of tasks that has no immediate successor tasks in $cls_s(i)$. Table 1 shows the notation for deriving the WSL. In particular, $dc(n_k, cls_s(i))$ is the set of descendant tasks of n_k in $cls_s(i)$, $S(n_k, cls_s(i))$ is the time span from the start time of the task in $top(cls_s(i))$ to the start time of n_k when $t_w(n_k, p_p) = 0$, and every task that has no dependencies on n_k executes before n_k. Furthermore, $TL(cls_s(i))$ is the latest start time of the task in $top(cls_s(i))$, and $tlevel(n_k)$ is the latest start time of n_k without data waiting time if $n_k \notin top(cls_s(i))$; $tlevel(n_k)$ is derived by $S(n_k, cls_s(i))$ if $n_k \notin top(cls_s(i))$, otherwise, it is the latest DRT. Moreover, $blevel(n_k)$ is the longest path length from n_k to the END task, and $BL(cls_s(i))$ is the maximum execution path length including $S(n_k, cls_s(i))$ and $blevel(n_k)$. If $LV(cls_s(i))$ is defined as the sum of $TL(cls_s(i))$ and $BL(cls_s)$, then the WSL at $M(G^s)$, i.e., $WSL(M(G^s))$, is defined as the maximum value of $LV(cls_s(i))$, where $cls_s(i) \in V_{cls}^s$.

Example 3.1. Fig. 1 presents an example of the WSL derivation; in particular, Fig. 1a shows the DAG at $M(G^0)$, and Fig. 1b shows the DAG at $M(G^4)$, i.e., four tasks are

TABLE 1
Notation for $WSL(M(G^s))$ ($n_k \in cls_s(i)$)

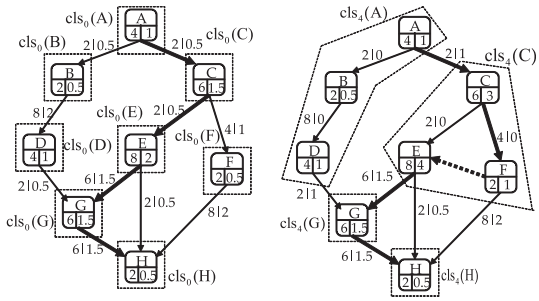
Parameter	Explanation	Definition
$dc(n_k, cls_s(i))$	The set of tasks in $cls_s(i)$ having paths from n_k.	$\{n_l \mid n_k \prec n_l, n_l \in cls_s(i)\} \cup \{n_k\}$.
$S(n_k, cls_s(i))$	The total processing time of tasks that can start execution before n_k in $cls_s(i)$.	$\sum_{n_l \in cls_s(i)} t_p(n_l, \alpha_p) - \sum_{n_l \in dc(n_k,i)} t_p(n_l, \alpha_p)$.
$tlevel(n_k)$	The start time of n_k when it is scheduled as late as possible.	$\begin{cases} \max_{n_l \in pred(n_k)}\{tlevel(n_l) + t_p(n_l, \alpha_p) + t_c(e_{l,k}, Lq, p)\}, \\ \quad if\ n_k \in top(cls_s(i)), \\ TL(cls_s(i)) + S(n_k, cls_s(i)),\ otherwise. \end{cases}$
$TL(cls_s(i))$	The latest start time of *top* tasks in $cls_s(i)$.	$\max_{n_k \in top(cls_s(i))}\{tlevel(n_k)\}$
$blevel(n_k)$	The maximum path length from n_k to the END task.	$\max_{n_l \in suc(n_k)}\{t_p(n_k, \alpha_p) + t_c(e_{k,l}, Lp, q) + blevel(n_l)\}$
$BL(cls_s(i))$	The maximum time duration from the time $cls_s(i)$ starts execution to the finish time of the END task.	$\max_{n_k \in out(cls_s(i))}\{S(n_k, cls_s(i)) + blevel(n_k)\}$
$level(n_k)$	The maximum time duration from a START task to the END task in case that n_k is scheduled as late as possible.	$tlevel(n_k) + blevel(n_k)$.
$LV(cls_s(i))$	The maximum *level* value in $cls_s(i)$.	$TL(cls_s(i)) + BL(cls_s(i)) = \max_{n_k \in cls_s(i)}\{level(n_k)\}$.
$WSL(M(G^s))$	*WSL* value at $M(G^s)$, i.e., the maximum LV value in all task clusters.	$\max_{cls_s(i) \in V^s_{cls}}\{LV(cls_s(i))\}$.

included in the task cluster. In both cases, the bold arrows correspond to the execution sequence dominating the WSL. Suppose there are three processors $(p_i, \alpha_i, \beta_i) = ((p_1, 4, 2), (p_2, 2, 4), (p_3, 2, 3))$. In Fig. 1a, each task belongs to a task cluster expressed by a dashed line box and assigned to a virtual processor having the maximum processing speed of 4 and maximum communication bandwidth of 4. The result for $cls_0(A)$ is as follows:

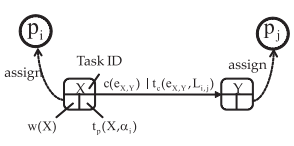
$$top(cls_0(A)) = out(cls_0(A)) = btm(cls_0(A)) = \{A\}.$$

From this state, $WSL(M(G^0)) = 9.5$ is decided by the path A, C, E, G, H, which is the same as the critical path length at $M(G^0)$. On the other hand, in Fig. 1b there are four task clusters. At $cls_4(A)$, suppose $cls_4(A)$, $cls_4(C)$, $cls_4(G)$, and $cls_4(H)$ are assigned to p_1, p_2, p_7^{vt}, and p_8^{vt}, respectively. Then we have the following results:

$$top(cls_4(A)) = \{A\},\ in(cls_4(A)) = \emptyset,$$
$$out(cls_4(A)) = \{A, D\},\ btm(cls_4(A)) = \{D\}.$$



(a)The state of $M(G^0)$. (b)The state of $M(G^4)$.



Meanings of assigned values.

Task Cluster	Processor (p_i, α_i, β_i)
$cls_4(A)$	$(p_1, 4, 2)$
$cls_4(C)$	$(p_2, 2, 4)$
$cls_4(G)$	$(p_7^{vt}, 4, 4)$
$cls_4(H)$	$(p_8^{vt}, 4, 4)$

Processor Assignment at (b).

Fig. 1. Example of the WSL derivation.

At $cls_4(C)$, we have

$$top(cls_4(C)) = \{C\},\ in(cls_4(C)) = \{C\},$$
$$out(cls_4(C)) = btm(cls_4(C)) = \{E, F\}.$$

The result expressed by

$$dc(C, cls_4(C)) = \{C, E, F\},\ dc(E, cls_4(C)) = \{E\},$$
$$dc(F, cls_4(C)) = \{F\},$$
$$TL(cls_4(c)) = tlevel(C) = 2,$$
$$S(E, cls_4(C)) = 8 - 4 = 4,$$
$$S(F, cls_4(C)) = 8 - 1 = 7,$$
$$blevel(E) = 9, blevel(F) = 3.5,$$
$$BL(cls_4(C)) = \max\{4 + 9, 7 + 3.5\} = 13.$$

Thus, $LV(cls_4(C)) = 2 + 13 = 15$. In the DAG in Fig. 1b, $cls_4(C)$ has two execution orders, i.e., C, E, F and C, F, E. $LV(cls_4(C))$ is taken when the execution order is the former case (the dashed arrow means that E starts executing after F is finished). In this case, we obtain $LV(cls_4(A)) = 14$, $LV(cls_4(G)) = LV(cls_4(H)) = 15$, and $WSL(M(G^4)) = 15$.

3.3 WSL Properties

The task clustering algorithm proposed in [14] minimizes the WSL instead of the schedule length because the schedule length cannot be decided until every task is scheduled by a task scheduling method. If the WSL and schedule length are related, a task cluster that minimizes the WSL also affects the schedule length. In [15], it is shown that minimizing the WSL also minimizes the upper and lower bounds of the schedule length (for proofs, we refer to [15]).

3.4 Lower Bound of the Cluster Processing Time

According to [14], $\Delta WSL_{up}(M(G^s))$ is a function of the lower bound, processing speed, and communication bandwidth as mentioned in Section 3.3. At $\Delta WSL(M(G^s))$, there are a number of task clusters exceeding the lower bound, i.e., $\delta(\alpha_p, \beta_p, G^s)$ is on a path belonging to the set of tasks dominating $WSL(M(G^{s-1}))$, where α_p and β_p are variables that must be determined. Thus, $\Delta WSL_{up}(M(G^s))$ assumes

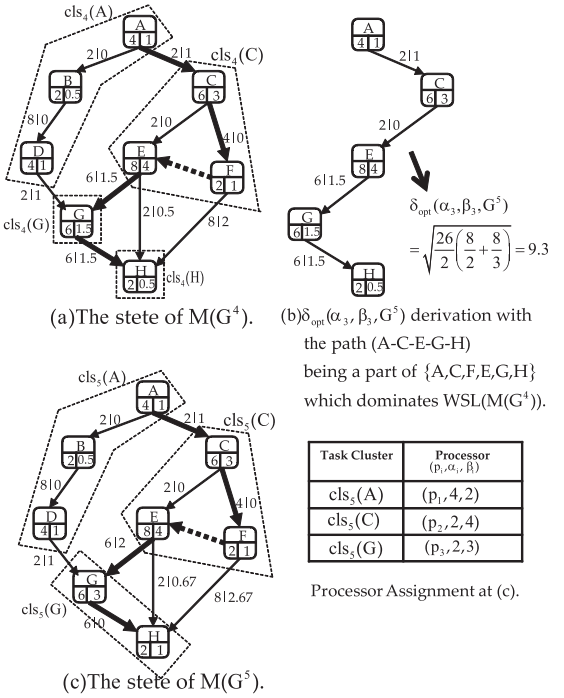


Fig. 2. Lower bound derivation at G^5.

the local minimum value when $\delta(\alpha_p, \beta_p, G^s)$ equals the following value:

$$\delta_{opt}(\alpha_p, \beta_p, G^s) = \sqrt{\frac{\sum_{n_k \in seq^{\prec}_{s-1}} w(n_k)}{\alpha_p}\left(\frac{\max_{n_k \in V}\{w(n_k)\}}{\alpha_p} + \frac{\max_{e_{k,l} \in E}\{c(e_{k,l})\}}{\beta_p}\right)}, \quad (8)$$

where $seq^{\prec}_{s-1}$ is a path where each task belongs to the set of tasks dominating $WSL(M(G^{s-1}))$. In (8), $\delta_{opt}(\alpha_p, \beta_p, G^s)$ is derived by tracing the path in the set of tasks and edges dominating $WSL(M(G^{s-1}))$, which means that the already clustered tasks are considered as unclustered tasks. Even if the total size of the unclustered tasks in a path is small and the data size among them is large, the lower bound is derived from the total task size in the path. As a result, the lower bound may be small, and consequently, a large amount of unlocalized data may exist. Therefore, the lower bound derivation in [14] does not consider the clustering state in a path.

Example 3.2. Fig. 2 shows an example of the lower bound derivation presented in [14]. In Fig. 2a, G and H are unclustered at $M(G^4)$. The path A, C, E, G, H is the path in which every task belongs to the set of tasks $\{E, C, F, E, G, H\}$ dominating $M(G^4)$. From this path, $\delta_{opt}(\alpha_p, \beta_p, G^5)$ is determined to be 9.3 by assuming the next assigned processor is p_3. Then, $cls_5(G)$ includes one of unclustered tasks, H. However, the total execution time at $cls_5(G)$ in Fig. 2c is $3 + 1 = 4 < 9.3$, and $cls_5(G)$ is clustered into either $cls_5(A)$ or $cls_5(C)$ in the next task clustering step.

4 CMWSL Algorithm

In this section, we present a cluster-based task scheduling method for obtaining a suitable schedule length without fully utilizing the given set of processors.

4.1 Algorithm Overview

CMWSL consists of three different procedures: (i) the processor assignment using the modified lower bound denoted by $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$, which is derived from the upper bound of the WSL difference at G^s and G^0 (denoted by $\Delta WSL^*_{up}(M(G^s))$), (ii) task clustering, and (iii) task scheduling. Algorithm 1 presents an overview of the CMWSL algorithm. UEX_{s-1} is the set of task clusters, each of which is assigned to a virtual processor; that is, at $M(G^0)$, every task cluster is assigned to a virtual processor implying $|UEX_0| = |V^0_{cls}| = |V|$. Lines 1 to 18 correspond to procedures (i) and (ii). These procedures are finished when $UEX_{s-1} = \emptyset$ or $\overline{P} = \emptyset$; otherwise, at line 2, a processor p_i, for which $\Delta WSL^*_{up}(M(G^s))$ becomes the minimum when $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$ is assigned, is selected for the sth task clustering step. Then, the task cluster $pivot_s$ is selected from the set of task clusters that is ready for task clustering, which is denoted by $FREE_s$.

Algorithm 1 Overall Procedures for CMWSL

Input: G^0 and P
Output: Schedule ς
Define: UEX_{s-1} as the set of task clusters assigned to the virtual processors
Define: $FREE_{s-1}$ as the set of task clusters that are candidates for $pivot_s$
Define: $\overline{P} \leftarrow P$, $s \leftarrow 1$ and $UEX_{s-1} \leftarrow V^0_{cls}$
1: **while** $UEX_{s-1} \neq \emptyset$ and $\overline{P} \neq \emptyset$ **do**
2: Find p_i s.t., $\Delta WSL^*_{up}(M(G^{s+r}))$ is the minimum by applying $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$ with α_i and β_i, where $p_i \in \overline{P}$ ▷ Detailed in Section 4.2
3: $pivot_s$ is selected by (15) ▷ $pivot_s$ is the clustering starting point at G^s (Detailed in Section 4.3)
4: Assign p_i to $pivot_s$, $\overline{P} \leftarrow \overline{P} - \{p_i\}$, and $UEX_{s-1} \leftarrow UEX_{s-1} - \{pivot_s\}$
5: **while** $T(pivot_s, p_i) < \delta^*_{opt}(\alpha_i, \beta_i, G^s)$ **or** Bottom task in $pivot_s$ belongs to seq_{s-1} with $pivot_s$ being linear **do**
6: $target_s \leftarrow getTarget(pivot_s)$ ▷ Detailed in Section 4.3
7: **if** $target_s \neq \emptyset$ **then**
8: $pivot_s \leftarrow clustering(pivot_s, target_s)$
9: Update $FREE_{s-1}$
10: $UEX_{s-1} \leftarrow UEX_{s-1} - \{target_s\}$
11: $s \leftarrow s + 1$
12: **else**
13: Update $FREE_{s-1}$
14: $s \leftarrow s + 1$
15: break
16: **end if**
17: **end while**
18: **end while**
19: $\varsigma \leftarrow schedule(M(G^s))$ ▷ Task scheduling

Definition 1. *Suppose $cls_s(j)$ has only one task, i.e., $cls_s(j) = \{n_j\}$. For $\forall n_{i'} \in pred(n_j)$ such that $n_{i'} \in cls_s(i) \neq cls_s(j)$, the condition that $cls_s(j)$ belongs to $FREE_s$ is given by*

$$FREE_s = \{cls_s(j) | \forall cls_s(i) \notin UEX_s\}. \quad (9)$$

Equation (9) implies that at every task cluster in $FREE_s$, every immediate predecessor task is assigned to an actual processor and belongs to a task cluster already applied in the task clustering steps, where $pivot_s$ is the source for the next task clustering step. Furthermore, p_i is assigned to $pivot_s$, and $pivot_s$ is removed from UEX_{s-1} at line 4. At lines 5 to 17, another task, $target_s$, is included in $pivot_s$ until $T(pivot_s, p_i) \geq \delta^*_{opt}(\alpha_i, \beta_i, G^s)$. At lines 9 and 13, task clusters satisfying (9) are added to $FREE_{s-1}$, and $target_s$ is removed from UEX_{s-1},. Then, s is incremented at line 11 or 14. After all task clustering steps are complete, each task is scheduled to minimize the schedule length at line 19, which will be discussed in greater detail in Section 4.4.

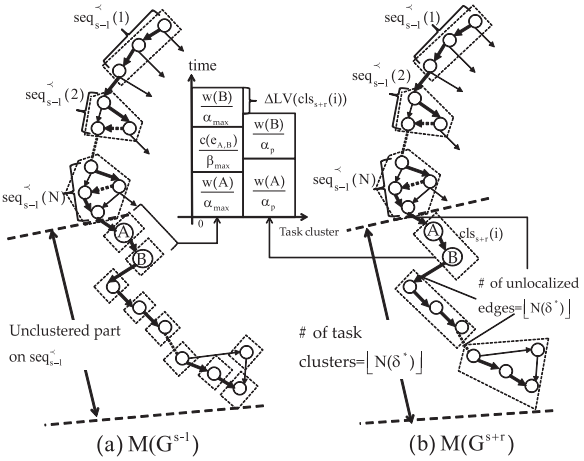


Fig. 3. Assumption for deriving $\Delta WSL^*_{up}(M(G^{s+r}))$.

4.2 Modified Lower Bound Definition

In this section, we present the definition of $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$, which is the lower bound of the sum of processing times of p_i at $M(G^s)$. In the derivation, we assume every unclustered task dominating $WSL(M(G^{s-1}))$ is clustered, and its size exceeds the lower bound at $M(G^{s+r})$. As presented in Section 3.4, $\delta_{opt}(\alpha_i, \beta_i, G^s)$ in [14] is derived from the path that is part of the set of tasks dominating $WSL(M(G^{s-1}))$. However, the difference between $WSL(M(G^{s+r}))$ and $WSL(M\ (G^{s-1}))$ depends on how unclustered tasks on the path at G^{s-1} are clustered and does not depend on already clustered tasks on the path at G^{s-1}.

The derivation of $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$ for minimizing $WSL(M(G^s))$ requires $\Delta WSL^*_{up}(M(G^{s+r}))$. On the path whose tasks belong to the set of tasks dominating $WSL\ (M(G^{s-1}))$, let the number of task clusters assigned to the actual processors be N. We define a path belonging to the execution sequence dominating $WSL(M(G^{s-1}))$ by $seq^{\prec}_{s-1}$. A subpath in $cls_{s-1}(i)$ that is part of $seq^{\prec}_{s-1}$ is defined by $seq^{\prec}_{s-1}(i)$. From these definitions, $\Delta WSL^*_{up}(M(G^{s+r}))$ is defined by imposing the following constraints:

- C1. Every task in $seq^{\prec}_{s-1}$ is clustered at the $(s+r)$th task clustering step, while $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$ is determined before the sth task clustering step; that is, the derivation of $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$ requires more than $(r+1)$ virtual task clustering steps.
- C2. When every unclustered task in $seq^{\prec}_{s-1}$ is clustered at $M(G^{s+r})$, the task clusters can be linear or non-linear, and the number of non-linear task clusters is y. A cluster is said to be linear if there is only one path between any two tasks in the cluster [18].
- C3. The sum of the processing times of tasks in $seq^{\prec}_{s-1}(i)$ exceeds $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$, where $cls_{s-1}(i) \in UEX_{s-1}$ and $cls_{s+r}(i) \notin UEX_{s+r}$.

Fig. 3 shows an assumption for the derivation of $\Delta WSL^*_{up}(M(G^{s+r}))$. In Fig. 3a, suppose that there are N task clusters that have been assigned to the processors and that each dashed arrow is the execution order among tasks for $WSL(M(G^{s-1}))$. The subpath at the bottom is the unclustered part of $seq^{\prec}_{s-1}$. In Fig. 3b, every task in the unclustered part of Fig. 3a is clustered, and each task cluster processing time exceeds $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$. If the number of such task clusters are defined by $\lfloor N(\delta^*) \rfloor$, $N(\delta^*)$ is defined as follows:

$$N(\delta^*) = \frac{1}{\alpha_p \delta^*} \left(\sum_{n_k \in seq^{\prec}_{s-1}} w(n_k) - \sum_{i=1}^{N} \sum_{\substack{n_k \in seq^{\prec}_{s-1}(i), \\ cls_{s-1}(i) \notin UEX_{s-1}}} w(n_k) \right), \tag{10}$$

where $\delta^* = \delta^*(\alpha_p, \beta_p, G^s)$ and $\alpha_p \delta^*$ is the task cluster size of p_p. For each task cluster $cls_{s+r}(i)$ on the bottom of Fig. 3b, we define $\Delta LV(cls_{s+r}(i))$ by

$$\Delta LV(cls_{s+r}(i)) = \sum_{n_k \in cls_{s+r}(i)} \frac{w(n_k)}{\alpha_p} - \left(\sum_{n_k \in seq^{\prec}_{s+r}(i)} \frac{w(n_k)}{\alpha_{max}} + \sum_{e_{k,l} \in seq^{\prec}_{s+r}(i)} \frac{c(e_{k,l})}{\beta_{max}} \right), \tag{11}$$

where

$$\alpha_{max} = \max_{p_p \in \overline{P}} \{\alpha_p\}, \beta_{max} = \max_{p_p \in \overline{P}} \{\beta_p\},$$

and $\overline{P}$ is the set of unassigned processors. In Fig. 3b, $cls_{s+r}(i)$ does not have the communication time between A and B, while the communication time between A and B exists in Fig. 3a; that is, $\Delta LV(cls_{s+r}(i))$ is the difference in the sum of processing times and communication times among tasks in $cls_{s+r}(i)$. We define $\Delta WSL^*(M(G^{s+r}))$ by $\Delta LV(cls_{s+r}(i))$ multiplied by $\lfloor N(\delta^*) \rfloor$. By considering constraints C1 to C3, note that $\Delta WSL^*(M(G^{s+r}))$ is bounded by the upper bound of $\Delta LV(cls_{s+r}(i))$ and also depends on $\Delta LV^{up}_{lnr}(cls_{s+r}(i))$ (if $cls_{s+r}(i)$ is linear) and $\Delta LV^{up}_{nlnr}(cls_{s+r}(i))$ (if $cls_{s+r}(i)$ is non-linear); in particular,

$$\begin{aligned} &\Delta WSL^*(M(G^{s+r})) \\ &\le \sum_{i=1}^{N(\delta^*)-y} \Delta LV^{up}_{lnr}(cls_{s+r}(i)) \\ &\quad + \sum_{i=N(\delta^*)-y+1}^{N(\delta^*)} \Delta LV^{up}_{nlnr}(cls_{s+r}(i)) \\ &\quad + \sum_{\substack{e_{k,l} \in seq^{\prec}_{s-1}, \\ e_{k,l} \notin seq^{\prec}_{s+r}(i)}} \left(\frac{c(e_{k,l})}{L_{p,q}} - \frac{c(e_{k,l})}{\beta_{max}} \right) \le \Delta WSL^*_{up}(M(G^{s+r})), \end{aligned} \tag{12}$$

$$\begin{aligned} &\Delta WSL^*_{up}(M(G^{s+r})) \\ &= y\delta^* + \frac{y w_{max}}{\alpha_p} + \frac{1}{\alpha_p \delta^*} \left\{ \varepsilon\lambda + \frac{w_{min}}{\alpha_p} \sum_{i=1}^{N} \sum_{n_k \in seq^{\prec}_{s-1}(i)} w(n_k) \right\} \\ &\quad - \frac{1}{w_{max}} \left(\frac{w_{min}}{\alpha_{max}} + \frac{c_{min}}{\beta_{max}} \right) - \frac{w_{min}}{w_{max} \alpha_p} \sum_{i=1}^{N} \sum_{n_k \in seq^{\prec}_{s-1}(i)} w(n_k) \\ &\quad + cp_w(\alpha_p), \end{aligned} \tag{13}$$

where $cp_w(\alpha_p)$ is the maximum of the total task size divided by α_p in a path and

$$w_{min} = \min_{n_k \in seq^{\prec}_{s-1}} \{w(n_k)\}, w_{max} = \max_{n_k \in seq^{\prec}_{s-1}} \{w(n_k)\},$$

$$c_{min} = \min_{e_{k,l} \in seq^{\prec}_{s-1}} \{c(e_{k,l})\}, c_{max} = \max_{e_{k,l} \in seq^{\prec}_{s-1}} \{c(e_{k,l})\},$$

$$\varepsilon = \sum_{n_k \in seq^{\prec}_{s-1}} w(n_k) - \sum_{i=1}^{N} \sum_{n_k \in seq^{\prec}_{s-1}(i)} w(n_k),$$

$$\lambda = \left(\frac{c_{max}}{\beta_p} - \frac{c_{max}}{\beta_{max}} + \frac{w_{min}}{\alpha_{max}} + \frac{c_{min}}{\beta_{max}} \right).$$

Note that the third term in (12) is the sum of the differences of communication times between each task cluster at $M(G^{s+r})$. In this case, $\Delta WSL^{*}_{up}(G^{s+r})$ can be defined by developing (12). For a detailed definition of $\Delta LV^{up}_{lnr}(cls_{s+r}(i))$, $\Delta LV^{up}_{nlnr}(cls_{s+r}(i))$, and $\Delta WSL^{*}_{up}(G^{s+r})$, see Appendix B in the supplemental file, available online.

By differentiating $\Delta WSL^{*}_{up}(G^{s+r})$ with respect to δ^{*}, we see that $\Delta WSL^{*}_{up}(G^{s+r})$ has a local minimum when δ^{*} assumes the following value (for more details, see Appendix B in the supplemental file, available online):

$$\delta^{*}_{opt}(\alpha_p, \beta_p, G^{s}) = \sqrt{\frac{1}{\alpha_p} \left\{ \varepsilon\lambda + \frac{w_{min}}{\alpha_p} \sum_{i=1}^{N} \sum_{n_k \in seq^{\prec}_{s-1}(i)} w(n_k) \right\}}. \tag{14}$$

4.3 Task Clustering Phase

In this section, we present the task clustering algorithm for lines 3 to 17 in Algorithm 1.

Algorithm 2 Procedures for $getTarget(pivot_s)$ at Line 6 in Algorithm 1 (As for $tlevel$, $blevel$, and LV, see their definitions in Table 1).

Input: $pivot_s$ and its assigned processor p_i
Output: A task cluster corresponding to $target_s$
Define: $n_{btm} \in btm(pivot_s), n_{top} \in top(pivot_s)$
Define: $S_{=}\{n_p | n_p \in cls_{s-1}(k), n_p \in suc(n_{btm})\}$
Define: $S_{cls} = \{cls_{s-1}(k) | n_p \in S, n_p \in cls_{s-1}(k)\}$
Define: $S_{suc} = \{n_p | n_p \in top(cls_{s-1}(k)), n_p \in suc(n_{btm})\}$
Define: $S^{lnr}_{suc} = \{n_p | n_p \in S_{suc}, n_p \in cls_{s-1}(k), cls_{s-1}(k)\ is\ linear\}$
Define: $S_{pred} = \{n_p | n_p \in btm(cls_{s-1}(k)), n_p \in pred(n_{top})\}$
Define: $S^{lnr}_{pred} = \{n_p | n_p \in S_{pred}, n_p \in cls_{s-1}(k), cls_{s-1}(k)\ is\ linear\}$
1: **if** $S^{lnr}_{suc} \neq \emptyset$ **then** ▷ maintain linearity if $pivot_s$ is linear
2: $target_s \leftarrow cls_{s-1}(l)$ s.t., $t_c(e_{btm,q}, L_{i,j}) + blevel(n_q)$

$$= \max_{n_{btm}} \{ \max_{n_p \in S^{lnr}_{suc}} \{t_c(e_{btm,p}, L_{i,j}) + blevel(n_p)\}\},$$

$$where\ n_p \in S^{lnr}_{suc} is\ assigned\ to\ p_j \in P \cup P^{vt},\ n_q \in S^{lnr}_{suc}$$

3: **else if** $S^{lnr}_{pred} \neq \emptyset$ **then** ▷ maintain linearity if $pivot_s$ is linear
4: $target_s \leftarrow cls_{s-1}(l)$ s.t., $tlevel(n_q) + t_p(n_q, \alpha_j) + t_c(e_{q,top}, L_{j,i})$

$$= \max_{n_{top}} \{ \max_{n_p \in S^{lnr}_{pred}} \{tlevel(n_p) + t_p(n_p, \alpha_j) + t_c(e_{p,top}, L_{j,i})\}\},$$

$$where\ n_p \in S^{lnr}_{pred}\ is\ assigned\ to\ p_j \in P \cup P^{vt},\ n_q \in S^{lnr}_{pred}$$

5: **else** ▷ minimize WSL with non-linearity
6: $pivot^{tmp}_s(k) \leftarrow pivot_s \cup cls_{s-1}(k)$, for each $cls_{s-1}(k) \in S_{cls}$
7: Decide $cls_{s-1}(m) \in S_{cls}$ s.t.,

$$LV(pivot^{tmp}_s(m)) = \min_{cls_{s-1}(k) \in S_{cls}} \{LV(pivot^{tmp}_s(k))\}$$

8: **if** $LV(pivot^{tmp}_s(m)) \leq LV(pivot_s)$ **then**
9: $target_s \leftarrow cls_{s-1}(m)$
10: **else**
11: $target_s \leftarrow \emptyset$
12: **end if**
13: **end if**
14: **return** $target_s$

4.3.1 Determining $pivot_s$ and $target_s$

Before the sth task clustering step, $pivot_s$ is selected as the starting point. The objective of the task clustering is to minimize the WSL in order to minimize both the upper bound and lower bound of the schedule length. At line 3 in Algorithm 1 the task cluster satisfying the following condition is selected for $pivot_s$: $pivot_s = cls_{s-1}(i)\ s.t.,$

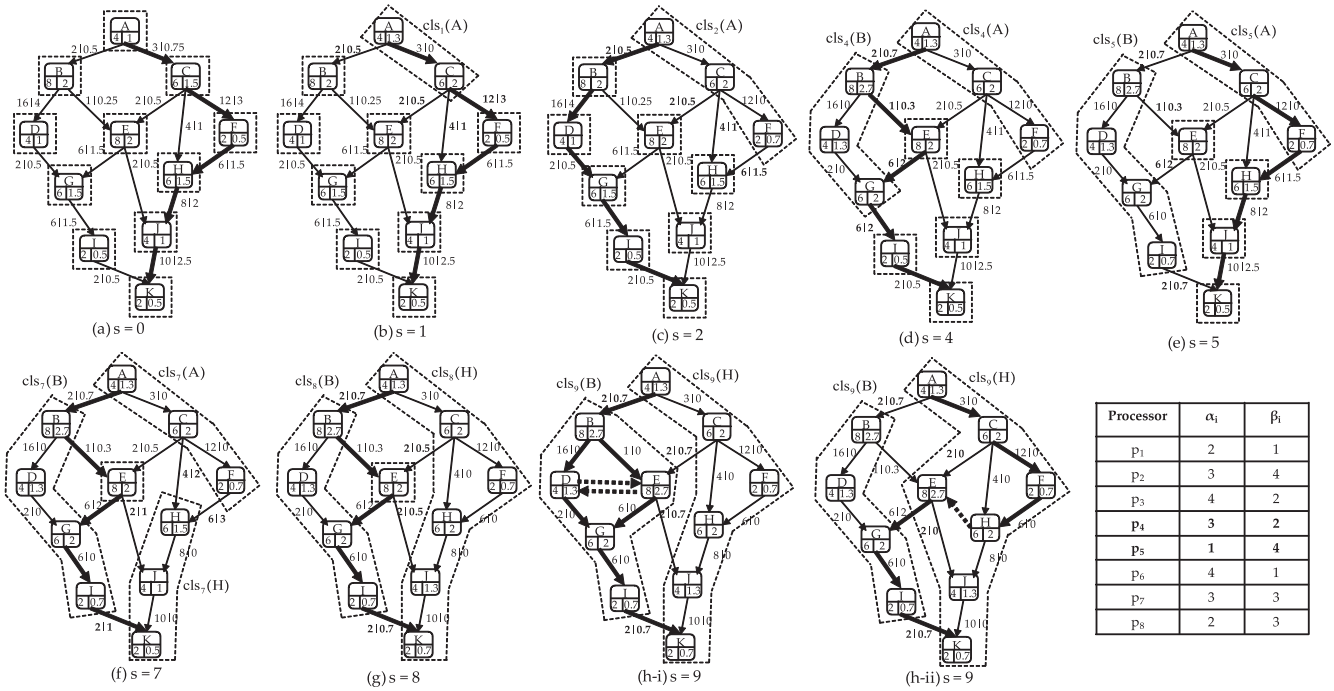
$$LV(cls_{s-1}(i)) = \max_{cls_{s-1}(i) \in FREE_{s-1}} \{LV(cls_{s-1}(i)\}. \tag{15}$$

Algorithm 2 shows the detailed procedures for $getTarget(pivot_s)$ at line 6 in Algorithm 1. For each task n_k in a linear task cluster $cls_{s-1}(i)$, $S(n_k, cls_{s-1}(i))$ depends only on the total processing time of its ancestor tasks and n_k; it does not depend on tasks that are not dependent on n_k. Therefore, $getTarget(pivot_s)$ tries to maintain the linearity of $pivot_s$ to minimize the WSL. Lines 1 to 4 in Algorithm 2 correspond to maintaining the linearity of the task cluster generated by $pivot_s$ and $target_s$. At line 2, $target_s$ is selected from the set of linear task clusters $cls_{s-1}(k)$, including successor tasks of the bottom task n_{btm} in $pivot_s$. Such successor tasks must belong to $top(cls_{s-1}(k))$ in order to maintain the linearity of the generated task cluster. The tasks belonging to S^{lnr}_{suc} defined in Algorithm 2 are also considered. From the set of tasks in S^{lnr}_{suc}, the one having the maximum impact on $blevel(n_{btm})$ is selected, and its task cluster corresponding to $target_s$ makes $LV(pivot_s)$ smaller after the task clustering step. There is only one task in $btm(pivot_s)$ if $pivot_s$ is linear, and $|btm(pivot_s)| > 1$ if $pivot_s$ is non-linear. If $pivot_s$ is non-linear, $getTarget(pivot_s)$ traces all tasks in $btm(pivot_s)$ to determine $target_s$. If there are no tasks in S^{lnr}_{suc}, at line 4, $target_s$ is selected from the set of task clusters $cls_{s-1}(k)$, including predecessor tasks of the top task n_{top} in $pivot_s$. Similar to line 2, such predecessor tasks must belong to $btm(cls_{s-1}(k))$ to maintain the linearity of the generated task cluster. The tasks belonging to S^{lnr}_{pred} defined in Algorithm 2 are also considered. The task cluster having the maximum impact on $tlevel(n_{top})$ is selected as $target_s$. If the number of tasks in $top(pivot_s)$ is two or more, $getTarget(pivot_s)$ tries to find $target_s$ by tracing all tasks in $top(pivot_s)$.

If both S^{lnr}_{suc} and S^{lnr}_{pred} are empty, the process goes to lines 5 to 13. For each task cluster $cls_{s-1}(k) \in S_{suc}$, the algorithm checks whether $BL(pivot_s)$ is made larger by creating a temporary task cluster denoted as $pivot^{tmp}_s(k)$. If $BL(pivot_s)$ does not increase, the task cluster $cls_{s-1}(k)$, by which the minimum of $BL(pivot_s)$ is obtained, is selected as $target_s$; otherwise, $target_s$ is not selected at line 11.

4.3.2 Clustering for $pivot_s$ and $target_s$

After both $pivot_s$ and $target_s$ have been determined, they are merged and several pieces of information are updated. This process corresponds to $clustering(pivot_s, target_s)$ at line 8 in Algorithm 1. Algorithm 3 shows the detailed procedures for $clustering(pivot_s, target_s)$. In this figure, S_{in} and S_{out} are the sets of tasks having immediate predecessor tasks and immediate successor tasks in the newly generated $pivot_s$, respectively. Since the assigned processor of $target_s$ is not always a virtual processor, whether the processor is virtual or not is verified. If it is an actual processor, at line 1, the processor is



Processor	α_i	β_i
p_1	2	1
p_2	3	4
p_3	4	2
p_4	**3**	**2**
p_5	**1**	**4**
p_6	4	1
p_7	3	3
p_8	2	3

Fig. 4. Example of task clustering in CMWSL.

restored to P, where $proc(target_s)$ is the assigned processor of $target_s$. Then, at line 4, $pivot_s$ and $target_s$ are merged and CMWSL updates $TL(pivot_s)$ and $BL(pivot_s)$ at lines 5 to 10. Though it is ideal for both $tlevel(n_k)$ and $blevel(n_k)$ ($\forall n_k \in V$) to be updated at each task clustering step, such an update procedure requires high time complexity, i.e., $O(|V|+|E|)$ for each task clustering step. In CMWSL, the update range is limited in order to suppress the time complexity. As for updating $tlevel(n_k)$, at line 7, n_k is traced from $top(pivot_s)$ to the bottom tasks in $pivot_s$. Then, every task in S_{out} is traced and its $tlevel$ value is updated. At line 9, the value of $blevel$ for the tasks in $pivot_s$ is updated by tracing from the bottom tasks to the top tasks. Then, every task in S_{in} is traced to update the $blevel$ values. Both $TL(pivot_s)$ and $BL(pivot_s)$ are updated at lines 8 and 10, respectively. At lines 11 to 16, the TL values and BL values of the task clusters whose tasks belong to S_{in} or S_{out} are updated to reflect the new priority value for each task cluster in $FREE_{s-1}$.

TABLE 2
Step Information for Fig. 4

step	$FREE_{s-1}$:LV	$pivot_s$	$target_s$	p_i	$\delta^*_{opt}(\alpha_i, \beta_i, G^s)$
a	$cls_0(A)$:15.75	$cls_0(A)$	$cls_0(C)$	p_2	$\sqrt{10}$
b	-	$cls_1(A)$	$cls_1(F)$	p_2	$\sqrt{10}$
c	$cls_2(B)$:13.8, $cls_2(H)$:13	$cls_2(B)$	$cls_2(D)$	p_7	$\frac{\sqrt{178}}{3}$
-	$cls_3(H)$:13	$cls_3(B)$	$cls_3(G)$	p_7	$\frac{\sqrt{178}}{3}$
d	$cls_4(H)$:13	$cls_4(B)$	$cls_4(I)$	p_7	$\frac{\sqrt{178}}{3}$
e	$cls_5(E)$:12.9, $cls_5(H)$:13	$cls_5(H)$	$cls_5(J)$	p_3	$\frac{5\sqrt{3}}{2}$
-	$cls_6(E)$:14.5	$cls_6(H)$	$cls_6(K)$	p_3	$\frac{5\sqrt{3}}{2}$
f	$cls_7(E)$:14.5	$cls_7(H)$	$cls_7(A)$	p_2	$\sqrt{10}$
g	$cls_8(E)$:13.1	$cls_8(E)$	$cls_8(B)$	p_7	$\frac{\sqrt{178}}{3}$

Algorithm 3 Procedures for $clustering(pivot_s, target_s)$ at Line 8 in Algorithm 1 (As for top, in, out, and btm, see their definitions in Section 3.2. As for dc, S, $tlevel$, TL, $blevel$ and BL, see Table 1).

Input: $pivot_s$ and $target_s$
Output: Newly generated $pivot_s$
Define: $n_{in} \in in(pivot_s), n_{out} \in out(pivot_s)$
Define: $S_{in} = \{n_p | n_p \in pred(n_{in}), n_p \in cls_{s-1}(k) \notin pivot_s \cup target_s\}$
Define: $S_{out} = \{n_p | n_p \in suc(n_{out}), n_p \in cls_{s-1}(k) \notin pivot_s \cup target_s\}$
1: **if** $proc(target_s) \in P$ **then** ▷ put the processor of $target_s$ back to $\overline{P}$
2: $\overline{P} \leftarrow \overline{P} \cup \{proc(target_s)\}$
3: **end if**
4: $pivot_s \leftarrow pivot_s \cup target_s$ ▷ put all tasks in $pivot_s$
5: Update $top(pivot_s)$, $in(pivot_s)$, $out(pivot_s)$, $btm(pivot_s)$
6: Update $dc(n_k, pivot_s)$ and calculate $S(n_k, pivot_s)$ for each $n_k \in pivot_s$
7: Update $tlevel(n_k)$ for $n_k \in pivot_s$. Then update $tlevel(n_k)$ for $n_k \in S_{out}$
8: Derive $TL(pivot_s)$ from $top(pivot_s)$;
9: Update $blevel(n_k)$ for $n_k \in pivot_s$. Then update $blevel(n_k)$ for $n_k \in S_{in}$
10: Derive $BL(pivot_s)$ from $out(pivot_s)$
11: **for** $n_p \in S_{in}$ **do**
12: Update $BL(cls_{s-1}(k))$, where $n_p \in cls_{s-1}(k)$
13: **end for**
14: **for** $n_p \in S_{out}$ **do**
15: Update $TL(cls_{s-1}(k))$, where $n_p \in cls_{s-1}(k)$
16: **end for**
17: **return** $pivot_s$

Example 4.1. Fig. 4 shows an example of task clustering in CMWSL, and Table 2 shows the information for each step. In particular, Figs. 4a, 4b, 4c, 4d, 4e, 4f, 4g, 4h-i, and 4h-ii corresponds to each task clustering state, where the step number is denoted by s. The table on the right in Fig. 4 corresponds to the given processor information. The bold arrows in Figs. 4a, 4b, 4c, 4d, 4e, 4f, 4g, 4h-i, and 4h-ii are the set of tasks and edges dominating the WSL. In Fig. 4a, every task cluster is assigned to a virtual processor having the maximum processing speed of 4 and the maximum communication bandwidth of 4. $FREE_0$ contains only $cls_0(A)$, so it is selected as $pivot_1$. From the path, i.e., $seq_0^{\prec}$ in Fig. 4a, we have $w_{max} = 6$, $w_{min} = 2$, $c_{max} = 12$, and $c_{min} = 3$, and the total task size on the

path is 24. At line 2 in Algorithm 1, CMWSL derives the minimum of $\Delta WSL^*_{up}(M(G^{1+r}))$ with $\delta^*_{opt}(\alpha_i, \beta_i, G^1)$ by applying the information from p_1 to p_8. The minimum of $\Delta WSL^*_{up}(M(G^{1+r}))$ is obtained when p_2 is applied; in this case, we have $\delta^*_{opt}(\alpha_2, \beta_2, G^1) = \delta^*_{opt}(3, 4, G^1) = \sqrt{10}$. Next, $cls_0(C)$ is selected as $target_1$ according to line 2 in Algorithm 2. In Fig. 4b, $pivot_1$ and $target_1$ are clustered and $cls_1(A)$ is generated. Then, the information of $cls_1(A)$ is updated according to line 5 in Algorithm 3, i.e.,

$$top(cls_1(A)) = \{A\}, in(cls_1(A)) = \{A\},$$
$$out(cls_1(A)) = \{A, C\}, btm(cls_1(A)) = \{C\},$$

and from line 7 in Algorithm 3, we have

$$tlevel(A) = 0, tlevel(C) = 1.3,$$
$$tlevel(B) = 1.8, tlevel(E) = 4.05, tlevel(H) = 4.3,$$
$$tlevel(F) = 6.3 \ for \ S_{out} = \{B, E, H, F\}.$$

According to line 8 in Algorithm 3, $TL(cls_1(A)) = tlevel(A) = 0$. Since $S_{in} = \emptyset$, only $blevel(A)$ and $blevel(C)$ are updated according to line 9 in Algorithm 3. Next, $TL(B)$, $TL(E)$, and $TL(H)$ are updated according to lines 14 to 16 in Algorithm 3, and $BL(cls_1(A)) = blevel(A) = 15.8$. From lines 11 to 16 in Algorithm 3, we obtain values for $LV(cls_1(B))$, $LV(cls_1(E))$, $LV(cls_1(H))$, and $LV(cls_1(F))$. Since $T(cls_1(A)), p_2) = 3.3 > \sqrt{10}$, $btm(cls_1(A)) = \{C\}$ is included in $seq_1 = A \rightarrow C \rightarrow F \rightarrow H \rightarrow I \rightarrow K$ and $cls_1(A)$ is still linear. This corresponds to the second condition at line 5 in Algorithm 1. Thus, $cls_1(A)$ becomes $pivot_2$, $cls_1(F)$ is selected as $target_2$, and they are clustered in Fig. 4c. Since $seq_1 \neq seq_2$ in Fig. 4c, $cls_2(B)$ is selected as $pivot_3$. In Fig. 4d, $cls_2(B)$ is clustered with $cls_2(D)$ and p_7 is assigned to $cls_2(B)$. Since $\delta^*_{opt}(\alpha_7, \beta_7, G^3)$ is $\frac{\sqrt{178}}{3}$ and $T(cls_4(B), p_7) > \frac{\sqrt{178}}{3}$ in Fig. 4d, we have $btm(cls_4(B)) = \{G\} \in seq_4$. In Fig. 4e, $cls_4(B)$ is clustered with $cls_4(I)$, and in Fig. 4f, $T(cls_7(H), p_3) = 3 < \delta^*_{opt}(\alpha_3, \beta_3, G^7) = \frac{5\sqrt{3}}{2}$, so more task clusters must be included in $cls_7(H)$. However, $S^{lnr}_{suc} = \emptyset$, and the process goes to line 3 in Algorithm 2. It follows that $target_8 = \{cls_7(A)\}$ and it is clustered with $cls_7(H)$ in Fig. 4g. Furthermore, $cls_8(H)$ is assigned to the processor of $cls_7(A)$, i.e., p_2, and p_3 is returned back to $\overline{P}$. From this state, only $cls_8(E)$ belongs to $FREE_8$, and both S^{lnr}_{suc} and S^{lnr}_{pred} are empty. Thus in Fig. 4g the process goes to line 5 in Algorithm 2. Since $S_{cls} = \{cls_8(B), cls_8(H)\}$, CMWSL determines $target_9$ from these two task clusters. Fig. 4(h-i) is the case when $cls_8(E)$ and $cls_8(B)$ are clustered into $cls_9(E)$. In this case, seq_9 is $A \rightarrow B \rightarrow E \rightarrow D \rightarrow G \rightarrow I \rightarrow K$ or $A \rightarrow B \rightarrow D \rightarrow E \rightarrow G \rightarrow I \rightarrow K$ and its WSL is 12.8, while the WSL in Fig. 4(h-ii) is 14.8. Since $12.8 < 13.1 = WSL(M(G^8))$, it follows that $target_9 = \{cls_8(B)\}$, and Fig. 4(h-i) is the result of task clustering.

4.4 Scheduling Phase

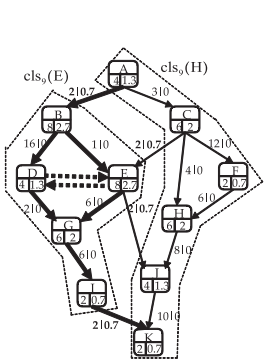
In this section, we present a task scheduling method that is performed after the task clustering described in Section 4.3. If the task cluster processing time exceeds the lower bound, i.e., $\delta^*_{opt}(\alpha_i, \beta_i, G^s)$, every task in the task cluster is assigned to an idle time slot on the processor, i.e., p_i. Otherwise, every task in the processor is assigned to an idle time slot of another processor that is already assigned to another cluster. This occurs when the process goes to line 11 in Algorithm 2, i.e., $target_s$ cannot be found in a task clustering step, so the process goes to the next step at line 15 in Algorithm 1. As a result, the task cluster processing time exceeds the lower bound after the scheduling phase is finished.

Next, we present how the scheduling priority is derived for each task. When the WSL is calculated, the data waiting time, i.e., $t_w(n_i, p_p)$, is taken into account for tasks only in $top(cls_s(k))$, where $n_i \in cls_s(k)$ (a *top* task is defined at Section 3.2). This means that the data waiting time should be zero; however, the data waiting time is not always zero in the actual execution. Thus, the objective of scheduling is to schedule a task n_i that can start executing first in the free task list (the set of tasks whose immediate predecessor tasks have been scheduled) in order to reduce the data waiting time of tasks in $suc(n_i)$.

Algorithm 4 Procedures for $schedule(M(G^s))$ at Line 19 in Algorithm 1.

Input: Clustered DAG $M(G^s)$
Output: Schedule of $M(G^s)$
Define: the task cluster to which n_i belongs by $C(n_i)$
Define: the processor to which n_i is assigned by $proc(n_i)$
Define: $USCHED$ to be the set of unscheduled tasks
Define: $FREE_{sched}$ to be the set of tasks whose immediate predecessor tasks have been scheduled
1: **while** $USCHED \neq \emptyset$ **do**
2: Find n_i having the minimum $t_{dr}(n_i, proc(n_i))$ in $FREE_{sched}$. If two or more tasks have the same minimum value, the task n_i having the maximum $blevel$ value is selected
3: $FREE_{sched} \leftarrow FREE_{sched} - \{n_i\}$
4: $USCHED \leftarrow USCHED - \{n_i\}$;
5: **if** $T(C(n_i), proc(n_i)) < \delta^*_{opt}$ **then**
6: Insert n_i into an idle time slot of another processor p_p in the set of assigned processors $(P - \overline{P} - \{proc(n_i)\})$ s.t, $t_f(n_i, p_p)$ is minimized by an insertion-based policy. Then update $t_s(n_i, p_p)$ and $proc(n_i) \leftarrow p_p$
7: **else**
8: Insert n_i into an idle time slot of $proc(n_i)$ s.t, $t_f(n_i, proc(n_i))$ is minimized by an insertion-based policy. Update $t_s(n_i, proc(n_i))$
9: **end if**
10: Set $t_f(n_i, proc(n_i))$
11: **for** $n_j \in suc(n_i)$ **do**
12: **if** $n_k \notin USCHED$ for $\forall n_k \in pred(n_j)$ **then**
13: Update $t_{dr}(n_j, proc(n_j))$
14: $FREE_{sched} \leftarrow FREE_{sched} \cup \{n_j\}$
15: **end if**
16: **end for**
17: **end while**

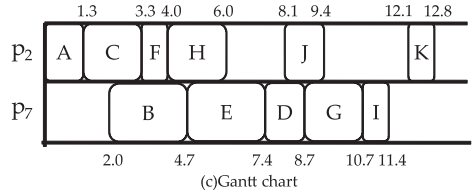
Algorithm 4 presents the procedures for scheduling in CMWSL. First, $FREE_{sched}$ includes all of the START tasks and $USCHED$ includes all of the tasks. This procedure finishes when $USCHED$ becomes empty. At line 2, the task to be scheduled is selected by the DRT, which is a lower bound for the start time for each task. If all of the immediate predecessor tasks of n_i have been scheduled, $t_{dr}(n_i, proc(n_i))$ can be derived. In the scheduling phase, the task having the minimum DRT in $FREE_{sched}$ is scheduled by inserting it into an idle time of the processor. If the task cluster processing time of the selected task is smaller than δ^*_{opt}, at line 6, it is assigned to an idle time of another processor in $P - \overline{P} - \{proc(n_i)\}$; otherwise, the task is assigned to an idle time of $proc(n_i)$ at line 8. As a result, every task cluster processing time of the processor exceeds the lower bound. After the task is scheduled, for each task in $suc(n_i)$, its DRT is updated if all of its predecessor tasks have been scheduled.



(a) Before scheduling tasks

step	$FREE_{sched}$: $t_{dr}(n_i, proc(n_i))$	task	$t_s(n_i, proc(n_i))$
1	A:0	A	0
2	B:2.0, C:1.3	C	1.3
3	B:2.0, F: 3.3	B	2.0
4	D:4.7, E:4.7, F: 3.3	F	3.3
5	D:4.7, E:4.7, H: 4.0	H	4.0
6	D:4.7, E:4.7	E	4.7
7	D:4.7, J:8.1	D	7.4
8	G:7.4, J:8.1	G	8.7
9	I:10.7, J:8.1	J	8.1
10	I:10.7	I	10.7
11	K:12.1	K	12.1

(b) Scheduling



(c)Gantt chart

Fig. 5. Example of the scheduling phase.

Example 4.2. Fig. 5 shows an example of scheduling tasks in Fig. 4(h-i). Figs. 5a, 5b, and 5c present the DAG before scheduling each task, the scheduling result, and the gantt chart, respectively. In Fig. 5a, the DAG has two task clusters, and their cluster processing time is the lower bound. Thus, the process proceeds to the case at lines 7 to 9. Since there is only one START task in Fig. 5a, A is included in $FREE_{sched}$, and it is selected for scheduling. Then, B and C become free, and their DRTs are 2.0 and 1.3, respectively; $t_f(A, p_2) = 1.3$. At step 6 in Fig. 5b, D and E are assigned to p_7, but their DRTs are the same. In this case, their $blevel$ values are compared at line 2 in Algorithm 4. Since $blevel(D) = 5.4 < blevel(E) = 6.8$, E is selected. At step 7 in Fig. 5b, D is selected, and its start time is calculated. There is no idle time between B and E according to Fig. 5c, and D is added after E. As a result, the start time of D is 7.4, while the DRT of D is 4.7. Similarly, at step 8 in Fig. 5b, G is selected, but there is no idle time in B-E or E-D. Thus, G is added after D. At steps 9, 10, and 11 in Fig. 5b, I , J, and K are added, and the schedule length is 12.8.

4.5 Complexity of CMWSL

In CMWSL, the dominating part of the time complexity is at lines 5 to 13 in Algorithm 2; task scheduling also dominates the time complexity. Since $|S_{cls}| \leq |suc(n_{btm})|$, lines 5 to 13 in Algorithm 2 require $|suc(n_{btm})|(|V| + |E|)\log|suc(n_{btm})|)$ or $|suc(n_{btm})|(|V| + |E|)\log|pred(n_{top})|)$ steps, which costs $O(|E|(|V| + |E|)\log|V|)$. The complexity of task scheduling is $O(|P||V|^2)$. Thus, the complexity of CMWSL is $\max\{O(|E|(|V| + |E|)\log|V|), O(|P||V|^2)\}$. For a more detailed complexity analysis, see Appendix C in the supplemental file, available online.

5 Experimental Results

In this section, we present our experimental results. The objective of CMWSL is to minimize the schedule length without fully utilizing the given processors; an appropriate number of processors can achieve a better schedule length than if all the given processors are fully utilized. In the experiment, we compared CMWSL with other methods in terms of the following criteria:

1) If the modified lower bound of each task cluster processing time has a positive effect on the schedule length.
2) If the criteria of the scheduling policy are adequate.
3) If the schedule length in real-world DAGs can be made smaller.

Our simulation environment was developed using jdk1.7.0.51, and was executed by an Intel(R) Core i7-5600U 2.6 GHz with an 8 GB memory size. As for the additional experimental results, see Appendix D in the supplemental file, available online.

5.1 Conditions and Metrics

The Schedule Length Ratio (SLR) [20], [21] metric was used to measure the performance of each task scheduling algorithm. The SLR is defined as follows:

$$SLR = \frac{SL}{\sum_{n_k \in CP} t_p(n_k, \max_{p_i \in P}\{\alpha_i\})}, \tag{16}$$

where CP is the critical path.

The efficiency metric was used to measure the degree of contribution of each processor to the reduction of the schedule length. The efficiency is defined as follows:

$$Efficiency = \frac{\sum_{n_k \in V} w(n_k)/\max_{p_i \in P}\{\alpha_i\}}{|P_{assigned}|SL_{algorithm}}, \tag{17}$$

where $P_{assigned}$ is the set of assigned processors and $SL_{algorithm}$ is the schedule length of the applied algorithm.

In Sections 5.2, 5.3, and 5.4, comparisons are made using random DAGs. For each random DAG, the Communication to Computation Ratio (CCR) [17], defined as the average data size divided by the average task size, was chosen from $\{0.1, 0.5, 1, 3, 5, 10\}$. The maximum number of tasks on a path, i.e., the depth, is defined by the Parallelism Factor (PF), which is denoted by $\frac{\sqrt{|V|}}{\alpha}$; in our experiments, α was randomly chosen from $\{0.5, 1.0, 2.0\}$ for each DAG. For each task, the out degree was randomly chosen from 1 to 5. The number of tasks in a random DAG was chosen from $\{50, 100, 300, 500, 1{,}000\}$, and in total, 1,000 random DAGs were generated to compare the average values of the SLR, WSL, and efficiency of CMWSL with other algorithms. Each task size and data size was determined by an uniform distribution, and the maximum to minimum ratio was set to 100.

As for the heterogeneity of the system, we defined $\overline{\alpha}$ as the ratio of the maximum processing speed to the minimum processing speed, and $\overline{\beta}$ as the ratio of the maximum communication bandwidth to the minimum communication bandwidth. In the experiments in Sections 5.2, 5.3, 5.4, 5.5, and 5.6, both $\overline{\alpha}$ and $\overline{\beta}$ were randomly chosen from $\{2, 5, 7, 10\}$ for each DAG. In our experiments, 16 patterns of $(\overline{\alpha}, \overline{\beta})$ were performed.

As for the processing speed, if $\alpha_i/\alpha_j = 10$, it is assumed that the time required to perform one arithmetic operation by p_j is ten times larger than the time required by p_i (e.g., α_i = 10 for a 3.0 GHz CPU and α_j=1 for a 300 MHz CPU as the

TABLE 3
Performance Ratios of (B)/(A)

CCR	SLR Ratio	WSL Ratio	Efficiency Ratio
0.1	1.038	1.151	0.982
0.5	1.041	1.163	0.979
1	1.082	1.193	0.955
3	1.090	1.206	0.929
5	1.098	1.244	0.911
10	1.123	1.247	0.906

reference processor). As for the communication bandwidth, if $\beta_i/\beta_j = 10$, this means that the time required to send one byte of data from p_j to a destination having a larger communication bandwidth is ten times larger than the time to send the same from p_i to the destination (e.g., $\beta_i = 10$ for 1.0 Gbps and $\beta_j = 1$ for 100 Mbps). The number of given processors in the system was set to $|P| = 3{,}000$ in Sections 5.2, 5.3, 5.4, and 5.5, because of the performance limitations in our simulation environment. Because the number of tasks in this experiment is less than 3,000, the number is large enough to assign each task without resource starvation.

5.2 Adequacy of the Lower Bound for each Cluster Size

Since the lower bound for each task cluster processing time in CMWSL is a modified version of that in [14], we evaluated the effects on the schedule length by investigating the differences of their lower bounds. In this comparison, we denote CMWSL by (A) and CMWSL with the lower bound derivation in [14] by (B).

Table 3 shows the comparison results for the SLR, WSL, and efficiency for (A) and (B). The values in the table are the average ratios for (A) and (B) with various CCR patterns. As mentioned in Section 5.1, for each CCR, 1,000 random DAGs were generated by randomly choosing a task number from $\{50, 100, 300, 500, 1{,}000\}$, $\alpha \in \{0.5, 1.0, 2.0\}$, and one combination from the 16 patterns of $(\overline{\alpha}, \overline{\beta})$. Notice that for every CCR pattern, both the SLR and WSL ratios are greater than 1.0, while the efficiency is less than 1.0. This implies that the lower bound of CMWSL has a greater effect on the schedule length than the lower bound derived in [14].

Table 4 shows the comparison results for various lower bounds of CMWSL. In this comparison, we tested $0.5\delta^*_{opt}$, $0.8\delta^*_{opt}$, $1.2\delta^*_{opt}$, and $2.0\delta^*_{opt}$ as the comparison targets. The values in the table correspond to the mean ratios for δ^*_{opt}. For $0.5\delta^*_{opt}$ and $0.8\delta^*_{opt}$, the SLR and efficiency ratios are worse than those of δ^*_{opt} as the CCR increases. For $1.2\delta^*_{opt}$ and $2.0\delta^*_{opt}$, the SLR ratios are approximately 8 percent worse than those of δ^*_{opt} for all of the CCR cases. On the other hand, the efficiencies for $1.2\delta^*_{opt}$ are better than those of δ^*_{opt} in select CCR cases, and the efficiencies for $2.0\delta^*_{opt}$ are better than those of all of the CCR cases. This is because larger lower bounds for each task cluster processing time lead to a reduction in the number of task clusters, i.e., assigned processors. Although the objective of CMWSL is to minimize the schedule length, we can choose a lower bound larger than δ^*_{opt} if the objective is to optimize the efficiency.

5.3 Adequacy of the Scheduling Phase in CMWSL

We compared the effects of various scheduling policies on the schedule length. In the scheduling phase of CMWSL, each task is scheduled to make the data waiting time zero. This policy is based on start time minimization [16] by updating the scheduling priority for each task [32]. Since we calculated only the difference of scheduling policies, all methods used in this comparison perform the same procedures before the scheduling phase.

The comparison targets for this evaluation were as follows: from the free task list, a task having (A) the minimum DRT is scheduled by CMWSL, (B) the maximum $rank^*_u$ value is scheduled, which is adopted in Heterogeneous Earliest Finish Time [20], (C) the maximum $DRT + rank_u$ is scheduled, and (D) the maximum DRT is scheduled, where $rank^*_u$ is given by

$$rank^*_u(n_k) = t_p(n_k, \alpha_i) + t_c(e_{k,l}, L_{i,j}) + \max_{n_l \in suc(n_k)} \{rank^*_u(n_l, \alpha_j)\}. \quad (18)$$

Table 5 shows the scheduling ratios for (A) for various CCR values. The average column is the (average SLR for each policy)/(average SLR of (A)), the better and worse columns correspond to the percentile of better cases and worse cases in terms of the scheduling length in 1,000 random DAGs, respectively. Furthermore, the DWT column is the sum of the data waiting times (DWTs) for each scheduled task. If the CCR value is 0.1, (B) outperforms (A) in the average SLR. However, for other cases, (A) outperforms all other policies. In CMWSL, the scheduling policy for start time minimization has a positive effect on the schedule length and has a smaller DWT. This implies that minimizing the DWT for each task effectively makes the schedule length small. For the assignment state obtained by task clustering in CMWSL, the task with the minimum DRT should be scheduled with the highest priority.

TABLE 4
Performance Comparison among Various Lower Bound Patterns

	$0.5\delta^*_{opt}$			$0.8\delta^*_{opt}$			$1.2\delta^*_{opt}$			$2.0\delta^*_{opt}$		
CCR	SLR Ratio	WSL Ratio	Effi. Ratio	SLR Ratio	WSL Ratio	Effi. Ratio	SLR Ratio	WSL Ratio	Effi. Ratio	SLR Ratio	WSL Ratio	Effi. Ratio
0.1	1.024	1.016	0.832	1.027	1.025	0.827	1.016	1.060	1.016	1.012	1.076	1.112
0.5	1.037	1.017	0.820	1.029	1.029	0.826	1.037	1.156	0.997	1.026	1.107	1.130
1	1.328	1.223	0.689	1.380	1.287	0.655	1.074	1.008	1.068	1.010	1.198	1.636
3	1.528	1.277	0.422	1.508	1.204	0.428	1.059	1.015	0.975	1.016	1.042	1.713
5	1.649	1.355	0.323	1.569	1.341	0.340	1.047	1.039	1.006	1.076	1.096	1.781
10	1.852	1.466	0.186	1.833	1.459	0.187	1.049	1.012	0.975	1.078	1.108	1.569

TABLE 5
Comparison of SLR among Scheduling Policies

	(B) maximum $rank_u^*$				(C) maximum $DRT + rank_u^*$				(D) maximum DRT			
CCR	average	better	worse	DWT	average	better	worse	DWT	average	better	worse	DWT
0.1	0.982	63%	35%	1.334	1.031	21%	77%	1.042	1.041	11%	88%	1.049
0.5	1.012	41%	58%	1.387	1.018	30%	68%	1.214	1.033	25%	72%	1.056
1	1.008	43%	57%	1.539	1.059	21%	78%	1.034	1.054	15%	84%	1.06
3	1.009	40%	58%	1.280	1.034	26%	74%	1.215	1.064	11%	87%	1.071
5	1.013	32%	66%	1.659	1.055	19%	79%	1.307	1.058	14%	83%	1.077
10	1.008	39%	61%	1.408	1.010	36%	64%	1.383	1.025	27%	73%	1.032

5.4 Comparison for Random Task Graphs

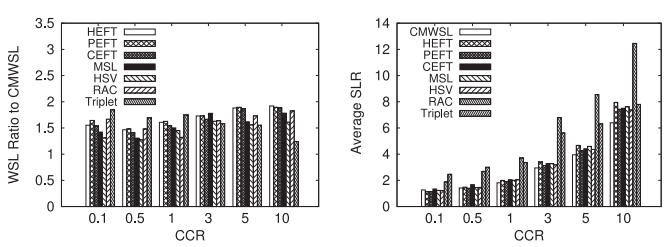
In this experiment, we compared the SLR with several task scheduling algorithms among random DAGs with various CCR values and various system heterogeneities. Comparison targets in this experiment are HEFT, PEFT, CEFT, MSL, HSV, RAC, and Triplet. Since both RAC and Triplet do not specify how to schedule each task after their task clustering phase, we used the scheduling policy of CMWSL for their algorithms. Fig. 6 shows the comparison results in terms of WSL, SLR for random DAGs. Fig. 6a is the WSL ratio to that of CMWSL. A value greater than 1.0 means that the WSL value is larger than that of CMWSL. Fig. 6b is the comparison results of SLR. As we described in Section 5.1, both $\overline{\alpha}$ and $\overline{\beta}$ were chosen from $\{2, 5, 7, 10\}$. Then WSL, SLR, and efficiency were averaged for the comparison in 1,000 random DAGs. Fig. 6a shows the WSL ratios for CMWSL in random DAGs. For every CCR, the WSL for CMWSL is less than that of the other algorithms. From this result, we conclude that the objective of task clustering in CMWSL is achieved in random DAGs. In Fig. 6b, CMWSL outperforms all of the other algorithms with a CCR value greater than 1.0.

5.5 Comparison for Real Task Graphs

We compared both the SLR and efficiency for real-world task graphs, i.e., Gaussian elimination DAGs and Fast Fourier Transform (FFT) DAGs, with varying CCR values. In both experiments, the average SLR and efficiency values were derived in 1,000 DAGs for each CCR case. For heterogeneity, $\overline{\alpha}$ and $\overline{\beta}$ were randomly chosen from $\{2, 5, 7, 10\}$ for each trial. Once $\overline{\alpha}$ and $\overline{\beta}$ were determined, each processing speed and communication bandwidth was assigned according to a normal distribution.

5.5.1 Gaussian Elimination

We generated kji Gaussian eliminations without pivoting DAGs [33] for $\frac{m(m+1)}{2}$ tasks, where m is the matrix size.



(a) Comparison of WSL (b) Comparison of SLR

Fig. 6. WSL and SLR comparison for random DAGs.

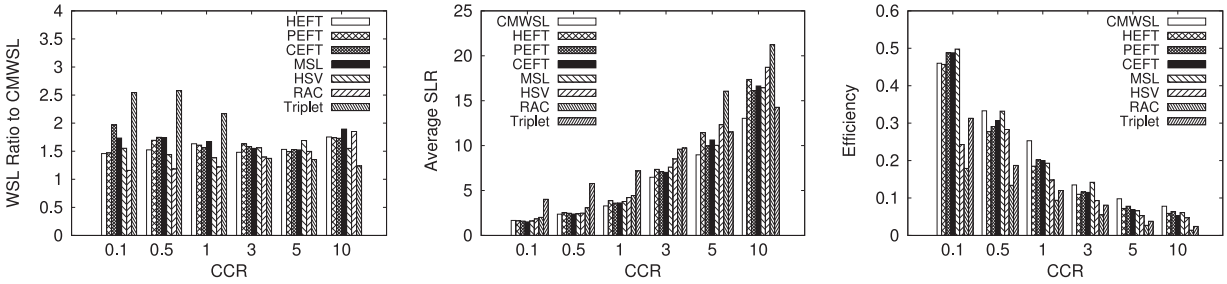
Fig. 7 shows the comparison results of WSL, SLR and efficiency in Gaussian elimination DAGs, where m was randomly chosen from 10, 30, and 50 for each DAG. In Fig. 7a, CMWSL has the smallest WSL in every case, and the objective of task clustering in CMWSL is achieved. In Fig. 7b, the SLR of CMWSL is not the lowest when CCR = 0.1 or CCR = 0.5, but it achieves the lowest SLR value when CCR is greater than or equal to 1.0. Fig. 7c shows the comparison results for the efficiency value. From this result, the efficiency obtained by CMWSL is not the highest if CCR = 0.1, but it achieves the highest efficiency for all other CCR values.

5.5.2 FFT

The number of FFT tasks was $m \log m$, where m is the matrix size [34]. In the FFT procedures, arithmetic additions and multiplications are performed for a task, and the resultant data is sent to a set of immediate successor tasks. According to [34], the number of such operations increases as the data size increases. Thus, we varied the CCR values from 0.1 to 10, similar to both random DAGs and Gaussian elimination DAGs. Fig. 8 shows the comparison results in terms of the WSL, SLR and efficiency where the matrix size m was randomly chosen from 32, 64, and 128 for each DAG. Fig. 8a shows the comparison results for the WSL in FFT DAGs, where m is the matrix size. Notice that CMWSL achieves the smallest WSL in every case. In Fig. 8b, the SLR of CMWSL is not the lowest if the CCR is 0.1 or 0.5, but it obtains the lowest SLR if the CCR is greater than or equal to 1.0. Fig. 8c shows the comparison results in terms of the efficiency. Notice that CMWSL does not achieve the highest efficiency if the CCR is less than 1.0; this suggests that CMWSL requires more processors than the other algorithms. If the CCR is greater than or equal to 3.0, CMWSL achieves the highest efficiency.

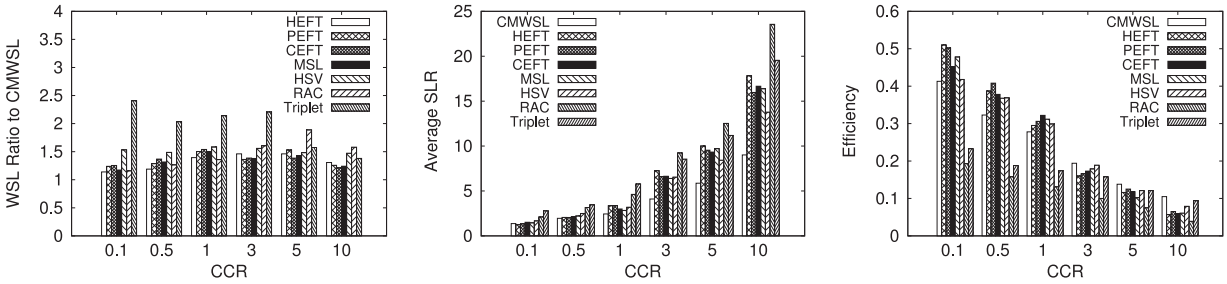
5.6 Comparison in Case of Processor Starvation

In CMWSL, we assume that the number of processors is sufficiently large for allocating tasks, i.e., $m \leq |P|$, where m is the actual number of processors required by CMWSL. In this comparison, we assume that $m > |P|$ for confirming the versatility of CMWSL. We define such a case as processor starvation. Processor starvation occurs when $\overline{P} = \emptyset$ at the first line in Algorithm 1. In this case, unassigned tasks are assigned to actual processors in the scheduling phase described in Section 4.4. Thus, the number of unassigned tasks at the task clustering phase may have an effect on the schedule length. That is, the schedule length can vary



(a) Comparison of WSL (b) Comparison of SLR (c) Comparison of Efficiency

Fig. 7. Comparison for Gaussian elimination DAGs.

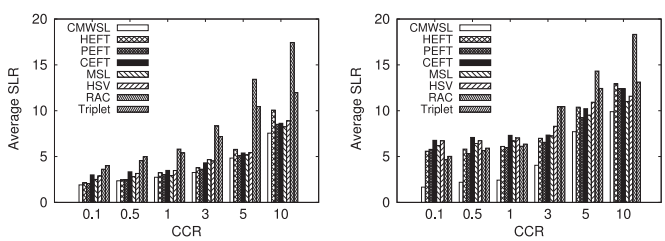


(a) Comparison of WSL (b) Comparison of SLR (c) Comparison of Efficiency

Fig. 8. Comparison for FFT DAGs.

depending on the degree of processor starvation. In this comparison, we varied the number of processors according to the following policy. First, CMWSL is performed with $|P| = 3{,}000$ in 1,000 random DAGs for each CCR, and m is obtained by averaging the number of required processors in CMWSL. Then, we performed a comparison in terms of SLR by varying $|P| = m/2$ and $m/5$ in 1,000 random DAGs for each CCR.

Fig. 9 presents the comparison results in terms of SLR; (a) is the case with $m/2$ and (b) is the case with $m/5$. In Fig. 9a, we see that CMWSL outperforms all of the other algorithms, even when CCR is equal to 0.1. Although SLR in CMWSL becomes less effective than in the case of non-starvation, such as in Fig. 6b, the performances of the other algorithms show a greater decline. From Fig. 9b, we see that the difference between CMWSL and the other algorithms is large when CCR is three or lower, and decreases when CCR is five or greater. However, in this case CMWSL still outperforms all of the other algorithms. Because a DAG with lower CCR requires more processors, the degree of processor starvation in such a case is higher. From these results, we conclude that CMWSL can be useful for minimizing the schedule length whether processor starvation occurs or not.



(a) SLR at $|P| = m/2$ (b) SLR at $|P| = m/5$

Fig. 9. Comparison in processor starvation.

Throughout the results from Sections 5.4 and 5.5, HEFT, PEFT, CEFT, MSL, and HSV have a good performance if CCR is under 0.5. However, their performances greatly degrade in case of a high processor starvation. From this fact, it is concluded that they can achieve a good schedule length in a computationally-intensive job without processor starvation.

6 Conclusion

In this paper, we proposed a clustering-based task scheduling algorithm in heterogeneous systems, namely, CMWSL, to obtain the near-optimal set of processors that minimizes the schedule length. In CMWSL, the scheduling priority is determined by the actual processor assignment obtained by task clustering. As a result, the schedule length can be made smaller than that of several existing algorithms if the CCR is greater than or equal to 0.5. In the task clustering step of CMWSL, the lower bound of the cluster processing time was derived by considering the task clustering state as well as the information for each task, each processor speed, and each communication bandwidth. Thus, the efficiency was improved compared to other existing lower bound derivation methods. In conclusion, the proposed CMWSL algorithm can be applied to execute data-intensive jobs in heterogeneous systems by utilizing processors.

As a future work, we explore the method to make WSL further smaller by applying other prioritizing schemes [35] for each task. Another target is to reduce the effect on communication overhead in a data-intensive DAG. This can be achieved by applying an graph partitioning such as METIS [36] to minimize the communication time among tasks.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 25730077.

REFERENCES

[1] S. Di and C. L. Wang, "Dynamic optimization of multiattribute resource allocation in self-organizing clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 3, pp. 464–478, Mar. 2013.

[2] M. Xu, et al., "A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl.*, 2009, pp. 629–634.

[3] H. M. Fard, et al., "A multi-objective approach for workflow scheduling in heterogeneous environments," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 300–309.

[4] J. Singh, et al., "Contention aware energy efficient scheduling on heterogeneous multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1251–1264, May 1, 2015.

[5] T. Ghafarian and B. Javadi, "Cloud-aware data intensive workflow scheduling on volunteer computing systems," *Future Gener. Comput. Syst.*, vol. 51, pp. 87–97, 2015.

[6] L. Zenga, et al., "SABA: A security-aware and budget-aware workflow scheduling strategy in clouds," *J. Parallel Distrib. Comput.*, vol. 75, pp. 141–151, 2015.

[7] H. J. Braun, et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[8] E. G. Coffman, *Computer and Job-Scheduling Theory*. Hoboken, NJ, USA: Wiley, 1976.

[9] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *J. Parallel Distrib. Comput.*, vol. 16, pp. 276–291, 1992.

[10] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. Cambridge, MA, USA: MIT Press, 1989.

[11] M. Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 330–343, Jul. 1990.

[12] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 951–967, Sep. 1994.

[13] J. C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling DAGs on multiprocessors," in *Proc. 8th Symp. Parallel Distrib. Process.*, 1996, pp. 152–156.

[14] H. Kanemitsu, et al., "A processor mapping strategy for processor utilization in a heterogeneous distributed system," *J. Comput.*, vol. 3, no. 11, pp. 1–8, 2011.

[15] H. Kanemitsu, et al., "On the effect of applying the task clustering for identical processor utilization to heterogeneous systems," in *Grid Computing*. Rijeka, Croatia: InTech, Mar. 2012, pp. 29–46.

[16] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Comput.*, vol. 19, pp. 1321–1344, 1993.

[17] O. Sinnen, *Task Scheduling for Parallel Systems*. Hoboken, NJ, USA: Wiley, 2007.

[18] A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 6, pp. 686–701, Jun. 1993.

[19] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surveys*, vol. 31, no. 4, 406–471, 1999.

[20] H. Topcuoglu, et al., "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[21] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.

[22] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Comput.*, vol. 38, no. 4-5, pp. 175–193, 2012.

[23] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 399–409, 2008.

[24] D. Sirisha and G. V. Kumari, "A new heuristic for minimizing schedule length in heterogeneous computing systems," in *Proc. IEEE Int. Conf. Elect., Comput. Commun. Technol.*, 2015, pp. 1–7.

[25] G. Xie, et al., "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, 2015.

[26] M. Shahid, et al., "Level based batch scheduling strategy with idle slot reduction under DAG constraints for computational grid," *J. Syst. Softw.*, vol. 108, pp. 110–133, 2015.

[27] B. Jedari and M. Dehghan, "Efficient DAG scheduling with resource-aware clustering for heterogeneous systems," *Comput. Inf. Sci.*, vol. 208, pp. 249–261, 2009.

[28] S. Chingchit, et al., "A flexible clustering and scheduling scheme for efficient parallel computation," in *Proc. 13th Int. 10th Symp. Parallel Distrib. Process.*, 1999, pp. 500–505.

[29] C. Boeres, et al., "A cluster-based strategy for scheduling task on heterogeneous processors," in *Proc. 16th Symp. Comput. Archit. High Perform. Comput.*, 2004, pp. 214–221.

[30] C. Boeres and V. E. F. Rebello, "On the design of clustering-based scheduling algorithms for realistic machine models," in *Proc. 15th Int. Parallel Distrib. Process. Symp.*, 2001.

[31] B. Cirou and E. Jeannot, "Triplet: A clustering scheduling algorithm for heterogeneous systems," in *Proc. Int. Conf. Parallel Process. Workshops*, 2001, pp. 231–236.

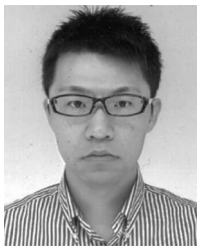
[32] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 17–186, Feb. 1993.

[33] A. K. Amoura, et al., "Scheduling algorithms for parallel gaussian elimination with communication costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 7, pp. 679–686, Jul. 1998.

[34] S. Chabridon and E. Gelenbe, "Dependable parallel computing with agents based on a task graph model," in *Proc. 3rd Euromicro Workshop Parallel Distrib. Process.*, 1995, pp. 350–357.

[35] Z. Henan and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *Proc. 9th Int. Euro-Par Parallel Process.*, 2003, pp. 189–194.

[36] K. George and V. Kumar, "Metis-Unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.



Hidehiro Kanemitsu received the BS degree in science from Waseda University, Japan, and the MS and DS degrees in global information and telecommunication studies from Waseda University, Japan. His research interests include parallel and distributed computing, grid, peer-to-peer computing, and web service technology. He is currently an assistant professor at the Global Education Center, Waseda University, Japan. He is a member of the IEEE.

Masaki Hanada received the BE degree in resources engineering from Waseda University in 1996, and the MS and DS degrees in global information and telecommunication studies from Waseda University in 2003 and 2007, respectively. He is an associate professor in the Department of Information Systems, Tokyo University of Information Sciences. His research interests include QoS/traffic control and resource management in communication networks. He is a member of the IEEE.

Hidenori Nakazato received the BE degree in electronics and telecommunications from Waseda University in 1982 and the MS and PhD degrees in computer science from the University of Illinois in 1989 and 1993, respectively. He was affiliated with Oki Electric from 1982 to 2000, where he developed equipment for public telephone switches, distributed environments for telecommunications systems, and communications quality control mechanisms. He is a professor at the Department of Communications and Computer Engineering, Waseda University, Japan. His research interests include information centric networking, cooperation mechanisms of distributed programs, and network QoS control. He is a member of the IEEE.