JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 10, No. 1, March 2016, pp. 9-20

A Task Scheduling Method after Clustering for Data Intensive Jobs in Heterogeneous Distributed Systems

Kazuo Hajikano*
Department of Information Technology and Electronics, Daiichi Institute of Technology, Kagoshima, Japan
k-hajikano@daiichi-koudai.ac.jp

Hidehiro Kanemitsu
Global Education Center, Waseda University, Tokyo, Japan
kanemih@ruri.waseda.jp

Moo Wan Kim
Department of Informatics, Tokyo University of Information Sciences, Chiba, Japan
mwkim@rsch.tuis.ac.jp

Hee-Dong Kim
Department of Information & Communications Engineering, Hankuk University of Foreign Studies, Yongin, Korea
kimhd@hufs.ac.kr

Abstract
Several task clustering heuristics are proposed for allocating tasks in heterogeneous systems to achieve a good response time in data intensive jobs. However, one of the challenging problems is the process in task scheduling after task allocation by task clustering. We propose a task scheduling method after task clustering, leveraging worst schedule length (WSL) as an upper bound of the schedule length. In our proposed method, a task in a WSL sequence is scheduled preferentially to make the WSL smaller. Experimental results by simulation show that the response time is improved in several task clustering heuristics. In particular, our proposed scheduling method with the task clustering outperforms conventional list-based task scheduling methods.

Category: Smart and intelligent computing

Keywords: Task clustering; Task scheduling; Heterogeneous; Data intensive

I. INTRODUCTION

Recently, task execution models are becoming more diverse, e.g., grid computing, cloud computing. Especially, as we enter the so called "Big Data" era where massive nonuniform data are gathered from the real world and webpages through internet, searched and analyzed in real time manner. These data are processed in parallel by heterogamous systems, i.e., many computational resources with various processing speeds and communication band-

Open Access http://dx.doi.org/10.5626/JCSE.2016.10.1.9 http://jcse.kiise.org

Received 09 November 2015; **Revised** 07 February 2016; **Accepted** 16 February 2016
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

widths, connected over the network. Jobs running over such environment are known to be data-intensive such as MapReduce, in which large data travel over tasks. Therefore, computational resource allocations and task scheduling methods for data-intensive jobs are important for efficient use of resources and quick response time (hereinafter, we call it 'schedule length') for real time application.

On task scheduling methods for a work-flow type job with precedence constraint among tasks over heterogeneous distributed environment, methods based on list scheduling such as Heterogeneous Earliest Finish Time (HEFT) [1], Predict Earliest Finish Time (PEFT) [2], and Constrained Earliest Finish Time (CEFT) [3]. These methods are effective for reducing the schedule length against computationally intensive jobs. Yet, these methods do not achieve the expected improvement in schedule length for data intensive jobs such as MapReduce because each task is inserted in the idle time for each processor without considering the actual data transfer time.

Considering data communication time among tasks, many task clustering heuristics have been proposed for heterogeneous systems with data communications localized in a cluster, e.g., Resource Aware Clustering (RAC) [4], Flexible Clustering and Scheduling (FCS) [5], Clustering for Heterogeneous Processors (CHP) [6], and Triplet [7]. These methods are considered to be effective for data intensive jobs, as actual processing time and communication bandwidth of each processor is not used, do not always achieve a good schedule length in all types of applications and systems, and do not specify task scheduling after clustering. Other task scheduling methods considering data communication time have been proposed, e.g., [8, 9]. These duplicate tasks to localize communications among tasks.

We propose a task scheduling method after completion of task clustering and processor assignment to the clusters, leveraging worst schedule length (WSL) [10, 11] affecting schedule length. It works with conventional clustering heuristics and improves their schedule length, using the actual processor time and the communication bandwidth of the assigned processor.

II. ASSUMED MODE

A. System Model

In this paper, we supposed off-line scheduling instead of on-line scheduling where tasks are assigned to processor after becoming ready to be executed, e.g., the method proposed in [12]. We assumed that a job is expressed as a directed acyclic graph (DAG), which is 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 i-th task is denoted as n_i, and let $w(n_i)$ be the size of n_i, i.e., $w(n_i)$ is the sum of unit times for processing by the reference processor. We define data dependency and direction of data transfer from n_i to n_j as $e_{i,j}$, where $c(e_{i,j})$ is the sum of unit times for transferring data from n_i to n_j over the reference communication link. One constraint imposed by a DAG is that a task cannot start for execution until all data from its predecessor tasks arrive. $pred(n_i)$ is the set of immediate predecessors of n_i, and $suc(n_i)$ is the set of immediate successors of n_i. If $pred(n_i) = \emptyset$, n_i is called START task, and if $suc(n_i) = \emptyset$, n_i is called END task. If there are one or more path form n_i to n_j, we denote such a relation as $n_i \prec n_j$.

We assume that each processor is completely connected to others over the network, with heterogeneous processing speeds and communication bandwidths. The set of processors is expressed as $P = \{p_1, p_2, \cdots p_n\}$, and the processing speed of p_i is denoted as α_i when that of the reference processor is set to 1. The execution time in the case when n_k is processed on p_i is expressed as $t_p(n_k, \alpha_i) = w(n_k) / \alpha_i$. Let the set of communication bandwidths be $\beta = \{\beta_1, \beta_2, \cdots \beta_n\}$, where that of the reference communication link is set 1, and if $c(e_{i,j})$ is sent from p_k to p_l, the communication time is defined as the communication speed $L_{k,l}$ such that $L_{k,l} = \min\{\beta_k, \beta_l\}$, then let the communication time be $t_c(e_{i,j}, L_{k,l}) = c(e_{i,j})/L_{k,l}$.

B. Task Clustering

A task cluster is a set of tasks explicitly grouped together before scheduling every task. As a result, every task in a task cluster comes to be assigned to the same processor. Let the i-th task cluster in V^s_{cls} be $cls_s(i)$. If n^s_k is included in $cls_s(i)$ by the (s + 1)th task clustering, it is expressed by $cls_{s+1}(i) \leftarrow cls_s(i) \cup \{n^s_k\}$. If any two tasks, i.e., n_i and n_j are included in the same cluster, they are assigned to the same processor. In this case the communication time between n^s_i and n^s_j becomes zero.

The total task size in a task cluster is called the task cluster size, and we call the value of the task cluster size divided by the processing time the task cluster processing time, or $T(cls_s(i), \alpha_p)$.

C. Schedule Length

In a DAG application, a task can start for execution if every data from its immediate predecessors has 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.

$$t_f(n_j, p_p) = t_s(n_j, p_p) + t_p(w(n_j), \alpha_p). \quad (1)$$

When all the data from $pred(n_j)$ has arrived at n_j, n_j can start for execution immediately. However, even if every

data from $pred(n_j)$ has arrived at n_j, n_j cannot be started until the execution of another task in the same processor is finished. The time when all the data from every immediate predecessor task has arrived at n_j is called as data ready time (DRT) [10]. If we define DRT of n_j on p_p when every task in $pred(n_i)$ is scheduled as $t_{dr}(n_i, p_q)$, it is derived as follows.

$$t_{dr}(n_i, p_q) = \max_{n_i \in pred(n_j), p_p \in P}\{t_f(n_i, p_p) + t_C(c(e_{i,j}), L_{p,q})\}. \quad (2)$$

From (2), $t_{dr}(n_j)$ 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 data transfer time. The start time of n_i , i.e., $t_s(n_j, p_p)$ is derived using DRT as follows.

$$t_s(n_j, p_q) = \max\ \{t_f(n_i, p_q), t_{dr}(n_j, p_q)\}, \quad (3)$$

where n_i has been scheduled. In (3), 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 which can accommodate $t_p(n_j, \alpha_q)$ and starts at $t_f(n_i, p_q)$ in 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 data waiting time for data arrival from other processors. If we define such a data waiting time as $t_w(n_j, p_p)$, it can be expressed as follows.

$$t_w(n_j, p_q) = \begin{cases} 0, & if\ t_f(n_i, p_q) \geq t_{dr}(n_j, p_q) \\ t_{dr}(n_j, p_q) - t_f(n_i, p_q), & otherwise. \end{cases} \quad (4)$$

During the time slot derived by (4), some tasks come to be inserted by an insertion-based policy in order to minimize their completion time. The schedule length is expressed by

$$Schedule\ length = \max_{suc(n_i)=\emptyset, p_p \in P}\{t_f(n_k, p_p)\}\ . \quad (5)$$

D. Task Scheduling in Clusters

Fig. 1 shows an example of a task clustering and scheduling tasks. In the figure, (a) represents the initial state of the DAG and (b)–(d) represent the states after a task clustering has been finished. Execution orders of tasks are different among (b), (c) and (d).

In Fig. 1(b), schedule length in the order of $n_2 \rightarrow n_3 \rightarrow n_5$ is 23. In Fig. 1(c), the schedule length in the order of $n_3 \rightarrow n_2 \rightarrow n_5$ is 24, because the data arrival time of $e_{2,7}$ at n_7 is delayed by the increase of the start time of n_2. In Fig. 1(d), the schedule length is larger than that of (b) and (c) by scheduling n_2 in $cls(1)$ at the latest execution order.

It can be concluded that the schedule length is varied depending on the execution order for each task in clusters, even though the set of tasks belonging to the cluster is the same among (b), (c), and (d).

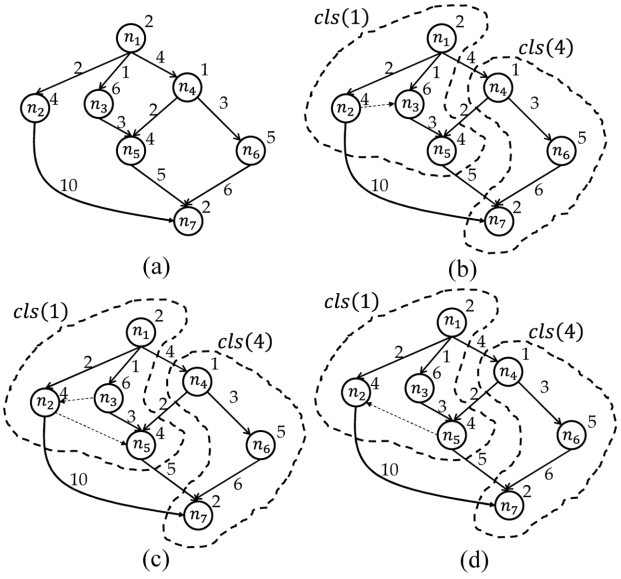


Fig. 1. Task scheduling after task clustering. (a) Initial state, (b) task scheduling (SL=23), (c) task scheduling (SL=24), and (d) task scheduling (SL=29).

III. PREVIOUS WORK

In our previous work [10, 11], we presented a processor assignment strategy for processor utilization. The number of processors is limited by imposing the lower bound for each cluster size. Under the constraint, we theoretically showed which processor should be assigned the cluster (assignment unit). The processor to be assigned is the one which has good impact on minimizing indicative value for the schedule length.

A. Related Results

Our previous works [10, 11] focus on how to determine the lower bound of each assignment unit size (sum of task sizes in the task cluster divided by the processing speed) in order to find a subset of given processors. Contributions of the literature [10, 11] are: (i) WSL was defined and its effect on the schedule length was proposed, and (ii) the lower bound of the assignment unit size for a processor was derived. In the following sections, we briefly describe those two factors.

B. Worst Schedule Length

WSL is the maximum execution path length in case each task is executed as late as possible in a processor, provided that there is no data waiting time for each task

Table 1. Notation for $WSL(M(G^s))$

Parameter	Definition
$top(cls_s(i))$	The set of tasks having no immediate predecessor tasks in $cls_s(i)$
$in(cls_s(i))$	The set of tasks having one or more immediate predecessor tasks in the cluster other than $cls_s(i)$
$out(cls_s(i))$	The set of tasks having one or more immediate successor tasks in the cluster other than $cls_s(i)$
$btm(cls_s(i))$	The set of tasks having no immediate successor tasks in $cls_s(i)$
$dc(n_k, cls_s(i))$	The set of tasks in $cls_s(i)$ having paths from n_k, i.e., $\{n_l \mid n_k \prec n_l, n_l \in cls_s(i)\} \cup \{n_k\}$
$S(n_k, cls_s(i))$	$\sum_{n_i \in cls_s(i)} t_p(n_i, \alpha_p) - \sum_{n_i \in dc(n_k, i)} t_p(n_i, \alpha_p)$
$tlevel(n_k)$	$\max_{n_l \in pred(n_k)} \{tlevel(n_l) + t_p(n_l, \alpha_p) + tc(e_{l,k}, \beta_{q,p})\}$, if $n_k \in top(cls_s(i))$. $TL(cls_s(i)) + S(n_k, cls_s(i))$, otherwise.
$TL(cls_s(i))$	$\max_{n_k \in top(cls_s(i))} \{tlevel(n_k)\}$.
$blevel(n_k)$	$\max_{n_l \in suc(n_k),\ n_l \notin cls_s(i)} \{t_p(n_k, \alpha_p) + tc(e_{k,l}, \beta_{p,q}) + blevel(n_l)\}$,
$BL(cls_s(i))$	$\max_{n_k \in out(cls_s(i))} \{S(n_k, cls_s(i)) + blevel(n_k)\}$.
$Level(n_k)$	$tlevel(n_k) + blevel(n_k)$
$LV(cls_s(i))$	$TL(cls_s(i)) + BL(cls_s(i)) = \max_{n_k \in cls_s(i)} \{level(n_k)\}$
$WSL(M(G_s))$	$\max_{cls_s(i) \in V_{scls}} \{LV(cls_s(i))\}$

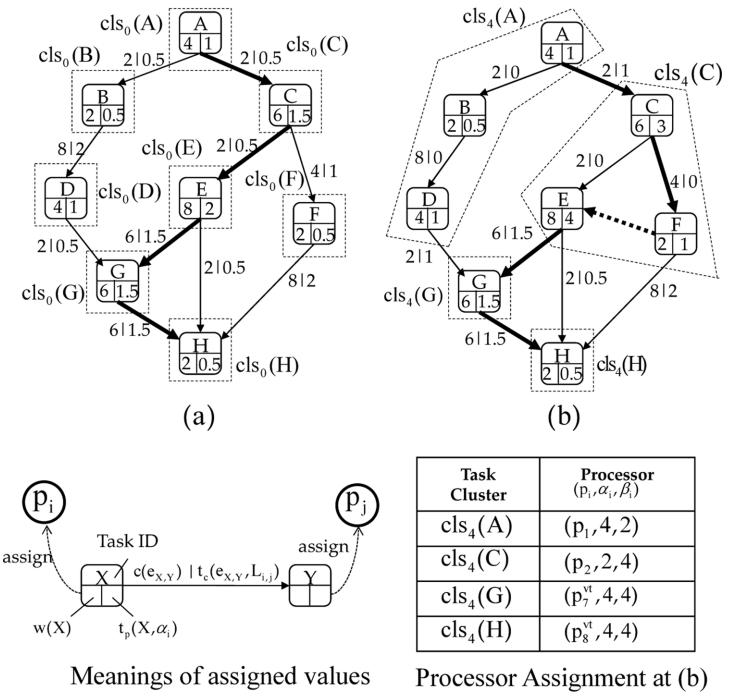
Here, note $n_k \in cls_s(i)$.

once the processor starts execution. Since WSL derivation requires the assignment and clustering state $M(G^s)$, first we present the definition of $M(G^s)$. At the initial clustering state G^0, each task belongs to a task cluster and suppose that it is assigned to the virtual processor $p_i^{vt} \in P^{vt}$ having the maximum processing speed and maximum communication bandwidth.

Let $M : G^s \rightarrow P$ be the assignment state after a processor assignment is performed to 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 p_i^{vt} is a virtual processor having the maximum processing speed and the maximum communication bandwidth. From the initial assignment state, each virtual processor is replaced with an actual processor and the number of task clusters is reduced by a processor assignment and a task clustering, i.e., $|M(G^s)| \leq |M(G^0)|$ for $s > 0$.

Table 1 shows notations for deriving WSL. $WSL(M(G^s))$ is the maximum of $LV(cls_s(i))$, which corresponds to the maximum of $level(n_k)$, where $n_k \in cls_s(i)$. $level(n_k)$ is the maximum time duration from the START task to the END task in case n_k is scheduled as late as possible. That is, $WSL(M(G^s))$ is the maximum of *level* value in all tasks at the clustering state of $M(G^s)$. For a task cluster $cls_s(i)$, $top(cls_s(i))$ is the set of tasks which can start execution first in $cls_s(i)$, and $in(cls_s(i))$ is the set of tasks having incoming edges from other task clusters and $out(cls_s(i))$ is the set of tasks having outgoing edges to other task clusters. $btm(cls_s(i))$ is the set of tasks having no immediate successor tasks in $cls_s(i)$, and $dc(n_k, cls_s(i))$ is the set of tasks being one 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 in case that $t_w(n_k, p_p) = 0$ and every task having no dependencies with n_k is executed before n_k. $TL(cls_s(i))$ 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 data ready time. $blevel(n_k)$ 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 $BL(cls_s(i))$, WSL at $M(G^s)$, i.e., $WSL(M(G^s))$ is defined as the maximum value of $LV(cls_s(i))$ where $cls_s(i) \in V^s{}_{cls}$.

EXAMPLE 3.1. Fig. 2 presents an example of WSL derivation. (a) is the DAG at $M(G^0)$, and (b) is the DAG at $M(G^4)$, i.e., four tasks have been included in task clusters. In both cases, bold arrows mean the execution sequence dominating WSL. Suppose there are three processors $(p_i, \alpha_i, \beta_i) = (p_1, 4, 2), (p_2, 2, 4), (p_3, 4, 4)$. In Fig. 2(a), each task belongs to a task cluster denoted by dashed box and assigned to a virtual processor having the maximum processing speed being 4 and the maximum communica-



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. 2. Example of worst schedule length derivation. (a) The state of $M(G^0)$ and (b) the state of $M(G^4)$.

tion bandwidth being 4. As for $cls_0(A)$, we have the following result.

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

From this state, $WSL(M(G^0)) = 9.5$ is determined by the path, A, C, E, G, and H, which is the same as the critical path length at $M(G^0)$. On the other hand, at (b) 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 result.

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

At $cls_4(C)$, we have

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

Since we have the following results as:

$dc(C, cls_4(C)) = \{C, E, F\}$. $dc(E, cls_4(C)) = \{E\}$.
$dc(F, cls_4(C)) = \{F\}$.

We obtain the following values:

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

Then we have $LV(cls_4(C)) = 2 + 13 = 15$. At the DAG of Fig. 2(b), $cls_4(C)$ 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 execution after F is finished). We obtain $LV(cls_4(A)) = 14$, $LV(cls_4(G)) = LV(cls_4(H)) = 15$. Then we have $WSL(M(G^4)) = 15$.

C. WSL Properties

The task clustering heuristic proposed in the literature [10] performs to minimize WSL instead of schedule length because the schedule length cannot be determined until every task is scheduled by a task scheduling method. According to the literature [11], it is proved that both the upper bound and the lower bound of the schedule length can be made smaller if WSL is small. Thus, minimizing WSL by a task allocation can lead to the reduction of the schedule length.

D. Lower Bound of Assignment Unit Size

In the literature [10], $\Delta WSL_{up}(M(G^s))$ is defined as an upper bound of $WSL(M(G^S)) - WSL(M(G^0))$. That is, a small value of $\Delta WSL_{up}(M(G^s))$ means that WSL can be made smaller by the s-th clustering step. According to the literature [10], $\Delta WSL_{up}(M(G^s))$ is a function of the lower bound, the processing speed, and the communication bandwidth.

At $\Delta WSL(M(G^s))$, there are a number of task clusters exceeding the lower bound, i.e., $\delta(\alpha_p, \beta_p, G^s)$ 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,

$$SL \geq \frac{WSL(M(G^s)) - \Delta WSL_{up}(M(G^s))}{1 + \frac{1}{g_{min}(M(G^0))}}, \quad (6)$$

$\Delta WSL_{up}(M(G^s))$ assumes the local minimum value when $\delta(\alpha_p, \beta_p, G^s)$ equals to the following value.

$$\delta_{opt}(\alpha_p, \beta_p, G^s) = \sqrt{\frac{\sum_{n_k \in seq_{s-1}^{\leftarrow}} w(n_k)}{\alpha_p}\left[\frac{\max_{n_k \in V}\{w(n_k)\}}{\alpha_p} + \frac{\max_{e_{k,l} \in E}\{c(e_{k,l})\}}{\beta_p}\right]}, \quad (7)$$

where $seq_{s-1}^{\leftarrow}$ is a path where each task belongs to the set of tasks dominating $WSL(M(G^{s-1}))$. In (7), $\delta_{opt}(\alpha_p, \beta_p, G^s)$ is derived by tracing the path in the set of tasks and edges dominating $WSL(M(G^{s-1}))$.

EXAMPLE 3.2. Fig. 3 shows an example of the lower bound derivation presented in [10]. In Fig. 3(a), 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, G, H*} dominating $M(G^4)$. From this path, $\delta_{opt}(\alpha_p, \beta_p, G^5)$ is derived as 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 $cls_5(G)$ at (c) is 3 + 1 = 4 < 9.3 and then $cls_5(G)$ will be clustered into one of cls_5(A) or $cls_5(C)$ in the next task clustering step.

E. Existing Clustering Heuristics

Many task clustering heuristics have been proposed for homogeneous system [13-16]. In homogenous systems, task assignment is not required. As a result, a clustering priority in a task clustering heuristic for homogeneous system directory affects the schedule length. However, in heterogeneous systems, system information, such as the processing speed and communication bandwidth, is required for deriving a clustering priority. Conventional task clustering heuristics for heterogeneous systems do not use actual processing time or communication time for the clustering priority [4-7]. The objective of a clustering is to localize data communications, and it is known that DAGs with larger data size have better schedule length. Even though RAC [4] and FCS [5] define the lower bound of task clusters, they cannot get good schedule length for all DAG.

On the other hand, in literature [10], we proposed the task clustering heuristic which derives the lower bound for each task cluster automatically and gets good schedule length for all DAGs. Proposed task clustering heuristic consists of three phases based on minimizing WSL. (i) Derive the lower bound for the cluster size as (7), (ii) decide the processor to be assigned, which minimizes ΔWSL. Then (iii) merge several tasks into a cluster until its size exceeds the lower bound derived in (i). In other words, the proposed method manages to generate the linear cluster to minimize WSL.

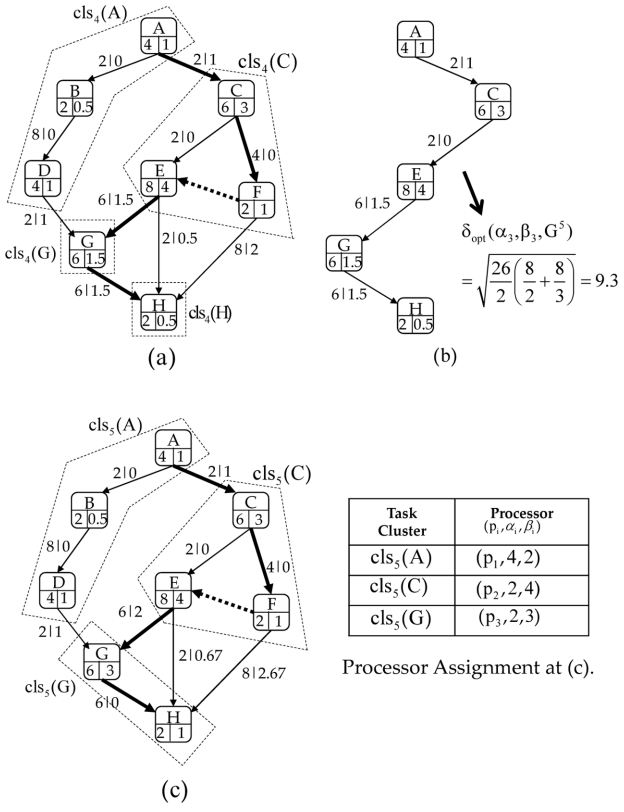


Fig. 3. Lower bound derivation at M(G^5). (a) The state of $M(G^4)$, (b) $\delta_{opt}(\alpha_3, \beta_3, G^5)$ derivation with the path (A-C-E-G-H) being a part of {A,C,F,E,G,H} which dominates WSL($M(G^4)$), and (c) the state of $M(G^5)$.

IV. PROPOSAL

A. Basic Idea

In this section we propose a task scheduling method that is performed after each task has been assigned to a processor by a task allocation. If task allocation is performed by a task clustering, each task is assigned to a task cluster, i.e., a task cluster is an assignment unit for each processor. The following shows features of our proposal.

- Our proposal minimizes the WSL to lower the upper and lower bounds of the schedule length.
- We use the actual processing speed and the communication bandwidths of each processor assigned to each task cluster for deriving the scheduling. The conven-

tional list-based task scheduling methods such as HEFT, CEFT and PEFT adopt average processing time and the average communication time for deriving the scheduling priority.

B. Proposed Task Scheduling

We present how the scheduling priority is derived for each task. We call a task as a free task, whose every immediate predecessor task has been scheduled. The objective of the proposed scheduling is to minimize WSL by choosing a task having the maximum ***level*** value from the free task list. By choosing such a task, we obtain the fact that WSL can be made smaller as follows.

THEOREM 4.1. *Let WSL after m tasks have been scheduled be* WSL_m.

If a task $n_k \in FREE_{sched}$ *s.t.,*

$$level(n_k) = \max_{n_i \in FREE_{sched}} \{level(n_i)\},$$

is selected at the m-th task selection phase, then we have

$$WSL_m \leq WSL_{m-1}. \quad (8)$$

***Proof* 4.1.** First we define the two sets of tasks as follows,

$$FREE_p = \{n_i | n_i \in FREE_{sched},\ pred(n_i) \cap pred(n_k) \neq \varnothing\},$$
$$FREE_d = \{n_i | n_i \in FREE_{sched},\ pred(n_i) \cap pred(n_k) = \varnothing\}. \quad (9)$$

Without loss of generality, suppose that n_k belongs to a task cluster $cls(K)$ and the level of $cls(K)$ is defined as $LV(K)$.

(i) Level of tasks in $FREE_p$,

For each task $n_i \in FREE_p$, there can be two cases, i.e., whether n_i belongs to $cls(K)$ or not.

(i-i) The case of $n_i \in cls(K)$.

If we have $LV(K) = \boldsymbol{level(n_k)}$ before the m-th task selection, we obtain $LV(K)$ after n_k is selected as follows.

$$LV(K) = \max_{n_i \in CLS(k)-\{n_k\}} \{level^m(n_i)\} \leq level^{m-1}(n_k), \quad (10)$$

where $\boldsymbol{level^m(n_i)}$ is $level(n_i)$ after m tasks have been scheduled.

(i-ii) The case of $n_i \notin cls(K)$.

In this case, $\boldsymbol{level(n_i)}$ is not affected by n_k selection. Thus, WSL is not increased.

(ii) Level of tasks in $FREE_p$.

For each task $n_i \in FREE_p$, $\boldsymbol{level(n_i)}$ is not affected by n_k selection. Thus, WSL is not increased.

From (i) and (ii), it leads that WSL is not increased by choosing the task having the maximum ***level*** in $FREE_{sched}$. □

INPUT: Clustered DAG G.
OUTPUT: Schedule of G.
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 set of unscheduled tasks;
Define $FREE_{sched}$ to be the set of tasks whose all immediate predecessor tasks have been scheduled;

1: **WHILE** *USCHED* $\neq \emptyset$ DO
2: Find n_i having a maximum of $level(n_i)$ in $FREE_{sched}$. If two or more tasks have the same maximum value, the task n_i having maximum *blevel* value is selected;
3: $FREE_{sched} \leftarrow FREE_{sched} - \{n_i\}$;
4: $USCHED \leftarrow FREE_{sched} - \{n_i\}$;
5: Insert *ni* 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.
6: Set $t_f(n_i, proc(n_i))$;
7: **FOR** $n_j \in suc(n_i)$ **DO**
8: **IF** $n_k \notin USCHED$ for $\forall n_k \ pred(n_j)$ **THEN**
9: $FREE_{sched} \leftarrow FREE_{sched} \cup \{n_j\}$;
10: **END IF**
11: **END FOR**
12: **END WHILE**

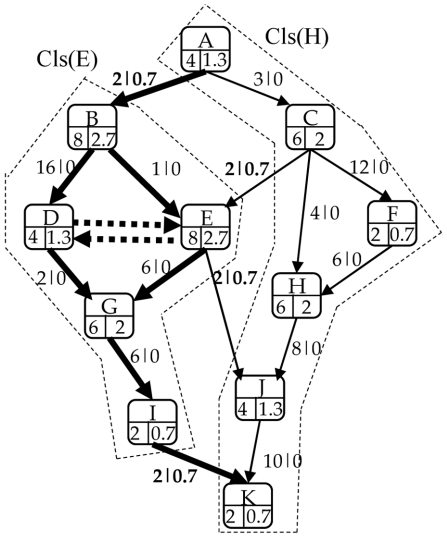
Fig. 4. Procedure for the scheduling.

As described in Section III-C, minimizing WSL contributes to lower the upper bound and the lower bound of the schedule length. To minimize the WSL, the strategy of our proposal is to reduce WSL for each scheduling step. However, how to minimize WSL is an NP-complete problem as with the schedule length minimization. That is, our proposal is based on a warranty for WSL reduction. Moreover, our proposal has a practical time complexity (see Section IV-D) and thus is said to be a cost-effective approach to reducing both the upper bound and the lower bound of the schedule length.

C. Procedure and Example

Fig. 4 presents the procedures for proposed scheduling. First, $FREE_{sched}$ includes all of the START tasks and USCED includes all of the tasks. This procedure finishes when *USCED* becomes empty.

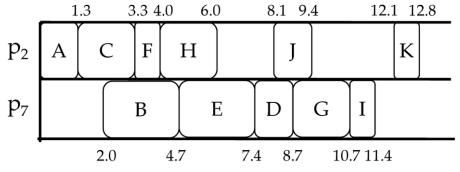
At line 2, the task to be scheduled is selected by the ***level***. After the completion of task clusterings and processor assignments to clusters, $\boldsymbol{level(n_i)}$ can be derived with *the actual processor speed and the communication bandwidth of the processor* which $\boldsymbol{n_i}$ has been assigned to. That is, we can derive WSL. In the scheduling phase, the task having the maximum ***level*** in $FREE_{sched}$ is scheduled by inserting it into an idle time of the processor. The task is assigned to an idle time of $proc(n_i)$ at line 5. After the task is scheduled, each task in $suc(n_i)$ becomes a part of $FREE_{sched}$ if all of its predecessor tasks have been scheduled.



(a)

cls(E)				cls(H)			
Step	FREE task	Level of FREE task	Selected task & FT	Step	FREE task	Level of FREE task	Selected task & FT
				1	A	*level*(A)=12.8	FT(A)=1.3
1	B	*level*(A)=12.8	FT(B)=4.7	2	C	*level*(C)=8	FT(C)=3.3
2	D, E	*level*(D)=12.8, *blevel*(D)=5.4 *level*(E)=12.8, *blevel*(E)=6.8	FT(E)=7.4	3	F	*level*(F)=8	FT(F)=4
				4	H	*level*(H)=8	FT(E)=6
3	D	*level*(D)=12.8	FT(D)=8.7				
4	G	*level*(G)=12.8	FT(G)=10.7	5	J	*level*(J)=8	FT(J)=9.4
5	I	*level*(I)=12.8	FT(I)=11.4				
				6	K	*level*(I)=12.8	FT(K)=12.8

(b)



(c)

Fig. 5. Example of the scheduling. (a) Before scheduling tasks, (b) the task scheduling results, and (c) Gantt chart.

EXAMPLE 4.1. Fig. 5 shows an example of scheduling tasks. Fig. 5(a)–(c) present the DAG before scheduling each task, the task scheduling result, and Gantt chart, respectively. In Fig. 5(a), the DAG has two task clusters and each cluster is assigned to each processor. Since there is only one START task in Fig. 5(a), A is included in $FREE_{sched}$, and it is selected for scheduling. Then, B in *cls*(*E*) and C in *cls*(*E*) become free, and their *levels* are 12.8 and 8, respectively; moreover, $t_f(A, p_2) = 1.3$. At step 2 of *cls*(*E*) in Fig. 5(b), D and E are assigned to p_7, but their *levels* are the same. In this case, their *blevel* values are compared at line 2 in Fig. 4. Since *blevel*(*D*) = 5.4 < *blevel*(*E*) = 6.8, E is selected. At step 3 of *cls*(*E*) in Fig. 5(b), D is selected and its finish time is calculated. There is no idle time between B and E according to Fig. 5(c), and D is added after E. As a result, the finish time of D is 8.7. Similarly, at step 4 of *cls*(*E*) in Fig. 5(c), there is no idle time in B-E or E-D. Thus, G is added after D. At step 5 of *cls*(*E*), step 5 of *cls*(*H*) and step 6 of *cls*(*H*), I, J, and K are added, and the schedule time is 12.8.

D. Complexity of the Proposed Method

In this section, we analyze the complexity of the proposed scheduling algorithm. At line 2 in Fig. 4, we have $|FREE_{sched}| \leq |V|$ and every task in $FREE_{sched}$ is ordered in nonincreasing order of *level*. Thus, one task is put in $FREE_{sched}$ by $\log|FREE_{sched}|$ steps. As a whole, this operation takes $O(|V|\log|V|)$.

At line 5 in Fig. 4, an idle slot can be found by at most the number of tasks assigned to the processor. This takes $|V|^2$.

At line 7 to 11, this requires $|suc(n_i)|\log|suc(n_i)|$ steps. As a whole, it takes $O(|E|\log|V|)$.

Therefore, the complexity of the proposed scheduling is $O|V|^2$, which is not higher than those of existing scheduling [1-3].

V. EXPERIMENT

A. Objectives

We conducted the experimental simulations to confirm advantages of our proposal against existing methods in terms of schedule length. Actually, the schedule length ratio (SLR) [1, 2] metric was used to measure the performance of each scheduling method. The SLR is defined as follows;

$$SLR = \frac{SL}{\sum_{n_k \in CP} t_p(n_k,\ \max_{P_i \in P}\{\alpha_i\})}. \qquad (11)$$

B. Comparison with Existing Scheduling Methods

Here, we conducted the experimental simulations to confirm advantages of the proposed scheduling method against existing methods in terms of schedule length ratio.

1) Existing Scheduling Methods

Task clustering heuristics does not specify the task scheduling method after task clustering. Here, we picked up following task scheduling methods after clustering for comparison.

-Method 1: The proposed scheduling method.

-Method 2: The task with minimum rank_down (the longest path length from START task to the task) is scheduled [1].

-Method 3: The task with minimum value of sum of rank_down and rank_up (the longest path length from the task to the END task) is scheduled [1].
-Method 4: The task with maximum rank_up is scheduled [1].

We supposed that Triplet [7] and RAC [4] are clustering heuristics working under task scheduling methods.

2) Experimental Environment

In the simulation, two types of DAGs, i.e., random DAGs and Gaussian elimination DAGs, were generated. We present each condition as follows.

a) Random DAGs: In the simulation, 100 DAGs were generated under following conditions and average of schedule lengths of DAGs were calculated after scheduling tasks. For each DAG, the number of tasks in the DAG was chosen from {50, 100, 300, 500, 1000} randomly, the max to min ratio in terms of task size was 100, the max to min ratio in terms of data size was 100, and the communication to computation ratio (CCR) [17], defined as the average communication bandwidth divided by the average processing speed, 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 $(\sqrt{|V|})/\alpha$; in our experiments, α was chosen from {0.5, 1.0, 2.0}. For each task, out degree was randomly chosen from 1 to 5. For the heterogeneity of the system, processing speed of a CPU was chosen as normal distribution where the max to min ratio were set to 2, 5 and 10, and communication bandwidth were chosen as normal distribution where the max to min ratio was set to 2, 5 and 10.

b) Gaussian elimination DAGs: In the simulation, 100 DAGs were generated in case matrix sizes were 10, 30 and 50, and averages of schedule lengths of DAGs were calculated after scheduling tasks. For each DAG, the max to min ratio in terms of task size was 100, the max to min ratio in terms of data size was 100, and the CCR was chosen from {0.1, 0.5, 1, 3, 5, 10}. For the heterogeneity of the system, processing speed of a CPU was chosen as normal distribution where the max to min ratios were set to 2, 5 and 10, and communication bandwidth was chosen as normal distribution where the max to min ratios were set to 2, 5 and 10.

The simulation environment was developed by JRE1.6.0_0, the operating system was Windows XP SP3, the CPU architecture was Intel Core 2 Duo 2.66 GHz, and the memory size is 2.0 GB.

Table 2. Comparison of SLR among scheduling method for random DAGs (1/2)

CCR	Triplet w/ method 1	Triplet w/ method 2	Triplet w/ method 3	Triplet w/ method 4
0.1	1.438	1.483	1.44	1.421
0.5	2.013	2.211	2.108	2.093
1	2.511	2.475	2.497	2.602
3	4.014	4.213	4.159	4.186
5	4.616	4.672	4.702	4.815
10	8.319	8.467	8.513	8.344

DAG: directed acyclic graph, SLR: schedule length ratio, CCR: communication to computation ratio.

Table 3. Comparison of SLR among scheduling method for random DAGs (2/2)

CCR	RAC w/ method 1	RAC w/ method 2	RAC w/ method 3	RAC w/ method 4
0.1	2.017	2.271	1.938	1.998
0.5	2.736	2.994	2.866	2.831
1	3.813	4.017	3.905	3.982
3	7.298	7.419	7.498	7.418
5	9.371	9.667	9.891	9.776
10	12.732	13.044	13.318	13.417

DAG: directed acyclic graph, SLR: schedule length ratio, CCR: communication to computation ratio.

Table 4. Comparison of SLR among scheduling methods for Gaussian elimination DAGs (1/2)

CCR	Triplet w/ method 1	Triplet w/ method 2	Triplet w/ method 3	Triplet w/ method 4
0.1	3.724	3.778	3.781	3.517
0.5	6.132	6.391	6.218	6.191
1	7.375	7.668	7.423	7.402
3	9.412	9.815	9.915	9.529
5	13.858	14.194	14.011	14.033
10	15.729	16.512	16.228	15.994

DAG: directed acyclic graph, SLR: schedule length ratio, CCR: communication to computation ratio.

Table 5. Comparison of SLR among scheduling methods for Gaussian elimination DAGs (2/2)

CCR	RAC w/ method 1	RAC w/ method 2	RAC w/ method 3	RAC w/ method 4
0.1	2.017	2.133	2.317	1.983
0.5	3.248	3.372	3.174	3.711
1	4.395	4.571	4.618	4.498
3	9.155	9.372	9.779	9.227
5	16.289	16.835	17.037	16.793
10	18.642	19.325	19.492	18.881

DAG: directed acyclic graph, SLR: schedule length ratio, CCR: communication to computation ratio.

3) Experimental Result for Each Clustering

Tables 2 and 3 show comparison results for random DAGs in terms of SLR. Tables 2 and 3 are cases of Triplet and RAC, respectively. For each value of CCR, SLRs for Radom DAGs are derived with different scheduling methods. Tables 4 and 5 show the comparison results for Gaussian Elimination DAGs in terms of SLR. Tables 4 and 5 are cases of Triplet and RAC, respectively. For each value of CCR, SLRs for Gaussian elimination DAGs are derived with different scheduling methods.

In any case, the proposed scheduling method gets better SLR than that of other scheduling methods if CCR is equal to or larger than 0.5. That is, the proposed method is suitable for data-intensive jobs with larger CCR.

C. Comparison of Clustering Heuristics

In this experiment, we compared the SLR by method 1 in the task clustering in [10], Triplet and RAC with conventional list-based task scheduling heuristics (HEFT, PEFT, and PEFT). We used the same experimental environment described in Section V-B-2). We call the proposed clustering heuristic in [10] as clustering 1 in this section.

1) Experimental Results

Fig. 6 shows the comparison results for SLR. For each value of CCR, SLRs for Radom DAGs are derived with the proposed scheduling methods working above clustering 1, Triplet and RAC and with HEFT, PEFT and CEFT. We can see that the proposed clustering heuristic [10] with the proposed scheduling method has better SLRs, if CCR is larger than 1.0. Fig. 7 shows experimental results for SLRs on Gaussian elimination DAGs. We can see that the clustering heuristic in [10] with the proposed scheduling method has better SLRs, if CCR is equal to and larger than 3. That is, the proposed scheduling method with the clustering heuristic proposed in [10] is suitable for data intensive jobs with larger CCR.

D. Discussion

For both of Gaussian elimination DAGs and Random DAGs, clustering 1 and xEFT (i.e., CEFT, PEFT, HEFT) does not make big difference in terms of SLR in case CCR is less than 3, because delays caused by data-waiting time at each task affect SLR a little. On the other hand, clustering 1 shows better SLRs remarkably in case

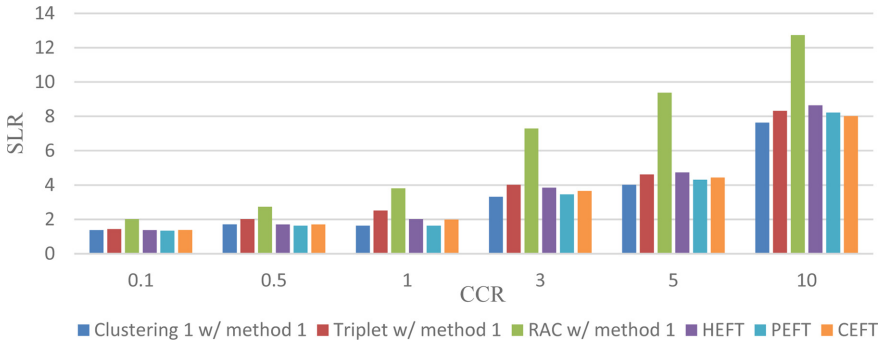


Fig. 6. Comparison of schedule length ratio (SLR) among clustering methods for random directed acyclic graph (DAG). CCR: communication to computation ratio.

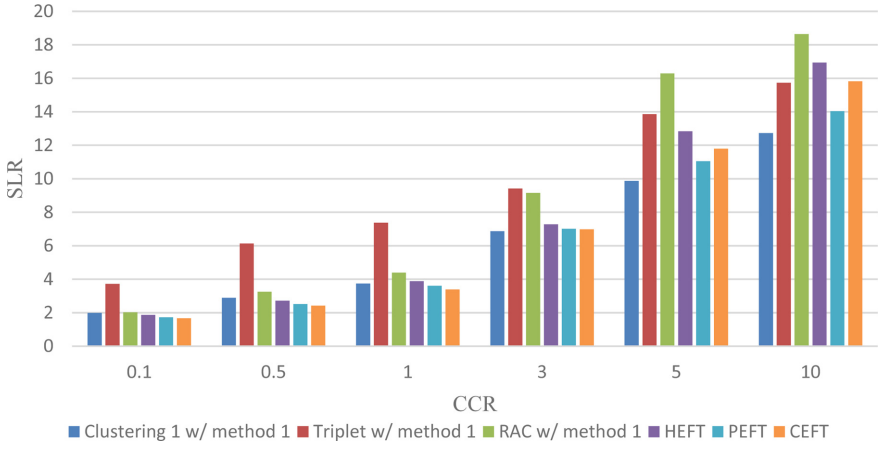


Fig. 7. Comparison of schedule length ratio (SLR) among clustering methods for Gaussian elimination. CCR: communication to computation ratio.

CCR is equal to or greater than 3. That is, bigger data-transferring time makes bigger data-waiting time at each processor and it is considered to contribute to make SLR worse remarkably. The combination of clustering 1 minimizing WSL and proposed scheduling method, i.e., method 1, which makes WSL smaller, was proved to be effective for larger CCR.

Therefore, the proposed scheduling method working over the clustering heuristic in [10] is suitable for workflow type jobs handling massive data.

VI. CONCLUSION

In this paper, we proposed a task scheduling method for use upon completion of task clustering in heterogeneous system. With the proposed method, tasks on the path dominating WSL, i.e., with maximum value of ***level***, are preferred to be scheduled. As a result, the SLR can be made smaller than in several existing methods, if the CCR is equal to or greater than 0.5.

Furthermore, the proposed scheduling method with the task clustering heuristic proposed in [10] has been confirmed to produce smaller SLR than that of well-known list scheduling method such as HEFT, CEFT and PEFT, if CCR is equal to or greater than 3 through the experiment. In conclusion the proposed scheduling method can be applied to execute data-intensive jobs in heterogeneous systems.

REFERENCES

1. H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.
2. H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682-694, 2014.
3. M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Computing*, vol. 38, no. 4, pp. 175-193, 2012.
4. B. Jedari and M. Dehghan, "Efficient DAG scheduling with resource-aware clustering for heterogeneous systems," in *Computer and Information Science 2009*, Heidelberg: Springer, pp. 249-261, 2009.
5. S. Chingchit, M. Kumar, and L. N. Bhuyan, "A flexible clustering and scheduling scheme for efficient parallel computation," in *Proceedings of the 13th International Parallel Processing and 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP 1999)*, San Juan, Puerto Rico, 1999, pp. 500-505.
6. C. Boeres and V. E. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," in *Proceedings of 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2004)*, Foz do Iguacu, Brazil, 2004, pp. 214-221.
7. B. Cirou and E. Jeannot, "Triplet: a clustering scheduling algorithm for heterogeneous systems," in *Proceedings of International Conference on Parallel Processing Workshops*, Valencia, Spain, 2001, pp. 231-236.
8. S. G. Ahmad, C. S. Liew, M. M. Rafique, E. U. Munir, and S. U. Khan, "Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems," in *Proceedings of 2014 IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud),* Sydney, 2014, pp. 129-136.
9. A. A. Nasr, N. A. El-Bahnasawy, and A. El-Sayed, "Task scheduling algorithm for high performance heterogeneous distributed computing systems," *International Journal of Computer Applications*, vol. 110, no. 16, pp. 23-29, 2015.
10. H. Kanemitsu, G. Lee, H. Nakazato, T. Hoshiai, and Y. Urano, "A processor mapping strategy for processor utilization in a heterogeneous distributed system," *Journal of Computing*, vol. 3, no. 11, pp. 1-8, 2011.
11. H. Kanemitsu, G. Lee, H. Nakazato, T. Hoshiai, and Y. Urano, "On the effect of applying the task clustering for identical processor utilization to heterogeneous systems," in *Grid Computing: Technology and Applications, Widespread Coverage and New Horizons*, Rijeka: Croatia, InTech, pp. 29-46, 2012.
12. W. Zheng, L. Tang, and R. Sakellariou, "A priority-based scheduling heuristic to maximize parallelism of ready tasks for DAG applications," in *Proceedings of 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Shenzhen, China, 2015, pp. 596-605.
13. A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 276-291, 1992.
14. V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, Cambridge, MA: MIT Press, 1989.
15. M. Y. Wu and D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Transactions on Parallel & Distributed System*, vol. 1, no. 3, pp. 330-343, 1990.
16. T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951-967, 1994.
17. O. Sinnen, *Task Scheduling for Parallel Systems*, Hoboken, NJ: John Wiley & Sons, 2007.

Kazuo Hajikano

Kazuo Hajikano received his B.E. and M.S. degrees in Electronics and Telecommunications from Waseda University in 1978 and 1980, respectively. He was affiliated with Fujitsu Ltd. from 1980 to 2014 where he researched and developed equipment for ATM switches, and automated resource allocation mechanism for a data center. He is a professor at the Department of Information Technology and Electronics, Daiichi Institute of Technology, Japan. His research interests include performance issues in distributed systems and networks, and distributed real-time systems over sensor network.

Hidehiro Kanemitsu

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.

Moo Wan Kim

Moo Wan Kim received B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan in 1974, 1977, and 1980, respectively. He joined Fujitsu Lab. in 1980 and had been engaged in research and development on multimedia communication systems, Intelligent Network, ATM switching system and operating system. In 1998 he joined Motorola Japan and had been engaged in research and development on CDMA2000 system. In 2000 he joined Lucent Japan and had been engaged in research and development on W-CDMA system, IMS and Parlay. In 2005 he joined Tokyo University of Information Sciences and has been engaged in research on Ubiquitous Network. He is currently a Professor in the Department of Informatics.

Hee-Dong Kim

Hee-Dong Kim is currently Vice President of Industry-University Cooperation of Hankuk University of Foreign Studies and a professor of Dept. of Information and Communication Engineering in Korea. He received the B.S. degree in Electric Engineering from Seoul National University in 1981 and Ph.D. degree in Electric and Electronics Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 1987. His current research interests include internet service, mobile communication & speech communication.