第一工業大学研究報告
第28号(2016)pp.19-30

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

Kazuo Hajikano[*1], Hidehiro Kanemitsu[*2], Moo Wan Kim[*3], and Hee-Dong Kim[*4]

*1 Department of Information Technology and Electronics, Daiichi Institute of Technology.
k-hajikano@daiichi-koudai.ac.jp
*2 Global Education Center, Waseda University, Tokyo Japan. kanemih@ruri.waseda.jp
*3 Department of Informatics, Tokyo University of Information Sciences, Chiba Japan.
mwkim@rsch.tuis.ac.jp
*4 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, it has been one of challenging problems that how each task should be scheduled after a task allocation by a 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 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 we proposed previously outperforms conventional list-based task scheduling methods.

Category: Task Scheduling
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, we are entering, so called, “Big Data” era where massive ununiformed 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 speed and communication bandwidth, connected over the network. Jobs running over such environment are called data-intensive job such as MapReduce, in which large data travel over tasks. Therefore, computational resource allocations and task scheduling methods for data-intensive jobs are important to get 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, e.g., HEFT [1], PEFT [2], CEFT [3] are well known. These methods are effective for reducing the schedule length against computationally intensive jobs. On the other hand, these are considered not to get improvement as expected about the schedule length in data intensive jobs such as MapReduce because they try to insert each task in the idle time for each processor without considering the actual data transfer time.

On the other hand, 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., RAC [4], FCS [5], CHP [6], Triplet [7]. These are considered to be effective for data intensive jobs. These do not use the actual processing time and communication bandwidth of each processor, 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 WSL (Worst Schedule Length) [10], [11] affecting schedule length. It works with conventional clustering heuristics and improve their schedule length, using the actual processor time and the communication bandwidth of the assigned processer.

II. Assumed Mode

A. System Model

In this paper, we supposed off-line scheduling instead of on-line scheduling where tasks are assigned to processer 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 time for processing by the reference processor. We define data dependency and direction of data transfer from n_i to n_j as $e_{i,j}$, $c(e_{i,j})$ is the sum of unit time 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 be started 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 $pred(n_i)$ =∅, n_i is called START task, and if $suc(n_i)$ =∅, 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 processor is expressed as $P = \{p_1, p_2, \cdot\cdot\cdot p_n\}$, 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 that 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 $\beta = \{\beta_1, \beta_2, \cdot\cdot\cdot\cdot \beta_n\}$, 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 $L_{k,l}$ is defined as $L_{k,l} = \min\{\beta_k, \beta_l\}$, 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 $cls_{s+1}(i) \leftarrow cls_s(i) \mathrm{U} \{n^s{}_k\}$. 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 let 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 have arrived. Let the start time of n_i on p_p be $t_s(n_i, p_p)$, 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 tasks 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 schedule as $t_{dr}(n_i, p_q)$, it is derived as follow.

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

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 $t_{dr}(n_i, p_q)$, 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 this figure, (a) represents the initial state of the DAG, and (b), (c), and (d) represent the state after a task clustering has been finished. Execution orders of tasks is different among (b), (c) and (d).

In (b), schedule length in the order of $n_2 \rightarrow n_3 \rightarrow n_5$ is 23. In (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 (d), the schedule length is larger than that of (b) and (c) by scheduling n_2 in $cls(1)$ at the latest execution order.

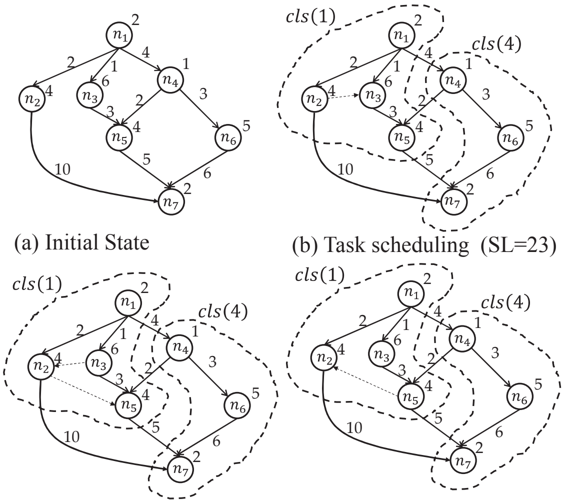
It can be concluded that the schedule length is varied depending on the execution order for each task in clusters, even though the set of tasks belonging to the cluster is same among (b), (c), and (d).

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 that 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 decide the lower bound of each assignment unit size (sum of task size 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) Worst Schedule Length (WSL) was defined



(a) Initial State

(b) Task scheduling (SL=23)

(c) Task scheduling (SL=24)

(d) Task scheduling (SL=29)

Fig. 1. Task Scheduling after Task Clustering

and its effect on the schedule length was proposed, (ii) the lower bound of the assignment unit size for a processor was derived. From the following sections, we briefly describe those two factors.

B. Worst Schedule Length (WSL)

WSL is the maximum execution path length in case that 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 execution. Since WSL derivation requires the assignment and clustering state $M(G^s)$, at 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, i.e.,

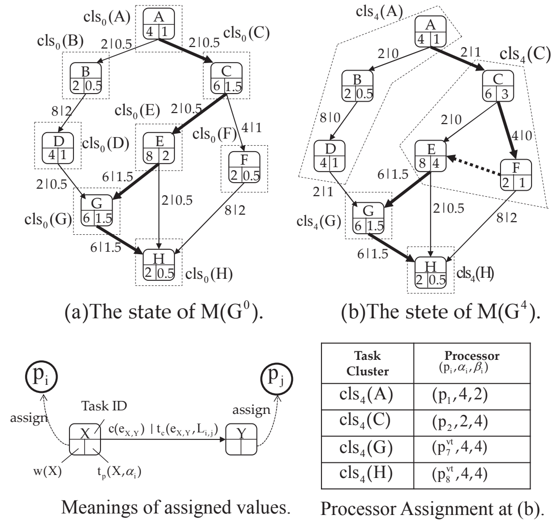
Let $M : G^s \rightarrow P$ be the assignment state after a processor assignment is performed to G^s. $M(G^0)$ corresponds to $\{(cls_0(1), p_1^{vt}), (cls_0(2), p_2^{vt}), \ldots, (cls_0(|V|), p_{|V|}^{vt})\}$, where p_i^{vt} is a virtual processor having the maximum processing speed and the maximum 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 that n_k is scheduled as late as possible. That is, $WSL(M(G^s))$ is the maximum of *level* value in all task 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 $cls_s(i) \in V^s_{cls}$.

Table 1. Notation for $WSL(M(G^s))$
(Here, note $n_k \in cls_s(i)$).

Parameter	Definition
$top(clss(i))$	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., $\{n_l \mid n_k \prec n_l, n_l \in cls_s(i)\} \cup \{n_k\}$
$S(n_k, cls_s(i))$	$\sum_{n_l \in cls_s(i)} t_p(n_l, \alpha_p)$ - $\sum_{n_l \in dc(nk,i)} t_p(n_l, \alpha_p)$
$tlevel(n_k)$	$\max_{n_l \in pred(n_k)} \{tlevel(n_l) + tp(n_l, \alpha_p) + tc(e_{l,k}, \beta_{q,p})\}$, $if\ n_k \in top(clss(i))$. $TL(cls_s(i)) + S(n_k, cls_s(i))$, *otherwise.*
$TL(cls_s(i))$	$\max_{n_k \in top(clss(i))} \{tlevel(n_k)\}$.
$blevel(n_k)$	$\max_{n_l \in suc(n_k), n_l \notin clss(i)} \{t_p(n_k, \alpha_p) + tc(e_{k,l}, \beta_{p,q}) + blevel(nl)\}$,
$BL(cls_s(i))$	$\max_{n_k \in out(cls_s(i))} \{S(n_k, clss(i)) + blevel(nk)\}$.
$Level(n_k)$	$tlevel(n_k) + blevel(n_k)$
$LV(cls_s(i))$	$TL(cls_s(i)) + BL(cls_s(i)) = \max_{n_k \in cls_s(i)} \{level(n_k)\}$.
$WSL(M(G_s))$	$\max_{cls_s(i) \in Vs_{cls}} \{LV(cls_s(i))\}$.

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



(a)The state of M(G^0).

(b)The stete 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. 2 Example of WSL derivation

in task clusters. In both cases, bold arrows mean the execution sequence dominating WSL. Suppose there are three processors (p_i, α_i, β_i) = (p_1, 4, 2), (p_2, 2, 4), (p_3, 4, 4). At (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 communication 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 decided 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
$top(cls_4(C)) = \{C\}$. $in(cls_4(C)) = \{C\}$.
$out(cls_4(C)) = \mathrm{btm}(cls_4(C)) = \{E, F\}$.

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

Then we have $LV(cls_4(C)) = 2 + 13 = 15$. At the DAG of (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 decided 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.

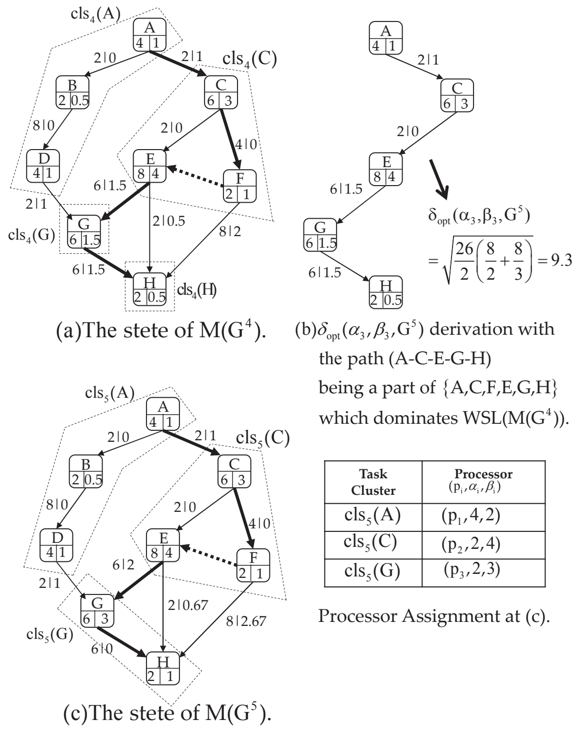
D. Lower Bound of Assignment Unit Size

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,

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

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



(a)The stete 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)).

(c)The stete of M(G^5).

Task Cluster	Processor (p_i, α_i, β_i)
cls$_5$(A)	(p_1,4,2)
cls$_5$(C)	(p_2,2,4)
cls$_5$(G)	(p_3,2,3)

Processor Assignment at (c).

Fig. 3. Lower Bound Derivation at M(G^5)

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

where $seq_{s-1}^{\prec}$ 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]. At (a), G and H are un-clustered 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.

D. Existing Clustering Heuristics

Many task clustering heuristics have been proposed for homogeneous system [14], [15], [16], [17]. 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], [5], [6], [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 can't get good schedule length for all DAG.

On the other hand, in literature [10], we proposed the task clustering heuristic which derivate the lower bound for each task cluster automatically and get 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 minimize 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.

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. Followings are features of our proposal.

- Our proposal minimizes the WSL (Worst schedule Length) to lower the upper and lower bound 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. In the conventional list-based task scheduling 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 tasks have been scheduled. The objective of the proposed scheduling is to minimize WSL by choosing a task having the maximum $\boldsymbol{level}$ 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*

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

***Proof4.1*.**

First we define the two set of tasks as follow,

$$FREE_p = \{n_i | n_i \in FREE_{sched},\ pred(n_i) \cap pred(n_k) \neq \emptyset\}.$$

$$FREE_d = \{n_i | n_i \in FREE_{sched}, pred(n_i) \cap pred(n_k) = \emptyset\}. (9)$$

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.

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

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 $FREE_p$.

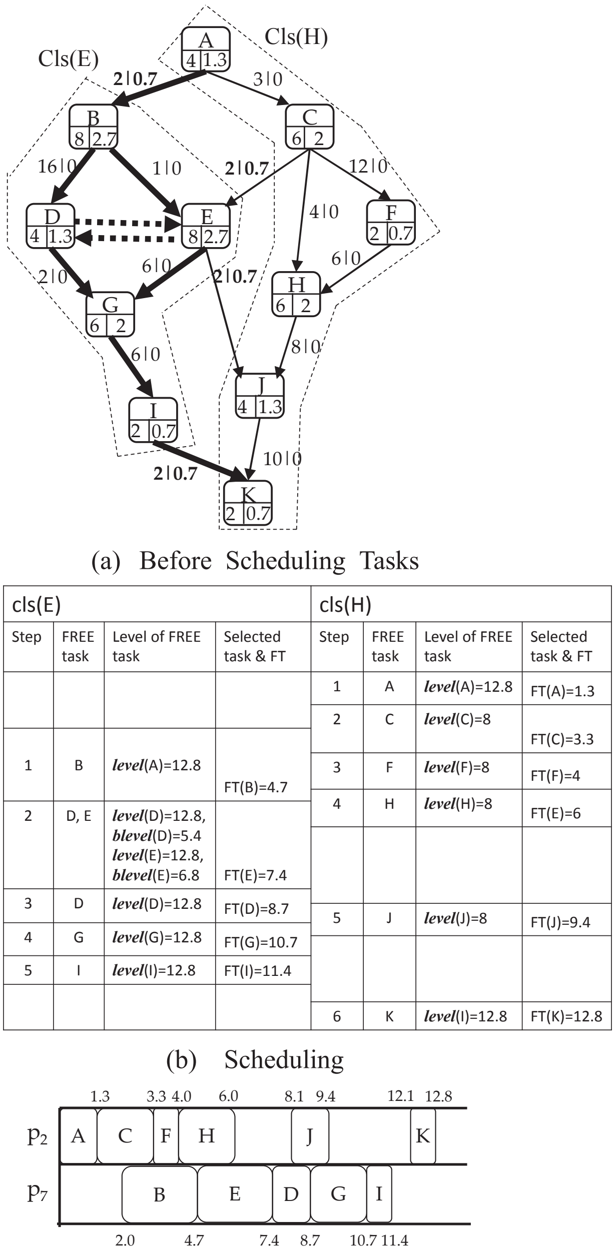
For each task $n_i \in FREE_p$, $\boldsymbol{level(n_i)}$ is not affected by n_k selection. Thus, WSL is not increased.

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

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 NP-complete problem as with the schedule length minimization. That is, our proposal is based on a

```
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;
--------------------------------------------------------------
WHILE USCHED ≠ ∅ DO
    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;
    FREE_sched ← FREE_sched - {n_i};
    USCHED ← FREE_sched - {n_i};
    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.
    Set t_f(n_i, proc(n_i));
    FOR n_j ∈ suc(n_i) DO
        IF n_k ∉ USCHED for∀ n_k pred(n_j) THEN
            FREE_sched ← FREE_sched ∪ {n_j};
        END IF
    END FOR
END WHILE
```

Fig. 4. Procedure for the Scheduling



(a) Before Scheduling Tasks

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
				3	F	***level***(F)=8	FT(F)=4
2	D, E	***level***(D)=12.8, ***blevel***(D)=5.4 ***level***(E)=12.8, ***blevel***(E)=6.8	FT(E)=7.4	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) Scheduling

(c) Gantt Chart

Fig. 5. Example of the Scheduling

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 for reducing both the upper bound and the lower bound of the schedule length.

C. Procedure and Example

Figure 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 $\boldsymbol{level}$. After the completion of task clusterings and processor assignments to clusters, $\boldsymbol{level(n_i)}$ can be derived with *the actual processer speed and the communication bandwidth of the processer* which $\boldsymbol{n_i}$ has been assigned to. That is, we can derivate WSL. In the scheduling phase, the task having the maximum $\boldsymbol{level}$ 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.

Example 4.1 Figure 5 shows an example of scheduling tasks. Figure 5(a), (b) and (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 $level$s 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 $level$s are 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 step3 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 according to 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 processer. 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], [2], [3].

V. Experiment

A. Objectives

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

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

B. Comparison with Existing Scheduling Methods.

Here, we conducted the experimental simulations to confirm advantages of the proposed scheduling method against existing methods in term of Schedule Length Ratio (SLR)**.**

1) Existing Scheduling Methods: Any task clustering heuristics doesn't 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 follow.

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 term of task size was 100, the max to min ratio in term of data size was 100, and the Communication to Computation Ratio (CCR) [13], 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 that matrix size were 10, 30 and 50, and average of schedule lengths of DAGs were calculated after scheduling tasks. For each DAG, the max to min ratio in term of task size was 100, the max to min ratio in term 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 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.

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.

3) Experimental Result for Each Clustering: Table

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

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

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

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

2 and 3 show comparison results for random DAGs in terms of SLR. Table 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. Table 4 and 5 show the comparison results for Gaussian Elimination DAGs in terms of SLR. Table 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 get 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 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

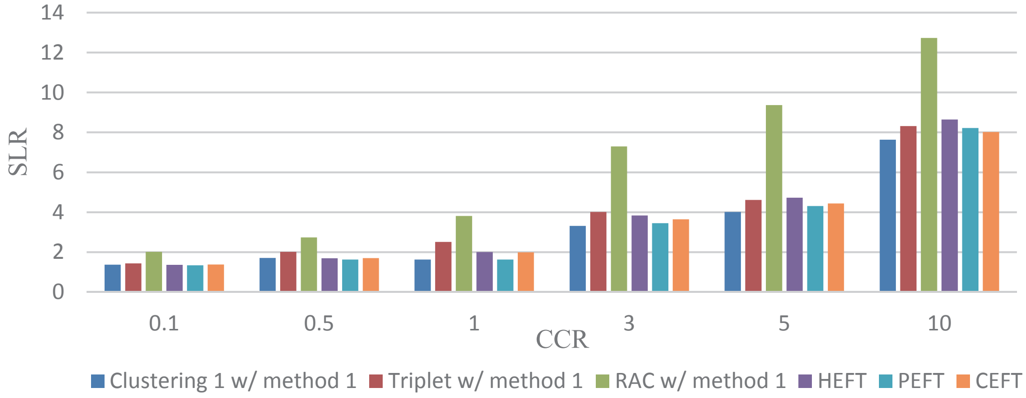


Fig. 6. Comparison of SLR among clustering methods for random DAG

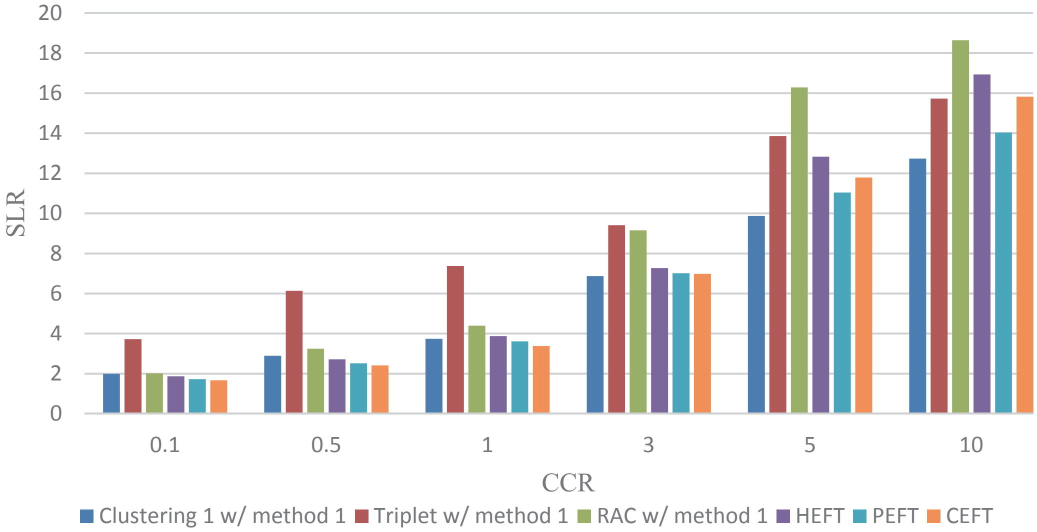


Fig. 7. Comparison of SLR among clustering methods for Gaussian elimination

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) doesn't make big difference in terms of SLR in case that 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 that CCR is equal to and larger than 3. That is, bigger data-transferring time makes bigger data-waiting time at each processer 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 work-flow type jobs handling massive data.

XI. CONCLUSION

In this paper, we proposed the task scheduling method after completion of task clustering in heterogeneous system. At 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 Schedule Length Ratio (SLR) can be made smaller than that of several existing method, 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 get smaller SLR than that of well-known list scheduling method such as HEFT, CEFT and PEFT, if CCR is equal to or bigger 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, et el., "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," IEEE *Trans. on Parallel and Distributed Systems*, Vol. 13, No. 3., pp. 260-274,2002.
2. H. Arabnejad, et.el, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," IEEE *Trans. on Parallel and distributed systems*, vol. 25, No. 3, March 2014.
3. M. A. Khan, "Schedule for heterogeneous systems using constrained critical paths," Parallel *Computing*, Vol 38, pp 175-193, 2012.
4. B. Jedari and M. Dehghan, "Efficient DAG Scheduling with Resource-Aware Clustering for Heterogeneous Systems," *Computer and Information Science,* Springer, Vol. 208, pp.249-261, 2009.
5. S. Chingchit. M. Kumar, and L. N. Bhuyan, "A Flexible Clustering and Scheduling Scheme for efficient Parallel Computation, " *Proc. of the 13th International and 10th Symposium on Parallel and Distributed Processing,* pp. 500-505, 1999.
6. C. Boeres, J. V. Filho, and V. E. F. Rebello, "A Cluster-based Strategy for Scheduling task on Heterogeneous Processors," *Proc. of the 16th Symposium on Computer Architecture and High performance Computing (SBAC-PAD'04),* pp. 214-221, 2004.
7. B. Cirou and E. Jeannot, "Triplet: a Clustering Scheduling Algorithm for Heterogeneous Systems," *Proc. of 2001 International Conference on Parallel Workshops (ICPPW'01),* pp. 231-236, 2001.
8. S. G. Ahmad et al., "Data-Intensive Workflow Optimization based on Application Task Graph Partitioning in Heterogeneous Computing Systems," *Proc. of IEEE 4th International Conference on Big Data and Cloud Computing*, pp. 129-136, 2014.
9. Aida A. Nasr et al., "Task Scheduling Algorithm for High Performance Heterogeneous Distributed Computing Systems," *International Journal of Computer Applications (0975 – 8887), Volume 110 – No. 16,* pp. 23-29, Jan' 2015
10. H. Kanemitsu et al., "A Processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System," *Journal of*

Computing, Vol. 3, Issue 11, pp. 1-8, 2011.

11. H. Kanemitsu et al., "On the Effect of Applying the Task Clustering for Identical processor Utilization to Heterogeneous Systems," Grid Computing – Technologies and Applications, *Widespread Converge and new Horizon*, Intech (ISBN: 978-953-51-0604-3), pp. 29 – 46, March, 2012.
12. W, Aheng and L. Tang, "A Priority-Based Scheduling Heuristic to Maximize Parallelism of Ready Tasks for DAG Applications," *Proc. of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid2015)*, pp. 596-605, 2015.
13. O, Sinnen "Task scheduling for Parallel systems," Willey, 2007.
14. 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, pp. 276-291, 1992.
15. V. Sarkar, "Partitioning and Scheduling Parallel programs for Execution on Multiprocessors," *Cambridge, MA: MIT Press,* 1989.
16. M. Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. on Parallel and Distributed Systems,* Vol. 1, No. 3, pp. 330-343, 1990.
17. T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," IEEE *Trans. on Parallel and Distributed Systems,"* Vol. 5, No. 9, pp. 951-967, 1994.