

A Task Cluster Size Determination in Heterogeneous Distributed Systems

Hidehiro Kanemitsu

Graduate School of Global Information and
Telecommunication Studies, Waseda University
Email: kanemih@ruri.waseda.jp

Abstract—There are many task scheduling algorithms for minimizing the schedule length in distributed systems. Especially, task clustering heuristics are well known as an approach for deciding the number of required processors. However, the actual number may become very large depending on characteristics of the input job and each processor's capability. Thus, currently there is no theoretical criterion for achieving processor utilization.

In this paper, we propose a method for deriving the lower bound of the cluster (assignment unit) size in heterogeneous distributed systems for limiting the number of processors. From results of experimental simulations, we discuss the applicability of the proposal to obtain better processor utilization.

Keywords—Task Scheduling, DAG, Heterogeneous Distributed Systems, Processor Utilization

I. INTRODUCTION

Actual task execution models over the networked processors, e.g., cluster, grid and utility computing have been studied and developed for maximizing the system throughput by utilizing computational resources. One of major trends in task execution types is to divide the required data into several pieces and then distribute them to workers like “master-worker model”. In contrast to such a data intensive job, how to divide a computational intensive job into several execution units for parallel execution is under discussion from theoretical points of view. If we take task parallelization into account in a grid environment such as a computational grid environment, an effective task scheduling strategy should be established. In the light of combining task scheduling concepts and grid computing methodologies, heterogeneity with respect to processing power, communication bandwidth and so on should be incorporated into a task scheduling strategy. If we assume the situation where multiple jobs are being submitted in the unknown number of computational resources over the Internet, objective functions can be considered as follows: (i) Minimization of the schedule length (the time duration from per each job, (ii) Minimization of the completion time of the last job, (iii) Maximization of the degree of contribution to the total speed up ratio for each computational resources. As one solution for those three objective functions, in the literature[4] we proposed a method for minimizing the schedule length per one job with a small number of computational resources (processors) for a set of identical processors. The objective of the method is “utilization of computational resources”. The method is based on “task clustering” [1], in which tasks are

merged into one “cluster” as an execution unit for one processor. As a result, several clusters are generated and then each of which becomes one assignment unit. The method proposes to impose the lower bound for every cluster size to limit the number of processors. Then the literature theoretically showed the near-optimal lower bound to minimize the schedule length.

However, which processor should be assigned to a cluster is not discussed because the proposal assumes identical processors. If we use one of conventional cluster assignment methods such as CHP[6], Triplet[7], and FCS[8], almost all processors may be assigned to clusters because they try to achieve the maximum task parallelism to obtain the minimized schedule length. Thus, the third objective function may not be achieved by those cluster assignment strategies.

In this paper, we propose a method for deriving the lower bound of the cluster size in heterogeneous distributed systems and a task clustering algorithm. From results of experimental simulations, we discuss the applicability of the proposal to obtain better processor utilization.

The remainder of this paper is organized as follows. Sec. II presents our assumed model, then the lower bound of the cluster size is derived in sec. III. Sec. IV presents a task clustering algorithm which adopts the lower bound shown in sec. III. Experimental results are shown in sec. V, and finally we present conclusion and future works in sec. VI.

II. ASSUMED MODEL

A. Job Model

We assume a job to be executed among distributed processor elements (PEs) is a Directed Acyclic Graph (DAG), which is one of task graphs. Let $G_{cls}^s = (V_s, E_s, V_{cls}^s)$ be the DAG, where s is the number of task merging steps (described in the latter half part in this subsection), V_s is the set of tasks after s task merging steps, E_s is the set of edges (data communications among tasks) after s task merging steps, and V_{cls}^s is the set of clusters which consists of one or more tasks after s task merging steps. An i -th task is denoted as n_i^s . Let $w(n_i^s)$ be a size of n_i^s , i.e., $w(n_i^s)$ is the sum of unit times taken for being processed by the reference processor element. We define data dependency and direction of data transfer from n_i^s to n_j^s as $e_{i,j}^s$. And $c(e_{i,j}^s)$ is the sum of unit times taken for transferring data from n_i^s to n_j^s over the reference communication link.

One constraint imposed by a DAG is that a task can not be started execution until all data from its predecessor tasks arrive. For instance, $e_{i,j}^s$ means that n_j^s can not be started until data from n_i^s arrives at the processor which will execute n_j^s . And let $pred(n_i^s)$ be the set of immediate predecessors of n_i^s , and $suc(n_i^s)$ be the set of immediate successors of n_i^s . If $pred(n_i^s) = \emptyset$, n_i^s is called START task, and if $suc(n_i^s) = \emptyset$, n_i^s is called END task. If there are one or more paths from n_i^s to n_j^s , we denote such a relation as $n_i^s \prec n_j^s$.

B. Task Clustering

We denote the i -th cluster in V_{cls}^s as $cls_s(i)$. If n_k^s is included in $cls_s(i)$ by “the $s+1$ th task merging”, we formulate one task merging as $cls_{s+1}(i) \leftarrow cls_s(i) \cup \{n_k^s\}$. If any two tasks, i.e., n_i^s and n_j^s , are included in the same cluster, they are assigned to the same processor. Then the communication between n_i^s and n_j^s is localized, so that we define $c(e_{i,j}^s)$ becomes zero. Task clustering is a set of task merging steps, that is finished when certain criteria have been satisfied.

Throughout this paper, we denote that $cls_s(i)$ is “linear” if and only if $cls_s(i)$ contains no independent task[11]. Note that if one cluster is linear, at least one path among any two tasks in the cluster exists and task execution order is unique.

C. System Model

We assume that each PE is completely connected to other PEs, with not identical processing speeds and communication bandwidths. The set of PEs is expressed as $P = \{P_1, P_2, \dots, P_m\}$, and let the set of processing speeds as $alpha$, i.e., $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$. Let the set of communication bandwidths as β , i.e.,

$$\beta = \begin{pmatrix} \infty & \beta_{1,2} & \beta_{1,3} & \dots & \beta_{1,m} \\ \beta_{2,1} & \infty & \beta_{2,3} & \dots & \beta_{2,m} \\ \beta_{3,1} & \beta_{3,2} & \infty & \dots & \beta_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_{m,1} & \beta_{m,2} & \beta_{m,3} & \dots & \infty \end{pmatrix} \quad (1)$$

$\beta_{i,j}$ means the bandwidth from P_i to P_j . The processing time in the case that n_k^s is processed on P_i is expressed as $t_p(n_k^s, \alpha_i) = w(n_k^s)/\alpha_i$. The data transfer time of $e_{k,l}^s$ over $\beta_{i,j}$ is $t_c(e_{i,j}^s, \beta_{k,l}) = c(e_{i,j}^s)/\beta_{k,l}$. This means that both processing time and data transfer time are not changed with time, and suppose that data transfer time within one PE is negligible.

III. PROCESSOR UTILIZATION

A. The Indicative Value for the Schedule Length

The schedule length depends on many factors, i.e., execution time for each task, communication time for each data exchanged among tasks, execution order after the task scheduling, processing speed, and bandwidth. Furthermore, whether a data transfer time can be localized or not depends on the cluster structure. The proposed method is that a cluster is generated after the lower bound of the cluster size (the total execution time of every task included in the cluster) has been derived. The lower bound is decided when the indicative

value for the schedule length is minimized. In this paper, the indicative value is defined as $sl_w(G_{cls}^s, \phi_s)$, that means the indicative value for the schedule length after s task merging steps and ϕ_s is the set of mapping between PEs and clusters after s task merging steps. $sl_w(G_{cls}^s, \phi_s)$ is the maximum value of the execution path length which includes both task execution time and data transfer time, provided that each task is scheduled as late as possible and every data from its immediate predecessors has been arrived before the scheduled time (its start time). Table I shows notations and definitions for deriving $sl_w(G_{cls}^s, \phi_s)$. In the table, assigned PEs for $cls_s(i)$ and $cls_s(j)$ are P_p and P_q , respectively. And suppose $n_k^s \in cls_s(i)$, $n_l^s \in cls_s(j)$. In table I, especially $S(n_k^s, i)$ means the degree of increase of execution time by independent tasks for n_k^s . Therefore, the smaller $S(n_k^s, i)$, the earlier n_k^s can be scheduled. The task n_k^s which dominates $sl_w(G_{cls}^s, \phi_s)$ (In the case of $sl_w(G_{cls}^s, \phi_s) = level(n_k^s)$) means that the schedule length may be maximized if n_k^s is scheduled as late as possible.

B. Relationship between $sl_w(G_{cls}^s, \phi_s)$ and the Schedule Length

Our objective is to minimize $sl_w(G_{cls}^s, \phi_s)$ with maintaining the certain size of each cluster for processor utilization. The schedule length can not be known before scheduling every task, we must estimate it by using $sl_w(G_{cls}^s, \phi_s)$. Thus, it must be proved that $sl_w(G_{cls}^s, \phi_s)$ can effect on the schedule length. In this section, we show that minimizing $sl_w(G_{cls}^s, \phi_s)$ leads to minimizing the schedule length to some extent. In this section we present that relationship between Table II shows notations for showing characteristics of $sl_w(G_{cls}^s, \phi_s)$. In an identical processor system, provided that every processor speed and communication bandwidth are 1, no processor assignment policy is needed. Thus, let $sl_w(G_{cls}^s, \phi_s)$ in an identical processor system as $sl_w(G_{cls}^s)$. In the literature [4], it is proved that minimizing $sl_w(G_{cls}^s)$ leads to minimizing the lower bound of the schedule length as follows.

Lemma 1: In an identical processor system, let $\Delta sl_{w,up}^{s-1}$ which satisfies $sl_w(G_{cls}^s) - cp \leq \Delta sl_{w,up}^{s-1}$ and be derived before s task merging steps. Then we obtain

$$sl(G_{cls}^s) \geq \frac{sl_w(G_{cls}^s) - \Delta sl_{w,up}^{s-1}}{1 + \frac{1}{g_{min}}}, \quad (2)$$

where cp and g_{min} are defined in table II, and $sl(G_{cls}^s)$ is the schedule length after s task merging steps. ■

As for $\Delta sl_{w,up}^{s-1}$, it is defined in the literature [4]. Furthermore, according to the literature [5], it can be proved that the upper bound of the schedule length can be reduced by reducing $sl_w(G_{cls}^s)$ by the following lemma.

Lemma 2: In an identical processor system, if $sl(G_{cls}^s) \leq cp$, then we obtain

$$sl(G_{cls}^s) \leq sl_w(G_{cls}^s) + \max_{p \in G_{cls}^s} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, \\ n_k^0 \in pred(n_l^0)}} c(e_{k,l}^0) \right\}. \quad \blacksquare \quad (3)$$

TABLE I
PARAMETER DEFINITION RELATED TO $sl_w(G_{cls}^s)$ (HERE $n_k^s \in cls_s(i)$).

Parameter	Definition
$top_s(i)$	$\{n_k^s \forall n_l^s \in pred(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup \{START\ Tasks \in cls_s(i)\}$.
$in_s(i)$	$\{n_k^s \exists n_l^s \in pred(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup \{START\ Tasks \in cls_s(i)\}$.
$out_s(i)$	$\{n_k^s \exists n_l^s \in suc(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup \{END\ Tasks \in cls_s(i)\}$.
$btm_s(i)$	$\{n_k^s \forall n_l^s \in suc(n_k^s), s.t., n_l^s \notin cls_s(i)\} \cup \{END\ Tasks \in cls_s(i)\}$.
$desc(n_k^s, i)$	$\{n_l^s n_k^s < n_l^s, n_l^s \in cls_s(i)\} \cup \{n_k^s\}$
$S(n_k^s, i)$	$\sum_{n_l^s \in cls_s(i)} t_p(n_l^s, \alpha_p) - \sum_{n_l^s \in desc(n_k^s, i)} t_p(n_l^s, \alpha_p)$
$tlevel(n_k^s)$	$\begin{cases} \max_{n_l^s \in pred(n_k^s)} \{tlevel(n_l^s) + t_p(n_l^s, \alpha_p) + t_c(e_{l,k}, \beta_{q,p})\}, & \text{if } n_k^s \in top_s(i). \\ TL_s(i) + S(n_k^s, i), & \text{otherwise.} \end{cases}$
$TL_s(i)$	$\max_{n_k^s \in top_s(i)} \{tlevel(n_k^s)\}$
$blevel(n_k^s)$	$\max_{n_l^s \in suc(n_k^s)} \{t_p(n_k^s, \alpha_p) + t_c(e_{k,l}, \beta_{p,q}) + blevel(n_l^s)\}$
$level(n_k^s)$	$tlevel(n_k^s) + blevel(n_k^s)$
$BL_s(i)$	$\max_{n_k^s \in out_s(i)} \{S(n_k^s, i) + blevel(n_k^s)\}$
$LV_s(i)$	$TL_s(i) + BL_s(i) = \max_{n_k^s \in cls_s(i)} \{level(n_k^s)\}$
ϕ_s	$\{\dots, < cls_s(i), P_p >, \dots\}$
$sl_w(G_{cls}^s, \phi_s)$	$\max_{cls_s(i) \in V_{cls}^s} \{LV_s(i)\}$

In the initial state of the DAG, assume that each task is assigned to “the fastest and the widest processor”, where its processing speed and bandwidth are maximum in α and β , respectively. Let such a mapping as ϕ_0 . Then we can assume the initial state is identical processor environment by ϕ_0 . Therefore, if and only if the initial state, lemma 1 and 2 are hold in ϕ_0 .

C. The Lower Bound of each Cluster Size

In this section we present how large each cluster size should be. According to the literature [4], the lower bound of each cluster size (δ_{opt}) in an identical processor system is derived as follows.

$$\delta_{opt} = \sqrt{cp_w \max_{n_k^0 \in V_0} \left\{ \frac{\max_{n_l^0 \in V_{cls}^0} \{w(n_l^0)\}}{g_{max}} \right\}}. \quad (4)$$

Eq. (4) is the lower bound of each cluster size when $sl_w(G_{cls}^R)$ can be minimized, provided that every cluster size is above a certain threshold, δ . And R corresponds to the number of merging steps when every cluster size is δ_{opt} or more. If taking the initial state (ϕ_0) of the DAG in a heterogeneous system into account, δ_{opt} is expressed by $\delta_{opt}(\phi_0)$ as follows.

$$\begin{aligned} & \delta_{opt}(\phi_0) \\ = & \sqrt{cp_w(\phi_0) \max_{n_l^0 \in V_0} \left\{ \frac{\max_{n_l^0 \in V_0} \left\{ t_p(n_l^0, \max_{\alpha_i \in \alpha} \{\alpha_i\}) \right\}}{g_{max}(\phi_0)} \right\}} \end{aligned} \quad (5)$$

By imposing $\delta_{opt}(\phi_0)$, it can be said that at least $sl_w(G_{cls}^0, \phi_0)$ can be minimized. However, for $s \geq 1$ $sl_w(G_{cls}^s, \phi_s)$ can not always be minimized by $\delta_{opt}(\phi_0)$, because the mapping of each cluster and each processor is changed and then

$sl_w(G_{cls}^s, \phi_s)$ is not equal to $sl_w(G_{cls}^0, \phi_0)$. In this paper, one heuristic of our method is to impose the same lower bound ($\delta_{opt}(\phi_0)$) for every cluster which will be generated by the task clustering.

IV. TASK CLUSTERING ALGORITHM

A. Overview of the Algorithm

In the previous section, we presented how large each cluster size should be set for processor utilization. In this section, we present the task clustering algorithm with incorporating the following two requirements.

- 1) Every cluster size is $\delta_{opt}(\phi_0)$ or more.
- 2) Minimize $sl_w(G_{cls}^R, \phi_R)$, where R is the total number of merging steps until the first requirement is satisfied.

Fig. 1 shows the task clustering algorithm. At first, the mapping ϕ_0 is applied to every task. Then $\delta_{opt}(\phi_0)$ is derived. Before the main procedures, two sets are defined, i.e., UEX_s and RDY_s . UEX_s is the set of clusters whose size is smaller than $\delta_{opt}(\phi_0)$, and RDY_s is defined as follow.

$$\begin{aligned} RDY_s = & \{cls_s(r) | cls_s(r) \in UEX_s, pred(n_{r'}^s) = \emptyset, cls_s(r) = \{n_{r'}^s\}\} \\ & \cup \left\{ \begin{array}{l} cls_s(r) | cls_s(r) \in UEX_s, cls_s(q) \notin UEX_s, \\ n_{q'}^s \in cls_s(q), n_{q'}^s \in pred(n_{r'}^s) \text{ for } \forall n_{r'}^s \in top_s(r) \end{array} \right\}. \end{aligned}$$

RDY_s is the set of clusters whose preceding cluster sizes are $\delta_{opt}(\phi_0)$ or more. That is, the algorithm tries to merge each cluster in top-to-bottom manner.

The algorithm is proceeded during $UEX_s \neq \emptyset$, which implies that at least one cluster in UEX_s exists. At line 3, one processor is selected by a processor selection method, e.g., by CHP[6] (In this paper, we do not present processor selection methods). At line 4, one cluster is selected as $pivot_s$, which corresponds to “the first cluster for merging”. Once the $pivot_s$ is selected, “the second cluster for merging”, i.e., $target_s$ is needed. Thus, during line 5 to 7, procedures for selecting

TABLE II
PARAMETER DEFINITIONS WHICH ARE USED IN ANALYSIS ON $sl_w(G_{cls}^s, \phi_s)$.

Parameter	Definition
p	One path of G_{cls}^0 , i.e., $\{n_0^0, n_1^0, n_2^0, \dots, n_k^0\} \cup \{e_{0,1}^0, e_{1,2}^0, \dots, e_{k-1,k}^0\}$, by which a sequence $\langle n_0^0, n_1^0, n_2^0, \dots, n_k^0 \rangle$ is constructed, where $e_{l-1,l}^0 \in E_0$, n_0^0 is a START task and n_k^0 is an END task.
cp	$\max_p \left\{ \sum_{n_k^0 \in p} w(n_k^s) + \sum_{e_{k,l}^s \in p} c(e_{k,l}^s) \right\}$.
$cp(\phi_s)$	$\max_p \left\{ \sum_{n_k^s \in p} t_p(n_k^s, \alpha_p) + \sum_{e_{k,l}^s \in p} t_c(c(e_{k,l}^s), \beta_p, q) \right\}$, where n_k^s, n_l^s are assigned to P_p, P_q .
cp_w	$\max_{p \in G_{cls}^0} \left\{ \sum_{n_k^0 \in p} w(n_k^0) \right\}$.
$cp_w(\phi_s)$	$\max_{p \in G_{cls}^s} \left\{ \sum_{n_k^s \in p} t_p(n_k^s, \alpha_p) \right\}$.
g_{min}	$\min_{n_k^0 \in V_{cls}^0} \left\{ \frac{\min_{n_j^0 \in pred(n_k^0)} \{w(n_j^0)\}}{\max_{n_j^0 \in pred(n_k^0)} \{c(e_{j,k}^0)\}}, \frac{\min_{n_l^0 \in suc(n_k^0)} \{w(n_l^0)\}}{\max_{n_l^0 \in suc(n_k^0)} \{c(e_{k,l}^0)\}} \right\}$.
g_{max}	$\max_{n_k^0 \in V_{cls}^0} \left\{ \frac{\max_{n_j^0 \in pred(n_k^0)} \{w(n_j^0)\}}{\min_{n_j^0 \in pred(n_k^0)} \{c(e_{j,k}^0)\}}, \frac{\max_{n_l^0 \in suc(n_k^0)} \{w(n_l^0)\}}{\min_{n_l^0 \in suc(n_k^0)} \{c(e_{k,l}^0)\}} \right\}$.

$target_s$ and merging $pivot_s$ and $target_s$ are performed. After those procedures, at line 7 RDY_s is updated to become RDY_{s+1} , and $pivot_s$ is also updated to become $pivot_{s+1}$. Procedures at line 6 and 7 are repeated until the size of $pivot_s$ is $\delta_{opt}(\phi_0)$ or more. The algorithm in fig. 1 has common parts with that of the literature [4], i.e., both algorithms use $pivot_s$ and $target_s$ for merging two clusters until the size of $pivot_s$ exceeds a lower bound of the cluster size. However, one difference among them is that the algorithm in fig. 1 keeps the same $pivot_s$ during merging steps until its size exceeds $\delta_{opt}(\phi_0)$, while the algorithm in [4] selects the new $pivot_s$ in every merging step. The reason of keeping the same $pivot_s$ is to reduce the time complexity in selection for $pivot_s$, which requires scanning every cluster in RDY_s . As a result, the number of scanning RDY_s can be reduced with compared to that of [4].

INPUT: G_{cls}^0
OUTPUT: G_{cls}^R
Set the mapping ϕ_0 to the input DAG.
Define UEX_s as a set of clusters whose size is under $\delta_{opt}(\phi_0)$;
Define RDY_s as a set of clusters which statisies eq. (6).;
For each $n_k \in V$, let $n_k^0 \leftarrow n_k, cls_0(k) = \{n_k^0\}$ and put $cls_0(k)$ into V_{cls}^0 .
0. Derive $\delta_{opt}(\phi_0)$ by eq. (5).
1. $UEX_0 \leftarrow V_{cls}^0, RDY_0 \leftarrow \{cls_0(k) | cls_0(k) = \{n_k^0\}, pred(n_k^0) = \emptyset\}$;
2. **WHILE** $UEX_s \neq \emptyset$ **DO**
3. select a processor P_p from P ;
4. $pivot_s \leftarrow getPivot(RDY_s)$;
5. **WHILE** size of $pivot_s < \delta_{opt}(P_p)$ **DO**
6. $target_s \leftarrow getTarget(pivot_s)$;
7. $RDY_{s+1} \leftarrow merging(pivot_s, target_s)$;
8. **ENDWHILE**
9. **ENDWHILE**
10. **RETURN** G_{cls}^R ;

Fig. 1. Procedures for the Task Clustering.

B. Selection for $pivot_s$ and $target_s$

As mentioned in IV-A, one objective of the algorithm is to minimize $sl_w(G_{cls}^R, \phi_R)$. Therefore, in RDY_s , $pivot_s$ should have maximum LV value (defined in table I), because such a cluster may dominate $sl_w(G_{cls}^s, \phi_s)$ and then $sl_w(G_{cls}^{s+1}, \phi_{s+1})$ after $s+1$ th merging may became lower than $sl_w(G_{cls}^s, \phi_s)$. Our heuristic behind the algorithm is that this policy for selecting $pivot_s$ can contribute to minimize $sl_w(G_{cls}^R, \phi_R)$. The same requirement holds to the selection of $target_s$, i.e., $target_s$ should be the cluster which dominates LV value of $pivot_s$.

C. Merging $pivot_s$ and $target_s$

After $pivot_s$ and $target_s$ have been selected, the merging procedure, i.e.,

$$pivot_{s+1} \leftarrow pivot_s \cup target_s \quad (6)$$

is performed. This procedure means that every task in $target_s$ is included in $pivot_{s+1}$. Then $pivot_s$ and $target_s$ are removed from UEX_s and RDY_s . After this merging step has been performed, clusters satisfying requirements for RDY_{s+1} (in eq. (6)) are included in RDY_{s+1} . Furthermore, every cluster's LV values are updated for selecting $pivot_{s+1}$ or $target_{s+1}$.

V. EXPERIMENTS

We conducted the experimental simulation to confirm advantages of our proposal. Thus, we compared with other conventional methods in terms of the following points of view.

- 1) Whether minimizing $sl_w(G_{cls}^R, \phi_R)$ leads to minimizing the schedule length or not.
- 2) The range of applicability by imposing $\delta_{opt}(\phi_0)$ as the lower bound of every cluster size.

No.	α	β	CCR	$ V_{cls}^R $	$sl_w(G_{cls}^R, \phi_R)$ Ratio			$sl(G_{cls}^R, \phi_R)$ Ratio		
					A	B	C	A	B	C
1	5	5	0.1	168	1.000	1.054	1.097	1.000	1.018	1.123
2			1.0	56	1.000	1.241	1.391	1.000	1.131	1.209
3			5.0	34	1.000	1.320	1.514	1.000	1.140	1.288
4			10.0	19	1.000	1.378	1.611	1.000	1.219	1.341
5	5	10	0.1	160	1.000	1.072	1.105	1.000	1.012	1.073
6			1.0	49	1.000	1.203	1.419	1.000	1.144	1.248
7			5.0	30	1.000	1.355	1.428	1.000	1.172	1.301
8			10.0	16	1.000	1.329	1.503	1.000	1.237	1.344
9	10	5	0.1	150	1.000	1.032	1.066	1.000	1.016	1.071
10			1.0	47	1.000	1.177	1.284	1.000	1.097	1.198
11			5.0	41	1.000	1.209	1.615	1.000	1.173	1.291
12			10.0	26	1.000	1.482	1.598	1.000	1.227	1.302
13	10	10	0.1	187	1.000	1.069	1.044	1.000	1.044	1.050
14			1.0	67	1.000	1.292	1.157	1.000	1.179	1.132
15			5.0	44	1.000	1.344	1.419	1.000	1.203	1.297
16			10.0	28	1.000	1.461	1.433	1.000	1.272	1.301

TABLE III
COMPARISON OF $sl_w(G_{cls}^R, \phi_R)$ AND THE SCHEDULE LENGTH IN RANDOM DAGs($|V_0| = 1000$).

We showed by theorem 1 and 1 that both the lower bound and the upper bound of the schedule length in an identical processor system can be expressed by $sl_w(G_{cls}^s)$. There is no way to confirm that minimizing $sl_w(G_{cls}^s, \phi_s)$ can lead to minimizing the schedule length in a heterogeneous distributed system. Thus, in this experiment we confirm that the actual relationship between the schedule length and $sl_w(G_{cls}^s, \phi_s)$.

A. Experimental Environment

In the simulation, a random DAG is generated. In the DAG, each task size and each data size are decided randomly. Also CCR (Communication to Computation Ratio)[9], [10] is changed from 0.1 to 10. The max to min ratio in terms of data size and task size is set to 100. Also we decided the Parallelism Factor (PF) is defined as ρ , taking values of 0.5, 1.0, and 2.0 [12]. By using PF, the depth of the DAG is defined as $\frac{\sqrt{|V_0|}}{\rho}$.

The simulation environment was developed by JRE1.6.0.0, the operating system is Windows XP SP3, the CPU architecture is Intel Core 2 Duo 2.66GHz, and the memory size is 2.0GB.

B. Comparison about $sl_w(G_{cls}^R, \phi_R)$ and the Schedule Length

In this experiment, we compared $sl_w(G_{cls}^R, \phi_R)$ and the schedule length to confirm the relationship between them. Comparison targets are as follows.

- At first, the lower bound of the cluster size is derived as $\delta_{opt}(\phi_0)$. Then the task clustering algorithm is performed, while processor assignment policy is based on CHP[6].
- The lower bound of the cluster size is derived as $\delta_{opt}(\phi_0)$. Then the task clustering policy is based on “load balancing” [13], while processor assignment policy is based on CHP[6], in which merging step for generating one cluster is proceeded until the cluster size exceeds $\delta_{opt}(\phi_0)$.
- The lower bound of the cluster size is derived as $\delta_{opt}(\phi_0)$. Then the task clustering policy is random-basis, i.e., two clusters smaller than $\delta_{opt}(\phi_0)$ are selected randomly to

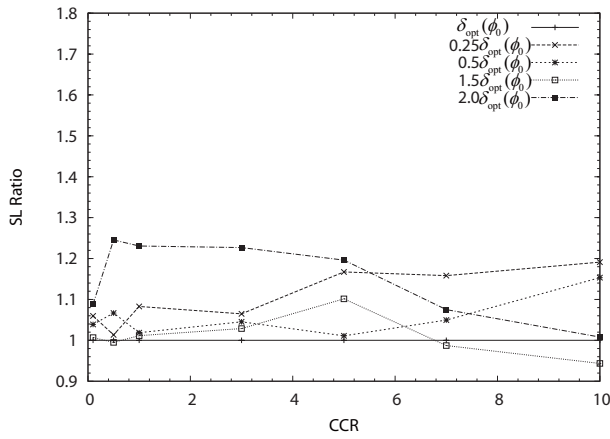
merge into one larger cluster, while processor assignment policy is based on CHP[6], in which merging step for generating one cluster is proceeded until the cluster size exceeds $\delta_{opt}(\phi_0)$.

The difference between A, B and C is how to merge clusters, while they have the common lower bound for the cluster size and the common processor assignment policy. We compared $sl_w(G_{cls}^R, \phi_R)$ and the schedule length by averaging them in 100 random DAGs.

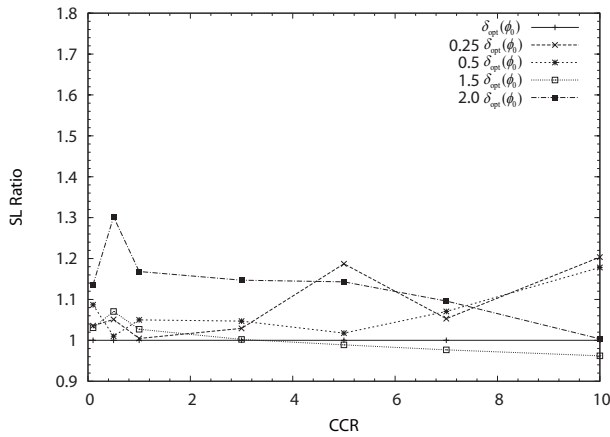
Table III shows comparison results in terms of $sl_w(G_{cls}^R, \phi_R)$ and the schedule length. The former is the result in the case of random DAGs. In both tables, α corresponds to max-min ratio for processing speed in P , and β corresponds to max-min ratio for communication bandwidth in P . “ $sl_w(G_{cls}^R, \phi_R)$ Ratio” and “ $sl(G_{cls}^R, \phi_R)$ Ratio” correspond to ratios to “A”, i.e., a value larger than 1 means that $sl_w(G_{cls}^R, \phi_R)$ or $sl(G_{cls}^R, \phi_R)$ is larger than that of “A”. In table III, it can be seen that both $sl_w(G_{cls}^R, \phi_R)$ and $sl(G_{cls}^R, \phi_R)$ in “A” are better than “B” and “C” as a whole. Especially, the larger CCR becomes, the better both $sl_w(G_{cls}^R, \phi_R)$ and $sl(G_{cls}^R, \phi_R)$ in “A” become. It can not be seen that noteworthy characteristics related to $sl_w(G_{cls}^R, \phi_R)$ and $sl(G_{cls}^R, \phi_R)$ with varying the degree of heterogeneity (i.e., α and β). From those results, it can be concluded that minimizing $sl_w(G_{cls}^R, \phi_R)$ leads to minimizing the schedule length.

C. Applicability of $\delta_{opt}(\phi_0)$

In this experiment, we confirmed that how optimal the lower bound of the cluster size, $\delta_{opt}(\phi_0)$ derived by eq. (5). Comparison targets in this experiment are based on “A” at sec. V-B, but only the lower bound of the cluster size is changed, i.e., $\delta_{opt}(\phi_0)$, $0.2\delta_{opt}(\phi_0)$, $0.5\delta_{opt}(\phi_0)$, $1.5\delta_{opt}(\phi_0)$, and $2.0\delta_{opt}(\phi_0)$. The objective of this experiment is to confirm the range of applicability of $\delta_{opt}(\phi_0)$, due to the fact that $\delta_{opt}(\phi_0)$ is not a value when $sl_w(G_{cls}^s, \phi_s)$ can be minimized for $1 \leq s$. Fig. 2 shows comparison results in terms of the



(a) $(\alpha, \beta) = (5, 5)$



(b) $(\alpha, \beta) = (10, 10)$

Fig. 2. Optimality for the Lower Bound of the Cluster Size.

optimality of $\delta_{opt}(\phi_0)$. (a) corresponds to the case of the degree of heterogeneity $(\alpha, \beta) = (5, 5)$, and (b) corresponds to $(10, 10)$. From (a), it can be seen that $\delta_{opt}(\phi_0)$ takes the best schedule length than other cases during CCR taken from 0.1 to 5.0. However, when CCR is 7 or more, $1.5\delta_{opt}(\phi_0)$ takes the best schedule length. This is because $\delta_{opt}(\phi_0)$ may be a small size for a data intensive DAG. Thus, it can be said that $1.5\delta_{opt}(\phi_0)$ is more appropriate size than $\delta_{opt}(\phi_0)$ when CCR exceeds a certain value. On the other hand, in (b), the larger CCR becomes, the better the schedule length by case of $1.5\delta_{opt}(\phi_0)$ becomes. However, during CCR is less than 3.0, $\delta_{opt}(\phi_0)$ can be the best lower bound of the cluster size.

From those results, it can be said that the lower bound for the cluster size should be derived according to the mapping state. For example, if the lower bound can be adjusted as a function of each assigned processor, better schedule length may be obtained. This point of view is an issue in the future works.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we presented a policy for deciding the assignment unit size to a processor and a task clustering for processor utilization in heterogeneous distributed systems. We defined the indicative value for the schedule length for heterogeneous distributed systems. Then we theoretically proved that minimizing the indicative value leads to minimization of the schedule length. Furthermore, we defined the lower bound of the cluster size by assuming the initial mapping state. From the experimental results, it is concluded that minimizing the indicative value has good effect on the schedule length. However, we found that the lower bound of the cluster size should be adjusted with taking an assigned processor's ability into account.

As a future work, we will study on how to adjust the lower bound of the cluster size for obtaining the better schedule length and more effective processor utilization.

REFERENCES

- [1] 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.
- [2] 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.
- [3] J. C. Liou, M. A. Palis., An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors, *Procs. of the 8th Symposium on Parallel and Distributed Processing*, October, 1996.
- [4] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, Static Task Cluster Size Determination in Homogeneous Distributed Systems, *Software Automatic Tuning: from concepts to state-of-the-art results*, Springer-Verlag (ISBN: 1441969349), pp. 229 – 252, 2010.
- [5] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, A Task Clustering for Utilizing Computational Resources, *IPSJ Trans. on Advanced Computing Systems*, Vol. 4, No. 1, pp. 111-146, 2011.
- [6] C. Boeres, J. V. Filho and V. E. F. Rebello, A Cluster-based Strategy for Scheduling Task on Heterogeneous Processors, *Procs. of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pp.214 – 221, 2004.
- [7] B. Cirou, E. Jeannot, Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems, *Procs. of 2001 International Conference on Parallel Processing Workshops (ICPPW'01)*, pp. 231 – 236, 2001.
- [8] S. Chingchit, M. Kumar and L.N. Bhuyan, A Flexible Clustering and Scheduling Scheme for Efficient Parallel Computation, *Procs. of the 13th International and 10th Symposium on Parallel and Distributed Processing*, pp. 500 – 505, 1999.
- [9] O. Sinnen and L. A. Sousa., Communication Contention in Task Scheduling, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 16, No. 6., pp. 503-515, 2005.
- [10] O. Sinnen., Task Scheduling for Parallel Systems, Wiley, 2007.
- [11] A. Gerasoulis and T. Yang., On the Granularity and Clustering of Directed Acyclic Task Graphs, *IEEE Trans. on Parallel And Distributed Systems*, Vol. 4, No. 6, June, 1993.
- [12] H. Topcuoglu, S. Hariri, M. Y. Wu. : 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.
- [13] J. C. Liou and M. A. Palis., A Comparison of General Approaches to Multiprocessor Scheduling, *Procs. of the 11th International Symposium on Parallel Processing*, pp. 152 – 156, 1997.