

Effective Use of Computational Resources in Multicore Distributed Systems

Hidehiro Kanemitsu
Media Network Center,

Waseda University
Email: kanemih@ruri.waseda.jp

Masaki Hanada

Department of Information Systems,
Tokyo University of Information Sciences
Email: mhanada@rsch.tuis.ac.jp

Takashige Hoshiai

Department of Computer and
Information Science, Sojo University
Email: hoshiai@cis.sojo-u.ac.jp

Hidenori Nakazato

Global Information and
Telecommunication Institute, Waseda University
Email: nakazato@waseda.jp

Abstract—In the last decades, many kinds of task execution models such as grid and cloud computing have been developed. In such distributed systems, each task is processed by respective processor in multicore computers e.g., household PCs which we can easily harness in recent years. If there is one policy to automatically decide the “best” combination and the number of processors (and computers), we effectively utilize those computational resources, thereby large number of jobs can be executed in parallel. In this paper, we propose a method for mapping of execution units for such environments. The method adopts a re-mapping technology after processor-execution unit mapping[6] is finished. Experimental comparisons by a simulation show the advantages of the proposed method.

Keywords—Task Clustering, Multicore, Distributed Systems, Parallel Computing

I. INTRODUCTION

Recently, task execution models are becoming diverse, e.g., grid computing[1], cloud computing. Such computational models often require parallel execution models such as parametric job execution, background data analysis for data mining, and searching for a target value in a key-value store[2]. Irrespective of dedicated computers or household computers, they should be effectively used when a parallel execution model is applied. We proposed a method for effective use of computational resources each of which has a single processor in a heterogeneous distributed system [6]. The method automatically derives set of mapping between each processor and each assignment unit (i.e., the set of tasks in a “DAG (Directed Acyclic Graph)” program such as a workflow type job), by which the degree of contribution for each processor toward the response time minimization is maximized. However, the method does not assume more realistic situations, e.g., each computer may have a parallel execution scheme such as multicore and multi OS by virtualization. Thus, our previously proposed method can not be applied to those environments. If a model which accommodates them, the response time can be minimized with a smaller number of computers. In

this paper, we propose a theoretical method for deciding the set of mapping between each computer and each assignment unit for effective use of computational resources, where each computer can have parallel execution scheme. The proposed method assumes that a computer with two or more processors as “two or more processors are completely connected over the network with infinite network bandwidth”. Then a lower bound of assignment unit size is mathematically decided for each processor for limiting the number of processors. Actual assignment units are generated by a task clustering algorithm until each size exceeds the lower bound. Among processors which belong to the same computer, every task in assignment units is locally scheduled without degrading task parallelism. Consequently, the required number of computer can be smaller than that of the method proposed in [6]. Experimental comparison is conducted by a simulation program. In the simulation, we evaluate the degree of contribution for each computer and show advantages of the proposed method over related works and the method we proposed in [6].

The remainder of this paper is organized as follows. Sec. II presents our assumed model. Then problem definitions and the objective of the proposal are presented in sec. III. The detailed content of the proposal is presented in sec. IV. Advantages of the proposal is shown by the experimental comparisons in sec. V. Finally, a conclusion and a future work are presented in sec. VI.

II. ASSUMED MODEL

A. Job Model

An assumed job is expressed as a Directed Acyclic Graph (DAG), which is known as one of task graph representations. 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

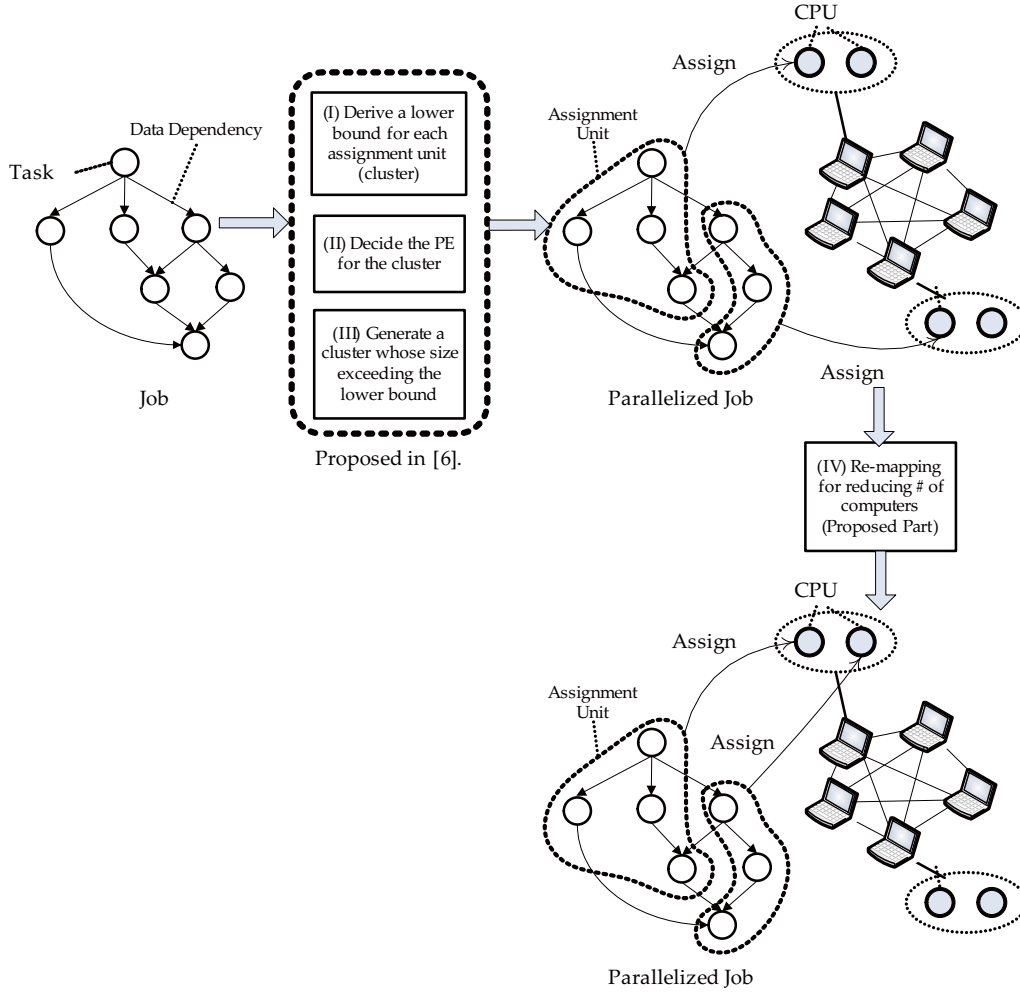


Fig. 1. Proposed part

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[3], [4], [5] is a set of task merging steps, that is finished when certain criteria have been satisfied.

C. System Model

We assume that each computer is completely connected to others, with not identical processing speeds and communication bandwidths. Each computer has one or more processing elements (PE). The set of computers is expressed as $M = \{M_1, M_2, \dots, M_m\}$. The set of PEs in M_i is expressed as $P_i = \{P_{i,1}, P_{i,2}, \dots, P_{i,|P_i|}\}$, and let the set of processing speeds in P_i as α_i , i.e., $\alpha_i = \{\alpha_{i,1}, \alpha_{i,2}, \dots, \alpha_{i,|P_i|}\}$, where $|P_i| = |\alpha_i|$.

As for data communication among computers, let the set of

communication bandwidths be 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 M_i to M_j . We assume that a communication bandwidth between two computers, e.g., M_i and M_j is the same between any one of P_i and any one of P_j . The processing time in the case that n_k^s is processed on $P_{i,p}$ is expressed as $t_p(n_k^s, \alpha_{i,p}) = w(n_k^s)/\alpha_{i,p}$. 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 computer is negligible.

III. PROBLEM DEFINITION

A. Processor Utilization in Singlecore Distributed systems

In this paper, we propose a method for processor utilization in a distributed system, where each computer has one or more PEs. The difference between the method proposed in [6] and a method in this paper is whether such multicore computers are considered or not. According to [6], “processor utilization”, in other words, “effective use of computational resources”, means to maximize the degree of contribution for reducing the schedule length for each PE. Fig. 1 shows a process flow we assume. A job is processed by three steps, i.e., (I) a lower bound for each cluster (set of tasks) execution time is derived, (II) mapping procedure to assign a PE to a cluster, (III) task clustering algorithm to generate actual clusters. Since these procedures were proposed in [6], we describe only its abstract.

1) *Lower Bound Derivation for each Cluster Execution Time*: According to fig. 1, the method proposed in [6] derives a lower bound for each cluster execution time (total task execution time in a cluster on a PE). To impose a lower bound, the number of required PEs is limited to some extent. At the same time, the response time should be minimized for achieving processor utilization.

Since the schedule length can not be derived until every task is scheduled, each lower bound should be decided with estimating the schedule length. This is because these three procedures in fig. 1 are performed before a task scheduling[8]. Thus, the method in [6] defines an indicative value for the schedule length as WSL (Worst Schedule Length) [6]. Let define “WSL after s -th task merging step” as $sl_w(G_{cls}^s, \phi_s)$, which means the largest value that the response time can take when every task is executed as late as possible after s task merging steps and the mapping state ϕ_s has been decided.

Fig. 2 shows one example of how WSL is decided. In this figure, (a) is the state after 4 task merging steps have been finished, and (b) is the state that each generated cluster is assigned to one actual PE. In (b), note that data communication time in the same PE is set to 0. Then we WSL is derived by tracing $n_1^5 \rightarrow n_3^5 \rightarrow n_5^5 \rightarrow n_2^5 \rightarrow n_7^5$, i.e., $sl_w(G_{cls}^6, \phi_6) = 25$.

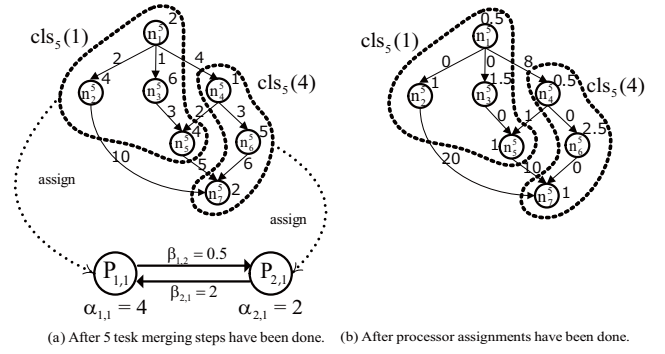


Fig. 2. Example of WSL

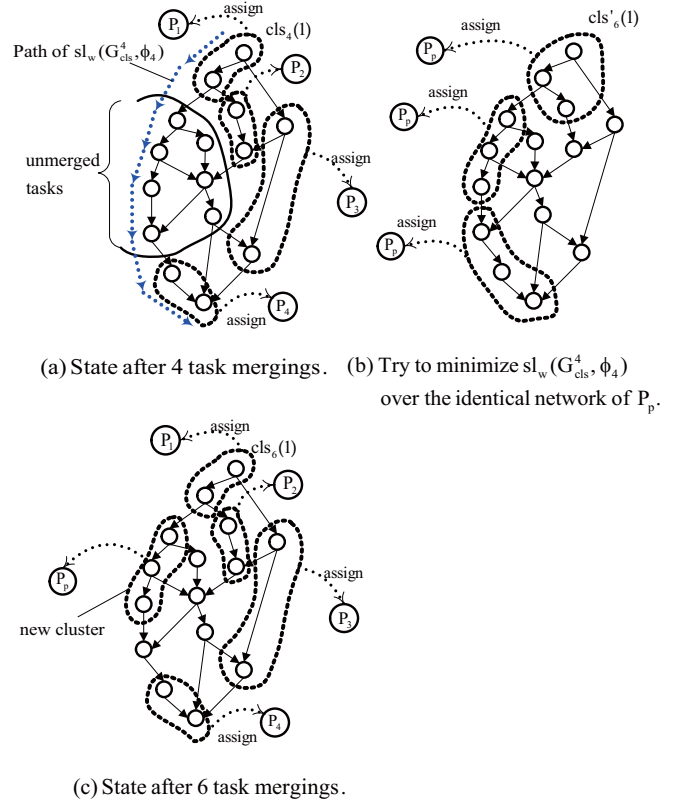


Fig. 3. Assumed model for WSL derivation

In [6], it is proved that minimizing WSL has good effect on the schedule length, i.e., it is necessary that WSL should be minimized to minimize the schedule length. Since actual WSL after s -th task merging step can not be decided until s task merging steps have been finished, we must estimate the upper bound of WSL after $s - 1$ task merging steps have been finished (i.e., before s -th task merging step). If we define an upper bound of the difference between WSL after s -th task merging step and the initial WSL (i.e., 0-th task merging step)

as $\Delta sl_w^{s-1}(\delta)$, we have

$$\Delta sl_w^{s-1}(\delta) = \left(\frac{\max_{n_k^0 \in V_0} \{w(n_k^0)\}}{\alpha_p} + \frac{\max_{e_{k,l}^0 \in E_0} \{c(e_{k,l}^0)\}}{\beta(p)} \right) \frac{\sum_{n_k^0 \in seq_{s-1}^<} w(n_k^0)}{\delta \alpha_p} + \delta - \frac{\max_{e_{k,l}^0 \in E_0} \{c(e_{k,l}^0)\}}{\max_{\beta_{p,q} \in \beta} \{\beta_{p,q}\}}, \quad (2)$$

where ϕ_0 is the initial mapping state that each task is assigned to a “virtual PE” having the maximum processing speed and the maximum communication bandwidth. At the mapping state ϕ_0 , the critical path length equals to the WSL at the initial state ϕ_0 (for more details, see [6]). At eq. (2), δ is the lower bound of each cluster execution time. δ is derived by temporarily assuming identical network, where we derive the lower bound for every cluster execution time on tasks dominating the WSL. $seq_{s-1}^<$ is a set of tasks on a path dominating $sl_w(G_{cls}^{s-1}, \phi_{s-1})$ (e.g., at (b) in fig. 2, $seq_5^<$ is $\{n_1^5, n_3^5, n_5^5, n_7^5\}$ or $\{n_1^5, n_2^5, n_7^5\}$).

Fig. 3 shows one example how δ is derived. In this figure, (a) is the state after 4 task merging steps have been finished. In (a) there are unmerged tasks each of which is assigned to the “virtual PE” having the maximum processing speed and communication bandwidth, respectively. On the other hand, other tasks are assigned to an actual PE. In this state, the dashed line corresponds to the path of WSL, i.e., $sl_w(G_{cls}^4, \phi_4)$. Then we temporarily assume these tasks will be clustered and each cluster is assigned to an identical PE (in (b), the PE is P_p). We have $\Delta sl_w^{s-1}(\delta)$, which is a function with respect to δ . From (b) we obtain the optimal value of δ (let call “ δ_{opt} ”) by which $\Delta sl_w^{s-1}(\delta)$ is minimized. Once δ_{opt} is obtained, every cluster in (b) is restored to clusters in (a) and then unmerged tasks in (a) are merged into one new cluster until its execution time exceeds δ_{opt} (see (c)). It is assumed that the cluster execution time of the new cluster in (c) exceeds δ_{opt} after 6 task merging steps (it was not enough after 5 task merging steps).

We obtain the minimum value of sl_w^{s-1} by differentiating eq. (2) with respect to δ . If we define the value of δ when sl_w^{s-1} takes the minimum as $\delta_{opt}^s(P_p)$, we have

$$\delta_{opt}^s(P_p) = \sqrt{\frac{\sum_{n_k^0 \in seq_{s-1}^<} w(n_k^0)}{\alpha_p} \left(\frac{\max_{n_k^0 \in V_0} \{w(n_k^0)\}}{\alpha_p} + \frac{\max_{e_{k,l}^0 \in E_0} \{c(e_{k,l}^0)\}}{\beta(p)} \right)},$$

where $\beta(p)$ is the minimum communication bandwidth of P_p . $\delta_{opt}^s(P_p)$ is the lower bound of the cluster execution time to be generated after s -th task merging step. If $\delta_{opt}^s(P_p)$ is applied to $\Delta sl_w^{s-1}(\delta)$, we have

INPUT: $\langle G_{cls}^R, \phi_R \rangle$, where R is # of task merging step required to obtain the output DAG of [6].

OUTPUT: $\langle G_{cls}^{R'}, \phi_{R'} \rangle$, where R' is R plus “# of the re-mapping required to obtain the output DAG of the proposal”.

Define $P(cls_R(k), \phi_R) = P_{i,p}$, where $cls_R(k)$ is assigned to $P_{i,p}$ at ϕ_R .

Define $M(cls_R(k), \phi_R) = M_i$, where $P(cls_R(k), \phi_R) = P_{i,p}$ is included in M_i .

Define $top_R(k)$ as the set of tasks in $cls_R(k)$ s.t., their all immediate predecessor tasks do not belong to $cls_R(k)$.

Define CHK is the set of clusters obtained by the method of [6].

```

1. FOR  $n_t^R \in seq_R^<$  s.t.,  $n_t^R \in cls_R(k)$ ,  $CHK \neq \emptyset$  DO
2.   IF  $cls_R(k) \in CHK$  THEN
3.     Set an immediate predecessor task of  $top_R(k)$  as  $n_u^R$ ,
      $n_u^R \in cls_R(l)$ ;
4.   FOR EACH  $cls_R(l)$  DO
5.      $\Delta(k, \alpha_{i,p}, \alpha_{j,q}) \leftarrow \varepsilon(k, \alpha_{i,p}, \alpha_{j,q}) - \omega(k, \alpha_{j,q})$ ;
6.   END FOR
7.   Find max of  $\Delta(k, \alpha_{i,p}, \alpha_{j,q})$  and move  $cls_R(k)$  to  $P_{j,q}$ ;
8.   Remove  $cls_R(l)$  from  $CHK$ ;
9. END IF
10. END FOR
```

Fig. 4. Procedures for re-mapping.

$$\Delta sl_w^{s-1}(\delta_{opt}^s(\alpha_p, \beta(p))) = \frac{2}{\alpha_p} \times \sqrt{\sum_{n_k^0 \in seq_{s-1}^<} w(n_k^0) \left(\frac{\alpha_p \max_{e_{k,l}^0 \in E_0} \{c(e_{k,l}^0)\}}{\beta(p)} + \max_{n_k^0 \in V_0} \{w(n_k^0)\} \right)} - \frac{\max_{e_{k,l}^0 \in E_0} \{c(e_{k,l}^0)\}}{\max_{\beta_{p,q} \in \beta} \{\beta_{p,q}\}}. \quad (3)$$

2) *Processor Selection:* At this phase, a PE to be assigned to a new cluster (e.g., “new cluster” at fig. 3) is decided. If $\Delta sl_w^{s-1}(\delta_{opt}^s(\alpha_p, \beta(p)))$ is minimized by applying actual values of α_p and $\beta(p)$, the PE having these values should be selected for processor assignment. Then the minimized $\Delta sl_w^{s-1}(\delta_{opt}^s(\alpha_p, \beta(p)))$ is obtained, by which the upper bound of WSL can be minimized.

3) *Task clustering:* Since the PE to be assigned has been decided, both values of $\delta_{opt}^s(P_p)$ and $\Delta sl_w^{s-1}(\delta_{opt}^s(\alpha_p, \beta(p)))$ are decided. The objective of the task clustering is to generate the cluster whose execution time exceeds $\delta_{opt}^s(P_p)$ and to minimize WSL, i.e., $sl_w(G_{cls}^s, \phi_s)$. One policy of the task clustering is to select *pivot* (a cluster in which at least one task belongs to $sl_w(G_{cls}^{s-1}, \phi_{s-1})$) and *target* (a cluster in which at least one task has data dependency with one of task in *pivot*). These two clusters are merged into a new cluster, and then if the cluster execution time of the cluster does not exceeds $\delta_{opt}^s(P_p)$, the cluster becomes the “new *pivot*”. Then the new *target* is selected for more merging steps.

By repeating (1) - (3), every cluster execution time becomes larger than each decided lower bound. In such an obtained output DAG, it was found that the degree of contribution for speed up can be maximized[6].

B. Objective of the Proposal

The method for processor utilization described above has good effect on a “single core distributed system”, where each

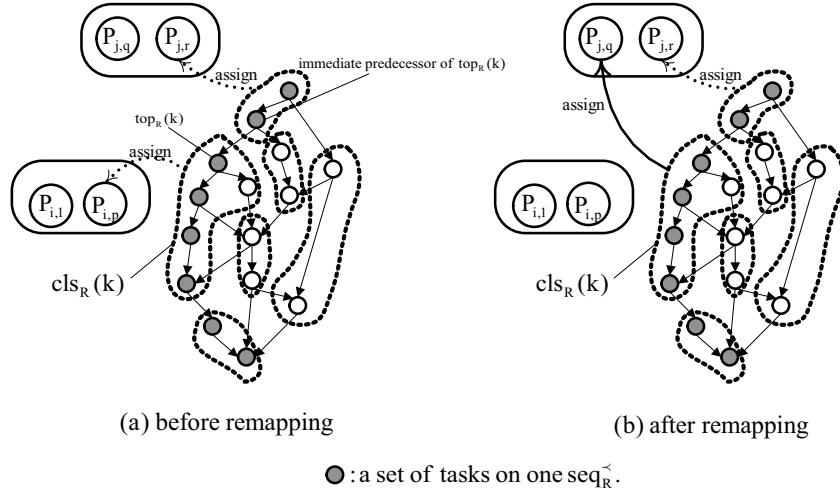


Fig. 5. Example of the cluster re-mapping algorithm

PE is completely connected over the network. However, the method does not take into account the locality among PEs. In fig. 1, a multicore distributed system is assumed. If the method proposed in [6] is applied in such an environment, each cluster (assignment unit) is assigned to different computers (see fig. 1). Thus, we propose a processor mapping policy taking into account the locality among PEs. In the top part of fig. 1, two clusters are assigned to different computers. This is because that the method proposed in [6] tries to find the PE to be assigned to a cluster by only applying its performance information to eq. (3). On the other hand, the bottom part of fig. 1 corresponds to the result by applying the proposal. In this case, both clusters are assigned to the same computer. The process of our proposal is performed after three steps ((I)-(III) in fig. 1).

IV. PROPOSAL

As stated in the previous section, the proposed method is performed after three steps proposed in [6] have finished. In this section, we present details in the proposal.

A. Cluster Selection

As stated above, it was proved that minimizing WSL can lead to reduction of the schedule length. Thus, at first clusters dominating WSL must be specified for “re-mapping”. This is because such a re-mapping can make WSL smaller than that obtained after three steps ((I)-(III) at fig. 1), thereby it is conceivable that the schedule length is also made smaller by the localization of the data communication.

Fig. 4 shows the cluster re-mapping algorithm, and fig. 5 shows one example of the algorithm with using the same notations as fig. 4. The algorithm firstly tries to find a cluster to be moved by tracing tasks in $seq_R^<$ (line 1 - 10 at fig. 4).

B. Cluster Remapping Procedure

It assumes that a cluster which will be checked is $cls_R(k)$ and $cls_R(k)$ is firstly assigned to $P_{i,p} \in M_i$. In such clusters,

the algorithm calculates the difference between $\varepsilon(k, \alpha_{i,p}, \alpha_{j,r})$ and $\omega(k, \alpha_{j,q})$. They are defined as follows.

$$\begin{aligned} \varepsilon(k, \alpha_{i,p}, \alpha_{j,r}) &= \frac{1}{\alpha_{i,p}} \sum_{n_u^R \in cls_R(k)} w(n_u^R) + \frac{1}{\beta_{q,p}} \sum_{\substack{n_x^R \in cls_R(k), \\ n_y^R \in cls_R(l)}} c(e_{y,x}) \\ &+ \frac{1}{\beta_{q,p}} \sum_{\substack{n_x^R \in cls_R(k), \\ n_y^R \in cls_R(l)}} c(e_{x,y}), \end{aligned} \quad (4)$$

$$\omega(k, \alpha_{j,q}) = \frac{1}{\alpha_{j,q}} \sum_{n_u^R \in cls_R(k)} w(n_u^R). \quad (5)$$

$\varepsilon(k, \alpha_{i,p}, \alpha_{j,r})$ is the sum of cluster execution time and data communication time (incoming communication and outgoing communication with a specific cluster (let define as $cls_R(l)$ and $cls_R(l)$ is assigned to $P_{j,r}$). That is, $\varepsilon(k, \alpha_{i,p}, \alpha_{j,r})$ means the total time taken to process $cls_R(k)$ and to communicate with $cls_R(l)$. On the other hand, $\omega(k, \alpha_{j,q})$ does not require data communication time with $cls_R(l)$ because it is included in the same computer (let assume no cluster is assigned to $P_{j,q}$). Then let define the evaluation value as follow.

$$\Delta(k, \alpha_{i,p}, \alpha_{j,q}) = \varepsilon(k, \alpha_{i,p}, \alpha_{j,q}) - \omega(k, \alpha_{j,q}). \quad (6)$$

For each cluster $cls_R(k)$, the algorithm finds the maximum value of $\Delta(k, \alpha_{i,p}, \alpha_{j,q})$ with changing j and q . This is the same meaning as changing l in $cls_R(l)$ (at line 5 in fig. 4). When the cluster $cls_R(l)$ or $P_{j,q}$ is found at line 7 in fig. 4, the cluster is removed from the set of clusters CHK . When CHK becomes empty, the algorithm is finished.

Selecting the target for moving the cluster $cls_R(k)$ by maximizing $\Delta(k, \alpha_{i,p}, \alpha_{j,q})$, the algorithm tries to find the PE by which the time taken for processing and data communication is effectively reduced.

V. PRELIMINARY EXPERIMENT

We conducted the preliminary experiment by a simulation to confirm advantages of our proposal, i.e., how the cluster re-mapping can reduce the schedule length. We compared with the method proposed in [6] in terms of the following points.

- 1) Effect on the cluster re-mapping with compared to the case without re-mapping.
- 2) Effect on the proposed re-mapping policy and algorithm.

Thus, comparison targets are, (A) the proposed cluster re-mapping, (B) each cluster is randomly re-mapped. (C) no re-mapping[6].

A. Experimental Environment

In the simulation, a random DAG is generated. In each DAG, each task size and each data size are decided randomly. Also CCR (Communication to Computation Ratio)[7], [8] 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 [9]. By using PF, the depth of the DAG is defined as $\frac{\sqrt{|V_0|}}{\rho}$. Since we assume a heterogeneous distributed system by multicore computers, each PE's performance and network bandwidth are varied by parameters. 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. Experimental Results

One comparison is done by executing 100 random DAGs and averaging the schedule length for each approach (A)-(C). Since (C) does not require cluster re-mapping, the schedule length by (C) is obtained. Then each re-mapping procedures is performed in (A) and (B), respectively. Then both approaches output each schedule length for the same DAG as (C). Table I shows comparison results by the experiment. In the table, α and β are ratios of maximum to minimum processing speed and maximum to minimum communication bandwidth, respectively. In the 6th column, values mean the schedule length ratio when the schedule length obtained by (A) is set to 1. Thus, a value over 1 is worse than (A). In every cases, it is observed that the proposed method (A) outperforms other approaches. If β is small such as No. 1-5 and No. 9-12, we can not observe noticeable characteristics among three approaches, except that the schedule length obtained by (A) is the smallest. On the other hand, β is large such as No. 5-8, the schedule length obtained by (C) is considerably large with increasing CCR. From this result, it is conceivable that data localization is necessary irrespective of cluster re-mapping methods if each data size is large. As for No. 13-16, the schedule length obtained by (B) is the largest with increasing CCR. From this result, the cluster re-mapping method has large effect on the schedule length if the heterogeneity of the network is large.

From these results, it is concluded that the proposed cluster re-mapping method has good effect on the schedule length by effectively localizing data communications among PEs.

TABLE I

COMPARISON RESULTS IN TERMS OF THE SCHEDULE LENGTH

No.	α	β	CCR	$ V_{cls}^R $	Schedule Length		
					A	B	C
1	5	5	0.1	156	1.000	1.002	1.002
2			1.0	58	1.000	1.011	1.008
3			5.0	42	1.000	1.031	1.048
4			10.0	16	1.000	1.109	1.144
5	5	10	0.1	165	1.000	1.001	1.044
6			1.0	55	1.000	1.076	1.071
7			5.0	32	1.000	1.103	1.142
8			10.0	14	1.000	1.107	1.281
9	10	5	0.1	139	1.000	1.001	1.003
10			1.0	48	1.000	1.068	1.042
11			5.0	37	1.000	1.103	1.009
12			10.0	27	1.000	1.198	1.201
13	10	10	0.1	188	1.000	1.004	1.002
14			1.0	67	1.000	1.143	1.107
15			5.0	40	1.000	1.170	1.140
16			10.0	29	1.000	1.179	1.151

VI. CONCLUSION AND FUTURE WORKS

In this paper, we presented a cluster re-mapping method in a multicore distributed system for processor utilization. The method tries to move clusters for data localization with taking into account both data dependencies which can have good effect on the schedule length. This procedure is performed by calculate the difference between time taken to execute every task in the cluster in the new PE and the time taken to execution and data communication in the PE decided by the method of [6]. From preliminary experiments, the proposed method was proved to have good effect on the schedule length.

As a future work, we develop more efficient method which does not use re-mapping in multicore distributed systems for processor utilization.

REFERENCES

- [1] I. Foster, The Grid: A new infrastructure for 21st century science, *Physics Today*, Vol.55(2), pp. 42-47, 2002.
- [2] G. DeCandia et al, Dynamo: amazon's highly available key-value store, *Procs. of ACM SIGOPS symposium on Operating systems principles (SOSP'12)*, pp. 205 - 220, 2007.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, A Processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System, *Journal of Computing*, Vol. 3, Issue 11, pp. 1 - 8, November 2011.
- [7] 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.
- [8] O. Sinnen., Task Scheduling for Parallel Systems, Wiley, 2007.
- [9] 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.