

Evaluation of Task Clustering Algorithm by FFT for Heterogeneous Distributed System

Shuya Hashimoto *, Emilia Ndilokelwa Weyulu**, Kazuo Hajikano***

Hidehiro Kanemitsu****, Moo Wan Kim *

* Department of Information Systems, Faculty of Informatics, Tokyo University of Information Sciences, 4-1 Onaridai, Wakaba-ku, Chiba-shi, Chiba, Japan

** Graduate School of Informatics, Tokyo University of Information Sciences, 4-1 Onaridai, Wakaba-ku, Chiba-shi, Chiba, Japan

*** Daiichi Institute of Technology, 1-10-2 Kokubunchuo, Kirishima-shi, Kagoshima, Japan

**** Global Education Center, Waseda University, 4-5-6 Totsuba-machi, Shinjuku-ku, Tokyo, Japan

j13343sh@gmail.com, eweyulu@gmail.com, k-hajikano@daiichi.ac.jp

ykim5jp@ybb.ne.jp, mwkim@rsch.tuis.ac.jp

Abstract— In this paper, the evaluation result of a task clustering heuristic algorithm proposed for heterogeneous distributed systems is shown. The proposed heuristic algorithm is based on our original concept, known as “Worst Schedule Length (WSL)”. It derives the lower bound of the total execution time of the cluster for each processor using WSL, then the processor which contributes to minimize WSL is chosen as an assignment target. Task clustering is then performed to get minimal response time (i.e., minimal schedule length). We show that our proposed method has advantages over existing conventional approaches through the evaluation results.

Keywords— Big Data, Distributed Processing, Fast Fourier Transform, Task Graph

I. INTRODUCTION

Distributed processing is indispensable when dealing with large scale of data like big data. Especially in cases of heterogeneous distributed systems with various processing speed and communication bandwidth, it is necessary to assign multiple tasks to multiple processors adequately. Assuming a job is expressed as DAG (Directed Acyclic Graph) it is necessary to develop a task clustering algorithm to assign multiple processors to multiple tasks. DAG is a directed graph where nodes represent tasks and directed edges represent the dependency among tasks. Assigning tasks in DAG to processors to minimize the response time is known as NP-complete problem [1], and several heuristic algorithms have already been proposed such as HEFT (Heterogeneous Earliest Finish Time) [2]. However these are not efficient for jobs that transfer large amounts of data.

In this paper, the evaluation result of the task clustering heuristic [5] algorithm proposed for data intensive jobs on heterogeneous distributed systems is shown. The proposed heuristic algorithm is based on our original concept, known as “Worst Schedule Length (WSL)”. We derive the lower bound of the cluster size assigned to each processor using WSL, then the processor which contributes to minimize WSL is chosen as an assignment target. Then the task clustering is performed to

get minimal response time (i.e., minimal schedule length). We show that our proposal has advantages over existing conventional approaches through the evaluation results.

II. ASSUMED MODEL

A. Job Model

We assume that a job is expressed as a directed acyclic graph (DAG), which is known as a workflow type job. $G_{cls}^s = (V_s, E_s, V_{cls}^s)$ (Fig 1) is the DAG, where V is the set of tasks, E is the set of edges (data communications among tasks), and V_{cls}^s is the set of task clusters including one or more tasks by task clustering steps.

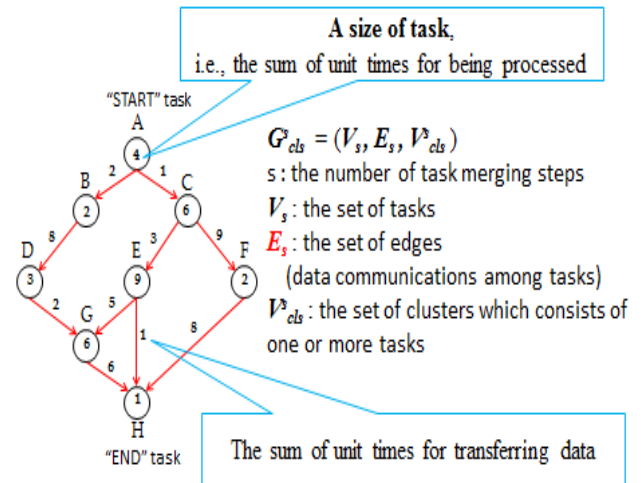


Figure 1. Example of Job Model [3]

This means that G_{cls}^s has $(|V| - s)$ unclustered tasks and $|V| = |V_{cls}^s|$ for $s = 0$. The i -th task is denoted as n_i , and let

$w(n_i^s)$ be the size of n_i^s , i.e., $w(n_i^s)$ is the sum of unit times for processing by the reference processor. We define data dependency and direction of data transfer from n_i^s to n_j^s as $e_{i,j}$, where $c(e_{i,j})$ is the sum of unit times 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 cannot start execution until all data from its predecessor tasks arrive. $pred(n_i^s)$ is the set of immediate predecessors of n_i^s , and $suc(n_i^s)$ is 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 path from n_i^s to n_j^s , we denote such a relation as $n_i^s < n_j^s$.

B. System Model

We assume each computer in the system model is connected to others with various processing speeds and communication bandwidths. Data transfer time within one computer is supposed to be negligible.

C. Task Clustering

Task clustering is to merge tasks into a cluster to localize data transfer and tasks in a cluster are assigned to the same PE (processing element). Also by localization of data transfer, $c(e_{i,j})$ i.e., data transfer time becomes zero in the cluster.

Procedure of task clustering is as follows;

- Let $cls(i)$ be the i -th cluster in V_{cls}^s
- If n_i^s is included in $cls(i)$ after the $(s + 1)$ -th task merging step, we formulate one task merging step as $cls_{s+1}(i) \leftarrow cls_s(i) \cup \{n_k^s\}$

Task clustering is finished when a certain criterion is satisfied, otherwise the process mentioned above is repeated.

III. MINIMIZING WORST SCHEDULE LENGTH(MWSL)

A. Worst Schedule Length(WSL)

WSL is the maximum execution path length, in case each task is executed as late as possible in a PE, provided that there is no data waiting time for each task once the PE starts execution. That is, WSL is a value determined for each task that is assigned to the PE, so that if it is assigned minimized WSL, it can be shown that the actual lower bound and the upper bound of the schedule length becomes smaller [4].

Table 1 shows notations for deriving WSL. $WSL(M(G_{cls}^s))$ is the maximum of $LV_s(cls_s(i))$, which corresponds to the maximum of $level(n_i^s)$, where $n_i^s \in cls_s(i)$. $level(n_i^s)$ is the maximum time duration from the START task to the END task in case n_i^s is scheduled as late as possible. That is, $WSL(M(G_{cls}^s))$ is the maximum level value in all tasks at the clustering state of $M(G_{cls}^s)$. For a task cluster $cls_s(i)$, $top_s(cls_s(i))$ is the set of tasks which can start execution first in $cls_s(i)$, and $ins(cls_s(i))$ is the set of tasks having incoming edges from other task clusters and $outs(cls_s(i))$ is the set of tasks having outgoing edges to other task clusters. $btm_s(cls_s(i))$ is the set of tasks having no immediate successor tasks in $cls_s(i)$, and $dc(n_i^s, cls_s(i))$ is the set of tasks being one of descendant tasks of n_i^s in $cls_s(i)$. $S(n_i^s, cls_s(i))$ is the time span from the start time of the task in $top_s(cls_s(i))$ to the start time of n_k in case that $t_w(n_i^s, pp) = 0$ and every task having no dependencies

with n_i^s is executed before n_k . $TL_s(cls_s(i))$ is the latest start time of the task in $top_s(cls_s(i))$, and $tlevel(n_i^s)$ is the latest start time of n_i^s without data waiting time if $n_i^s \notin top_s(cls_s(i))$. $tlevel(n_i^s)$ is derived by $S(n_i^s, cls_s(i))$ if $n_i^s \notin top_s(cls_s(i))$, otherwise it is the latest data ready time. $blevel(n_i^s)$ is the longest path length from n_i^s to the END task, and $BL_s(cls_s(i))$ is the maximum execution path length including $S(n_i^s, cls_s(i))$ and $blevel(n_i^s)$. If $LV_s(cls_s(i))$ is defined as the sum of $TL_s(cls_s(i))$ and $BL_s(cls_s(i))$, WSL at $M(G_{cls}^s)$, i.e., $WSL(M(G_{cls}^s))$ is defined as the maximum value of $LV_s(cls_s(i))$ where $cls_s(i) \in V_{cls}^s$.

TABLE 1. Notation for $WSL(M(G_{cls}^s))$ [4]

Parameter	Definition
$top_s(cls_s(i))$	The set of tasks having no immediate predecessor tasks in $cls_s(i)$
$ins(cls_s(i))$	The set of tasks having one or more immediate predecessor tasks in the cluster other than $cls_s(i)$
$outs(cls_s(i))$	The set of tasks having one or more immediate successor tasks in the cluster other than $cls_s(i)$
$btm_s(cls_s(i))$	The set of tasks having no immediate successor tasks in $cls_s(i)$
$dc(n_i^s, cls_s(i))$	The set of tasks in $cls_s(i)$ having paths from n_i^s , i.e., $\{n_j^s \mid n_i^s < n_j^s, n_j^s \in cls_s(i)\} \cup \{n_i^s\}$
$S(n_i^s, cls_s(i))$	$\sum_{n_i^s \in cls_s(i)} w(n_i^s) - \sum_{n_i^s \in dc(n_i^s, cls_s(i))} w(n_i^s)$
$tlevel(n_i^s)$	$\max_{n_i^s \in pred(n_i^s)} \{ tlevel(n_i^s) + w(n_i^s) + c(e_{i,j}^s) \},$ if $n_i^s \in top_s(i)$ $TL_s(cls_s(i)) + S(n_i^s, cls_s(i)), otherwise$
$TL_s(cls_s(i))$	$\max_{n_i^s \in top_s(cls_s(i))} \{ tlevel(n_i^s) \}$
$blevel(n_i^s)$	$\max_{n_i^s \in suc(n_i^s), n_i^s \notin cls_s(i)} \{ w(n_i^s) + c(e_{i,j}^s) + blevel(n_i^s) \}$
$level(n_i^s)$	$tlevel(n_i^s) + blevel(n_i^s)$
$BL_s(cls_s(i))$	$\max_{n_i^s \in outs(cls_s(i))} \{ S(n_i^s, cls_s(i)) + blevel(n_i^s) \}$
$LV_s(cls_s(i))$	$TL_s(cls_s(i)) + BL_s(cls_s(i)) = \max_{n_i^s \in cls_s(i)} \{ level(n_i^s) \}$
$WSL(M(G_{cls}^s))$	$\max_{cls_s(i) \in V_{cls}^s} \{ LV_s(cls_s(i)) \}$

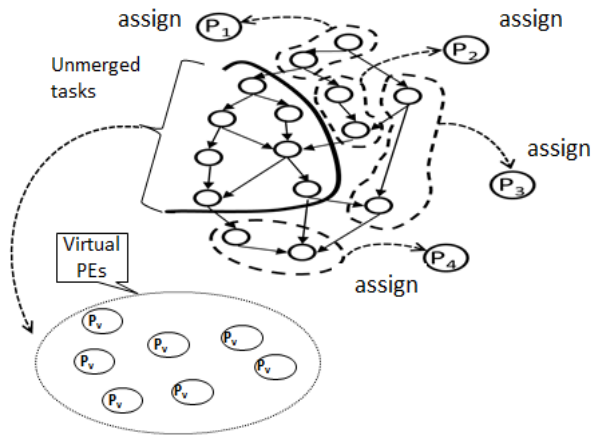


Figure 2. State after 4 task merging [3]

B. Algorithm

MWSL consists of the following 2 phases:

1) Phase-1: We define δ as a lower bound of the execution time at a task cluster. At first, we identify the path generating WSL by using virtual PE with the maximum processing speed and maximum communication bandwidth (we shall call "virtual PE"). We derive the upper bound of the WSL for each clustering step. Such upper bound of WSL is a function of δ , processing speed and communication bandwidth of PE. Next, we choose a PE minimizing the upper bound of WSL from PEs that have not been assigned to clusters, then differentiate the upper bound of the WSL with respect to δ and get the minimum value of the upper bound of WSL.

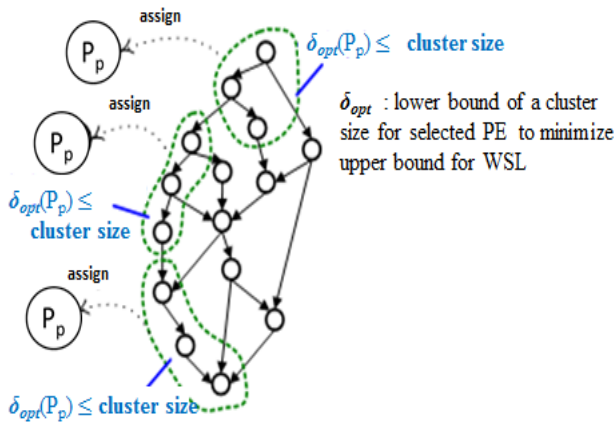


Figure 3. Try to minimize WSL [3]

2) Phase-2: In MWSL, two clusters are merged into a new task cluster in order to reduce WSL. The first one is called "pivot", and the other is called as "target" [4]. At first, a task cluster, which dominates WSL is selected as pivot. Then a task cluster, by which WSL reduction is achieved by

merging it with pivot, is selected as a target. This merging step is repeated until execution time of the new cluster exceeds the δ_{opt} . Phases 1 and 2 are repeated until all tasks are merged into clusters and assigned to PEs.

IV. EVALUATION RESULT

A. Evaluation Method

We compare the proposed algorithm with HEFT by using specific jobs. We chose FFT (Fast Fourier Transform) as the exemplary job. Next, we generated FFT DAG and implemented it using a simulation program. The maximum to minimum ratio in terms of processing speed and communication bandwidth are set to 10, respectively. Then we derived the response time by averaging 100 DAGs for the comparison. As for the characteristic of each DAG, we varied CCR (Communication to Computation Ratio) ranging from 1, 5, and 10. CCR in this simulation is derived by the averaged data transfer time to the averaged processing time, i.e., CCR=1 means transferred data is the same as processing tasks, CCR=10 is in the case of transferred data is 10 times of processing tasks.

B. DAG of FFT

In the simulation, we define Butterfly Computation as one task. Therefore, the FFT DAG generates $N/2 \log N$ tasks, where N is the matrix size [5].

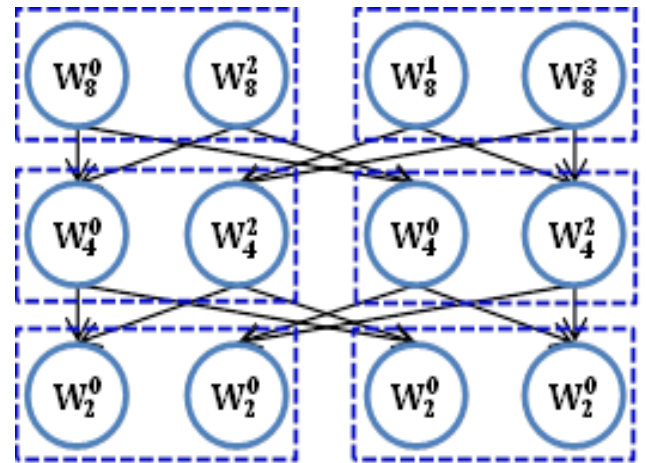


Figure 4. The DAG of FFT in the case of $N = 8$

If the matrix size of the FFT DAG is 8, this DAG generates 12 tasks. Moreover, each task executes the same operation, so that execution time of each task is the same if the assigned processor has the same processing speed.

C. Simulation Results

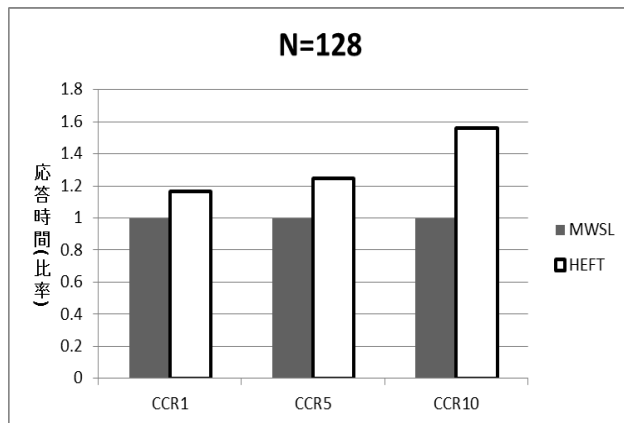
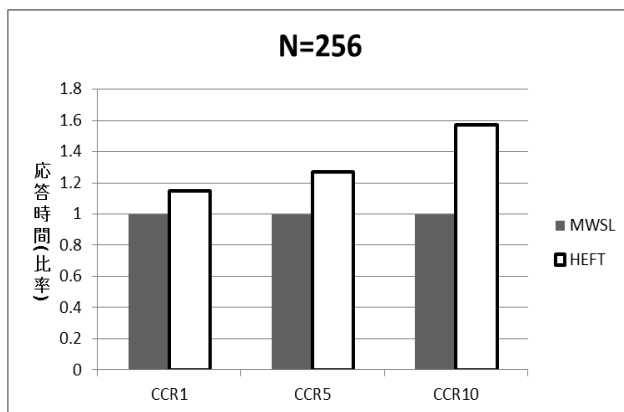
The simulation of FFT has been performed in the case that $N = 128$ and $N = 256$. Table 1 shows the result in case that $N = 128$ and $N = 256$, and Table 2 shows the data showing Table 1 as a ratio of MWSL. Fig.5 shows the result of simulation in case that $N = 128$ and Fig.6 shows in case that $N = 256$.

TABLE 2. The data by simulation in case that $N = 128$ and $N = 256$

Matrix Size	CCR	MWSL	HEFT
N=128	1	993.04	1159.58
	5	1132.02	1412.83
	10	1132.71	1766.61
N=256	1	1150.94	1323.49
	5	1314.97	1666.49
	10	1337.76	2101.93

TABLE 3. The data showing Table 1 as a ratio of MWSL

Matrix Size	CCR	MWSL	HEFT
N=128	1	1	1.1677
	5	1	1.2481
	10	1	1.5596
N=256	1	1	1.1499
	5	1	1.2673
	10	1	1.5712

**Figure 5.** The result of simulation in the case of $N = 128$ **Figure 6.** The result of simulation in the case of $N = 256$

From Table 1, Table2, Fig. 5and Fig. 6, it can be seen that MWSL shows better response time as CCR becomes larger. Therefore, it can be concluded that MWSL is more suitable algorithm than HEFT for data intensive jobs.

V. CONCLUSIONS

In this paper, we have shown the evaluation result of our proposed algorithm.

We performed a simulation of FFT 100 times in each case $N = 128$, $N = 256$ we compared response time of HEFT with MWSL. From the simulation results, we have shown that our proposed algorithm is a more suitable algorithm than HEFT in the case of large application of data.

REFERENCES

- [1] H.J. Braun et al, "of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," Journal of Parallel and Distributed Computing, pp. 810-837, Vol.61, No.6, 2001.
- [2] H. Topcuoglu, et al., "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.
- [3] H. Moon, et al., "Evaluation of Task Clustering Heuristic Algorithm in Heterogeneous Distributed System," Proc. "RISP International Workshop on, Nonlinear Circuits, Communications and Signal Processing 2016," Honolulu, Mar 2016.
- [4] H. Kanemitsu, et el, "A processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System", Journal of Computing, Vol. 3, Issue 11, pp.1-8, Nov 2011.
- [5] James W. Cooley, and John W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol.19, no.90, pp.297-301, Apr 1965.

Shuya Hashimoto is student with the Tokyo University of Information Sciences in Chiba, Japan.

Emilia Weyulu is student with the Graduate school of Informatics at the Tokyo University of Information Sciences in Chiba, Japan.

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

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

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