A Task Scheduling Method for Data Intensive Jobs in Multicore Distributed System

Kazuo Hajikano*[1]　Hidehiro Kanemitsu*[2]　Moo Wan Kim*[3]

*[1] Department of Information Technology and Electronics, Daiichi Institute of Technology
1-10-2, Kokubu-Chuo, Kirishima, Kagoshgima, 899-4395, Japan

*[2] Global Education Center, Waseda University
1-104, Totsuka-Chou, Shinjuku, Tokyo, 169-8050, Japan

*[3] Department of Informatics, Tokyo University of Information Sciences
4-1, Onaridai, Wakaba, Chiba, 265-8501, Japan

Abstract: On task scheduling methods for a work-flow type job with precedence constraint among tasks over heterogeneous distributed environment, methods based on list scheduling are well known. These are considered to not effective as expected about the response time in data intensive jobs. We propose a task scheduling method for data intensive jobs in Multicore Distributed System, which can reduce the response time with keeping parallelism in execution. We show advantage of proposed method against existing methods with experimental simulations.
Key word: task scheduling, multicore, heterogeneous, data intensive

1. Introduction

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(Predict Earliest Finish Time)[2], CEFT(Constrained Earliest Finish Time)[3] are well known. These methods are effective for reducing the response time against computationally intensive jobs. On the other hand, these are considered to not to get improvement as expected about the response time 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. We propose a task scheduling method for data intensive jobs in Multicore Distributed System, which can reduce the response time with keeping parallelism in execution.

2. Assumed Model

2.1 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 be the DAG, G^s_{cls} = (V_s, E_s, V^s_{cls}), where s is the number of task merging steps (described in 2.3), V_s is the set of tasks after the s-th task merging step, E_s is the set of edges (data communications among tasks) after the s-th task merging step, and V^s_{cls} is the set of clusters which consists of one or more tasks after the s-th task merging step. An i-th task is denoted as n^s_i. Let $W(n^s_i)$ be a size of n^s_i, i.e., $W(n^s_i)$ is the sum of unit times taken for being processed by the reference processor. We define data dependency and direction of data transfer from n^s_i to n^s_j as e^s_{ij}. And $C(e^s_{ij})$ is the sum of unit times taken for transferring data from n^s_i to n^s_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. If a task does not have any immediate predecessor, it is called START task, and if a task does not have any immediate successors, it is called END task.

2.2 System model

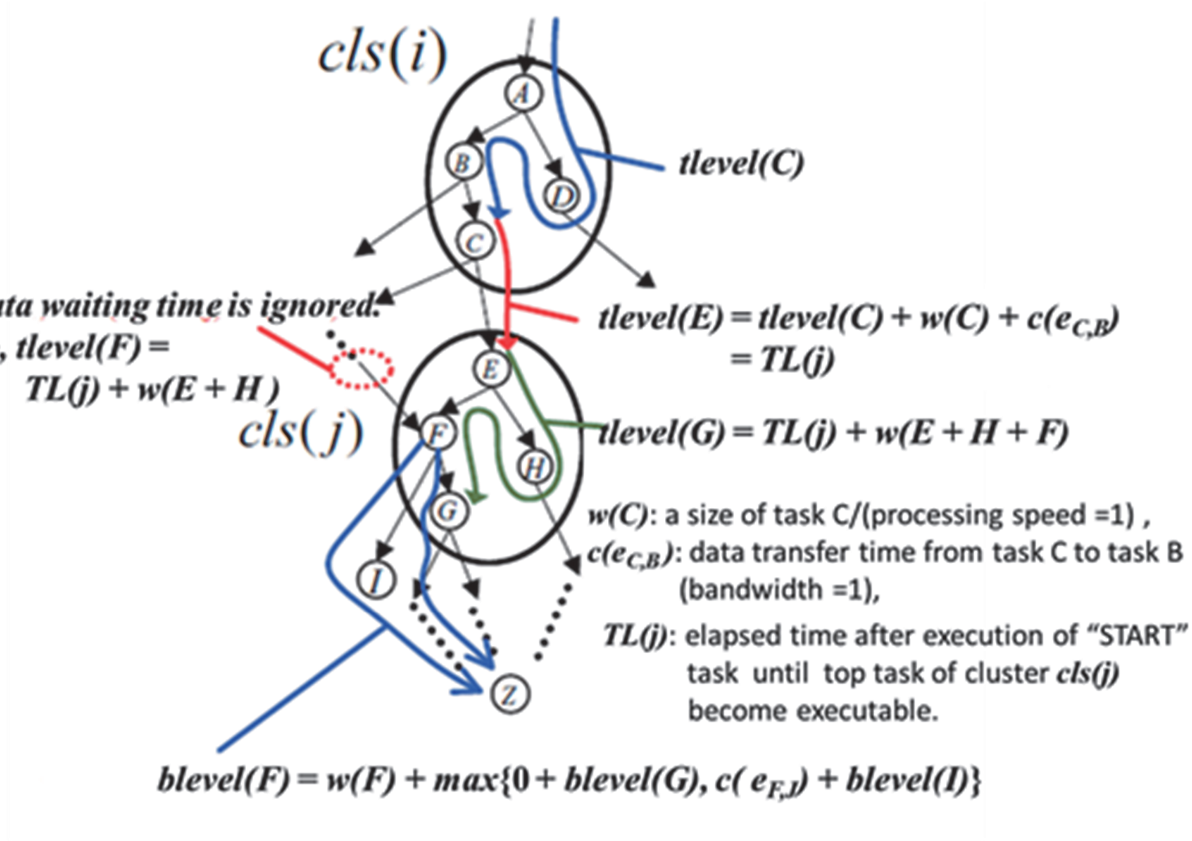
We assumed each computer is completely connected to others, with heterogeneous processing speed and communication bandwidths. Each computer has one or more PE, i.e., core, with heterogeneous processing speed. Data transfer time within one computer is supposed to be negligible.

2.3 Definitions of a Cluster and Task Clustering

We denote the i-th cluster in V^s_{cls} as $cls_s\ (i)$. If n^s_k is included in $cls_s\ (i)$ by "the s + 1th task merging", we formulate one task merging as $\{cls_{s+1}\ (i) \leftarrow cls_s\ (i)$ U $\{n^s_k\}$. If any two tasks, i.e., n^s_i and n^s_j are included in the same cluster, this means that they are assigned to the same processing element. Then the communication between n^s_i and n^s_j is localized, so that communication time between those is zero. Task clustering is a set of task merging steps, that is finished when a certain criteria is satisfied.

3. Previous work

We proposed a method for efficient use of computational resources each of which has a single



Data waiting time is ignored. So, $tlevel(F) = TL(j) + w(E + H)$

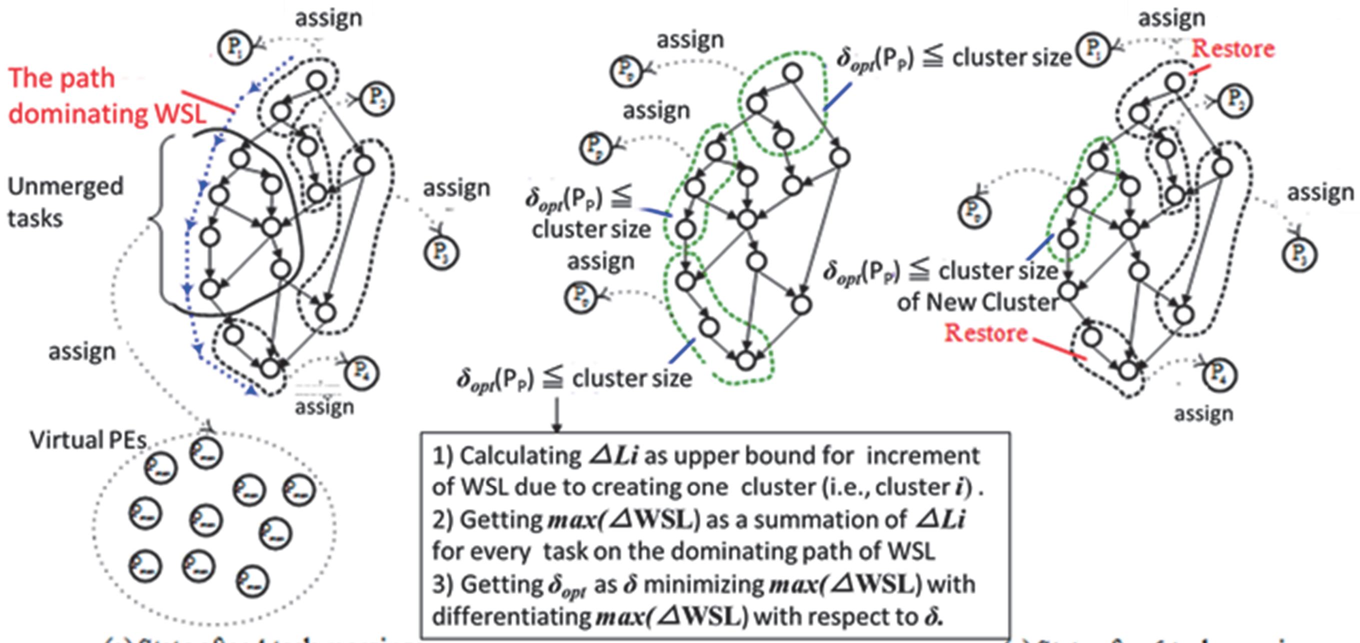
$tlevel(E) = tlevel(C) + w(C) + c(e_{C,B}) = TL(j)$

$tlevel(G) = TL(j) + w(E + H + F)$

$w(C)$: a size of task C/(processing speed =1) ,
$c(e_{C,B})$: data transfer time from task C to task B (bandwidth =1),
$TL(j)$: elapsed time after execution of "START" task until top task of cluster $cls(j)$ become executable.

$blevel(F) = w(F) + max\{0 + blevel(G), c(e_{F,I}) + blevel(I)\}$

Fig. 1 Level and WSL (Worst Scheduling Length)


assign
P_1
P_2
P_3
P_4
The path dominating WSL
Unmerged tasks
assign
assign
assign
Virtual PEs
assign
$\delta_{opt}(P_p) \leqq$ cluster size
assign
P_p
$\delta_{opt}(P_p) \leqq$ cluster size
assign
$\delta_{opt}(P_p) \leqq$ cluster size
$\delta_{opt}(P_p) \leqq$ cluster size of New Cluster
Restore
Restore
assign
assign

1) Calculating ΔLi as upper bound for increment of WSL due to creating one cluster (i.e., cluster i) .
2) Getting $max(\Delta \mathrm{WSL})$ as a summation of ΔLi for every task on the dominating path of WSL
3) Getting δ_{opt} as δ minimizing $max(\Delta \mathrm{WSL})$ with differentiating $max(\Delta \mathrm{WSL})$ with respect to δ.

(a) State after 4 task mergings.　(b) Try to minimize WSL with P_p　(c) State after 6 task mergings.

Fig. 2 Example of δ_{opt} (P_p) derivation

processor in a heterogeneous distributed system [4]. The method automatically derives the set of mapping between each processor and each assignment unit (i.e., the set of tasks in a DAG).

For each PE, we derivate the lower bound of a cluster size to be assigned to the PE theoretically, considering amount of data and load for a job , and processing performance and bandwidth for each PE. With those lower bounds, this method can keep parallelism even for data intensive jobs with adequate number of PEs involved in execution.

We also introduced WSL (Worst Scheduling Length) as the index of lower bound and upper bound of the response time, which can be calculated before scheduling. It is proved that WSL should be minimized to minimize the response time.

3.1 WSL and level of task

WSL means the largest value that the response time can take when every task is executed as late as possible.

When a task in a cluster is executed as late as possible and a path including the task from START task to END task is identified, the level of the task means the response time along the path, that is, summation of the maximum start time of the task and the maximum elapsed time of the task from its starting time to completion of the END task. Largest value of the level in a cluster is defined as the level of that cluster. Largest value of the level among clusters is defined as WSL and we call the cluster with the WSL as "the cluster dominating WSL" as well as call the task with WSL as "the task dominating WSL". Also, the path that "the task dominating WSL" belongs to is called as "the path dominating WSL "

In Fig.1, level of task ***F = tlevel(F) + blevel(F),*** where ***tlevel(F)*** means elapsed time after execution of START task until task F become ready to execute, while ***blevel(F)*** means expected longest elapsed time after executing task F until completion of END task . ***LV(j)***, i.e., the level of Cluster cls(j) is largest level among those of task E, F, G and H.

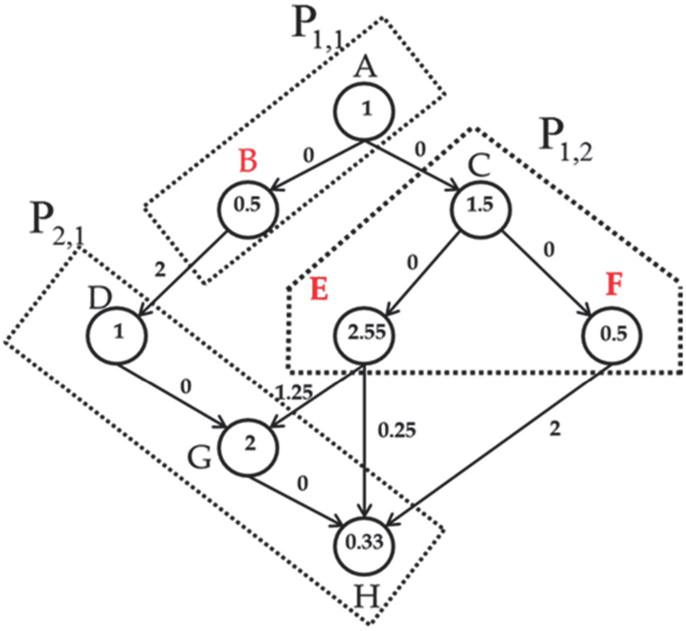


Fig.3 State after completion of all task mergings

4. Proposal

To get more reduction of the response time for data intensive jobs, we enhance the method proposed in [4] as follows.

- Considering data transfer time, generated new cluster is assigned to unassigned core belonging to the computer having already assigned other cores.
- Using WSL as priority for task scheduling.

Our proposal is consist of following 3 processes.

4.1 Process 1 : Lower bound derivation for each cluster execution time and PE selection

At first, we define $\boldsymbol{\delta}$ is a lower bound of cluster execution time. Fig. 2 (a) shows state after 4 task mergings. There are unmerged tasks each of which is assigned to the "virtual PE" with the maximum processing speed and communication bandwidth, respectively. On the other hand, other tasks are assigned to actual PEs. In Fig. 2 (b), we temporarily assume that tasks on the path dominating WSL will be clustered and each cluster is assigned to an identical PE (in (b), the PE is P_P). From procedure 1) to 3) in (b), We get $\boldsymbol{\delta_{opt}}$ for selected P_P to minimize upper bound for WSL(For more details about $\boldsymbol{\delta_{opt}}$ derivation, refer to the literature[2].). Once $\boldsymbol{\delta_{opt}}$ is obtained, every cluster in (b) is restored to clusters in (a). We calculate $\boldsymbol{\delta_{opt}}$ and ⊿WSL for every candidate. Candidates are unassigned cores belonging to computers, one of whose cores have already been assigned to the existing cluster at least. If there is no such core, unassigned cores belonging to any computers would be candidates. Among candidates, the core with minimized ⊿WSL is selected to be assigned to new cluster in (c).

4.2 Process 2: Task clustering

We select the cluster with maximum level as pivot and the succeeding cluster dominating level of pivot as target. These two are merged into a new cluster. This merging step is repeated until the new cluster's execution time exceeds the $\boldsymbol{\delta_{opt}}$. In (c), it is supposed that we get the new cluster after six task mergings.

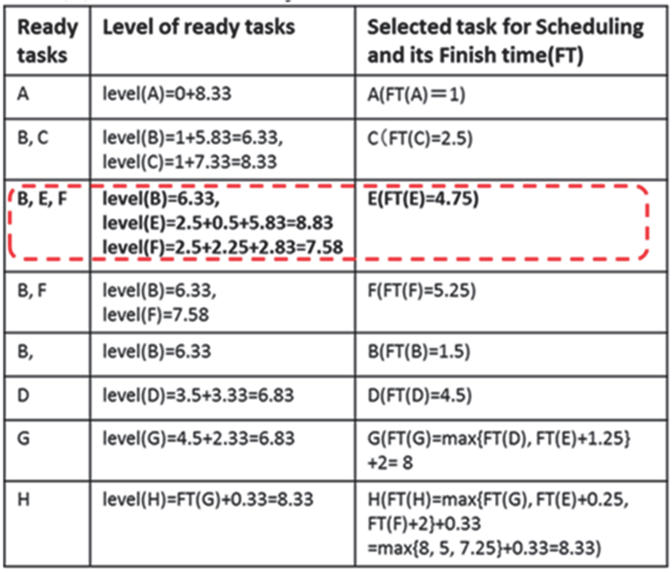
Process 1 and 2 are repeated until all tasks are merged into clusters assigned to cores.

4.3 Process 3: Task scheduling.

At actual scheduling phase, the level for each ready task is recalculated with actual processor performance and communication bandwidth, then the task with the maximum level is chosen to be executed next.

Fig. 3 shows state after completing all task mergings. i.e., all tasks are merged into clusters assigned to different cores. Table 1 shows the level of each ready task. There are three ready tasks at 3rd row at Table 1, i.e., B, E and F. Task B belongs to the cluster assigned to core $P_{1,1}$ while task E and task F belong to the cluster assigned to core $P_{1,2}$. Therefore, task B can be executed

Table 1 Ready tasks and those levels



Ready tasks	Level of ready tasks	Selected task for Scheduling and its Finish time(FT)
A	level(A)=0+8.33	A(FT(A)＝1)
B, C	level(B)=1+5.83=6.33, level(C)=1+7.33=8.33	C(FT(C)=2.5)
B, E, F	level(B)=6.33, level(E)=2.5+0.5+5.83=8.83 level(F)=2.5+2.25+2.83=7.58	E(FT(E)=4.75)
B, F	level(B)=6.33, level(F)=7.58	F(FT(F)=5.25)
B,	level(B)=6.33	B(FT(B)=1.5)
D	level(D)=3.5+3.33=6.83	D(FT(D)=4.5)
G	level(G)=4.5+2.33=6.83	G(FT(G)=max{FT(D), FT(E)+1.25} +2= 8
H	level(H)=FT(G)+0.33=8.33	H(FT(H)=max{FT(G), FT(E)+0.25, FT(F)+2}+0.33 =max{8, 5, 7.25}+0.33=8.33)

independently of task E and F. Because of task E and F assigned to same core, we have to choose task E or F to be executed next. Level of task E is 8.83 while that of task F is 7.58 (middle of row), therefore Task E is chosen to be executed next (3rd column).

5. Experiment

5.1 Objectives

We conducted the experimental simulation to confirm advantages of our proposal against existing methods, i.e., HEFT and PEFT in term of response time.

5.2 Experimental Environment

In the simulation, a random DAG is a generated. In the DAG, each task size and data size are decided randomly. Also CCR (Communication to Computation Ration) [5] is chosen as 1, 5 and 10. The max to min ratio in term of data size is set to 2, 5 and 10. Also, the max to min ratio in term of communication bandwidth is set to 2, 5 and 10.

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 2.66 GHz, and the memory size is 2.0 GB.

5.3 Comparison about response time

Fig. 4 shows comparison results. In the Figures, **α** and **β** mean max to min ration of the processing speed and communication bandwidth, respectively. In both figures, vertical axis show relative response time where response time of proposed method is “1.00”.

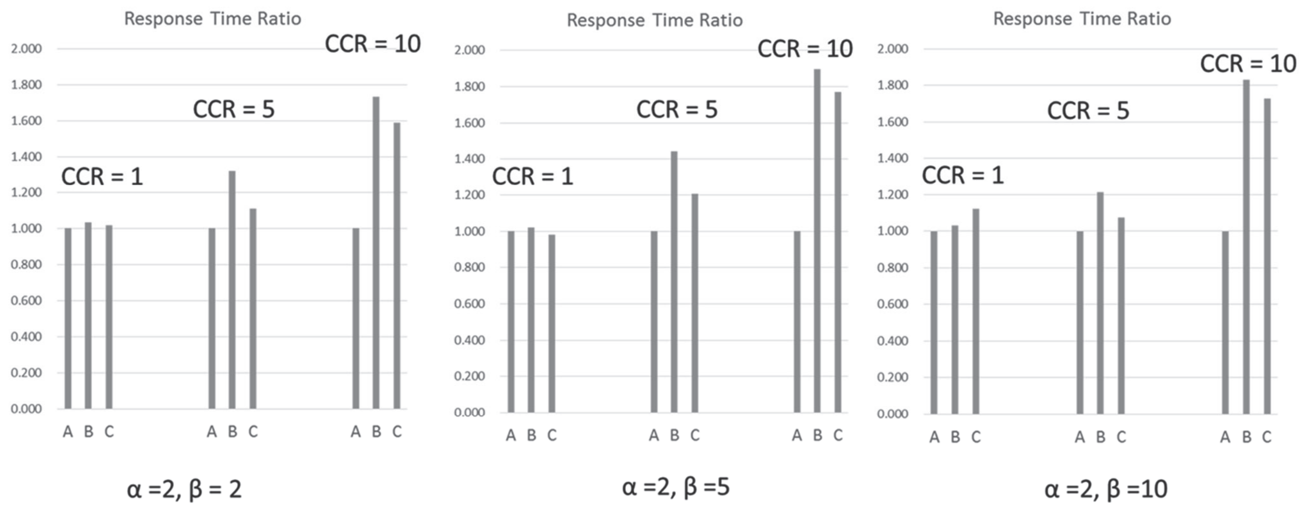
From comparison result in Fig. 4, it is concluded that response time of proposed method is better than that of existing method. Especially, As CCR is larger, that is, in more data intensive case, proposed method shows better performance.

6. Conclusion

We presented a task scheduling method for data intensive jobs in Multicore Distributed System. We confirmed that advantage of proposed method against existing methods with experimental simulations.

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”, IEE Trans. on Parallel and distributed systems, vol. 25, No. 3, pp. 682-694, March 2014
[3] M. A. Khan, “Schedule for heterogeneous systems using constrained critical paths”, Parallel Computing, vol. 38, pp 175-193, 2012
[4] H. Kanemitsu, et el, “ A processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System”, Journal of Computing, Vol. 3, Issue 11, pp1-8,
[5] O. Sinnen and l. A. “Sousa, Communication Contention in Task Scheduling”, IEE Trans. on parallel and Distributed Systems, Vol. 16, No. 6, pp503-515, 2005



α: max to min ration of the processing speed, β:max to min ration of the communication bandwidth
CCR: Communication to Computation Ration

Fig. 4 Comparison of proposed method and existing method