Workflow Scheduling to Minimize Data Movement Using Multi-Constraint Graph Partitioning

Masahiro Tanaka*‡ and Osamu Tatebe†*‡
*Center for Computational Sciences
University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
†Faculty of Engineering, Information and Systems
University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
‡Core Research for Evolutional Science and Technology
Japan Science and Technology Agency, Kawaguchi, Saitama, Japan
Email: tanaka@hpcs.cs.tsukuba.ac.jp, tatebe@cs.tsukuba.ac.jp

***Abstract*—Among scheduling algorithms of scientific workflows, the graph partitioning is a technique to minimize data transfer between nodes or clusters. However, when the graph partitioning is simply applied to a complex workflow DAG, tasks in each parallel phase are not always evenly assigned to computation nodes since the graph partitioning algorithm is not aware of edge directions that represent task dependencies. Thus, we propose a new method of task assignment based on Multi-Constraint Graph Partitioning. This method relates the dimension of weight vectors to the rank of a task phase defined by traversing the task graph. Our algorithm is implemented in the Pwrake workflow system and evaluated the performance of the Montage workflow using a computer cluster. The result shows that the file size accessed from remote nodes is reduced from 88% to 14% of the total file size accessed during the workflow and that the elapsed time is reduced by 31%.**

I. Introduction

In various science fields, such as astronomy, elementary particle physics, and life science, the data volume is increasing as the observation instruments evolve. Such data-intensive science requires a high-performance system to maximize I/O throughput in scientific workflow. One of important issues in workflow systems is workflow scheduling. Workflow scheduling is the problem of deciding where to execute a process [1], [2]. Many scheduling algorithms such as Min-Min [3] and HEFT [4] have objectives to minimize the completion time (*makespan*) of the workflow. The makespan depends on *computation costs* and *communication costs*.

In this paper, we focus on workflow scheduling to minimize data movement during workflow execution using the graph partitioning algorithm. Graphs are widely used to represent the data dependencies in a computation. Recall that a graph, $G = (V, E)$, comprises of a set of vertices, $V = \{v_1, v_2, ..., v_n\}$, and a set of pairwise relationships called edges $E \subset V \times V$. If $(v_i, v_j) \in E$, then vertices v_i and v_j are neighbors. In general, a workflow can be represented as a Directed Acyclic Graph (DAG) based on the dependencies of the tasks constituting the workflow. For our purposes, the vertices of a graph represent units of computation, and the edges encode data dependencies. The graph partitioning is used to determine the division of the computation and data for an efficient parallel computation. Our objective is to distribute tasks evenly over n computation nodes consisting a distributed system by partitioning the vertices into n subsets while minimizing internode communication represented by edges crossing between partitions. Our main contribution is the application of the Multi-Constraint Graph Partitioning (MCGP) [5] for scientific workflow system, including the implementation of the workflow system and the evaluation using a use case of the Montage astronomy workflow. Using our workflow system with scheduling based on MCGP, we show that our algorithm minimizes the communication cost and effectively reduces the workflow execution time.

This paper is organized as follows. In Section II, we discuss the scheduling problem and the data movement in workflow. In Section III, we propose a workflow scheduling to minimize data movement using MCGP. In Section IV, the implementation of the proposed workflow scheduling is described. In Section V, the performance of the workflow accelerated by our method is evaluated. Section VII gives conclusion and future work.

II. Data Movement in Workflow

A. Scheduling Problem and Data Movement

We discuss data movement in an example workflow shown in Figure 1. This workflow requires the four input files stored at a storage node. The workflow consists of two phases, Task A and Task B. In the Task A phase, each task receives an input file. In the Task B phase, each task receives inputs from two of Task A. All the tasks are executed on a distributed system consisting of two computation nodes. For the scheduling of this workflow, we make the following two assumptions; Firstly, the distributed system is homogeneous; two computation nodes have the same performance. Secondly, all the tasks in each phase require the same computation cost and all the data transfer in each phase requires the same communication cost. Although

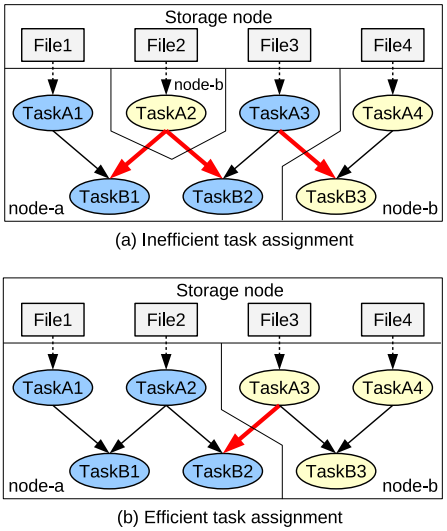


Figure 1. Task assignment and data movement

this system is homogeneous, we consider the HEFT scheduling algorithm [4], as one of the well-studied DAG-based scheduling algorithms. In the HEFT scheme, the first phase is the calculation of an upward rank $rank_u$. However, the $rank_u$ of Task A1-A4 are the same since the computation costs of Task A1-A4 are equal and those of Task B1-B3 are equal, respectively. In this case, one of Task A1-A4 are selected randomly. Assume that Task A1 is selected first. The next phase is the selection of a computation node where Task A1 is to be executed, based on the Earliest Finished Time (EFT). However, EFT is the same regardless of the node allocation if the input file sizes are all the same. Then a computation node for Task A1 is selected randomly. After that, the other node is selected for the next-selected task. In conclusion, under the HEFT scheme, Task A1-A2 are randomly assigned to computation nodes. Therefore, the chances of task assignment are equal between Figure 1 (a) and (b). Furthermore, the target function of the HEFT algorithm, *makespan*, also results in the same values for (a) and (b), as is shown in Figure 2.

However, the amount of data movement is different between Figure 1 (a) and (b). Data movement in Figure 1 (a) is three times, while that in Figure 1 (b) is only once. This example demonstrates that the node allocation in the former phase of Task A has an impact on the communication costs at the subsequent dependencies (e.g. A1,A2→B1).

B. Montage Workflow

Real scientific workflows are more complex than Figure 1. In this study, we investigate the Montage workflow [6]. Montage is a collection of programs to combine multiple shots of astronomical images and to generate a custom mosaic image. Every task of Montage is an individual program written in the C language. This design enables parallel execution

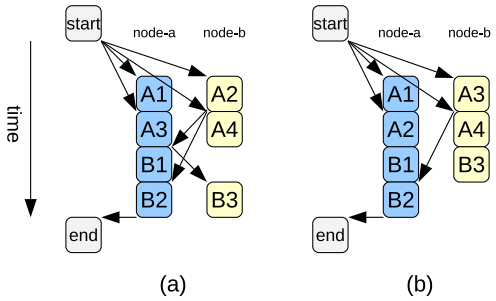


Figure 2. HEFT scheduling applied to Fig. 1.

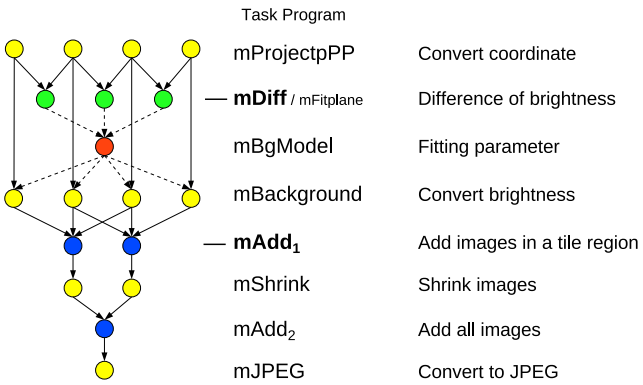


Figure 3. DAG of the Montage Workflow

of independent tasks. The Montage workflow represented as a DAG is shown in Figure 3. The Montage workflow consists of eight phases according to kinds of tasks. For example, the first phase is mProjectPP, a task to project an image onto a target plane. The number of tasks in each phase depends on the number of input files, which can be more than ten thousand. Two of the parallel phases in Figure 3 have a similar dependency pattern to Figure 1, where one task receives multiple inputs from prior tasks. One of these patterns is mProjectPP→mDiff, and the other is mBackground→mAdd_1 (the former mAdd). These phases are combied through task dependencies and constitute a single workflow DAG. In order to minimize data movement in such a complex workflow, it is necessary to determine which node to assign tasks, based on all the dependencies of the DAG.

C. Standard Graph Partitioning

The *graph partitioning* is a possible algorithm to assign tasks to computation nodes for the purpose to minimize data movement in a complex workflow. It has been used for dividing large data for simulation and data processing [7].

The graph partitioning is a problem, given a graph $G = (V, E)$, to partition V into n groups, $V_1, V_2, ..., V_n$, evenly (i.e., $|V_1|, |V_2|, ..., |V_n|$ are equal), so as to minimize the number of edges whose endpoint vertices belong to different

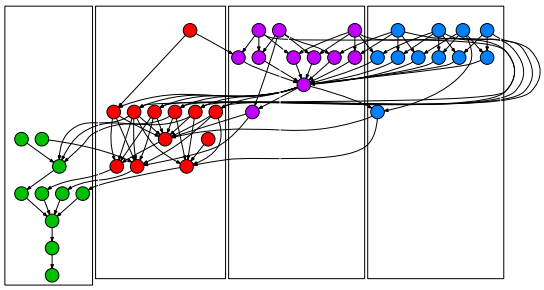


Figure 4. The Standard Graph Partitioning: unbalanced tasks in each phase

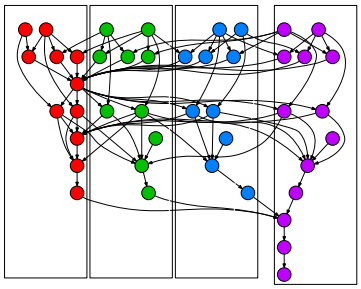


Figure 5. The Balanced Graph Partitioning using the proposed method

subsets. It is known that the graph partitioning problem is NP-complete. The graph partitioning problem can be extended to a graph in which each vertex or edge has a weight. If the vertices have weights, the sum of the vertex-weights in each subset is equal. If the edges have weights, the sum of the edge-weights whose endpoint vertices belong to different subsets is minimized. We refer to this basic graph partitioning problem as *the standard graph partitioning*. The standard graph partitioning is applicable to undirected graphs.

For workflow DAGs, the vertices and the edges represent tasks and data dependencies, respectively. The purpose of this study to minimize data movement is the same as the purpose of the graph partitioning, i.e., to minimize the edge cut. The standard graph partitioning is applied to a DAG of the Montage workflow, and its result is shown in Figure 4. The figure shows that the former tasks are assigned to two groups on the right, and the final task are assigned to a group on the left. This is because all the vertices are evenly partitioned regardless of the task phases. The standard graph partitioning algorithm does not reflect task parallelism. Instead, Figure 5 shows an intended partitioning where vertices are balanced in each phase. Therefore, the partitioning algorithm of workflow DAG requires the condition of equipartition of tasks in each parallel phase as well as the edge cut minimization.

III. Workflow Scheduling using MCGP

We propose that task assignment to computation nodes is determined using the Multi-Constraint Graph Partitioning

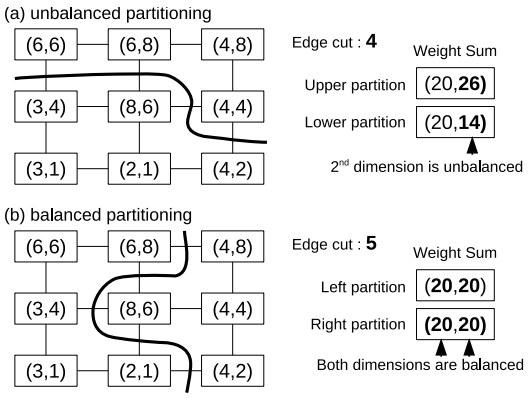


Figure 6. Example of Multi-Constraint Graph Partitioning

(MCGP) [5]. The idea of the use of MCGP for partitioning multi-phase tasks appeared in [8]. However, the application of MCGP to workflow scheduling has not been reported. The contribution of this paper is the implementation and evaluation of MCGP for workflow scheduling on a real workflow system.

A. Multi-Constraint Graph Partitioning

The MCGP algorithm has been studied and implemented in the METIS library [5], [9]. Contrary to the standard graph partitioning in which a vertex weight is a scalar value, a vertex weight in MCGP is a vector consisting of multiple values. The purpose of MCGP is to balance the summation of weight values in each dimension, The example of MCGP algorithm is shown in Figure 6. In this example, two-dimensional weight vectors are given to the vertices of the graph. Figure 6 (a) shows the partitioning with an edge-cut cost of 4. However, the weight summations of the second dimension are 26 and 14 in the upper and lower partition, respectively. This case is unbalanced partitioning. On the other hand, Figure 6 (b) shows the partitioning with a larger edge-cut cost of 5 than Figure 6 (a). However, the weight summations of both dimensions are balanced. The MCGP algorithm aims at partitioning in Figure 6 (b), where all the dimensions of weight vectors are balanced simultaneously. One of the example applications of MCGP proposed in [5] is to compute a partitioning of the mesh among the processors such that both of the amount of computations and memory are balanced simultaneously.

B. Determining Task Phase

Before applying MCGP to a workflow DAG, we define *task phases* in which tasks can be executed in parallel, based on the dependency of the workflow. This is equivalent to the procedure that the vertex V of the workflow DAG is partitioned into subset $P_1, P_2, ..., P_l$. The determination of the phase is as follows; The tasks with no precedent task are defined as the first phase. This is expressed as $P_1 = \{v_k \mid (v_j, v_k) \notin E \text{ for } \forall v_j \in V\}$. The next tasks to the first phase

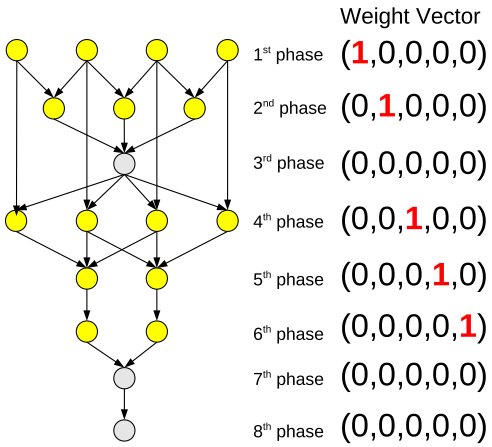


Figure 7. Weight Vector in Workflow DAG

are defined as the second-phase tasks. Thus, the i-th phase P_i is defined by tracing task dependencies. If a task depends on multiple tasks in different phases, its phase is defined as the next of the maximum phase. This algorithm is expressed as $P_i = \{v_k \mid (v_j, v_k) \in E \text{ for } \exists v_j \in P_{i-1} \text{ and } \nexists v_j \in P_{i'} \text{ for } i' \geq i\}$ for $i \geq 2$. Figure 7 shows the task phases defined for Montage workflow.

C. Weight Vector

After the task phase is determined, the weight vector w_i for the i-th phase is defined as follows. First, we define a subset of task phases as $Q = \{P_i \mid |P_i| \geq n\}$, here $|P_i|$ is the number of tasks in the i-th phase and n is the number of partitions. Then, we count the order of P_i in Q and define it as d_i. For example, if $|P_1| \geq n$, $|P_2| < n$ and $|P_3| \geq n$, then $Q = \{P_1, P_3\}$, $d_1 = 1$ and $d_3 = 2$.

Now we can define the weight vectors w_i. The length of weight vectors is the number of the phases of Q, i.e., $|Q| = \max\{d_i\}$. Then, for the i-th phases $P_i \in Q$, the d_i-th dimension of w_i is set to 1, all the other dimensions of w_i is set to 0. For the other phases $P_i \notin Q$, the weight vectors w_i is set to zero vector. For example, again if $|P_1| \geq n$, $|P_2| < n$ and $|P_3| \geq n$, the weight vectors are defined as $w_1 = (1, 0, ...)$, $w_2 = (0, 0, ...)$ and $w_3 = (0, 1, ...)$. For the Montage workflow, weight vectors are defined as shown in Figure 7.

We here explain how this proposed method works for equipartition in each phase. Consider a scalar-weighted DAG; each vertex weight is substituted with the value at the d_i-th dimensional of the weight vector. As a result, the vertex weight is 1 for the i-th phase, 0 for the other phases. If the standard graph partitioning is applied to this scalar-weighted graph, equipartition of vertices is achieved only for the i-th phase, and no constraint is imposed on the partitioning of the other phases since their weight is 0. This discussion holds for all the phases assigned with the dimensions of the weight vector. At the same time, the objective of minimizing the edge-cut cost is applied to the entire DAG. The MCGP algorithm thus achieves equipartition for all the phases simultaneously as well as minimizing the edge-cut cost. In the proposed method, the dimension of the weight vector is *not* allocated in task phases where the number of vertices is less than the number of partitions, i.e., P_i with $|P_i| < n$. It means that, in these phases, there is more than one partition to which no vertex belongs. We allocate zero vector to the vertex weight of these phases due to limitation of the METIS library, which is used as a graph partitioning algorithm.

In the standard graph partitioning, the vertex-weight represents the computation cost and the edge-weight represents the communication cost. This discussion still holds for the MCGP. It is noted that the comparison of the weight value makes sense only in one phase, and comparison of the weight values between different phases does not makes sense. In order to utilize the vertex-weight and/or edge-weight for the graph partitioning of the workflow DAG, the estimation of computation and communication cost is required. However, such precise estimation is not always possible. Therefore, in this paper, we assume that the communication cost and the computation cost are almost equal with each other. This assumption holds true in many cases of scientific application, especially exploiting data-parallelism.

IV. Workflow System

For the implementation and evaluation of the workflow scheduling proposed in the previous section, we use the Pwrake parallel workflow system [10], the METIS graph-partitioning library [11], and the Gfarm wide-area distributed file system [12].

A. Pwrake Workflow System

In this study, we use Pwrake [10], [13] for the evaluation of the proposed method. Pwrake is a workflow system to execute process in parallel using distributed computers. Pwrake is based on Rake, a build tool in written Ruby, a similar to UNIX *make*. Since the Makefile syntax is powerful also for defining scientific workflows, *make*-based workflow systems such as GXP make [14] have been developed. We employ Rake since it is more powerful and flexible than *make*. Rake is a standard tool which is bundled in Ruby version 1.9 or later. The syntax of Rakefile (the script file of Rake tasks) is same as the Ruby language. This feature is called an internal Domain Specific Language (DSL), where a host language is directly used as a DSL. The scripting power of Ruby enables the description of complicated science workflows. Pwrake extends Rake with the following features; the parallel execution with a thread pool, the remote execution by SSH, and the utilization of the Gfarm file system. However, Pwrake has evolved after that. The syntax extension described in [10] is abandoned.

The current version of Pwrake is same as Rake and all independent tasks can be executed in parallel. Pwrake has been already used in Bioinfomatics [15] and Earth science as well as Astronomy.

The Ruby language has a feature that the behavior of existing classes can be modified dynamically. Utilizing this feature, we add a proposed scheduling method using MCGP to Pwrake.

B. METIS Graph Partitioning Library

As an implementation of MCGP, we employ the METIS library version 5.0pre2. In order to use the APIs of METIS from Ruby, we develop a Ruby wrapper over the METIS APIs. The MCGP APIs of METIS are `METIS_mCPartGraphRecursive` and `METIS_mCPartGraphKway`. However, these APIs does not receive a parameter of partition weight, which is required for applying this function to un-uniform machine environment. Therefore, `METIS_mCPartGraphRecursive2` provided in ParMETIS is modified for METIS ver.5.0pre2, and used for the evaluation of this study.

C. Gfarm Distributed File System

The Pwrake system assumes the use of the Gfarm wide-area distributed file system [12]. The Gfarm file system utilizes local storage of distributed computation nodes. This design has an advantage in data locality. The Gfarm also brings benefits to distributed workflow execution in the following ways. The Gfarm provides the gfarm2fs command which mounts the Gfarm file system on a local file system. This feature enables an application program to access a Gfarm file in the same way as a normal file without any modification of the program. The Gfarm also provides a global directory tree for distributed storage. Files can be accessed from application programs without specifying data transfer explicitly. These features enable users to write workflow in a same way to execute on a local file system.

The Gfarm system has mechanisms to exploit data locality. When writing a file on the Gfarm file system, the same storage node is selected if possible. However, file systems do not manage task execution, but workflow systems do. The previous works on Pwrake [10] achieved an implementation of locality-aware task assignment which is based on the locality of an input file. It is noted that this assignment is *not* based on a workflow DAG in contrast to this work. The assignment for each task is determined immediately before the task execution. In the next section, this *immediate locality* scheme is compared with the proposed method of this paper.

V. Evaluation

In this section, the proposed method is evaluated for the Montage workflow using a cluster system.

Table I
Description of Input data

Each image file	
File size	2.1 MB or 1.7 MB
Pixel	512 × (1024 or 830)
Image area	8.5′ × (17.1′ or 13.8′)
Overall data set	
# of files	609
Total data size	1270 MB
Final image area	250′ × 250′

Table II
System used for Measurement

Cluster	InTrigger Kobe site
CPU	Xeon E5410 (2.33GHz)
Main Memory(GB)	16
# of used nodes	8
# of used cores	32

A. Experiment

As input data to the Montage workflow for this evaluation, a subset of 2MASS[16] image data is used. Table I shows the details of the image data set. The environment used for the performance evaluation is the Kobe site of InTrigger platform [17]. Table II shows the details of the used platform. The Gfarm file system is used for all of the measurement. The metadata server of Gfarm is installed in the Kobe site. In this evaluation, eight nodes are used for workflow execution. Therefore, the workflow DAG is partitioned into eight groups. As an initial condition, all the input files are stored in storage of a single node.

We investigate the following three kinds of task assignment scheme:

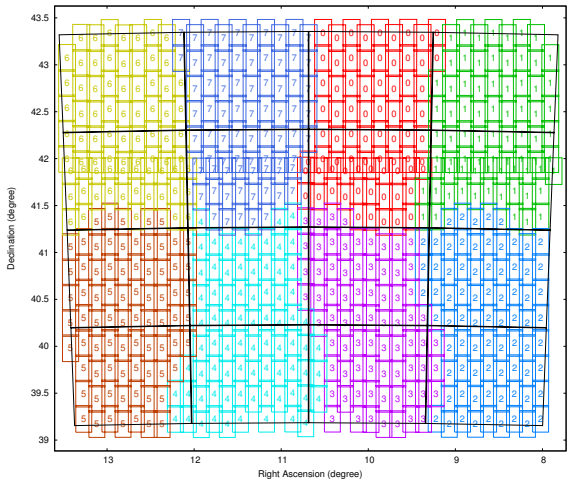
1) **Round-robin**: No data locality is considered.
2) **Immediate**: The locality is probed from the location of an input file immediate before task execution. This is studied in the previous work [10] (Section IV-C).
3) **MCGP**: The method proposed in this paper.

In addition, we investigate different two patterns of the Montage workflow which produce the same result image. The difference is in the mAdd_1 phase. In this phase, the target region is divided into $n \times m$ subregions or *tiles*. Then input images that overlap one of tiles are combined into a subimage by the mAdd program. The tile size can be specified as a parameter of a workflow. In this study, we investigate two patterns; 4×4 tiles and 8×4 tiles. Since the workflow is executed on eight nodes, the number of the tile regions assigned to each node is 2 and 4, respectively.

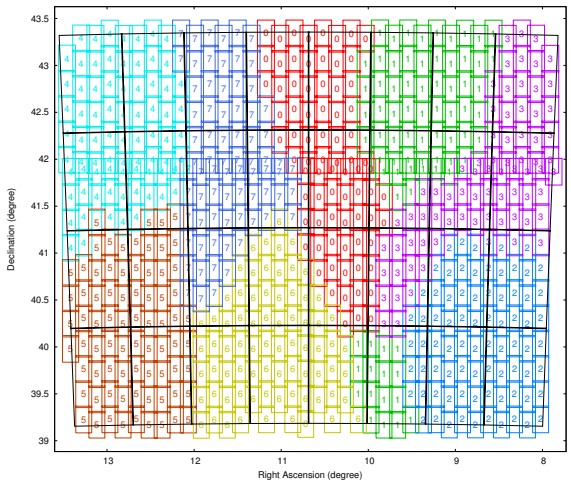
B. Task Assignment and Image Position

Before evaluating the workflow performance, we investigate how the proposed method assigns workflow tasks to computation nodes. Figure 8 shows the positions of input images in the celestial coordinate. In these figures, colors and numbers denote allocated nodes where the mProjectPP

(a) MCGP: 4×4 tiles in mAdd phase



(b) MCGP: 8×4 tiles in mAdd phase



(c) Assignment other than MCGP

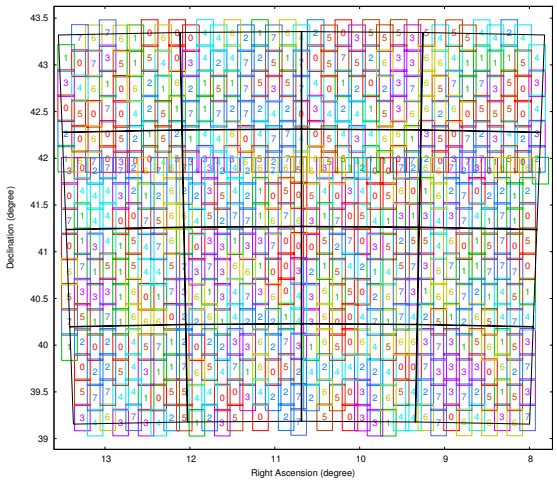


Figure 8. Image Position in Celestial Coordinate

task reads the corresponding image file. The black lattice grid indicates the tiles at the mAdd_1 phase. Figure 8 (a) and (b) show the result of the proposed method using MCGP. Among these figures, Figure 8 (a) shows the good accordance of node allocation with tile regions. This means that the graph partitioning of DAG corresponds to grouping in the spatial coordinate. It is noted that this node allocation is produced from only the workflow DAG. The proposed method does not require any spatial information.

On the other hand, the partition of Figure 8 (b) (8×4 tiles) does not always correspond to tile regions. In this figure, data locality in the mAdd_1 phase seems to be inferior to partition according to the tile grid. The effect of this partition on workflow performance is evaluated in the following subsections.

For comparison, Figure 8 (c) shows the result of task assignment other than MCGP. Both the scheduling schemes of 1) Round-robin and 2) Immediate result in a random task assignment as shown in Figure 8 (c). In the Immediate scheme, tasks in the first phase are randomly assigned due to the initial condition in which all the input files are stored in a single node.

C. Data Movement

The data size of files accessed for read during workflow execution is counted for each scheduling scheme. In every case, the total size of all data access is about 24 GB. The ratios of remote access during the workflow execution are plotted in Figure 9. In the case of 4×4 tiles, the ratios of remote access are 1) Round-robin: 87.9%, 2) Immediate: 47.4% and 3) MCGP: 14.0%. The result demonstrates that the data movement between nodes is dramatically reduced by the proposed method using MCGP. The Immediate scheme is worse than the MCGP because the initial node allocation pattern is like Figure 8 (c). In the case of 8×4 tiles, the remote access ratio of 3) MCGP is 15.4%. This is slightly larger from the 4×4 tile case, but still much smaller than the other scheme. From the data position shown in Figure 8 (b), the result of MCGP for 8×4 tiles seems not to be the best allocation. However, in view of data movement, the proposed method using the MCGP is enough for workflow scheduling. The result shows that the proposal method is effective for reducing the amount of data movement during workflow execution.

D. Workflow Execution Time

The elapsed time of the workflow is measured for each scheduling scheme, and the result is shown in Figure 10. The measurement is performed three times and average values are indicated. In the case of 3) MCGP, the elapsed time includes the graph partitioning time which is about 0.03 sec, small enough compared with the whole workflow.

In the case of 4×4 tiles, 3) MCGP scheme reduces the elapsed time by 31% (53 sec) from 1) Round-robin scheme

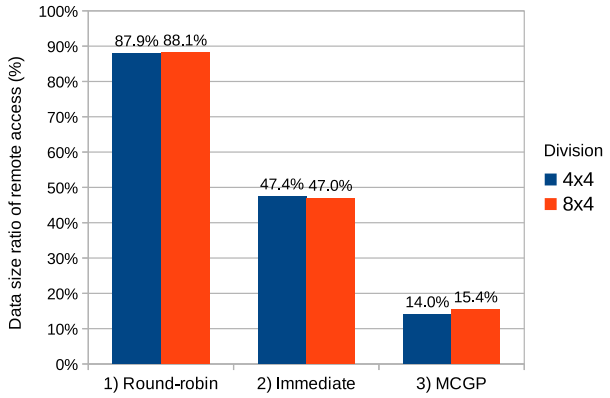


Figure 9. Ratio of Remote Data Access

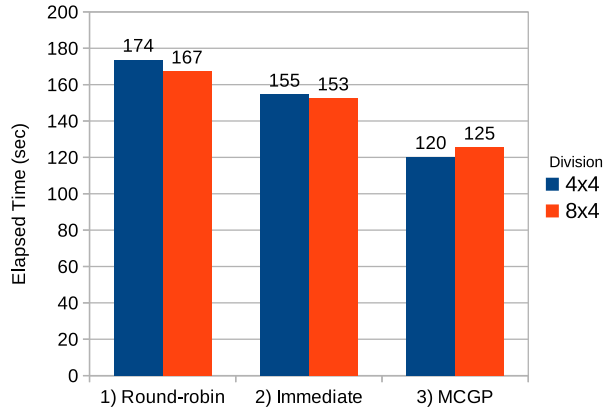


Figure 10. Elapsed Time of Workflow

and by 22% (34 sec) from 2) Immediate scheme. This result shows that the proposed method is effective to reduce the execution time of a workflow. The reduction ratio in elapsed time (31%) is smaller than that in remote access (74%) since the elapsed time includes time for read, write and compute, and the reduction of remote access affects only time for read. The small increase in elapsed time from 4×4 tiles to 8×4 tiles in the MCGP scheme is due to inferior partitioning, while it shows that both the cases of the MCGP are still effective for reducing elapsed time. It is noted that the reduction of 53 sec is achieved by paying the time for graph partitioning of 0.03 sec. From these results, the proposed method using the MCGP is powerful to improve the performance of workflow execution.

VI. Related Work

The idea using MCGP for partitioning multi-phase tasks was appeared in [8]. However, the application of MCGP to workflow scheduling has not been reported after that. The contribution of this paper is the implementation and evaluation of MCGP for workflow scheduling on a real workflow system. The use of the graph partitioning for the workflow DAG was proposed in the papers [18], [19], [20]. However, these papers did not propose the use of MCGP. The papers [21], [22] proposed the use of graph partitioning for partition the resources of a distributed system, but not the workflow DAG. In the study [23], the workflow clustering was applied to the Pegasus workflow system. They divided parallel tasks to generate abstraction workflows. The workflow clustering based on the spatial information of data was studied in [24] for a workflow of astronomical data analysis. However, such a method forces users to handle spatial information from data to give it to the workflow system. In contrast, our proposed method using MCGP requires only the workflow DAG. Given the workflow DAG, the workflow system automatically perform task scheduling to minimize the communication cost during workflow execution.

VII. Conclusion and Future Fork

Data-intensive science requires high-performance workflow systems where data-aware task scheduling is an important issue. We proposed a method applying *Multi-Constraint Graph Partitioning* (MCGP) to the workflow DAG, aiming at task assignment that minimizes data movement between processors and reduces workflow execution time. This method was implemented in the Pwrake workflow system with the Gfarm distributed file system. The proposed method was evaluated using the Montage astronomy workflow. As a result, the proposed method reduced the remote file access from 88% to 14% of total file access in data size and decreased workflow execution time by 31% (53 sec) in comparison to locality-unaware task assignment. This reduction was achieved by paying the graph partitioning time of only 0.03 sec. Our proposed method bases on only the workflow DAG and no other information such as data positions. This method is widely applicable to other scientific workflows.

Future work includes (1) the evaluation of workflows in which file sizes and computation costs are uneven, (2) the algorithm to define the order of task execution in each task phase (this is not defined in this study), (3) the multi-level partitioning for a platform where processors are connected by networks with different throughput, (e.g., a cluster of clusters), and (4) the application for heterogeneous clusters.

Acknowledgment

This work is supported by JST CREST research area, "Development of System Software Technologies for Post-Peta Scale High Performance Computing," and the MEXT Promotion of Research for Next Generation IT Infrastructure "Resources Linkage for e-Science (RENKEI)."

REFERENCES

[1] T. L. Casavant, Jon, and G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, pp. 141–154, 1988.

[2] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Record*, vol. 34, no. 3, p. 45, 2005.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.

[4] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 260–274, 2002.

[5] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, ser. Supercomputing '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–13.

[6] Montage, http://montage.ipac.caltech.edu/.

[7] K. Schloegel, G. Karypis, and V. Kumar, "Sourcebook of parallel computing," J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, ch. Graph partitioning for high-performance scientific simulations, pp. 491–541.

[8] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519 – 1534, 2000, graph Partitioning and Parallel Computing.

[9] K. Schloegel, G. Karypis, and V. Kumar, "Parallel multilevel algorithms for multi-constraint graph partitioning," in *Euro-Par 2000 Parallel Processing*. Springer, 2000, pp. 296–310.

[10] M. Tanaka and O. Tatebe, "Pwrake: A Parallel and Distributed Flexible Workflow Management Tool for Wide-area Data Intensive Computing," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 356–359.

[11] METIS, http://www.cs.umn.edu/~metis.

[12] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm Grid File System," *New Generation Computing*, vol. 28, no. 3, pp. 257–275, 2004.

[13] M. Tanaka, "Pwrake," http://masa16.github.com/pwrake.

[14] K. Taura, T. Matsuzaki, M. Miwa, Y. Kamoshida, D. Yokoyama, N. Dun, T. Shibata, C. S. Jun, and J. Tsujii, "Design and Implementation of GXP Make – A Workflow System Based on Make," *eScience, IEEE International Conference on*, vol. 0, pp. 214–221, 2010.

[15] H. Mishima, K. Sasaki, M. Tanaka, O. Tatebe, and K.-i. Yoshiura, "Agile parallel bioinformatics workflow management using pwrake," *BMC Research Notes*, vol. 4, no. 1, p. 331, 2011.

[16] M. F. Skrutskie, R. M. Cutri, R. Stiening, M. D. Weinberg, S. Schneider, J. M. Carpenter, C. Beichman, R. Capps, T. Chester, J. Elias, J. Huchra, J. Liebert, C. Lonsdale, D. G. Monet, S. Price, P. Seitzer, T. Jarrett, J. D. Kirkpatrick, J. E. Gizis, E. Howard, T. Evans, J. Fowler, L. Fullmer, R. Hurt, R. Light, E. L. Kopan, K. A. Marsh, H. L. McCallon, R. Tam, S. Van Dyk, and S. Wheelock, "The Two Micron All Sky Survey (2MASS)," *Astronomical Journal*, vol. 131, pp. 1163–1183, Feb. 2006.

[17] H. Saito, Y. Kamoshida, S. Sawai, K. Hironaka, K. Takahashi, T. Sekiya, N. Dun, T. Shibata, D. Yokoyama, and K. Taura, "InTrigger: A Multi-Site Distributed Computing Environment Supporting Flexible Configuration Changes," *IPSJ SIG Technical Report 2007-HPC-111*, vol. 2007, no. 80, pp. 237–242, 2007.

[18] F. Dong and S. Akl, "Two-phase computation and data scheduling algorithms for workflows in the grid," in *Parallel Processing, 2007. ICPP 2007. International Conference on*. IEEE, 2007, pp. 66–66.

[19] S. Kalayci, G. Dasgupta, L. Fong, O. Ezenwoye, and S. Sadjadi, "Distributed and adaptive execution of condor dagman workflows."

[20] O. Sonmez, N. Yigitbasi, S. Abrishami, A. Iosup, and D. Epema, "Performance analysis of dynamic workflow scheduling in multicluster grids," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 49–60.

[21] S. Kumar, S. Das, and R. Biswas, "Graph partitioning for parallel applications in heterogeneous grid environments," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. IEEE, 2002, pp. 66–72.

[22] C. Lin, C. Shih, and C. Hsu, "Adaptive dynamic scheduling algorithms for mapping ongoing m-tasks to pr 2 grid," *Journal of information science and engineering*, vol. 26, pp. 2107–2125, 2010.

[23] E. Deelman, J. Blythe, A. Gil, C. Kesselman, G. Mehta, S. Patil, M. hui Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," 2004, pp. 11–20.

[24] L. Meyer, J. Annis, M. Wilde, M. Mattoso, and I. Foster, "Planning spatial workflows to optimize grid performance," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2006, pp. 786–790.