File-Access Patterns of Data-Intensive Workflow Applications and their Implications to Distributed Filesystems

Takeshi Shibata
University of Tokyo
Department of Information and Communication Engineering
Graduate School of Information Science and Technology
shibata@logos.ic.i.u-tokyo.ac.jp

SungJun Choi
University of Tokyo
Department of Information and Communication Engineering
Graduate School of Information Science and Technology
demyan@logos.ic.i.u-tokyo.ac.jp

Kenjiro Taura
University of Tokyo
Department of Information and Communication Engineering
Graduate School of Information Science and Technology
tau@logos.ic.i.u-tokyo.ac.jp

ABSTRACT

This paper studies five real-world data intensive workflow applications in the fields of natural language processing, astronomy image analysis, and web data analysis. Data intensive workflows are increasingly becoming important applications for cluster and Grid environments. They open new challenges to various components of workflow execution environments including job dispatchers, schedulers, file systems, and file staging tools. The keys to achieving high performance are efficient data sharing among executing hosts and locality-aware scheduling that reduces the amount of data transfer. While much work has been done on scheduling workflows, many of them use synthetic or random workload. As such, their impacts on real workloads are largely unknown. Understanding characteristics of real-world workflow applications is a required step to promote research in this area. To this end, we analyse real-world workflow applications focusing on their file access patterns and summarize their implications to schedulers and file system/staging designs.

Keywords

workflow applications, distributed filesystems

1. INTRODUCTION

Workflow facilitates integration of individually developed executables, making parallel processing readily accessible to domain experts. Thus it has become an important discipline in various fields including natural science, engineering, and information processing. Many systems for executing workflows have been developed [1, 2, 3]. More recently, programming paradigms and systems specifically designed for large data processing such as MapReduce [4], Hadoop[1], and Dryad [5][2] made it popular to utilize parallel processing without an involved effort of parallel programming. An obvious common goal of these systems is efficient execution of workflows. To this end, there have been efforts on various components of workflow engines including scheduling algorithms [6, 7, 8, 9, 10, 11, 12], data transfers [13, 14], and fast dispatchers [15]. There are efforts focusing on scheduling with data transfer costs taken into account [16, 17, 18, 19] A good survey on scheduling algorithm is in [20].

Despite their importance, practical evaluation of workflow systems have been rare and remain difficult, mainly due to lack of commonly accessible benchmarks. Even with a real application, translating the result on a particular platform into a generally acceptable observation on workflow systems is difficult because performance of a workflow depends on so many components of the environment such as nodes, networks, file systems, and so on. This is particularly so because workflows typically consist of many sequential executables each of which may have unknown sensitivities to their environments. Most existing studies on scheduling algorithms therefore have been based on simulation with synthetic workloads such as randomly generated task graphs. Bharathi et al. [21] is one of the few studies that systematically characterizes several real workflow applications. The present paper shares the same spirit as theirs, but pays a special attention to IO (file access) behaviors of applications.

A single workflow generally consists of a set of tasks each of which may communicate with (i.e. depends on or is depended upon) another task in the workflow. Since a task is typically a sequential (single node) application, a data transfer among tasks is generally handled by the workflow system. Data may be implicitly transferred via a shared file system or explicitly moved by a staging subsystem. Either case, they generally go through a secondary storage to ensure a certain degree of fault tolerance—that a workflow

[1]http://hadoop.apache.org/
[2]http://research.microsoft.com/en-us/projects/Dryad/

can restart from a point where a job failed. As such, understanding IO behaviours of real workflow applications is very important for implementing efficient workflow systems. It is particularly so for data-intensive workflows, which are becoming major targets of workflow systems.

To this end, this paper shows the result of our study on five real workflow applications in the fields of natural language processing, web data analysis, and astronomy image analysis. With this study, we like to provide data points by which we can investigate important design questions for implementing workflow and related subsystems such as file systems. Relevant questions include:

- How effective is it to place jobs according to file affinity, and what is the tradeoff between the complexity and benefit of scheduling algorithms?
- What are advantages and disadvantages of using shared file systems (such nas NFS[22], Lustre[3], GPFS [23][4], and Ceph[24]) over using explicit staging?
- How much we can gain from staging systems that transfer files directly between compute hosts rather than indirectly through a central home node?
- Likewise, how much we can gain from user-level file systems such as Gfarm [25] or GMount [26] that can use *any* compute host as a storage server?

2. RELATED WORK

2.1 Workflow Studies

Bharathi et al. [21] is the closest to the present work. It studies five real workflow applications. It analyses dependency graphs of each workflow and identifies common communication and dependency patterns such as pipeline, data distribution, aggregation, and redistribution. It also shows statistics on job granularities. In addition to dependency patterns, we also study file access behaviors and discuss their implications to file system or staging designs.

2.2 Executable Workflows

Montage [27] is one of the few workflow applications that can be readily downloaded and executed by others. Both Bharathi et al. [21] and our work include this application as a target of their studies. We will provide four more executable applications with annotations that can be shared by researchers in this community.

2.3 Workflow Collections

For publicly accessible collection of workflows, Bharathi et al. also provides WorkflowGenerator web site[5], which generates DAGs whose nodes are tasks annotated with runtime and files used. Files are annotated with their sizes, so dependencies and communication volumes can be reconstructed from the DAGs. We are preparing a web site that provides similar data for applications studied in this paper. In addition, we will provide more comprehensive traces of file access behaviors, as well as their executables.

myExperiment[6] is a repository of workflow descriptions, currently providing more than 750 workflows. Each workflow is given as a description for a particular system (mostly for Taverna [3]), and not annotated with profile information such as runtimes and file sizes. The number of workflows we are going to provide is much smaller, but we are trying to provide them in a form executable by others.

2.4 Workflow Scheduling Techniques and Distributed File Systems

There have been efforts on various components of workflow engines including scheduling algorithms [6, 7, 8, 9, 10, 11, 12]. However, most of those methods require the complete form of DAGs for workflows and completion time, input/output file size for each jobs, etc. It is difficult to apply scheduling techniques such as HEFT without those information.

Especially, The complete form of DAGs requires workflow descriptions to have explicit declarations of input/output files for each job. Because the workflow descriptions we examined in this paper has no explicit form of input/output files, some systems which help dispatched jobs find required files and register output files is required. Distributed file systems are able to be used for that purpose.

3. WORKFLOW EXECUTION AND FILE ACCESS LOGGING

3.1 Workflow Engine

Workflows described in this paper are described in Makefiles and a parallel and distributed make engine called GXP Make[7] was used to execute them. Make is a convenient tool for describing jobs with dependencies and can naturally be used for describing executing workflows.

Except for Montage, all workflows are written in Makefile by their original authors. For Montage, one of the tools for it generates an abstract workflow in an XML format called DAX, used Pegasus. Output DAX files are converted to Makefiles. Every job description by the tag ⟨job⟩ is converted to a "rule" of makefile; commands of a rule are extracted from ⟨job⟩ tags, whereas dependency (prerequisite) and target filenames from ⟨uses⟩ tags with arguments 'link = "input"' and 'link = "output"', respectively.

GXP Make runs on top of an existing Make implementation GNU Make, which supports parallel execution for a single multiprocessor node. GXP Make extends this function to parallel execution in distributed environments by intercepting its child process invocations. Processes forked by the underlying GNU make enter a queue managed by GXP make; It then dispatches jobs to available workers that have been acquired by any underlying resource manager such as TORQUE, SGE, SSH, etc. Input and output files of tasks are stored in a single network file system, such as NFS, Lustre, GFarm, etc.

The choice of this particular workflow system is largely orthogonal to the results presented in this paper. It is not difficult to automatically generate descriptions for other work-

[3]http://www.sun.com/software/products/lustre/
[4]http://www-03.ibm.com/systems/clusters/software/gpfs/
[5]http://vtcpc.isi.edu/pegasus/index.php/WorkflowGenerator
[6]http://www.myexperiment.org/
[7]http://www.logos.ic.i.u-tokyo.ac.jp/gxp/

flow system such as DAGMan, given file access logging methods presented in the next section.

3.2 File Access Logging

In data intensive applications, it is important to examine how much data are transferred over the network, when and where these transfers happen, and where bottlenecks are. When data are transferred mainly as files through a distributed file system, the detail of file access is required to extract such information. In order to record file access operations, we developed a logging tool which is specialized for GXP Make using a tool called Filesystem in Userspace (FUSE). Table 1 shows the recorded data for each access.

Table 1: logged data par a file access operation

job id	hostname:processId
operation	read, write, open, create, getattr, mkdir, etc.
data id	filename
date	execution time of each operation in nsec. from the epoc
time	blocking time during each operation
size	size for each read and write

A file system using our logging tool provides a mount point under which the target directory is mirrored. When a process accesses a file or a directory under the mount point, one of our registered callback routines is called by FUSE, in which the access is logged along with the client process id.

We need to translate the process id to the identity of a job (note that a job may consist of multiple processes). GXP Make gives a unique number to each job through an environment variable (`GXP_WORK_IDX`). When our logging tool receives a callback, it records `GXP_WORK_IDX` of the client process if it is the first callback from that process.

When the file system is unmounted after logging all file access of a workflow, access logs by all hosts are collected to form a database. Dependencies among dispatched jobs and accessed files are inferred from those files. Dependencies are inferred as follows. (1) If a job X accessed a file A with any file operation, X is related to A. (2) If X created A, then A depends on X ($X \rightarrow A$), otherwise X depends on A ($A \rightarrow X$).

After the database is created, the workflow DAG is generated as a clickable HTML file with information for the workflow execution. Statistics for file-access patterns of each job and each category for jobs are also extracted from the database. A category for jobs is a set of jobs which are derived from the same line in the makefile, or the same command without input file.

All dispatched command lines, or jobs, are divided into categories by the following steps.

1. Strings which is called normal commands are extracted from the given makefile by shallow pursing. Each normal command is basically corresponding to single line of the makefile.
2. The distance of dispatched command lines from normal commands are measured by edit distance. Each dispatched command line is assigned to a category which is represented by the nearest normal command.

For instance, if the following block is in a makefile :

```
DIR := some_dir
$(DIR)/%.input : $(DIR)/%.output
        cmd $< > $@
```

then the corresponding normal command is

```
cmd some_dir/%.input > some_dir/%.output.
```

Although another FUSE-based tool ParaTrac [28] offers similar results to ours, our tool is implemented independently for different purpose. While ParaTrac is a general purposes profiling tool, this profiling tool is specialized for GXP Make workflow system and has several different functions from ParaTrac. For example, while our tool can not produce fine-grade analysis such as statistics for time when 'read' operations occur in processes, it is able to divide jobs into categories for given makefiles and show their statistics, extract user cpu time of each job and show critical path of the workflow, produce workflow DAGs in HTML files with information of workflow execution, and so on.

4. WORKFLOW APPLICATIONS

In this section, details and profiling result for the five workflow applications [8] are given. They are:

- Medline to Medie Indexing
- Case Frames Construction
- Similar Pages Extraction
- Supernovae Explosion Detection
- Montage

The first two are applications in natural language processing; Similar Pages Extraction is a data mining application applied to a large collection of web pages; The last two are image analysis applications in astronomy.

The experimental environment is a single cluster composed of 15 nodes (120 CPU cores) with a single NFS server. They are connected by Gigabit Ethernet with bonding (2Gbps/node). We use GXP make as the workflow engine.

Details of file-access patterns are logged and analysed by means of the previous section.

Following subsections describe each application with more details.

[8] https://www.intrigger.jp/wiki/index.php/Applications

4.1 MEDLINE to MEDIE

MEDLINE to MEDIE Indexing (hereafter M2M) workflow creates indexing database for a running search engine called MEDIE.[9] It indexes all registered abstracts in the MEDLINE, the world largest biomedical text database. It indexes and retrieves sentences with their semantic subjects, verbs, and objects, enabling queries such as "what does p53 activate?" To this end, its indexing involves deep natural language processing including parsing, requiring a large amount of CPU times and intermediate data storage. M2M takes a compressed XML text files as its input, each containing a number of biomedical papers in MEDLINE. Most tasks in M2M input and output compressed XML files.

Figure 1 shows a sub-workflow of M2M for a single input file. A circle is a single task, and a square represents a file created or read by tasks. An arrow from a task to a file indicates that the task created the file, and one from a file to a task that the task opened the file for reading. The most time consuming part is parsing done by a parser called Enju. This is parallelized within each input file by splitting the intermediate file right before the parsing into small chunks. This can be seen as a number of horizontally aligned boxes around the middle of the graph.

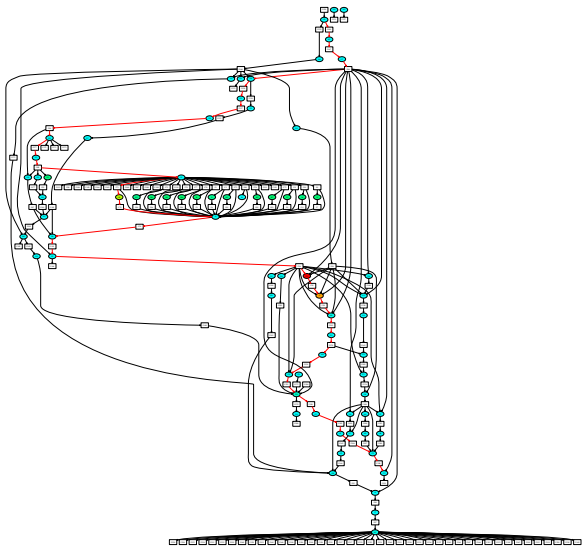
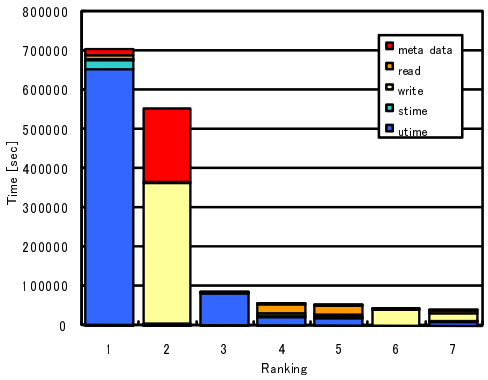


Figure 1: MEDLINE to MEDIE

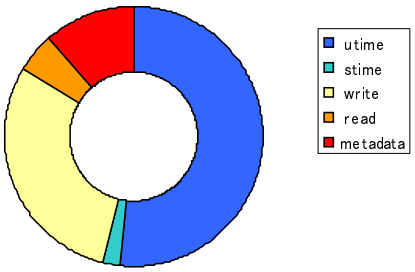
Each input file results in 68 independent jobs, including natural language processing, database creation, managing and conversion of files described in XML, etc. Resource consuming tasks are listed below in the decreasing order for execution time.

1. Enju : HPSG parser, which constructs of predicate argument structures.
2. Akane : Protein-protein interaction inference from sentences.
3. Genia-tagger : part-of-speech tagging and shallow parsing for biomedical text.

[9]http://www-tsujii.is.s.u-tokyo.ac.jp/medie/



(a) Time detail of each job



(b) Time detail for whole workflow

Figure 2: MEDLINE to MEDIE :Time detail of each job and of the whole workflow

4. Medie-ner-disamb.: named entity recognition for biomedical terms, such as protein names, and removing ambiguity from them.
5. Medie-ner-filter: preprocess for Medie-ner-disamb.
6. MakeDB : creation index database for MEDIE.
7. Som-merge : merging intermediate XML files.

Figure 2(a) shows a timing detail of the fifteen most time consiming tasks for 99 input files (491 MB). In the experiment, Enju is executed in 3584 times and each of the other tasks is executed 99 times (once for each input file). While the most time-consuming task, Enju, is clearly cpu-intensive, the second one, Akane, is very IO intensive. Moreover, it spends a significant amount of time on meta-data accesses. Overall, 46% of the whole time are consumed by file IO and metadata access (Figure 2(b)). Details of metadata accesses are shown in Table 2.

4.2 Similar Pages Extraction

Similar Pages is a workflow that takes a large number of texts collected from the web as input and analyses them. It outputs all pairs of 'similar' sentences, which have Hamming distances smaller than a specified threshold. The key algorithm, called Sachica, is able to enumerate all such pairs

Table 2: detail of file metadata access for M2M

	getattr	readdir	open
count	1,749,482	99	3,418,394
aggregate time[sec.]	84,266	4	128,671
average time[msec.]	48	40	38
time ratio	0.39	0	0.59
	create	mkdir	unlink
count	21,642	496	4,871
aggregate time[sec.]	1,199	33	1,069
average time[msec.]	55	67	219
time ratio	0.01	0	0.01

rapidly [29]. The dependency DAG of Similar Pages Extraction workflow is shown in Figure 3. It has three kind of tasks:

- Mkbins : gather all input text files and split them into a given number (N) of files.
- Sachica1 : find pairs of similar subsequences in a single file.
- Sachica2 : find pairs of similar subsequences in two files, one from each of them.

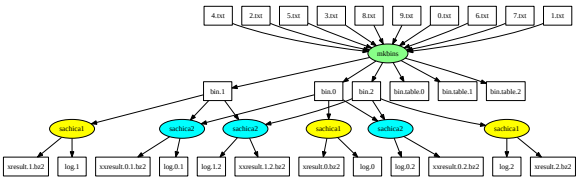


Figure 3: Similar Pages Extraction workflow

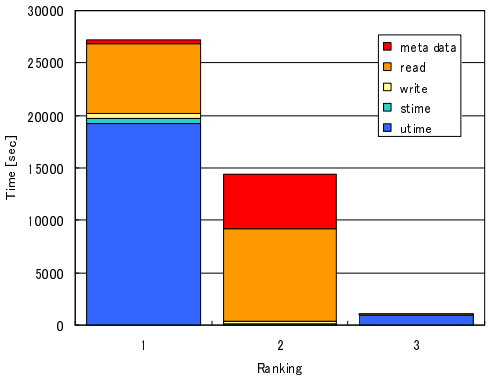
Characteristic of the dependency DAG for Similar Pages is that Sachica1 and Sachica2 have N and $N(N-1)/2$ jobs respectively, where N is the number of files Mkbins outputs. Each split file is read N times from Sachica1 and Sachica2.

Figure 4 shows an experimental result for about 1 million (total 5 GB) input files with $N = 30$ (Mkbin:1, Schica1:30, Schica2:435). Mkbin is very file-access intensive, whose cost are mainly IO operation 'read' and metadata operations 'get attribute' and 'open'. Overall, file-access cost occupies the half of its execution time and 'read' is the main reason of the file-access costs.

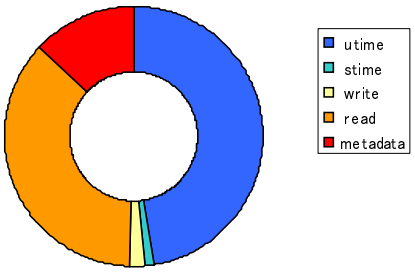
4.3 Case Frames Construction

Case Frames Construction is a workflow to create data structures in natural language processing called case frames [30, 31]. Case frames, which are frequent patterns of predict-argument structures, are generated by analysing large mount of text collected from the web. The workflow is mainly constructed from the following jobs in the increasing order with consuming time(Figure 7(a)):

1. parse-comp : a parser for Japanese texts with KNP [30] including named entity recognition.



(a) Time detail of each job



(b) Time detail for whole workflow

Figure 4: Similar Pages :Time detail of each job and of the whole workflow

2. make-caseframe : a job that creates case frames by extracting reliable predicate-argument examples from the parsing result.
3. cf-distribute-rest : a job that adds all case frames to clusters made by cf-clustering.
4. cf-clustering : clustering highly frequent case frames and merging them with similarity.
5. calc-of-similarity : computing similarities for each pair of case frames.

The DAG for the workflow with 4 input files is shown in Figure 5. As the figure shows, all files output by the upstream jobs are gathered into one large file, and it is split into files with other size, and finally they are combined into a result file. So parallelism of the workflow has two peaks. Figure 6 shows the number of concurrent jobs with respect to the elapsed time (in seconds) of an execution.

In the experiment, 59 % of the whole time is spent for file access, while the ratio of time for each job has large variance (Figure 7).

4.4 Finding Supernovae

Finding Supernovae is a workflow to find coordinates where supernovae appears to be from astronomical data. The workflow uses the same software and data with Data Analysis

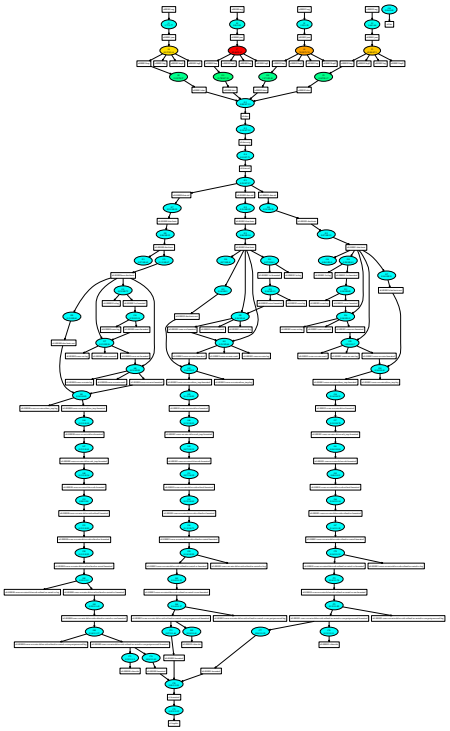


Figure 5: Case Frames Construction

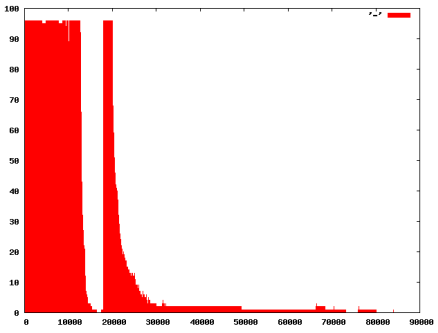
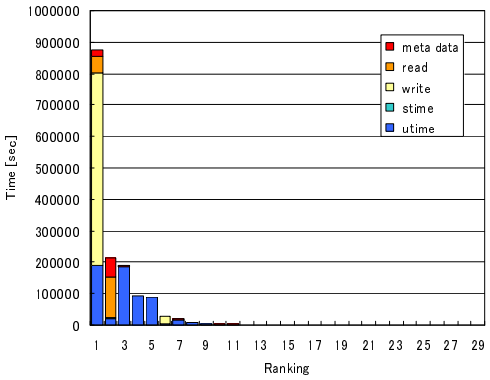
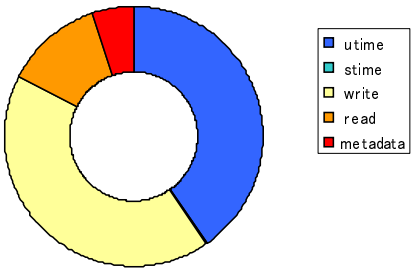


Figure 6: Parallelism of an execution for Case Frames Construction



(a) Time detail of jobs



(b) Time detail for whole workflow

Figure 7: Details for Case Frames Construction

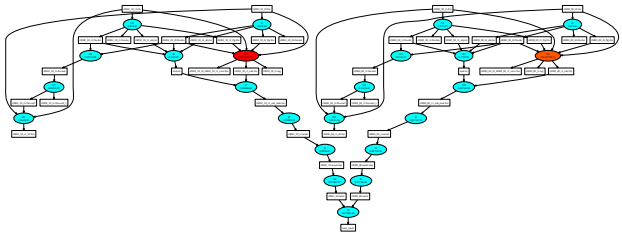


Figure 8: Finding Supernovae

Challenge on IEEE Cluster/Grid 2008 [10]. Each pair of astronomical images which are for the same position but different time is compared with each other and the positions of candidates for supernovae are listed up. Finally the candidate positions are gathered into one file. Figure 8 shows the DAG of the Finding Supernovae for 2 pairs of input images.

In this workflow, the most heavy job (imbus3vp3) is relatively compute-intensive while other jobs are not.

4.5 Montage

We execute Montage DAX and, take log and generate the dependency graph(Figure 11).

Each input file is a astronomic image file, and the workflow gathers them into one picture. In the experiment we use 3156 2MASS(J)images as input, where each size of them is about 2MB .

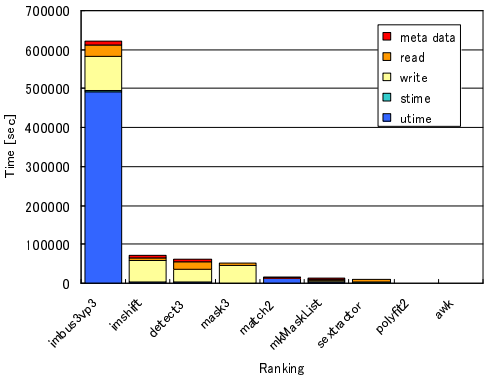
As Figure 11(b) shows, almost of the aggregate time is consumed for file access. Concentrating all files to single file server becomes the bottleneck and degrades performance of the workflow.

Percentage of cpu usage time to aggregate time becomes larger as the input size (Figure 11(c)), which means that concentrating data access degrades throughput of the file server.

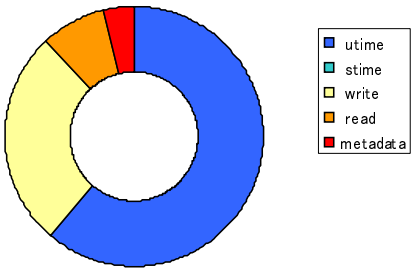
5. COMPARISON FOR DESIGNS OF WORKFLOW SOLVERS

In this section, we investigate various cost for designs of workflow solvers such as staging, centorized file system, schedul-

[10]http://www.cluster2008.org/challenge/



(a) Time detail of jobs



(b) Time detail for whole workflow

Figure 9: Details for Finding Supernovae

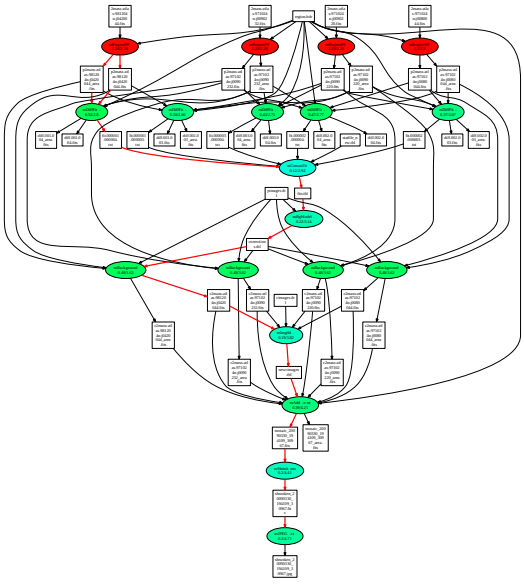
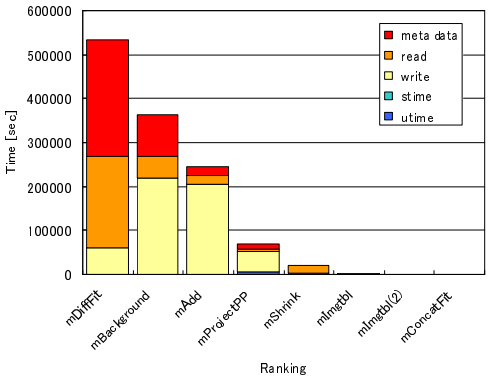


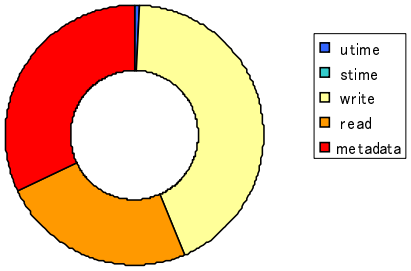
Figure 10: montage workflow

ing, based on logged data for workflows in the previous section.

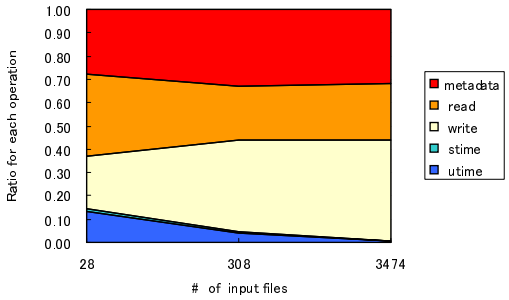
5.1 Centralization Costs



(a) Time detail of each job



(b) Time detail for whole workflow



(c) ratio of time for each oparations w.r.t. the number of input files

Figure 11: Montage :Time detail of each job and of the whole workflow

A staging is a process to transfer required files to nodes in advance. Many staging methods send all updated and created files to central servers when jobs finish. Since intermediate files are not sent directly from the node who output them to other nodes in those methods, called centralized staging, bandwidth and latency to access for central servers frequently becomes a bottleneck especially for data intensive workflows.

Many centralized file systems, such as NFS and LUSTRE, also have the same problem, whereas distributed file systems, such as Gmount and Gfarm, can be more effective since they locate files on local disks of nodes and send them

if necessary.

While input and output files of a workflow are usually required to be located on file system, intermediate files, which are both read and written by jobs of the workflow, are not.

We assume that files which are only read during the workflow execution are the input files and that files which are only written are the output files. Table 3 shows input, intermediate and output aggregate file size and the intermediate rate ($\mathbf{intermediate}/(\mathbf{input}+\mathbf{intermediate}+\mathbf{output})$) for each workflow in the previous section. It is possible to avoid 40-88% of file transfer to central servers on centralized file systems and staging systems that can be become a bottleneck for the whole workflow performance if intermediate files does not send to the central servers.

Table 3: ratio for the aggregate size of intermediate files

	input	interm.	output [GB]	interm. rate
m2m	0.51	97.43	13.25	0.88
similarpages	2.49	2.49	0.97	0.42
caseframes	0.73	46.74	58.51	0.44
supernovae	77.91	103.48	78.26	0.40
montage	7.26	72.84	21.44	0.72

5.2 Locality-aware Scheduling

When a scheduler for workflows decides the compute nodes where each job are dispatched, it is important to consider the cost to access files located on remote nodes. Jobs are preferred to be lunched at a compute node where as many input files are located on the local disk or memory.

In many scheduling methods like HEFT [7], the cost to access files on remote nodes are considered when jobs are assigned to compute nodes, as well as predicted computation time for jobs, length of critical path, etc. Let us define that a job Y depends on a job X if some output file of X is input file of Y. If a job X has some jobs that depend on only X, it is very likely that, at least, one of jobs which depends on X is assigned to the same node with X with any criterion for assignments of jobs. In order to measure how much size of files are able to read from local in locality-aware scheduling methods, for each job X, a job Y which depends on X are randomly selected among jobs depends on X, and X and Y are assigned the same node. Talbe 4 shows the result of the assignment for each workflow. While Case Frames Construction has high local read rate $\mathbf{local}/(\mathbf{local}+\mathbf{input}+\mathbf{remote})$, Similar Pages Extraction does not. The difference of them mainly arise from the difference of the average number of branches in the DAGs of the workflows. If a workflow has the small averaged number of branches for a job in the DAG, considering locality awareness became more important. As Figure 3 and 5 shows, most jobs of Case Frames Construction have one branch while those of Similar Pages Extraction have 30 branches.

5.3 Costs for Metadata Operations

Metadata operations often be non-negligible costs on a single file system. Table 5 shows the experimental result in the previous section shows. The reason for the cost of metadata operations hardly depends on bandwidth limitation but on access latency to the metadata server on the file system with keeping consistency. Thus metadata operation cost should be able to reduce much less if staging is used in stead of a single file system, since metadata access is only in local nodes and keeping consistency with other nodes is not required with stating methods.

Table 4: local read for locality-aware scheduling

	read size [GB]			
	input	local	remote	local read rate
m2m	0.51	69.56	119.35	0.36
similarpages	2.49	0.17	74.61	0.002
caseframes	0.73	46.35	0.99	0.96
supernovae	233.73	37.50	67.54	0.11
montage	9.13	56.93	235.86	0.18

Table 5: the ratio of metadata access costs

	ratio for whole	ratio for file-access
m2m	0.12	0.21
similarpages	0.13	0.18
caseframes	0.05	0.09
supernovae	0.04	0.10
montage	0.32	0.32

6. CONCLUSION

In Section 5, advantages of modifying file systems, staging methods and scheduling methods are estimated by analysing dependency graphs generated from logs. Since those workflows have no explicit declaration of input and output files for each job, dependencies between files and jobs are created by analysing file access logs. Three key insights for performance are drawn from the experiments. First, the total load on the central file server decreases by $40-88\%$ if intermediate files are sent not going through the central file servers, but directly from the producer to the consumer. Secondary, if locality-aware scheduling methods are applied, their effectiveneses highly depend on the shapes of DAGs (the reductions are from $0.002-0.96$). In particular, it largely depends on the number of children per job. Lastly, $9-32\%$ of whole file access cost can be eliminated when a staging is used instead of a file system, because a large part of file access costs comes from the metadata operation latency between clients and servers.

Our future direction includes using profiled information to statistically infer various information about jobs such as names of input files, names of output files, size of output files, and the execution time of the job from the command line. If many of them are successfully estimated, it is possible to build a framework in which jobs can be described without explicit staging directives, files transparently shared across nodes, yet executed very efficiently.

Acknowledgment

This work was supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan) and the MEXT Grant-in-Aid for Scienti?c Research on Priority Areas project “New IT Infrastructure for the Information-explosion Era”

7. REFERENCES

[1] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the condor experience." *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005.

[2] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz., "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming Journal*, vol. 13, no. 3, pp. 219–237, 2005.

[3] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004, p. 137 窶 50.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2007 Eurosys Conference*, 2007.

[6] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson, "Scheduling strategies for mapping application workflows onto the grid," in *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, July 2005, pp. 125–134.

[7] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[8] Z. Yu and W. Shi, "An adaptive rescheduling strategy for grid workflow applications," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, March 2007, pp. 1–8.

[9] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of Computer Science*, vol. 3, no. 2, pp. 94–103, 2007.

[10] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, April 2004, pp. 111–.

[11] W. Guo, W. Sun, W. Hu, and Y. Jin, "Resource allocation strategies for data-intensive workflow-based applications in optical grids," in *Communication systems, 2006. ICCS 2006. 10th IEEE Singapore International Conference on*, Oct. 2006, pp. 1–5.

[12] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 2, pp. 759–767, 2005.

[13] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2009, pp. 39–48.

[14] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, 2004, pp. 342–349.

[15] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a fast and light-weight task execution framework," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–12.

[16] G. Khanna, N. Vydyanathan, U. Catalyurek, T. Kurc, S. Krishnamoorthy, P. Sadayappan, and J. Saltz, "Task scheduling and file replication for data-intensive jobs with batch-shared i/o," in *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, 0-0 2006, pp. 241–252.

[17] I. Raicu, I. T. Foster, Y. Zhao, P. Little, C. M. Moretti, A. Chaudhary, and D. Thain, "The quest for scalable support of data-intensive workloads in distributed systems," in *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2009, pp. 207–216.

[18] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002, pp. 352–358.

[19] T. Kosar, "A new paradigm in data intensive computing: Stork and the data-aware schedulers," in *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, 0-0 2006, pp. 5–12.

[20] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Tech. Rep., January 2006.

[21] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, Nov. 2008, pp. 1–10.

[22] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "Nfs version 3 - design and implementation," in *In Proceedings of the Summer USENIX Conference*, 1994, pp. 137–152.

[23] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *In Proceedings of the 2002 Conference on File and Storage Technologies (FAST*, 2002, pp. 231–244.

[24] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006, p. 307 窶 20.

[25] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, "Grid datafarm architecture for petascale data intensive computing," in *Cluster Computing and*

the Grid, 2002. 2nd IEEE/ACM International Symposium on, May 2002, pp. 102–102.

[26] N. Dun, K. Taura, and A. Yonezawa, "Gmount: An ad hoc and locality-aware distributed file system by using ssh and fuse," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 188–195.

[27] "Montage: An astronomical image mosaic engine," http://montage.ipac.caltech.edu/.

[28] N. Dun, K. Taura, and A. Yonezawa, "Paratrac: A fine-grained profiler for data-intensive workflows," in *to appear; 19th IEEE Symposium on High Performance Distributed Computing*, 2010.

[29] T. Uno, "A new approach for speeding up enumeration algorithms," in *Proceeding of ISAAC98(Lecture Note in Computer Science 1533, Springer-Verlag, Algorithms and Computation)*, 1998, pp. 287–296.

[30] N. K. Daisuke Kawahara and S. Kurohashi, "Japanese case structure analysis by unsupervised construction of a case frame dictionary," in *International Conference on Computational Linguistics (COLING2000)*, 2000, pp. 432–438.

[31] R. Sasano, D. Kawahara, and S. Kurohashi, "The effect of corpus size on case frame acquisition for discourse analysis," in *the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT2009)*, 2009, pp. 521–529.