OPTIMIZING TASK GRANULARITY FOR LARGE-SCALE SCIENTIFIC WORKFLOWS

by

Weiwei Chen

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

Jun 2014

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

Abstract

Scientific workflows are a means of defining and orchestrating large, complex, multi-stage computations that perform data analysis and simulation. Task granularity optimization is a key problem in the execution of workflows because they often involve large overheads and computations that must be optimized. Traditionally, optimizing task granularity at the workload sphere without consideration of data dependencies has been widely discussed in the literature. However, the recent emergence of executing large-scale scientific workflows on modern distributed environments, such as grids and clouds, brings new challenges to the optimization of task granularity and requires novel mechanisms and new knowledge in several aspects. Our work investigates the key concern of task granularity studies in scientific workflows and proposes a series of innovative partitioning and clustering methods to improve the overall workflow performance, including runtime performance, fault tolerance, and data locality. Simulation-based and experiment-based evaluation verifies the effectiveness of our approaches.

Chapter 1

Introduction

Over the years, with the emerging of the fourth paradigm of science discovery [39], scientific workflows continue to gain their popularity among many science disciplines, including physics [27], astronomy [77], biology [50, 66], chemistry [98], earthquake science [56] and many more. Scientific workflows increasingly require tremendous amounts of data processing and workflows with up to a few million tasks are not uncommon [13]. Among these large-scale, loosely-coupled applications, the majority of the tasks within these applications are often relatively small (from a few seconds to a few minutes in runtime). However, in aggregate they represent a significant amount of computation and data [27]. For example, the CyberShake workflow [58] is used by the Southern California Earthquake Center (SCEC) [86] to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. CyberShake workflows composed of more than 800,000 jobs have been executed on the TeraGrid [91]. When executing these applications on a multi-machine parallel environment, such as the Grid or the Cloud, significant system overheads may exist. An overhead is defined as the time of performing miscellaneous work other than executing the user's computational activities. Overheads adversely influence runtime performance of large-scale scientific workflows and cause significant resource underutilization [17]. In order to minimize the impact of such overheads, task clustering [82, 40, 105] and workflow partitioning [47, 38] have been developed to increase their computational granularity and thereby reduce the system overheads. Task clustering is a technique that merges small tasks into larger jobs so that the number of computational activities is reduced. It has been widely used in executing such large-scale scientific workflows and has demonstrated its great effort [82]. Workflow partitioning is the other technique that divides a large workflow into several sub-workflows such that the overall number of tasks is reduced and the resource requirements of these sub-workflows

can be satisfied in the execution environments. The performance of workflow partitioning has been evaluated in [58].

However, these existing methods have a simple approach to optimize the task granularity in a job or in a sub-workflow. For example, Horizontal Clustering (HC) [82] merges multiple tasks that are at the same horizontal level of the workflow. The clustering granularity (number of tasks within a cluster) of a clustered job is controlled by the user, who defines either the number of tasks per clustered job (clusters.size), or the number of clustered jobs per horizontal level of the workflow (clusters.num). However, such an empirical approach of tuning task granularity has ignored or underestimated the dynamic features of distributed environments. First of all, such a naive setting of clustering granularity may cause significant imbalance between of run-time and data since it heavily relies on the users' knowledge. Second, these techniques have ignored the occurrence of task failures, which will counteract the benefit from task clustering. The workflow partitioning approach [58] has also ignored that the execution environments have limited resources to host some large-scale scientific workflows, in particular, the data storage constraints.

These developments have introduced many challenges for the management of large-scale scientific workflows. In next section, we generalize these requirements to the three research challenges.

1.1 Problem Space

In this thesis, we identify the new challenges when executing complex scientific applications:

The first challenge has to do with the **data management within a workflow**. Scientific applications are often data intensive and usually need collaborations of scientists from different institutions, hence application data in scientific workflows are usually distributed. Particularly with the emergence of grids and clouds, scientists can upload their data and launch their applications on scientific cloud workflow systems from anywhere in the world via the Internet. However, merging tasks that have input data from or have output data to different data centers

is time-consuming and inefficient. Merging tasks that have no intermediate data between them seems safe at the first sight. However, the subsequent tasks that rely on the output data that their parent tasks produce may suffer a data locality problem since data may be distributed poorly and the data transfer time is increased. Communication-aware scheduling [84, 42] has taken the communication cost into the scheduling/clustering model and have achieved some significant improvement. The workflow partitioning approach [38, 101, 98, 31] represents the clustering problem as a global optimization problem and aims to minimize the overall runtime of a graph. However, the complexity of solving such an optimization problem does not scale well. Heuristics [57, 12] are used to select the right parameters and achieve better runtime performance but the approach is not automatic and requires much experience in distributed systems.

The second challenge users face when merging workflow tasks is the **imbalance of runtime and data dependency**. Tasks may have diverse runtimes and such diversity may cause significant load imbalance. To address this challenge, researchers have proposed several approaches. Bag-of-Tasks [40, 16, 68] dynamically groups tasks together based on the task characteristics but it assumes tasks are independent, which limits its usage in scientific workflows. Singh [82] and Rynge [58] examine the workflow structure and groups tasks together into jobs in a static way for best effort systems. However, this work ignores the computational requirement of tasks and may end up with an imbalanced load on the resources. A popular technique in workload studies to address the load balancing challenge is over-decomposition [52]. This method decomposes computational work into medium-grained tasks. Each task is coarse-grained enough to enable efficient execution and reduce scheduling overheads, while being fine-grained enough to expose significantly higher application-level parallelism than that is offered by the hardware.

The third challenge has to do with **fault tolerance**. Existing clustering strategies ignore or underestimate the impact of the occurrence of failures on system behavior, despite the increasing impact of failures in large-scale distributed systems. Many researchers [104, 89, 79, 76] have emphasized the importance of fault tolerance design and indicated that the failure rates in modern distributed systems are significant. The major concern has to do with transient failures because

they are expected to be more prevalent than permanent failures [104]. For example, denser integration of semiconductor circuits and lower operating voltage levels may increase the likelihood of bit-flips when circuits are bombarded by cosmic rays and other particles [104]. In a faulty environment, there are usually three approaches for managing workflow failures. First, one can simply retry the entire job when its computation is not successful as in the Pegasus Workflow Management System [25]. However, some of the tasks within the job may have completed successfully and it could be a waste of time and resources to retry all of the tasks. Second, the application process can be periodically check-pointed so that when a failure occurs, the amount of work to be retried is limited. However, the overheads of checkpointing can limit its benefits [104]. Third, tasks can be replicated to different nodes to avoid location-specific failures [103]. However, inappropriate clustering (and replication) parameters may cause severe performance degradation if they create long-running clustered jobs.

1.2 Thesis Statement

This thesis states that **optimizing task granularity in scientific workflows can significantly improve the overall performance of execution**. We distinguish our work by exploring novel mechanism and new knowledge in several aspects. First, we propose data aware workflow partitioning to divide large workflows into sub-workflows that satisfy the data storage limit in the execution environments. Second, we propose a series of balanced task clustering strategies to address the tradeoff of dependency imbalance and runtime imbalance. Third, we propose fault tolerant clustering algorithms to automatically adjust the task granularity in a faulty environment and thereby improve the fault tolerance of executing scientific workflows.

This thesis use a wide range of scientific workflows to demonstrate the efficiency and effectiveness of our approaches. We believe our approaches can be used by and will inspire new ways for future optimization of task granularity in scientific workflows.

1.3 Supporting the Thesis Statement

In this section, we substantiate the thesis statement through four specific studies, each gaining new insights and knowledge about the task clustering in scientific workflows.

In our first work [17] (Chapter 3), we **extend the existing DAG model to be overhead aware** and quantitatively analyze the relationship between the workflow performance and overheads. Previous research has established models to describe system overheads in distributed systems and has classified them into several categories [73, 74]. In contrast, we investigate the distributions and patterns of different overheads and discuss how the system environment (system configuration, etc.) influences the distribution of overheads.

In our second work [19, 20] (Chapter 4), we introduce **data aware workflow partitioning** to reduce the data transfer between clustered jobs. Data-intensive workflows require significant amount of storage and computation. For these workflows, we need to use multiple execution sites and consider their available storage. Data aware partitioning aims to reduce the intermediate data transfer in a workflow while satisfying the storage constraints. Heuristics and algorithms are proposed to improve the efficiency of partitioning and experiment-based evaluation is performed to validate its effectiveness.

In our third work [21, 22] (Chapter **??**), to **solve the runtime and data dependency imbalance problem**, we introduce a series of balanced clustering methods. We identify the two challenges: runtime imbalance due to the inefficient clustering of independent tasks and dependency imbalance that is related to dependent tasks. What makes this problem even more challenging is that solutions to address these two problems are usually conflicting. For example, balancing runtime may aggravate the dependency imbalance problem, and vice versa. A quantitative measurement of workflow characteristics is required to serve as a criterion to select and balance these solutions. To achieve this goal, we propose a series of imbalance metrics to reflect the internal structure (in terms of runtime and dependency) of the workflow.

In our fourth work [18] (Chapter **??**), we propose **fault tolerant clustering** that dynamically adjusts the clustering strategy based on the current trend of failures. During the runtime, this

approach uses Maximum Likelihood Estimation to estimate the failure distribution among all the resources and dynamically merges tasks into jobs of moderate size and recluster failed jobs to avoid further failures.

Overall, this thesis aims to **improve the overall performance of task clustering and workflow partitioning in large-scale scientific workflows**. We present both experiment-based and simulation-based evaluation of a wide range of scientific workflows.

1.4 Research Contributions

The main contribution of this thesis is a framework for task clustering and workflow partitioning in distributed autonomous systems. Specially

1. We have developed an overhead aware workflow model to investigate the performance of task clustering in distributed environments. We present the overhead characteristics for a wide range of widely used workflows.

2. We have developed partitioning algorithms that use heuristics to divide large-scale workflows into sub-workflows to satisfy resource constraints such as data storage constraint.

3. We have built a statistical model to demonstrate that transient failures can have a significant impact on the runtime performance of scientific workflows. We have developed a Maximum Likelihood Estimation based parameter estimation approach to integrate both prior knowledge and runtime feedbacks. We have proposed fault tolerant clustering algorithms to dynamically adjust the task granularity and improve the runtime performance.

4. We have examined the reasons that cause runtime imbalance and dependency imbalance in task clustering. We have proposed quantitative metrics to evaluate the severity of the two imbalance problems and a series of balanced clustering methods to address the load balance problem for five widely used scientific workflows.

5. We have developed an innovative workflow simulator called WorkflowSim with the implementation of popular scheduling algorithms and task clustering algorithms. We have built an open source community for the users and developers of WorkflowSim.

Chapter 2

Related Work

2.1 Workflow Modeling and Analysis

Workflow Management Systems (WMS) such as Askalon [98], Taverna [66], and Pegasus [25] are designed to run scientific workflows on distributed environments. Many workflow systems use a particular workflow language or representation (BPEL [4], SCUFL [66], DAGMan [45], DAX [28]), which has a specification that can be composed by hand using a plain text editor. BPEL (Business Process Execution Language) [4] is the de facto standard for Web-service-based workflows with a number of implementations from corporates as well as open source organizations. DAGs (Directed Acyclic Graph) [28] is one of the task graph representations that are widely used as the programming model for many parallel applications because it is effective in expressing and optimizing irregular computations. Moreover, algorithms expressed as DAGs have the potential to alleviate the user from focusing on the architectural issues, while allowing the engine to extract the best performance from the underlying architecture. In this work, we extend the DAG model to be overhead aware, which assists us to model the process of task clustering in scientific workflows. Among these workflow management systems, Pegasus [82] has implemented and used a basic algorithm of task clustering called Horizontal Clustering (HC) that merges task at the same horizontal levels of the workflow. The clustering granularity (number of tasks within a cluster) of a clustered job is controlled by the user, who defines either the number of tasks per clustered job (*clusters.size*), or the number of clustered jobs per horizontal level of the workflow (*clusters.num*). We further extend HC to consider data transfer cost, the balancing between dependencies and computation, and the failure occurrence.

Workflow Patterns and Characteristics [100, 43, 54] are used to capture and abstract the common structure within a workflow and they give insights on designing new workflows and

optimization methods. Yu and Buyya [100] proposed a taxonomy that characterizes and classifies various approaches for building and executing workflows on Grids. They also provided a survey of several representative Grid workflow systems developed by various projects worldwide to demonstrate the comprehensiveness of the taxonomy. Juve et al. [43] provided a characterization of workflow from 6 scientific applications and obtained task-level performance metrics (I/O, CPU, and memory consumption). They also presented an execution profile for each workflow running at a typical scale and managed by the Pegasus workflow management system [28]. Liu et al. [54] proposed a novel pattern based time-series forecasting strategy which utilizes a periodical sampling plan to build representative duration series. We illustrate the relationship between the workflow patterns (asymmetric or symmetric workflows) and the performance of our balancing algorithms. Some work in the literature has further attempted to define and model workflow characteristics with quantitative metrics. In [1], the authors proposed a robustness metric for resource allocation in parallel and distributed systems and accordingly customized the definition of robustness. Tolosana et al. [93] defined a metric called Quality of Resilience to assess how resilient workflow enactment is likely to be in the presence of failures. Ma et al. [55] proposed a graph distance based metric for measuring the similarity between data oriented workflows with variable time constraints, where a formal structure called time dependency graph (TDG) is proposed and further used as representation model of workflows. Similarity comparison between two workflows can be reduced to computing the similarity between TDGs. In this paper, we focus on novel quantitative metrics that are able to demonstrate the imbalance problem in scientific workflows.

Overhead Analysis [70, 74, 17] is a topic of great interest in the Grid community. Stratan et al. [87] evaluate in a real-world environment Grid workflow engines including DAGMan/Condor and Karajan/Globus. Their methodology focuses on five system characteristics: overhead, raw performance, stability, scalability, and reliability. They pointed out that head node consumption should not be negligible and the main bottleneck in a busy system is often the head node. Prodan et al. [74] offered a complete Grid workflow overhead classification and a systematic measurement of overheads. In Chen et al. [17], we extended [74] by providing a measurement of major

overheads imposed by workflow management systems and execution environments and analyzed how existing optimization techniques improve runtime by reducing or overlapping overheads. The prevalent existence of system overheads is an important reason why task clustering provides significant performance improvement for workflow-based applications. In this paper, we aim to further improve the performance of task clustering using such a classification of overheads.

2.2 Workflow Paritioning

Workflow Partitioning. For convenience and cost-related reasons, scientists execute scientific workflows [8, 31] in distributed large-scale computational environments such as multi-cluster grids, that is, grids comprising multiple independent execution sites. Topcuoglu [96] presented a classification of widely used task scheduling approaches. Such scheduling solutions, however, cannot be applied directly to multi-cluster grids. First, the data transfer delay between multiple execution sites is more significant than that within an execution site and thus a hierarchical view of data transfer is necessary. Second, they do not consider the resource availability experienced in grids, which also makes accurate predictions of computation and communication costs difficult. Sonmez [85] extended the traditional scheduling problem to multiple workflows on multi-cluster grids and presented a performance of a wide range of dynamic workflow scheduling policies in multi-cluster grids. Duan [31] and Wieczorek [98] have discussed the scheduling and partitioning scientific workflows in dynamic grids with challenges such as a broad set of unpredictable overheads and possible failures. Duan [31] then developed a distributed service-oriented Enactment Engine with a master-slave architecture for de-centralized coordination of scientific workflows. Kumar [47] proposed the use of graph partitioning for partition the resources of a distributed system, but not the workflow DAG, which means the resources are provisioned into different execution sites but the workflows are not partitioned at all. Dong [29] and Kalayci [45] have discussed the use of graph partitioning algorithms for the workflow DAG according to features of the workflow itself and the status of selected available resource clusters. Our work focuses on the workflow partitioning problem with resource constraints, in particular, the data

storage constraint. Compared to Dong [29] and Kalayci [45], we extend their work to estimate the overall runtime of sub-workflows and then schedule these sub-workflows based on the estimates.

Data Placement techniques try to strategically manage placement of data before or during the execution of a workflow. Kosar et al. [46] presented Stork, a scheduler for data placement activities on grids and proposed to make data placement activities as first class citizens in the Grid. In Stork, data placement is a job and is decoupled from computational jobs. Amer et al. [3] studied the relationship between data placement services and workflow management systems for data-intensive applications. They proposed an asynchronous mode of data placement in which data placement operations are performed as data sets become available and according to the policies of the virtual organization and not according to the directives of the workflow management system (WMS). The WMS can however assist the placement services with the placement of data based on information collected during task executions and data transfers. Shankar [81] presented an architecture for Condor in which the input, output and executable files of jobs are cached on the local disks of the machines in a cluster. Caching can reduce the amount of pipelines and batch I/O that is transferred across the network. This in turn significantly reduces the response time for workflows with data-intensive workloads. In contrast, we mainly focus on the workflow partitioning problem but our work can be extended to consider the data placement strategies they have proposed in the future.

Data Throttling. Park et al. [71] limits the amount of parallel data transfer to avoid overloading supporting services such as data servers, which is called data throttling. Throttling is especially useful for unbalanced workflows in which one task might be idle while waiting for data to arrive. However, as discussed in [71], data throttling has an impact on the overall workflow performance depending on the ratio between computational and data transfer tasks. Therefore, performance analysis is necessary after the profiling of data transfers so that the relationship between computation and data transfers can be identified more explicitly. Rodriguez [75] proposed an automated and trace-based workflow structural analysis method for DAGs. Files transfers are accomplished as fast as the network bandwidth allows, and once transferred,

the files are buffered/stored at their destination. To improve the use of network bandwidth and buffer/storage within a workflow, they adjusted the speeds of some data transfers and assured that tasks have all their input data arriving at the same time. Compared to our work, data throttling has a limit in performance gain by the amount of data transfer that can be reduced, while our partitioning approach can improve the overall workflow runtime and resource usage.

2.3 Task Clustering

Workflow Scheduling. There have been a considerable amount of work trying to solve workflow-mapping problem using DAG scheduling heuristics such as HEFT [96], Min-Min [9], MaxMin [10], MCT [10], etc. Duan [31] and Wieczorek [98] have discussed the scheduling and partitioning scientific workflows in dynamic grids. The emergence of cloud computing [5] has made it possible to easily lease large-scale homogeneous computing resources from commercial resource providers and guarantee the quality of services. In our case, since we assume resources are homogeneous, the resource selection is not a major concern and thus the scheduling problem can be simplified as a task clustering problem. More specifically, a plethora of scheduling algorithms have been developed in the networking and operating system domains. Many of these schedulers have been extended to the hierarchical setting. Lifflander et al. [52] proposed to use work stealing and a hierarchical persistence-based rebalancing algorithm to address the imbalance problem in scheduling. Zheng et al. [106] presented an automatic hierarchical load balancing method that overcomes the scalability challenges of centralized schemes and poor solutions of traditional distributed schemes. There are other scheduling algorithms [10] (e.g. list scheduling) that indirectly achieve load balancing of workflows through makespan minimization. However, the benefit that can be achieved through traditional scheduling optimization is limited by its complexity. The performance gain of task clustering is primarily determined by the ratio between system overheads and task runtime, which is more substantial in modern distributed systems such as Clouds and Grids.

Task Granularity Control has also been addressed in scientific workflows. For instance, Singh et al. [82] proposed a level- and label-based clustering. In level-based clustering, tasks at the same level can be clustered together. The number of clusters or tasks per cluster are specified by the user. In the label-based clustering, the user labels tasks that should be clustered together. Although their work considers data dependencies between workflow levels, it is done manually by the users, which is prone to errors. Recently, Ferreira da Silva et al. [33] proposed task grouping and ungrouping algorithms to control workflow task granularity in a non-clairvoyant and online context, where none or few characteristics about the application or resources are known in advance. Their work significantly reduced scheduling and queuing time overheads, but did not consider data dependencies. The low performance of *fine-grained* tasks is a common problem in widely distributed platforms where the scheduling overhead and queuing times at resources are high, such as Grid and Cloud systems. Several works have addressed the control of task granularity of bags of tasks. For instance, Muthuvelu et al. [62] proposed a clustering algorithm that groups bags of tasks based on their runtime—tasks are grouped up to the resource capacity. Later, they extended their work [61] to determine task granularity based on task file size, CPU time, and resource constraints. Recently, they proposed an online scheduling algorithm [60, 63] that groups tasks based on resource network utilization, user's budget, and application deadline. Ng et al. [64] and Ang et al. [92] introduced bandwidth in the scheduling framework to enhance the performance of task scheduling. Longer tasks are assigned to resources with better bandwidth. Liu and Liao [53] proposed an adaptive fine-grained job scheduling algorithm to group fine-grained tasks according to processing capacity and bandwidth of the current available resources. Although these techniques significantly reduce the impact of scheduling and queuing time overhead, they did not consider data dependencies.

Failure Analysis and Modeling [89] presents system characteristics such as error and failure distribution and hazard rates. Schroeder et al. [79] has studied the statistics of the data, including the root cause of failures, the mean time between failures, and the mean time to repair. Sahoo et al. [76] analyzed the empirical and statistical properties of system errors and failures from a network of heterogeneous servers running a diverse workload. Oppenheimer et al. [67] analyzed

the causes of failures from three large-scale Internet services and the effectiveness of various techniques for preventing and mitigating service failure. McConnel [59] analyzed the transient errors in computer systems and showed that transient errors follow a Weibull distribution. In [88, 41] Weibull distribution is one of the best fit for the workflow traces they used. Based on these work, we measure the inter-arrival time of failures in a workflow and then provide methods to improve task clustering. More and more workflow management systems are taking fault tolerance into consideration. The Pegasus workflow management system [25] has incorporated a task-level monitoring system and used job retry to address the issue of task failures. They also used provenance data to track the failure records and analyzed the causes of failures [78]. Plankensteiner et. al. [72] have surveyed the fault detection, prevention and recovery techniques in current grid workflow management systems such as ASKALON [32], Chemomentum [80], Escogitare [48] and Triana [90]. Recovery techniques such as replication, checkpointing, task resubmission and task migration etc. have been provided. We are specifically joining the work of failure analysis and the optimization in task clustering. To be best of our knowledge, none of existing workflow management systems have provided such features.

Load Balancing. With the aim of dynamically balancing the computational load among resources, some jobs have to be moved from one resource to another and/or from one period of time to another, which is called task reallocation [95]. Caniou [14] presented a reallocation mechanism that tunes parallel jobs each time they are submitted to the local resource manager (which implies also each time a job is migrated). They only needed to query batch schedulers with simple submission or cancellation requests. Its authors also presented different reallocation algorithms and studied their behaviors in the case of a multi-cluster Grid environment. In [102], a pre-emptive process migration method was proposed to dynamically migrate processes from overloaded nodes to lightly-loaded nodes. However, it can achieve a good balance only when there are some idle compute nodes (e.g. when the number of task processes is less than that of compute nodes). In our case with large scale scientific workflows, usually we have more tasks than available compute nodes . Guo et al. [37] presented mechanisms to dynamically split and

consolidate tasks to cope with load balancing and break through the concurrency limit resulting from fixed task granularity. They have proposed algorithms to address the load balancing problem in single-job systems and prior knowledge is not required. For multi-job cases, they used a shortest-job-first algorithm to minimize job turnaround time when combined with task splitting. Similarly, Ying et al. [99] proposed a load-balancing algorithm based on collaborative task clustering. The algorithm divides the collaborative computing tasks into subtasks and then dynamically allocates them to the servers. Compared to these approaches, our work selects tasks to be merged based on their task runtime distribution and their data dependencies initially, without introducing additional overheads during the runtime. Also, a quantitative approach of imbalance measurement provides us a general view of multiple workflow instances with different runtime characteristics.

Machine Learning in Workflow Optimization has been used to predict execution time [30, 15, 51, 23] and system overheads [17], and develop probability distributions for transient failure characteristics. Duan et.al. [30] used Bayesian network to model and predict workflow task runtimes. The important attributes (such as the external load, arguments etc.) are dynamically selected by the Bayesian network and fed into a radial basis function neural network to make further predictions. Ferreira da Silva et. al. [23] used regression trees to dynamically estimate task behavior including process I/O, runtime, memory peak and disk usage. We reuse the knowledge gained from prior work on failure analysis, overhead analysis and task runtime analysis. We then use prior knowledge based Maximum Likelihood Estimation to integrate both the knowledge and runtime feedbacks and adjust the estimation accordingly.

Chapter 3

Background

In this chapter, we first introduce how we extend the existing DAG model to be overhead aware and we also describe the system model that we use in this thesis. Second we present our overhead analysis on a series of widely used workflows, which is a base of our optimization methods that will be used in simulations. Third, we introduce the experiment environments used in this thesis including FutureGrid, a distributed platform and a workflow simulator as an example of the importance of an overhead model when simulating workflow execution. The simulation of a widely used workflow verifies the necessity of taking overhead into consideration.

3.1 Workflow and System Model

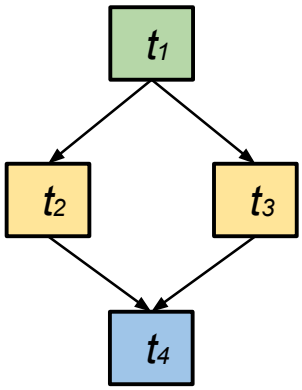

Figure 3.1: Simple DAG with four tasks (t_1, t_2, t_3, t_4). The edges represent data dependencies between tasks.

Traditionally a workflow is modeled as a Directed Acyclic Graph (DAG). As shown in Figure 3.1, each node in the DAG represents a workflow task, and the edges represent dependencies

between the tasks (t) that constrain the order in which the tasks are executed. Each task is a program and a set of parameters that need to be executed. The dependencies typically represent data flow dependencies in the application, where the output files produced by one task are needed as inputs of another task.

Workflows are typically prepared and executed in a Workflow Management System such as Pegasus [25], ASKALON [69] and Taverna [94]. The components of a WMS are listed below:

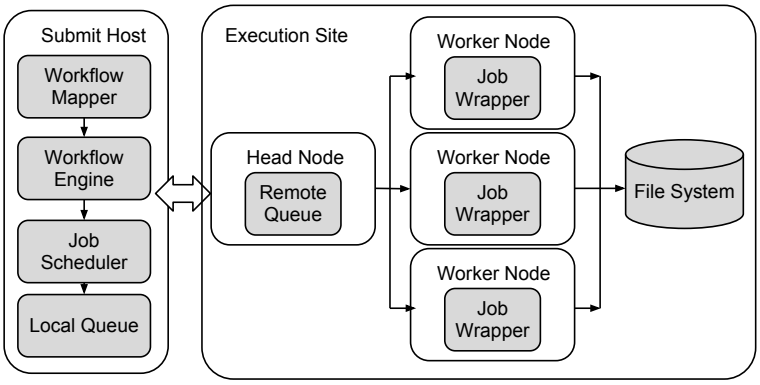

Figure 3.2: System Model

Workflow Mapper generates an executable workflow based on an abstract workflow provided by the user or workflow composition system.

Workflow Engine executes the jobs defined by the workflow in order of their dependencies. Only free jobs that have all their parent jobs completed are submitted to Job Scheduler.

Job Scheduler and **Local Queue** manage individual workflow jobs and supervise their execution on local and remote resources.

Job Wrapper extracts tasks from clustered jobs and executes them at the worker nodes.

A DAG models the computational activities and data dependencies within a workflow and it fits with most workflow management systems such as Pegasus [25] and DAGMan [45]. However, the preparation and execution of a scientific workflow in distributed environments often involves multiple components and the system overheads within and between these components are ignored. An overhead is defined as the time of performing miscellaneous work other than executing the users computational activities. To address this challenge, in this section, we extend

the existing Directed Acyclic Graph (DAG) model to be overhead aware (o-DAG), in which an overhead is also a node in the DAG and the control dependencies are added as directed edges. We utilize o-DAG to provide a systematic analysis of the performance of workflow optimization methods and provide a series of novel optimization methods to further improve the overall workflow performance.

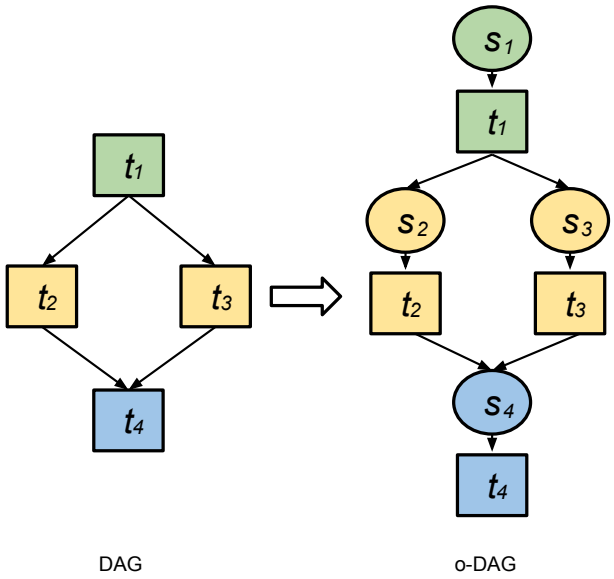

Figure 3.3: Extending DAG to o-DAG

Fig 3.3 shows how we augment a DAG to be an o-DAG with the capability to represent scheduling overheads (s) such as workflow engine delay, queue delay, and postscript delay. The classification of overheads is based on the model of a typical workflow management system shown in Fig 3.2.

3.2 Modeling Task Clustering

With an o-DAG model, we can explicitly express the process of task clustering. In this paper, we address task clustering horizontally and vertically. **Horizontal Clustering** (HC) merges multiple tasks that are at the same horizontal level of the workflow, in which the horizontal level of a task is defined as the longest distance from the entry task of the DAG to this task. **Vertical Clustering**

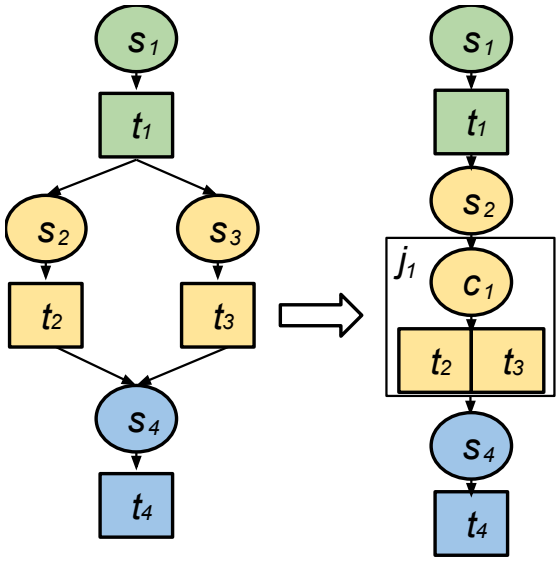

Figure 3.4: Task Clustering

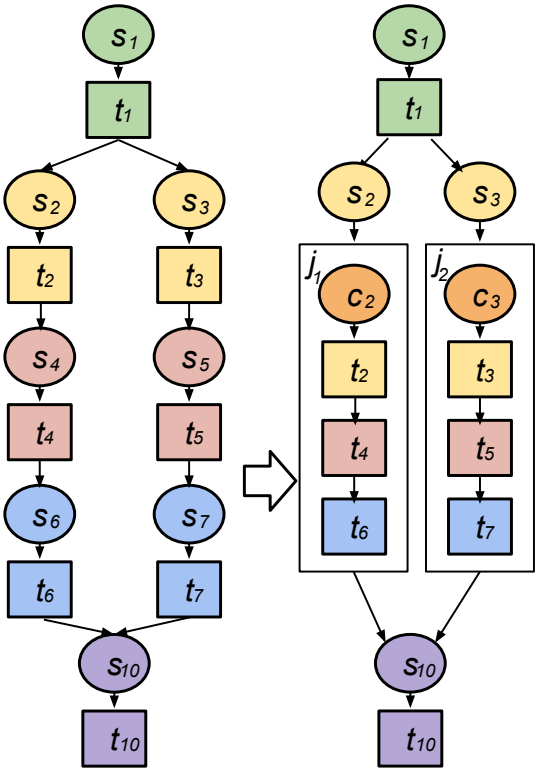

Figure 3.5: An example of vertical clustering.

(VC) merges tasks within a pipeline of the workflow. Tasks at the same pipeline share a single-parent-single-child relationship, which means a task t_a is the unique parent of a task t_b, which is the unique child of t_a.

Figure 3.4 shows a simple example of how to perform HC, in which two tasks t_2 and t_3, without a data dependency between them, are merged into a clustered job j_1. A job is a single execution unit composed by one or multiple task(s). Job wrappers are commonly used to execute clustered jobs, but they add an overhead denoted by the clustering delay c. The clustering delay measures the difference between the sum of the actual task runtimes and the job runtime seen by the job scheduler. After horizontal clustering, t_2 and t_3 in j_1 can be executed in sequence or in parallel, if parallelism in one compute node is supported. In this work, we consider sequential executions only. Given a single resource, the overall runtime for the workflow in Figure 3.4 (left) is $runtime_l = \sum_{i=1}^{4}(s_i + t_i)$, and the overall runtime for the clustered workflow in Figure 3.4 (right) is $runtime_r = s_1 + t_1 + s_2 + c_1 + t_2 + t_3 + s_4 + t_4$. $runtime_l > runtime_r$ as long as $c_1 < s_3$, which is often the case in many distributed systems since the clustering delay within a single execution node is usually shorter than the scheduling overhead across different execution nodes.

Figure 3.5 illustrates an example of vertical clustering, in which tasks t_2, t_4, and t_6 are merged into j_1, while tasks t_3, t_5, and t_7 are merged into j_2. Similarly, clustering delays c_2 and c_3 are added to j_1 and j_2 respectively, but system overheads s_4, s_5, s_6, and s_7 are removed.

3.3 Workflow Overheads

3.3.1 Overhead Classification

The execution of a job is comprised of a series of events as shown in Figure 3.6 and they are defined as:

1. Job Release is defined as the time when the workflow engine identifies that a job is ready to be submitted (when its parents have successfully completed).

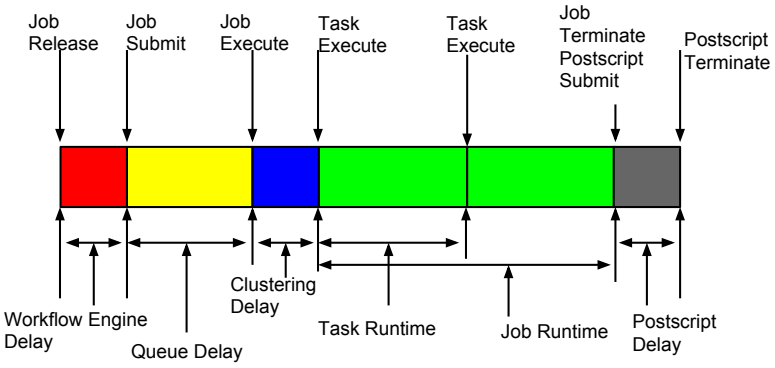

Figure 3.6: Workflow Events

2. Job Submit is defined as the time when the workflow engine submits a job to the local queue.

3. Job Execute is defined as the time when the workflow engine sees a job is being executed.

4. Task Execute is defined as the time when the job wrapper sees a task is being executed.

5. Postscript Start is defined as the time when the workflow engine starts to execute a postscript.

6. Postscript Terminate is defined as the time when the postscript returns a status code (success or failure).

Figure 3.6 shows a typical timeline of overheads and runtime in a job. We do not specify the data transfer delay in this timeline because data transfer is handled by data transfer jobs (stage-in and stage-out jobs).

We have classified workflow overheads into five categories as follows.

1. Workflow Engine Delay measures the time between when the last parent job of a job completes and the time when the job gets submitted to the local queue. The completion time of the last parent job means this job is released to the ready queue and is waiting for resources to be assigned to it. The workflow engine delay reflects the efficiency of a workflow engine (i.e., DAGMan [24]).

2. Queue Delay is defined as the time between the submission of a job by the workflow engine to the local queue and the time the local scheduler sees the job running. This overhead reflects the efficiency of the local workflow scheduler (e.g. Condor [35]) to execute a job and the availability of resources for the execution of this job.

3. Postscript Delay is the time taken to execute a lightweight script under some execution systems after the execution of a job. Postscripts examine the status code of a job after the computational part of this job is done.

4. Data Transfer Delay happens when data is transferred between nodes. It includes three different types of processes: staging data in, cleaning up, and staging data out. Stage-in jobs transfer input files from source sites to execution sites before the computation starts. Cleanup jobs delete intermediate data that is no longer needed by the remainder of the workflow. Stage-out jobs transfer workflow output data to archiving sites for storage and analysis.

5. Clustering Delay measures the difference between the sum of the actual task runtime and the job runtime seen by the job wrapper. The cause of Clustering Delay is usually because we use a job wrapper in worker nodes to execute a clustered job that requires some delay to extract the list of tasks.

3.3.2 Overhead Distribution

We examined the overhead distributions of a widely used astronomy workflow called Montage [7] that is used to construct large image mosaics of the sky. Montage was run on FutureGrid [34]. FutureGrid is a distributed, high-performance testbed that provides scientists with a set of computing resources to develop parallel, grid, and cloud applications. Figure 3.8, 3.7 and 3.9 show the overhead distribution of the Montage workflow run on the FutureGrid. The postscript delay concentrates at 7 seconds, because the postscript is only used to locally check the return status of a job and is not influenced by the remote execution environment. The workflow engine

delay tends to have a uniform distribution, which is because the workflow engine spends a constant amount of time to identify that the parent jobs have completed and insert a job that is ready at the end of the local queue. The queue delay has three decreasing peak points at 8, 14, and 22 seconds. We believe this is because the average postscript delay is about 7 seconds (see details in Figure 3.8) and the average runtime is 1 second. The local scheduler spends about 8 seconds finding an available resource and executing a job; if there is no resource idle, it will wait another 8 seconds for the current running jobs to finish, and so on.

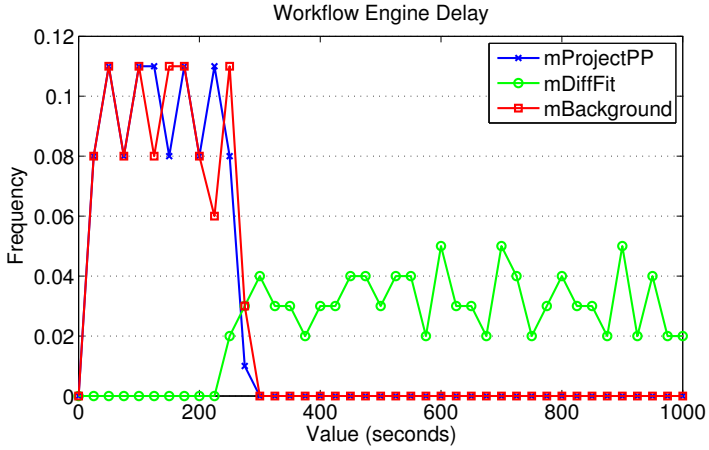

Figure 3.7: Distribution of Workflow Engine Delay in the Montage workflow

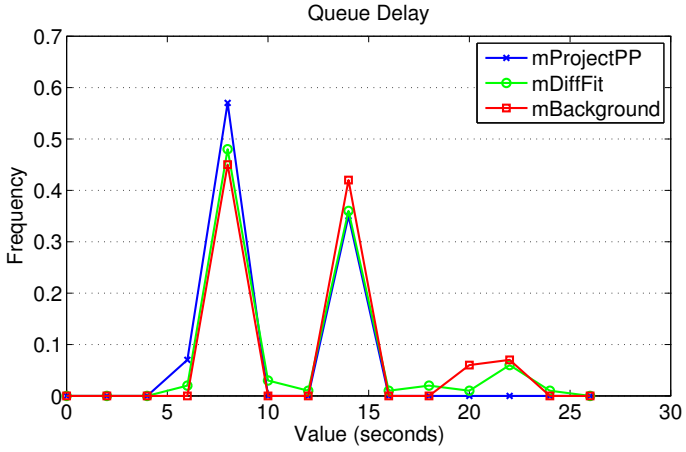

Figure 3.8: Distribution of Queue Delay in the Montage workflow

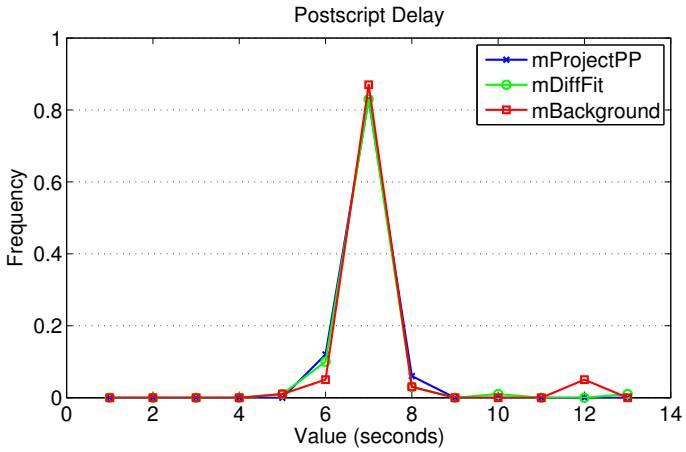

Figure 3.9: Distribution of Postscript Delay in the Montage workflow

We use Workflow Engine Delay as an example to show the necessity to model overheads appropriately. Figure 3.10 shows a real trace of overheads and runtime in the Montage 8 degree workflow (for visibility issues, we only show the first 15 jobs at the mProjectPP level). We can see that Workflow Engine Delay increases steadily after every five jobs. For example, the Workflow Engine Delay of jobs with ID from 6 to 10 is approximately twice of that of jobs ranging from ID1 to ID5. Figure 3.11 further shows the distribution of Workflow Engine Delay of the first 20 jobs at the mProjectPP level in the Montage workflow. After every five jobs, the Workflow Engine Delay increases by 8 seconds approximately. We call this special nature of workflow overhead as cyclic increase. The reason is that Workflow Engine (in this trace it is DAGMan) releases five jobs by default in every working cycle. Therefore, simply adding a constant delay after every job execution has ignored its potential influence on the performance.

Figure 3.12 shows the average value of Clustering Delay of mProjectPP, mDiffFit, and mBackground. It is clear that with the increase of *clusters.num* (the maximum number of jobs per horizontal level), since there are less and less tasks in a clustered job, the Clustering Delay for each job decreases. For simplicity, we use an inverse proportional model in Equation 3.1 to describe this trend of Clustering Delay with *clusters.num*. Intuitively we assume that the average delay per task in a clustered job is constant (n is the number of tasks in a horizontal level). An inverse proportional model can estimate the delay when $clusters.num = i$ directly if we have

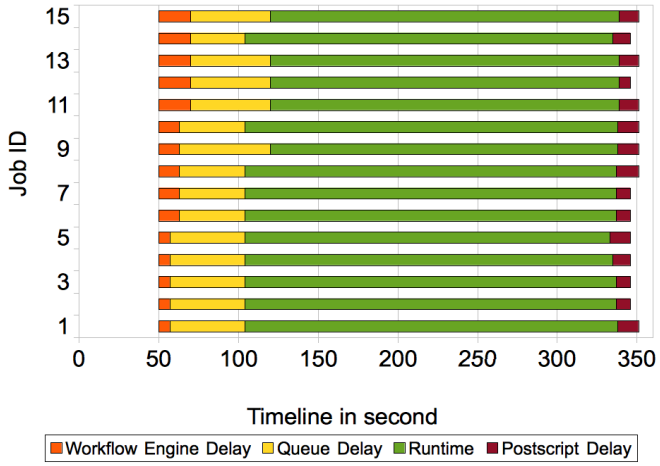

Figure 3.10: Workflow Overhead and Runtime. Clustering delay and data transfer delay are not shown

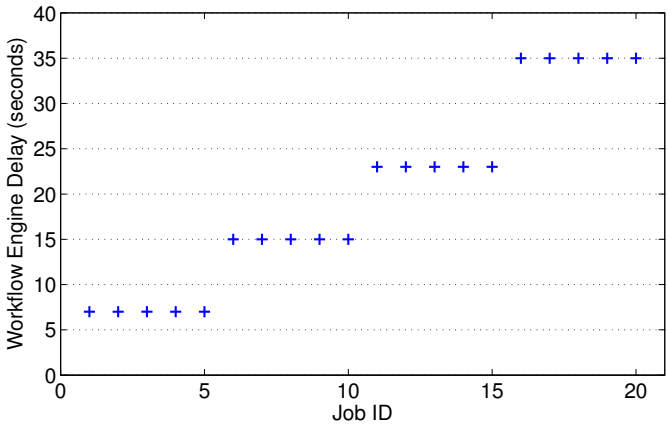

Figure 3.11: Workflow Engine Delay of mProjectPP

known the delay when $clusters.num = j$. Therefore we can predict all the clustering cases as long as we have gathered one clustering case.

$$\frac{ClusteringDelay|_{clusters.num=i}}{ClusteringDelay|_{clusters.num=j}} = \frac{n/i}{n/j} = \frac{j}{i} \tag{3.1}$$

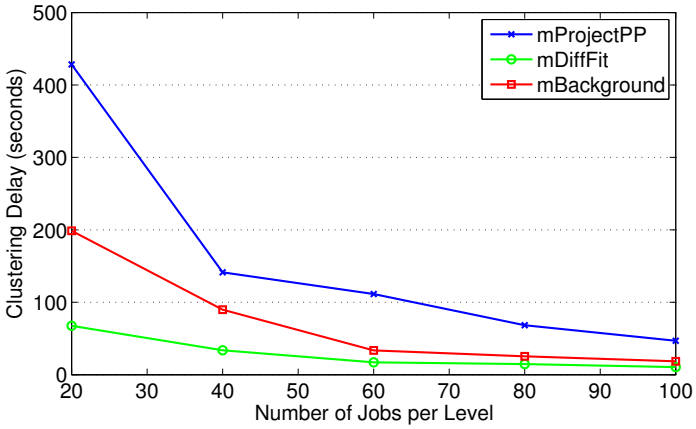

Figure 3.12: Clustering Delay of mProjectPP, mDiffFit, and mBackground

3.4 Experiment Environments

In this section, we introduce the experiment environments to evaluate our approaches in this thesis. We first introduce FutureGrid, a distributed cloud platform and then WorkflowSim, a traced based simulator developed by us.

3.4.1 FutureGrid

FutureGrid [34] is a distributed, high-performance testbed that provides scientists with a set of computing resources to develop parallel, grid, and cloud applications. We used FutureGrid to evaluate our workflow partitioning algorithms in Chapter 4 and we also collected the traces of workflows that were run on FutureGrid to feed WorkflowSim such that it can simulate workflows with better accuracy.

FutureGrid is built out of a number of clusters of different types and sizes that are interconnected with up to a 10GB Ethernet between its sites. In our experiments of workflow partitioning, we have used five execution sites including Indiana University (IU), University of Chicago (UC), San Diego Supercomputing Center (SDSC), Texas Advanced Computing Center (TACC), and University of Florida (UF). The FutureGrid policy allows us to launch 20 virtual machines in each site. Table 3.1 shows the overview of these clusters.

Table 3.1: Overview of the Clusters

Site	Name	Nodes	CPUs	Cores	RAM(GB)	Storage (TB)
IU	india	128	256	1024	3072	335
UC	hotel	84	168	672	2016	120
SDSC	sierra	84	168	672	2688	96
UF	foxtrot	32	64	256	768	0
TACC	alamo	96	192	768	1152	30

3.4.2 WorkflowSim

We have developed WorkflowSim as an open source workflow simulator that extends CloudSim [11] by providing a workflow level support of simulation. It models workflows with a DAG model with support an elaborate model of node failures, a model of delays occurring in the various levels of the WMS stack [17], the implementations of several most popular dynamic and static workflow schedulers (e.g., HEFT, Min-Min) and task clustering algorithms (e.g., runtime-based algorithms, data-oriented algorithms and fault tolerant clustering algorithms). Parameters are imported from traces of workflow executions that were run on FutureGrid [34].

In heterogeneous distributed systems, workflows may experience different types of overheads, which are defined as the time of performing miscellaneous work other than executing users computational activities. Since the causes of overheads differ, the overheads have diverse distributions and behaviors. For example, the time to run a post-script that checks the return status of a computation is usually a constant. However, queue delays incurred while tasks are waiting in a batch scheduling systems can vary widely. By classifying these workflow overheads in different layers and system components, our simulator can offer a more accurate result than simulators that do not include overheads in their system models.

Whats more, many researchers [104, 89, 79, 76, 67, 59] have emphasized the importance of fault tolerant design and concluded that the failure rates in modern distributed systems should not be neglected. A simulation with support for randomization and layered failures is supported in WorkflowSim to promote such studies.

Finally, progress in workflow research also requires a general-purpose framework that can support widely accepted features of workflows and optimization techniques. Existing simulators

such as CloudSim/GridSim [11] fail to provide fine granularity simulations of workflows. For example, they lack the support of task clustering, which is a popular technique that merges small tasks into a large job to reduce task execution overheads. The simulation of task clustering requires two layers of execution model, on both task and job levels. It also requires a workflow-clustering engine that launches algorithms and heuristics to cluster tasks. Other techniques such as workflow partitioning and task retry are also ignored in these simulators. These features have been implemented in WorkflowSim.

To the best of our knowledge, none of the current distributed system simulators support these rich-features and techniques. Below, we introduce our early work on simulating scientific workflows satisfying these requirements. We evaluate the performance of WorkflowSim with an example of task clustering.

3.4.3 Components and Functionalities

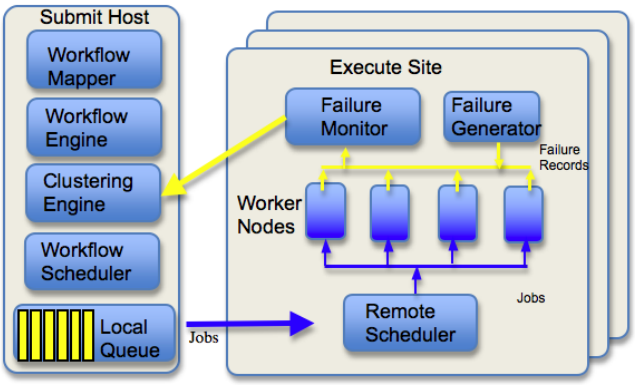

Figure 3.13: Overview of WorkflowSim.

Deelman et. al. [26] have provided a survey of popular workflow systems for scientific applications and have classified their components into four categories: composition, mapping, execution, and provenance. Based on this survey, we identified the mandatory functionalities/components and designed the layers in our WorkflowSim. In our design as shown in Figure 3.13, we add multiple layers on top of the existing task scheduling layer of CloudSim, which

include Workflow Mapper, Workflow Engine, Clustering Engine, Workflow Scheduler, Failure Generator and Failure Monitor etc.

1. Workflow Mapper is used to import DAG files formatted in XML (called DAX in WorkflowSim) and other metadata information such as file size from Workflow Generator. Workflow Mapper creates a list of tasks and assigns these tasks to an execution site. A task is a program/activity that a user would like to execute.

2. Workflow Engine manages tasks based on their dependencies between tasks to assure that a task may only be released when all of its parent tasks have completed successfully. The Workflow Engine will only release free tasks to Clustering Engine.

3. Clustering engine merges tasks into jobs such that the scheduling overhead is reduced. A job is an atomic unit seen by the execution system, which contains multiple tasks to be executed in sequence or in parallel if applicable. Different to the clustering engine in Pegasus WMS, Clustering Engine in WorkflowSim also perform task reclustering in a faulty environment with transient failures. If there are failed tasks returned from Workflow Scheduler, they are merged again into a new job.

4. Workflow Scheduler is used to match jobs to a worker node based on the criteria selected by users. WorkflowSim relies on CloudSim to provide an accurate and reliable job-level execution model, such as time-shared model and space-shared model. However, WorkflowSim has introduced different layers of overheads and failures based on our prior work [17], which improves the accuracy of simulation.

5. Failure Generator is introduced to inject task/job failures at each execution site during the simulation. After the execution of each job, Failure Generator randomly generates task/job failures based on the distribution and average failure rate that a user has specified.

6. Failure Monitor collects failure records (e.g., resource id, job id, task id) to return these records to Clustering Engine to adjust the scheduling strategies dynamically. In a failure-prone environment, there are several options to improve workflow performance. First, one

can simply retry the entire job or only the failed part of this job when its computation is not successful. This functionality is added to the Workflow Scheduler, which checks the status of this job and takes action based on the strategies that a user selects.

WorkflowSim extracts the common features exposed by various workflow systems and supports widely used workflow features and optimization techniques. Below we list the main functionalities supported by WorkflowSim. Note that WorkflowSim is an open source community and more functionalities are added at the same time.

1. Overhead Modeling: Based on our prior studies on workflow overheads [17], we add layered overhead to the workflow simulation. We have classified workflow overheads into five categories as follows. Workflow Engine Delay measures the time between when the last parent job of a job completes and the time when the job gets submitted to the local queue. Queue Delay is defined as the time between the submission of a job by the workflow engine to the local queue and the time the local scheduler sees the job running. Data Transfer Delay happens when data is transferred between nodes. Clustering Delay measures the difference between the sum of the actual task runtime and the job runtime seen by the Workflow Scheduler.

2. Failure Modeling: WorkflowSim supports two types failures. A task failure which means a task fails while other tasks in the same job may not fail. A job failure means a job fails while all of its tasks fail. The reason why we have this classification of transient failures is that they usually have different causes. Task failure is usually generated by an error during the execution while a job failure happens during the preparation of a job. Users can specify the distribution (Weibull, Uniform, Normal and Gamma) of the failure rates.

3. Shared and Distributed File Systems: Modern file systems are usually classified into two types. The first one is a central file system in which the data storage is shared among all the worker nodes in a data-center. The communication cost is thereby included in the execution time of a task and scheduling algorithms do not necessary consider the data

transfer delay between tasks. The second is a distributed file system in which the data storage is constructed with multiple separate local data storages. The communication cost should not be ignored and data-aware scheduling algorithms can apply to this environment to optimize the data locality and the overall makespan. In either way, WorkflowSim uses a Replica Catalog to keep track of the files including their replications.

4. Dynamic Scheduling Algorithm: While CloudSim has already supported static scheduling algorithms, we added the support of dynamic workflow algorithms in WorkflowSim. For static algorithms, jobs are assigned to a worker node at the workflow planning stage. When a job reaches the remote scheduler, it will just wait until the assigned worker node is free. For dynamic algorithms, jobs are matched to a worker node in the remote scheduler whenever a worker node becomes idle. This classification proposes new options for researchers to evaluate the dynamic performance of their algorithms.

5. Task Clustering: Compared to CloudSim and other workflow simulators, WorkflowSim provides support of task clustering that merges tasks into a clustered job. Users can specify different criteria to optimize the overall performance. For example, in a fault environment, a long job would end up with running forever even though the overheads are reduced.

6. Energy aware:

3.4.4 Results and Validation

We use task clustering as an example to illustrate the necessity of introducing overheads into workflow simulation. The goal was to compare the simulated overall runtime of workflows in case the information of job runtime and system overheads are known and extracted from prior traces. In this example, we collected real traces generated by the Pegasus Workflow Management System while executing workflows on FutureGrid [34]. We built an execution site with 20 worker nodes and we executed the Montage workflow five times in every single configuration of *clusters.num*, which is the maximum number of clustered jobs in a horizontal level. These five traces of workflow execution with the same *clusters.num* is a training set or a validation set.

We ran the Montage workflow with a size of 8-degree squares of sky. The workflow has 10,422 tasks and 57GB of overall data. We tried different *clusters.num* from 20 to 100, leaving us 5 groups of data sets with each group having 5 workflow traces. First of all, we adopt a simple approach that selects a training set to train WorkflowSim and then use the same training set as validation set to compare the predicted overall runtime and the real overall runtime in the traces. We define accuracy in this section as the ratio between the predicted overall runtime and the real overall runtime:

$$Accuracy = \frac{Predicted\ Overall\ Runtime}{Real\ Overall\ Runtime} \tag{3.2}$$

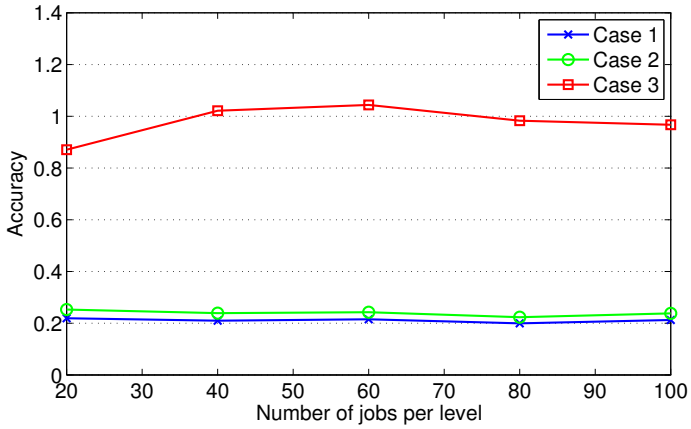

Figure 3.14: Performance of WorkflowSim with different support levels

Performance of WorkflowSim with different support levels. To train WorkflowSim, from the traces of workflow execution (training sets), we extracted information about job runtime and overheads, such as average/distribution and, for example, whether it has a cyclic increase. We then added these parameters into the generation of system overheads and simulated them as close as possible to the real cases. Here, we do not discuss the randomization or distribution of job runtime since we rely on CloudSim to provide a convincing model of job execution.

To present an explicit comparison, we simulated the cases using WorkflowSim that has no consideration of workflow dependencies or overheads (Case 1), WorkflowSim with Workflow Engine that has considered the influence of dependencies but ignored overheads (Case 2), and

WorkflowSim, that has covered both aspects (Case 3). Intuitively speaking, we expect that the order of the accuracy of them should be Case 3 > Case 2 > Case 1.

Fig 3.14 shows the performance of WorkflowSim with different support levels is consistent to our expectation. The accuracy of Case 3 is quite close to but not equal to 1.0 in most points. The reason is that to simulate workflows, WorkflowSim has to simplify models with a few parameters, such as the average value and the distribution type. It is not efficient to recur every overhead as is present in the real traces. It is also impossible to do since the traces within the same training set may have much variance. Fig 3.14 also shows that the accuracy of both Case 1 and Case 2 are much lower than Case 3. The reason why Case 1 does not give an exact result is that it ignores both dependencies and multiple layers of overheads. By ignoring data dependencies, it releases tasks that are not supposed to run since their parents have not completed (a real workflow system should never do that) and thereby reducing the overall runtime. At the same time, it executes jobs/tasks irrespective of the actual overheads, which further reduces the simulated overall runtime. In Case 2, with the help of Workflow Engine, WorkflowSim is able to control the release of tasks and thereby the simulated overall runtime is closer to the real traces. However, since it has ignored most overheads, jobs are completed and returned earlier than that in real traces. The low accuracy of Case 1 and Case 2 confirms the necessity of introducing overhead design into our simulator.

3.5 Workflows Used

Five widely used scientific workflow applications are used in this thesis: LIGO Inspiral analysis [49], Montage [7], CyberShake [36], Epigenomics [97], and SIPHT [83]. In this section, we describe each workflow application and present their main characteristics and structures.

LIGO Laser Interferometer Gravitational Wave Observatory (LIGO) [49] workflows are used to search for gravitational wave signatures in data collected by large-scale interferometers. The observatories' mission is to detect and measure gravitational waves predicted by general relativity (Einstein's theory of gravity), in which gravity is described as due to the curvature of the

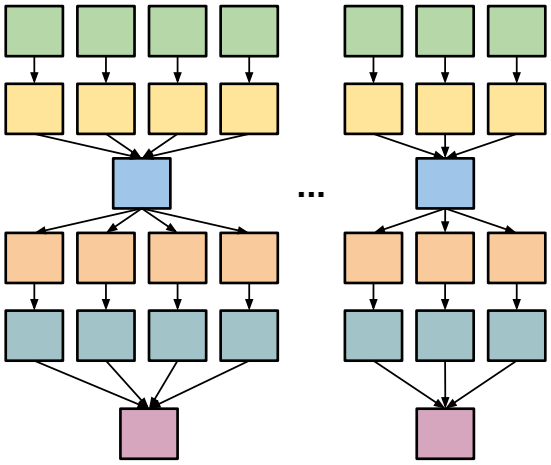

Figure 3.15: A simplified visualization of the LIGO Inspiral workflow.

fabric of time and space. The LIGO Inspiral workflow is a data-intensive workflow. Figure 3.15 shows a simplified version of this workflow. The LIGO Inspiral workflow is separated into multiple groups of interconnected tasks, which we call branches in the rest of our paper. However, each branch may have a different number of pipelines as shown in Figure 3.15.

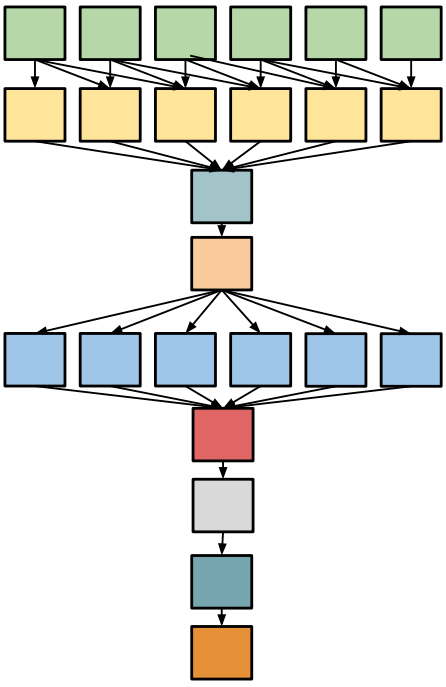

Figure 3.16: A simplified visualization of the Montage workflow.

Montage Montage [7] is an astronomy application that is used to construct large image mosaics of the sky. Input images are reprojected onto a sphere and overlap is calculated for each input image. The application re-projects input images to the correct orientation while keeping background emission level constant in all images. The images are added by rectifying them to a common flux scale and background level. Finally the reprojected images are co-added into a final mosaic. The resulting mosaic image can provide a much deeper and detailed understanding of the portion of the sky in question. Figure 3.16 illustrates a small Montage workflow. The size of the workflow depends on the number of images used in constructing the desired mosaic of the sky.

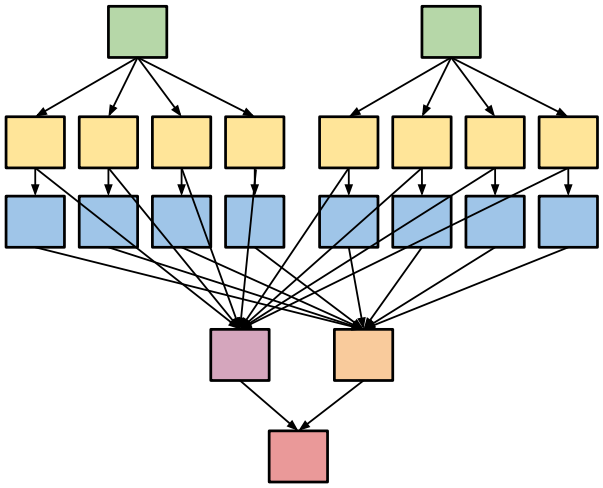

Figure 3.17: A simplified visualization of the CyberShake workflow.

Cybershake CyberShake [36] is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region. It identifies all ruptures within 200km of the site of interest and converts rupture definition into multiple rupture variations with differing hypocenter locations and slip distributions. It then calculates synthetic seismograms for each rupture variance, and peak intensity measures are then extracted from these synthetics and combined with the original rupture probabilities to produce probabilistic seismic hazard curves for the site. Figure 3.17 shows an illustration of the Cybershake workflow.

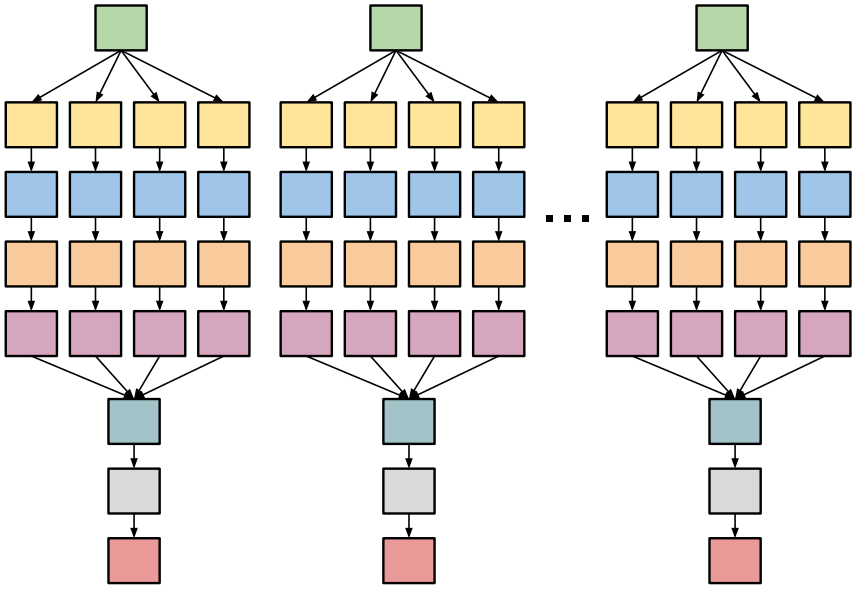

Figure 3.18: A simplified visualization of the Epigenomics workflow with multiple branches.

Epigenomics The Epigenomics workflow [97] is a data-parallel workflow. Initial data are acquired from the Illumina-Solexa Genetic Analyzer in the form of DNA sequence lanes. Each Solexa machine can generate multiple lanes of DNA sequences. These data are converted into a format that can be used by sequence mapping software. The mapping software can do one of two major tasks. It either maps short DNA reads from the sequence data onto a reference genome, or it takes all the short reads, treats them as small pieces in a puzzle and then tries to assemble an entire genome. In our experiments, the workflow maps DNA sequences to the correct locations in a reference Genome. This generates a map that displays the sequence density showing how many times a certain sequence expresses itself on a particular location on the reference genome. Epigenomics is a CPU-intensive application and its simplified structure is shown in Figure 3.18.

SIPHT The SIPHT workflow [83] conducts a wide search for small untranslated RNAs (sRNAs) that regulates several processes such as secretion or virulence in bacteria. The kingdom-wide prediction and annotation of sRNA encoding genes involves a variety of individual programs

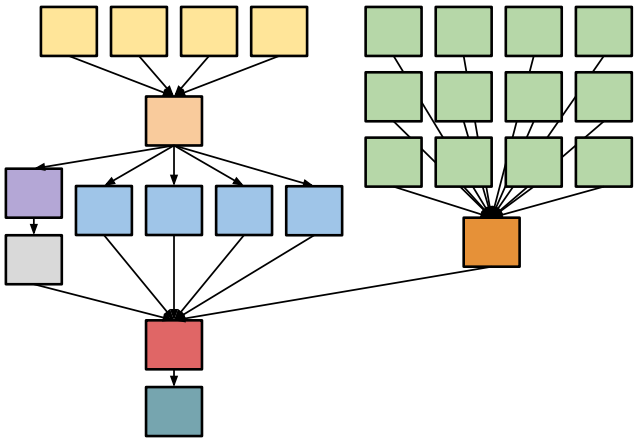

Figure 3.19: A simplified visualization of the SIPHT workflow.

that are executed in the proper order using Pegasus [25]. These involve the prediction of ρ-independent transcriptional terminators, BLAST (Basic Local Alignment Search Tools [6]) comparisons of the inter genetic regions of different replicons and the annotations of any sRNAs that are found. A simplified structure of the SIPHT workflow is shown in Figure 3.19.

Workflow	Number of Tasks	Average Data Size	Average Task Runtime
LIGO	800	5 MB	228s
Montage	300	3 MB	11s
CyberShake	700	148 MB	23s
Epigenomics	165	355 MB	2952s
SIPHT	1000	360 KB	180s

Table 3.2: Summary of the scientific workflows characteristics.

Table 3.2 shows the summary of the main **workflow characteristics**: number of tasks, average data size, and average task runtimes for the five workflows.

3.6 Summary

In this chapter, we have introduced the workflow and system models including an overhead aware DAG model that we have proposed, the execution environments including a workflow simulator that we have developed, and the workflows used in this thesis. As indicated by Table 3.1,

resources particular the data storage are limited in a data center. In next chapter, we will introduce the first approach called Workflow Partitioning to address the optimization of task granularity with resource constraints.

Chapter 4

Data Aware Workflow Partitioning

When mapping data-intensive tasks to compute resources, scheduling mechanisms need to take into account not only the execution time of the tasks, but also the overheads of staging the dataset. This also applies to the task clustering problem since merging tasks usually means data transfers associated with these tasks have to be merged as well. In this chapter, we introduce our work on data aware workflow partitioning that divides large scale workflows into several sub-workflows that are fit for execution within a single execution site. Three widely used workflows have been used to evaluate the effectiveness of out methods and the experiments show an runtime improvement of up to 48.1%.

4.1 Motivation

Data movement between tasks in scientific workflows has received less attention compared to task execution. Often the staging of data between tasks is either assumed or the time delay in data transfer is considered to be negligible compared to task execution, which is not true in many cases, especially in data-intensive applications. In this chapter, we take the data transfer into consideration and propose to partition large workflows into several sub-workflows where each sub-workflows can be executed within one execution site.

The motivation behind workflow partitioning starts from a common scenario where a researcher at a research institution typically has access to several research clusters, each of which may consist of a small number of nodes. The nodes in one cluster may be very different from those in another cluster in terms of file system, execution environment, and security systems. For example, we have access to FutureGrid [34], Teragrid/XSEDE [91] and Amazon EC2 [2] but each cluster imposes a limit on the resources, such as the maximum number of nodes a user

can allocate at one time or the maximum storage. If these isolated clusters can work together, they collectively become more powerful.

Additionally, the input dataset could be very large and widely distributed across multiple clusters. Data-intensive workflows require significant amount of storage and computation and therefore the storage system becomes a bottleneck. For these workflows, we need to use multiple execution sites and consider their available storage. For example, the entire CyberShake earthquake science workflow has 16,000 sub-workflows and each sub-workflow has more than 24,000 individual jobs and requires 58 GB of data. In this chapter, we assume we have Condor installed at the execution sites. A Condor pool can be either a physical cluster or a virtual cluster.

The first benefit of workflow partitioning is that this approach reduces the complexity of workflow mapping. For example, the entire CyberShake workflow has more than 3.8×10^8 tasks, which is a significant load for workflow management tools to maintain or schedule. In contrast, each sub-workflow has 24,000 tasks, which is acceptable for workflow management tools. A sub-workflow is a workflow and also a job of a higher-level workflow. What is more, workflow partitioning provides a fine granularity adjustment of workflow activities so that each sub-workflow can be adequate for one execution site. In the end, workflow partitioning allows us to migrate or retry sub-workflows efficiently. The overall workflow can be partitioned into sub-workflows and each sub-workflow can to be executed in different execution environments such as a hybrid platform of Condor/DAGMan [24] and MPI/DAGMan [58]) while the traditional task clustering technique requires all the tasks can be executed in the same execution environment.

4.2 Approach

To efficiently partition workflows, we proposed a three-phase scheduling approach integrated with the Pegasus Workflow Management System to partition, estimate, and schedule workflows onto distributed resources. Our contribution includes three heuristics to partition workflows respecting storage constraints and internal job parallelism. We utilize three methods to estimate

and compare runtime of sub-workflows and then we schedule them based on two commonly used algorithms (MinMin and HEFT).

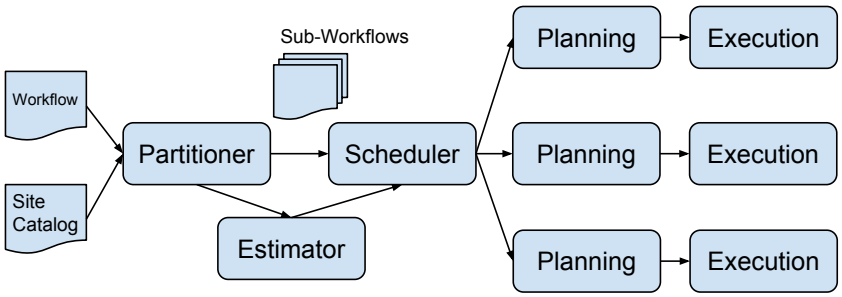

Figure 4.1: The steps to partition and schedule a workflow

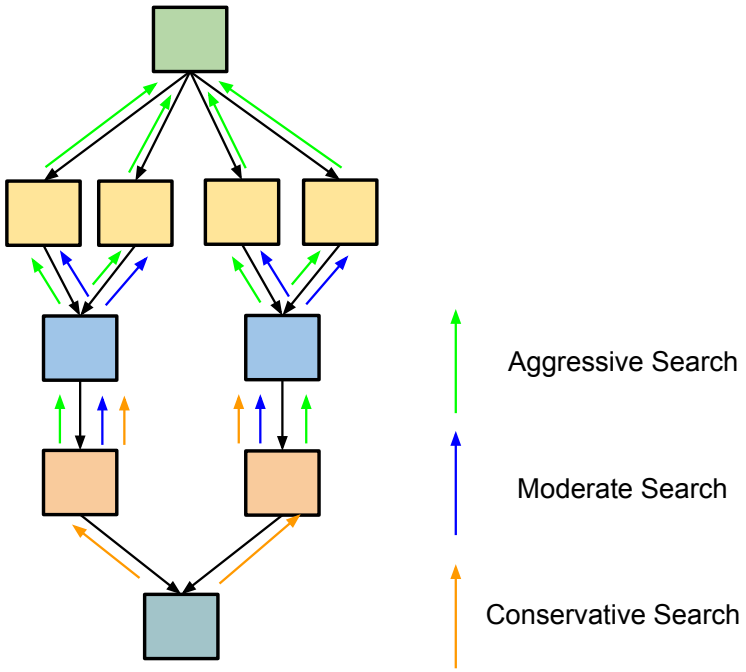

Figure 4.2: Three Steps of Search

Our approach (see Figure 4.1) has three phases: partition, estimate and schedule. The partitioner takes the original workflow and site catalog as input, and outputs various sub-workflows that respect the storage constraintsthis means that the data requirements of a sub-workflow are

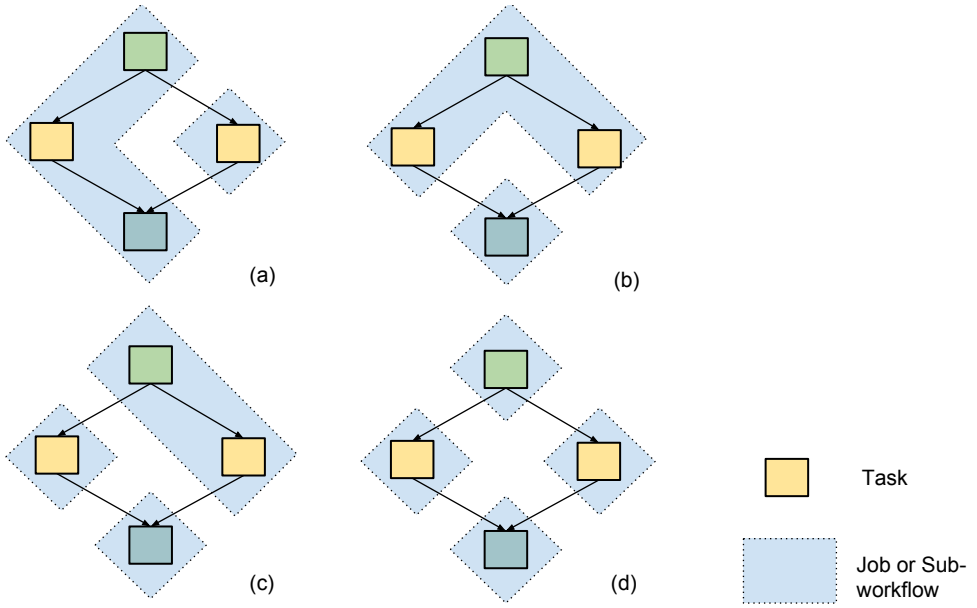

Figure 4.3: Four Partitioning Methods

within the data storage limit of a site. The site catalog provides information about the available resources. The estimator provides the runtime estimation of the sub-workflows and supports three estimation methods. The scheduler maps these sub-workflows to resources considering storage requirement and runtime estimation. The scheduler supports two commonly used algorithms. We first guarantee to find a valid mapping of sub-workflows satisfying storage constraints. Then we optimize performance based on these generated sub-workflows and schedule them to appropriate execution sites if runtime information for individual jobs is already known. If not, a static scheduler maps them to resources merely based on storage requirements.

The major challenge in partitioning workflows is to avoid cross dependency, which is a chain of dependencies that forms a cycle in graph (in this case cycles between sub-workflows). With cross dependencies, workflows are not able to proceed since they form a deadlock loop. For a simple workflow depicted in Figure 4.3, we show the result of four different partitioning. Partitioning (a) does not work in practice since it has a deadlock loop. Partitioning (c) is valid but not efficient compared to Partitioning (b) or (d) that have more parallelism.

Usually jobs that have parent-child relationships share a lot of data since they have data dependencies. Its reasonable to schedule such jobs into the same partition to avoid extra data transfer and also to reduce the overall runtime. Thus, we propose Heuristic I to find a group of parents and children. Our heuristic only checks three particular types of nodes: the fan-out job, the fan-in job, and the parents of the fan-in job and search for the potential candidate jobs that have parent-child relationships between them. The check operation means checking whether one particular job and its potential candidate jobs can be added to a sub-workflow while respecting storage constraints. Thus, our algorithm reduces the time complexity of check operations by n folds, while n is the average depth of the fan-in-fan-out structure. The check operation takes more time than the search operation since the calculation of data usage needs to check all the data allocated to a site and see if there is data overlap. Similar to [96], the algorithm starts from the sink job and proceeds upward.

To search for the potential candidate jobs that have parent-child relationships, the partitioner tries three steps of searches. For a fan-in job, it first checks if its possible to add the whole fan structure into the sub-workflow (aggressive search). If not, similar to Figure 4.3(d), a cut is issued between this fan-in job and its parents to avoid cross dependencies and increase parallelism. Then a less aggressive search is performed on its parent jobs, which includes all of its predecessors until the search reaches a fan-out job. If the partition is still too large, a conservative search is performed, which includes all of its predecessors until the search reaches a fan-in job or a fan-out job. Figure 4.2 depicts an example of three steps of search while the workflow in it has an average depth of 4. Pseudo-code of Heuristic I is depicted in Algorithm 1 .

We propose two other heuristics to solve the problem of cross dependency. The motivation for Heuristic II is that Partitioning (c) in Figure 4.3 is able to solve the problem. The motivation for Heuristic III is an observation that partitioning a fan structure into multiple horizontal levels is able to solve the problem. Heuristic II adds a job to a sub-workflow if all of its unscheduled children can be added to that sub-workflow without causing cross dependencies or exceed the storage constraint. Heuristic III adds a job to a sub-workflow if two conditions are met:

1. For a job with multiple children, each child has already been scheduled.

2. After adding this job to the sub-workflow, the data size does not exceed the storage constraint.

To optimize the workflow performance, runtime estimation for sub-workflows is required assuming runtime information for each job is already known. We provide three methods.

1. Critical Path is defined as the longest depth of the sub-workflow weighted by the runtime of each job.

2. Average CPU Time is the quotient of cumulative CPU time of all jobs divided by the number of available resources (its the number of Condor slots in our experiments, which is also the maximum number of Condor jobs that can be run on one machine).

3. The HEFT estimator uses the calculated earliest finish time of the last sink job as makespan of sub-workflows assuming that we use HEFT to schedule sub-workflows.

The scheduler selects appropriate resources for the sub-workflows satisfying the storage constraints and optimizes the runtime performance. Since the partitioning step has already guaranteed that there is a valid mapping, this step is called re-ordering or post-scheduling. We select HEFT[96] and MinMin[9], which represent global and local optimizations respectively. But there are two differences compared to their original versions. First, the data transfer cost within a sub-workflow is ignored since we use a shared file system in our experiments. Second, the data constraints must be satisfied for each sub-workflow. The scheduler selects an optimal set of resources in terms of available Condor slots since its the major factor influencing the performance. This work can be easily extended to considering more factors. Although some more comprehensive algorithms can be adopted, HEFT or MinMin are able to find an optimal schedule in terms that the sub-workflows are already generated since the number of sub-workflows has been greatly reduced compared to the number of individual jobs.

Algorithm 1 Workflow Partitioning algorithm

Require: G: workflow; $SL[index]$: site list, which stores all information about a compute site
Ensure: Create a subworkflow list SWL that does not exceed storage constraints

```
1:  procedure ParWorkflow(G, SL)
2:      index ← 0
3:      Q ← new Queue()
4:      Add the sink job of G to Q
5:      S ← new subworkflow()
6:      while Q is not empty do
7:          j ← the last job in Q
8:          Aggressive-Search(j)                          ▷ for fan-in job
9:          C ← the list of potential candidate jobs to be added to S in SL[index]
10:         P ← the list of parents of all candidates
11:         D ← the data size in SL[index] with C
12:         if D > storage constraint of SL[index] then
13:             Less-Aggressive-Search(j), update C, P, D
14:             if D > storage constraint of SL[index] then
15:                 Conservative-Search(j), update C, P, D
16:             end if
17:         end if
18:         ...                                           ▷ for other jobs
19:         if S causes cross dependency in SL[index] then
20:             S = new subworkflow()
21:         end if
22:         Add all jobs in C to S
23:         Add all jobs in P to the head of Q
24:         Add S to SWL[index]
25:         if S has no enough space left then
26:             index + +
27:         end if
28:         ...                                           ▷ for other situations
29:         Remove j from Q
30:     end while
31:     return SWL
32: end procedure
```

4.3 Experiments and Discussion

In order to quickly deploy and reconfigure computational resources, we use a private cloud computing resource running Eucalyptus [65]. Eucalyptus is an infrastructure software that provides on-demand access to Virtual Machine (VM) resources. In all the experiments, each VM has

4 CPU cores, 2 Condor slots, 4GB RAM and has a shared file system mounted to make sure data staged into a site is accessible to all compute nodes. In the initial experiments we build up four clusters, each with 4 VMs, 8 Condor slots. In the last experiment of site selection, the four virtual clusters are reconfigured and each cluster has 4, 8, 10 and 10 Condor slots respectively. The submit host that performs workflow planning and which sends jobs to the execution sites is a Linux 2.6 machine equipped with 8GB RAM and an Intel 2.66GHz Quad CPUs. We use Pegasus to plan the workflows and then submit them to Condor DAGMan [24], which provides the workflow execution engine. Each execution site contains a Condor pool and a head node visible to the network.

Table 4.1: CyberShake with Storage Constraint

storage constraint	site	Disk Usage(GB)	Percentage
35GB	A	sub0:0.06; sub1:33.8	97%
	B	sub2:28.8	82%
30GB	A	sub0:0.07;sub1:29.0	97%
	B	sub2:29.3	98%
	C	sub3:28.8	96%
25GB	A	sub0:0.06;sub1:24.1	97%
	B	sub2:24.4	98%
	C	sub3:19.5	78%
20GB	A	sub0:0.06;sub1:18.9	95%
	B	sub2:19.3	97%
	C	sub3:19.6	98%
	D	sub4:15.3	77%

Performance Metrics. To evaluate the performance, we use two types of metrics. Satisfying the Storage Constraints is the main goal of our work in order to fit the sub-workflows into the available storage resources. We compare the results of different storage constraints and heuristics. Improving the Runtime Performance is the second metric that is concerned with in order to minimize the overall makespan. We compare the results of different partitioners, estimators and schedulers.

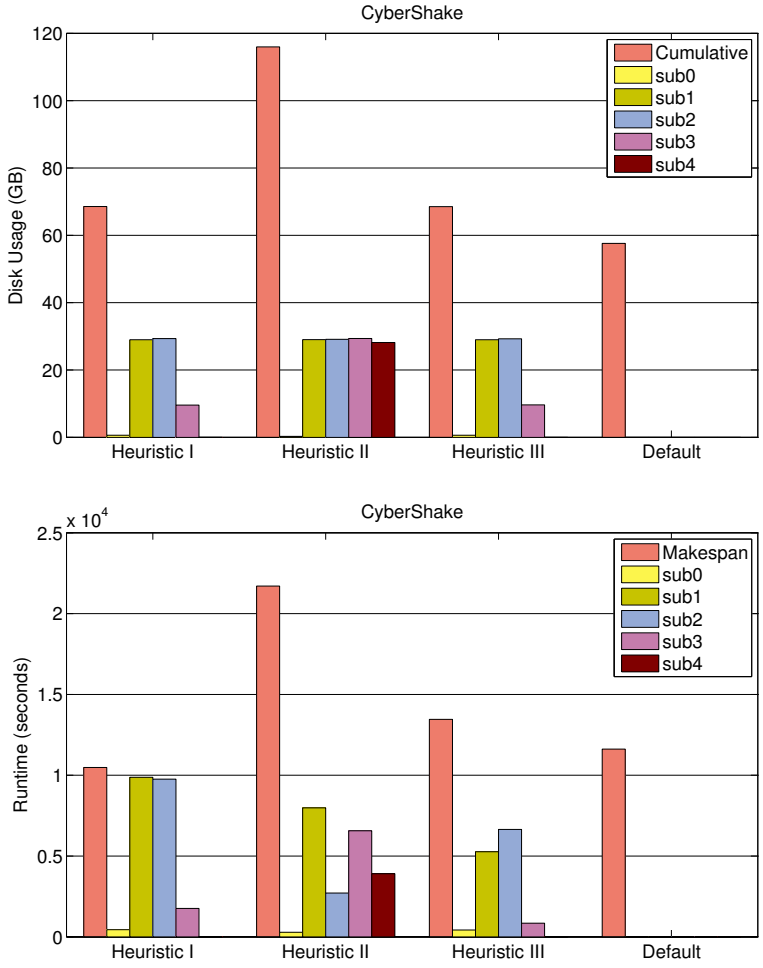

Figure 4.4: Performance of the three heuristics. The default workflow has one execution site with 4 VMs and 8 Condor slots and has no storage constraint.

Workflows Used. We ran three different workflow applications: an astronomy application (Montage), a seismology application (CyberShake) and a bioinformatics application (Epigenomics). They were chosen because they represent a wide range of application domains and a variety of resource requirements [44].

Performance of Different Heuristics. We compare the three heuristics with the CyberShake application. The storage constraint for each site is 30GB. Heuristic II produces 5 sub-workflows with 10 dependencies between them. Heuristic I produces 4 sub-workflows and 3 dependencies. Heuristic III produces 4 sub-workflows and 5 dependencies. The results are shown in Figure 4.4

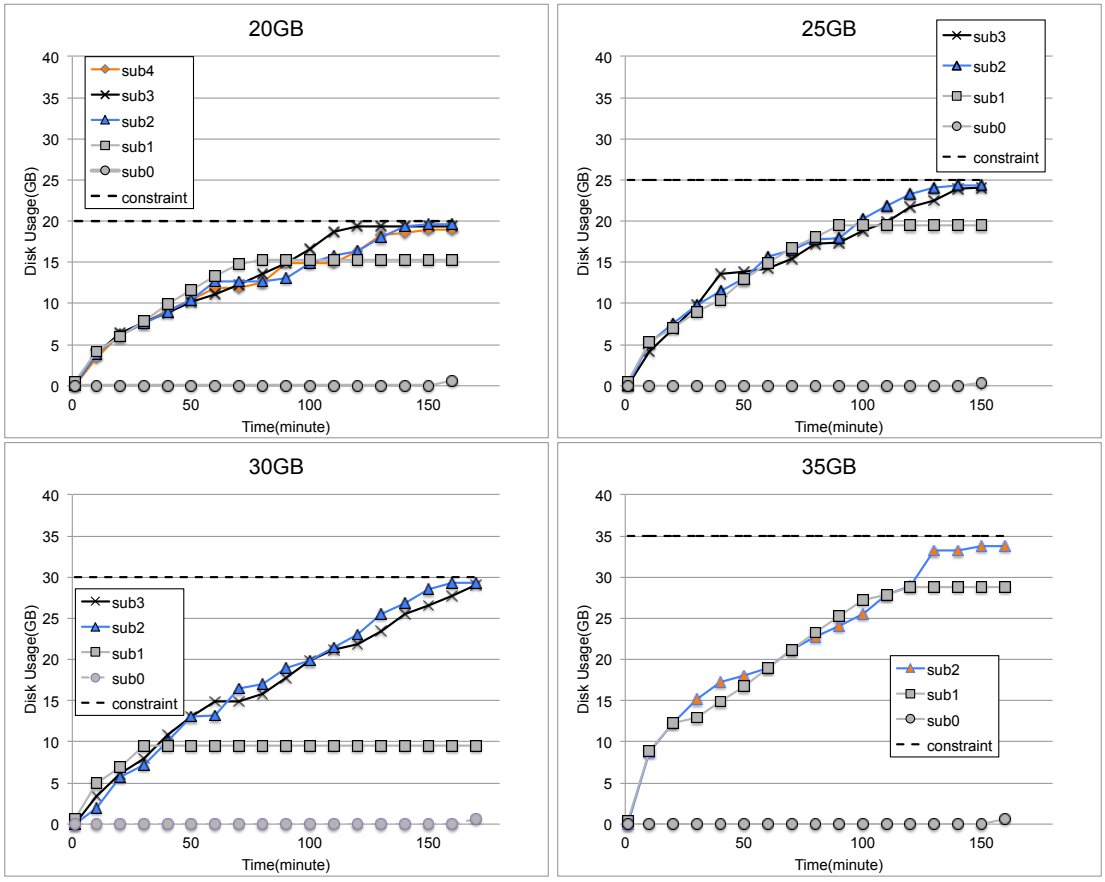

Figure 4.5: CyberShake with storage constraints of 35GB, 30GB, 25GB, and 20GB. They have 3, 4, 4, and 5 sub-workflows and require 2, 3, 3, and 4 sites to run respectively.

and Heuristic I performs better in terms of both runtime reduction and disk usage. This is due to the way it handles the cross dependency. Heuristic II or Heuristic III simply adds a job if it does not violate the storage constraints or the cross dependency constraints. Furthermore, Heuristic I puts the entire fan structure into the same sub-workflow if possible and therefore reduces the dependencies between sub-workflows. From now on, we only use Heuristic I in the partitioner in our experiments below.

Performance with Different Storage Constraints. Figure 4.5 and Table 4.1 depict the disk usage of the CyberShake workflows over time with storage constraints of 35GB, 30GB, 25GB, and 20GB. They are chosen because they represent a variety of required execution sites. Figure 4.6 depicts the performance of both disk usage and runtime. Storage constraints for all

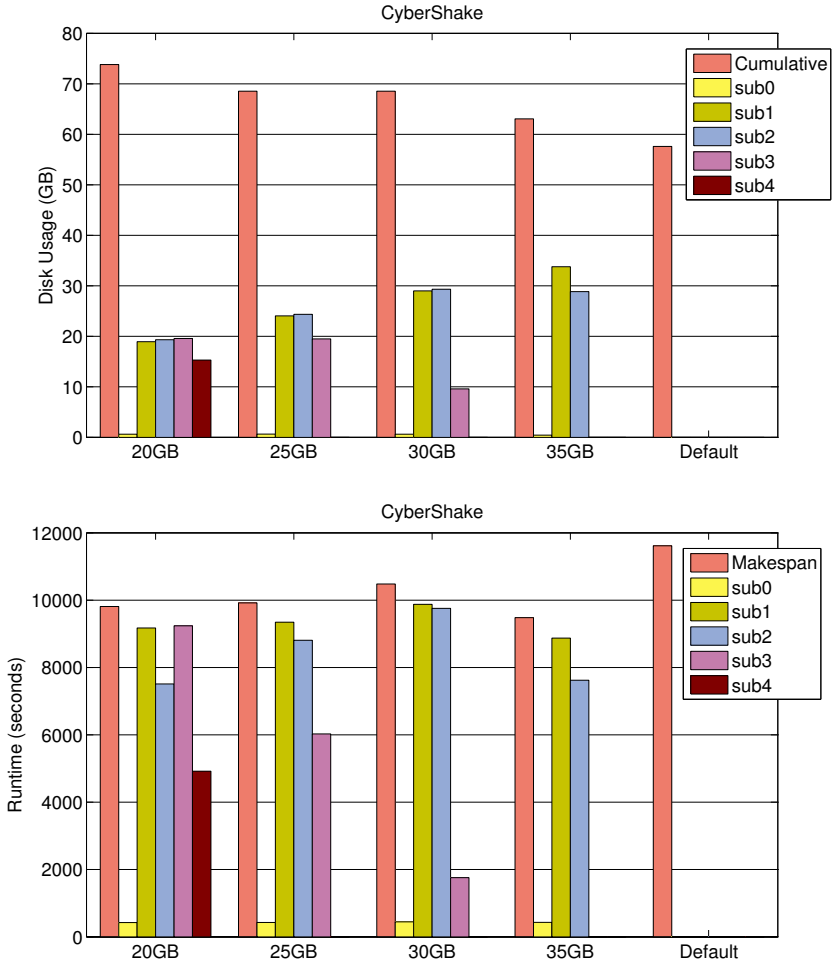

Figure 4.6: Performance of the CyberShake workflow with different storage constraints

Table 4.2: Performance of estimators and schedulers

Combination	Estimator	Scheduler	Makespan(second)
HEFT+HEFT	HEFT	HEFT	10559.5
PATH+HEFT	Critical Path	HEFT	12025.4
CPU+HEFT	Average CPU Time	HEFT	12149.2
HEFT+MIN	HEFT	MinMin	10790
PATH+MIN	Critical Path	MinMin	11307.2
CPU+MIN	Average CPU Time	MinMin	12323.2

of the sub-workflows are satisfied. Among them sub1, sub2, sub3 (if exists), and sub4 (if exists) are run in parallel and then sub0 aggregates their work. The CyberShake workflow across two sites with a storage constraint of 35GB performs best. The makespan (overall completion time)

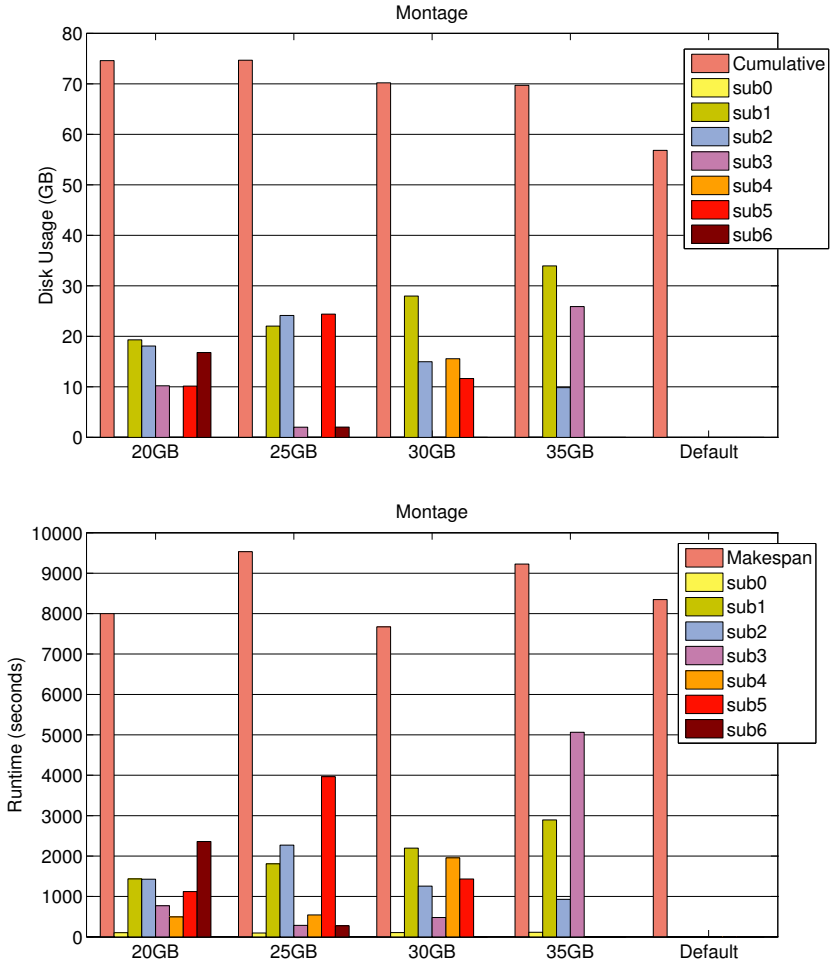

Figure 4.7: Performance of the Montage workflow with different storage constraints

improves by 18.38% and the cumulative disk usage increases by 9.5% compared to the default workflow without partitioning or storage constraints. The cumulative data usage is increased because some shared data is transferred to multiple sites. Workflows with more sites to run on do not have a smaller makespan because they require more data transfer even though the computation part is improved.

Figure 4.8 depicts the performance of Montage with storage constraints ranging from 20GB to 35GB and Epigenomics with storage constraints ranging from 8.5GB to 15GB. The Montage workflow across three sites with 30GB disk space performs best with 8.1% improvement in makespan and the cumulative disk usage increases by 23.5%. The Epigenomics workflow across

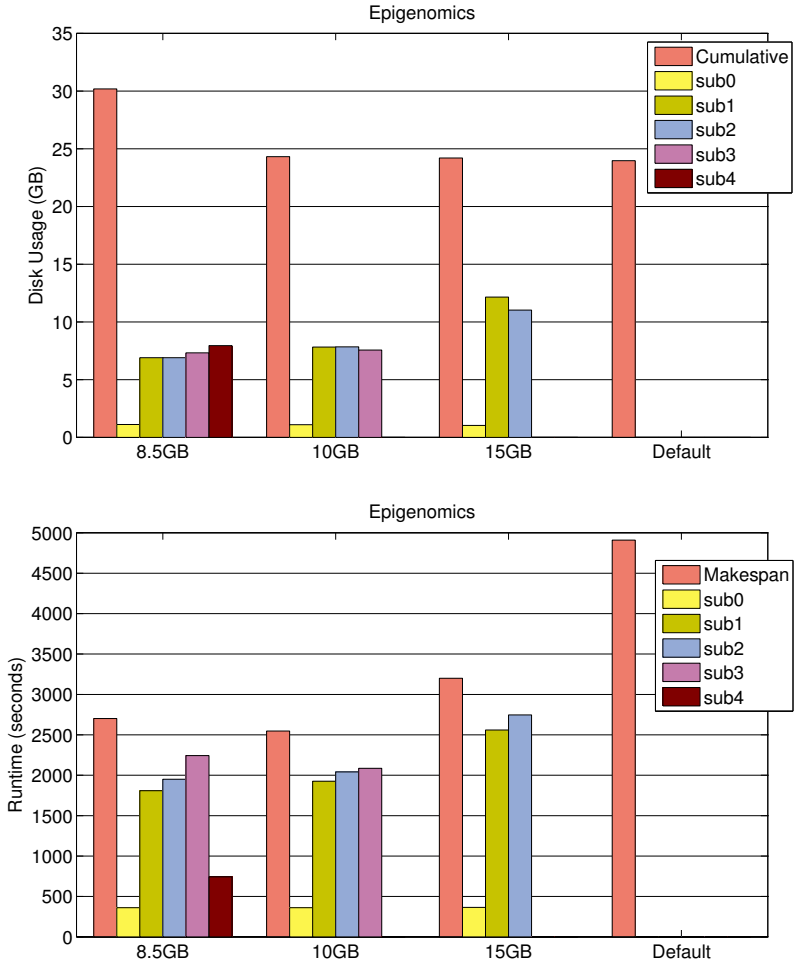

Figure 4.8: Performance of the Epigenomics workflow with different storage constraints

three sites with 10GB storage constraints performs best with 48.1% reduction in makespan and only 1.4% increase in cumulative storage. The reason why Montage performs worse is related to its complex internal structures. Montage has two levels of fan-out-fan-in structures and each level has complex dependencies between them.

Site selection. To show the performance of site selection for each sub-workflow, we use three estimators and two schedulers together with the CyberShake workflow. We build four execution sites with 4, 8, 10 and 10 Condor slots respectively. The labels in Figure 4.9 and Table 4.2 are defined in a way of Estimator + Scheduler. For example, HEFT+HEFT denotes a combination of HEFT estimator and HEFT scheduler, which performs best as we expected. The

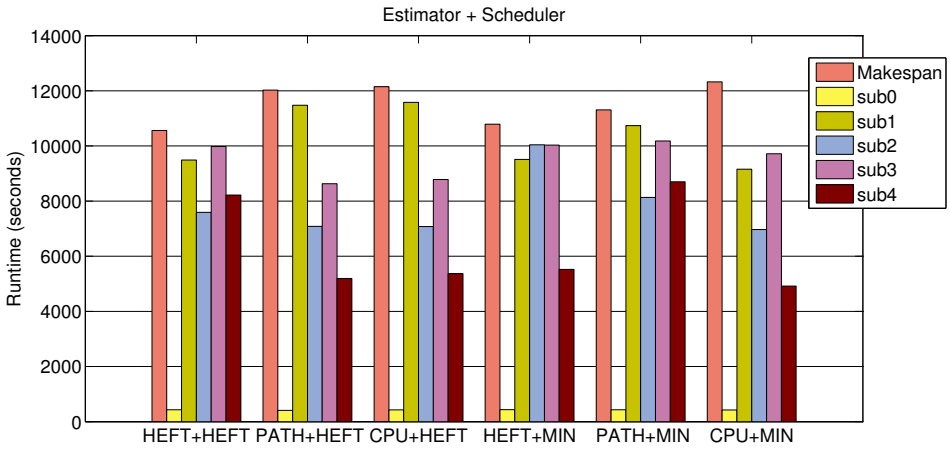

Figure 4.9: Performance of estimators and schedulers

Average CPU Time (or CPU in Figure 4.7) does not take the dependencies into consideration and the Critical Path (or PATH in Figure 4.9) does not consider the resource availability. The HEFT scheduler is slightly better than MinMin scheduler (or MIN in Figure 4.9). Although HEFT scheduler uses a global optimization algorithm compared to MinMins local optimization, the complexity of scheduling sub-workflows has been greatly reduced compared to scheduling a vast number of individual tasks. Therefore, both local and global optimization algorithms are able to handle such situations well.

In conclusion, we provide a solution to address the problem of scheduling large workflows across multiple sites with storage constraints. The approach relies on partitioning the workflow into valid sub-workflows. Three heuristics are proposed and compared to show the close relationship between cross dependency and runtime improvement. The performance with three workflows shows that this approach is able to satisfy the storage constraints and reduce the makespan significantly especially for Epigenomics which has fewer fan-in (synchronization) jobs. For the workflows we used, scheduling them onto two or three execution sites is best due to a tradeoff between increased data transfer and increased parallelism. Site selection shows that the global optimization and local optimization perform almost the same.

4.4 Summary

for other constraints

Bibliography

[1] S. Ali, A. Maciejewski, H. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *Parallel and Distributed Systems, IEEE Transactions on*, 15(7):630–641, 2004.

[2] Amazon.com, Inc. Amazon Web Services. `http://aws.amazon.com`.

[3] M. Amer, A. Chervenak, and W. Chen. Improving scientific workflow performance using policy based data placement. In *Policies for Distributed Systems and Networks (POLICY), 2012 IEEE International Symposium on*, pages 86–93, 2012.

[4] T. Andrews, F. Curbera, H. Dholakian, Y. Goland, J. Kleini, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[6] Basic local alignment search tools. `http://blast.ncbi.nlm.nih.gov/Blast.cgi`.

[7] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M. Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *SPIE Conference on Astronomical Telescopes and Instrumentation*, June 2004.

[8] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi. Characterization of scientific workflows. In *3rd Workshop on Workflows in Support of Large Scale Science (WORKS 08)*, 2008.

[9] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, 2005.

[10] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, and R. F. Freund. A comparison of eleven static heuristic for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, Jan. 2011.

[12] S. Callaghan, E. Deelman, D. Gunter, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner, R. Graves, E. Field, et al. Scaling up workflow-based applications. *Journal of Computer and System Sciences*, 76(6):428–446, 2010.

[13] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T. Jordan, E. Deelman, G. Mehta, K. Vahi, D. Gunter, K. Beattie, and C. X. Brooks. Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. *International Journal of High Performance Computing Applications*, 25(3):274–285, 2011.

[14] Y. Caniou, G. Charrier, and F. Desprez. Evaluation of reallocation heuristics for moldable tasks in computational grids. In *Proceedings of the Ninth Australasian Symposium on Parallel and Distributed Computing - Volume 118*, AusPDC '11, pages 15–24, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

[15] J. Cao, S. Jarvis, D. Spooner, J. Turner, D. Kerbyson, and G. Nudd. Performance prediction technology for agent-based resource management in grid environments. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002)*, page 14, April 2002.

[16] J. Celaya and L. Marchal. A fair decentralized scheduler for bag-of-tasks applications on desktop grids. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:538–541, 2010.

[17] W. Chen and E. Deelman. Workflow overhead analysis and optimizations. In *The 6th Workshop on Workflows in Support of Large-Scale Science*, Nov. 2011.

[18] W. Chen and E. Deelman. Fault tolerant clustering in scientific workflows. In *IEEE Eighth World Congress on Services (SERVICES)*, pages 9–16, 2012.

[19] W. Chen and E. Deelman. Integration of workflow partitioning and resource provisioning. In *The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, May 2012.

[20] W. Chen and E. Deelman. Partitioning and scheduling workflows across multiple sites with storage constraints. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics - Volume Part II*, PPAM'11, pages 11–20, Berlin, Heidelberg, 2012. Springer-Verlag.

[21] W. Chen, E. Deelman, and R. Sakellariou. Imbalance optimization in scientific workflows. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, ICS '13, pages 461–462, 2013.

[22] W. Chen, R. Ferreira da Silva, E. Deelman, and R. Sakellariou. Balanced task clustering in scientific workflows. In *2013 IEEE 9th International Conference on eScience*, pages 188–195, 2013.

[23] R. F. da Silva, G. Juve, E. Deelman, T. Glatard, F. Desprez, D. Thain, B. Tovar, and M. Livny. Toward fine-grained online task characteristics estimation in scientific workflows. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, pages 58–67. ACM, 2013.

[24] DAGMan: Directed Acyclic Graph Manager. `http://cs.wisc.edu/condor/dagman`.

[25] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Across Grid Conference*, 2004.

[26] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: an overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, May 2009.

[27] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, and S. Koranda. GriPhyN and LIGO: building a virtual data grid for gravitational wave scientists. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, 2002.

[28] E. Deelman, G. Singh, M. 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*, 13(3):219–237, 2005.

[29] F. Dong and S. G. Akl. Two-phase computation and data scheduling algorithms for workflows in the grid. In *2007 International Conference on Parallel Processing*, page 66, Oct. 2007.

[30] R. Duan, F. Nadeem, J. Wang, Y. Zhang, R. Prodan, and T. Fahringer. A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 339–347, 2009.

[31] R. Duan, R. Prodan, and T. Fahringer. Run-time optimisation of grid workflow applications. In *7th IEEE/ACM International Conference on Grid Computing*, pages 33–40, Sept. 2006.

[32] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, et al. Askalon: A development and grid computing environment for scientific workflows. In *Workflows for e-Science*, pages 450–471. Springer, 2007.

[33] R. Ferreira da Silva, T. Glatard, and F. Desprez. On-line, non-clairvoyant optimization of workflow activity granularity on grids. In *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 255–266. Springer Berlin Heidelberg, 2013.

[34] G. C. Fox, G. V. Laszewski, J. Diaz, K. Keahey, R. Figueiredo, S. Smallen, and W. Smith. Futuregrid - a reconfigurable testbed for cloud, hpc and grid computing. In *Contemporary High Performance Computing: From Petascale toward Exascale*, 2013.

[35] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: a computation management agent for Multi-Institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[36] R. Graves, T. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi. CyberShake: A Physics-Based Seismic Hazard Model for Southern California. *Pure and Applied Geophysics*, 168(3-4):367–381, May 2010.

[37] Z. Guo, M. Pierce, G. Fox, and M. Zhou. Automatic task re-organization in mapreduce. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 335–343, 2011.

[38] M. K. Hedayat, W. Cai, S. J. Turner, and S. Shahand. Distributed execution of workflow using parallel partitioning. In *Parallel and Distributed Processing with Applications, 2009 IEEE International Symposium on*, pages 106–112. IEEE, 2009.

[39] T. Hey, S. Tansley, and K. Tolle. *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research, 2009.

[40] M. Hussin, Y. C. Lee, and A. Y. Zomaya. Dynamic job-clustering with different computing priorities for computational resource allocation. In *The 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010.

[41] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, HPDC '08, pages 97–108, New York, NY, USA, 2008. ACM.

[42] W. M. Jones, L. W. Pang, W. Ligon, and D. Stanzione. Bandwidth-aware co-allocating meta-schedulers for mini-grid architectures. In *Cluster Computing, 2004 IEEE International Conference on*, pages 45–54. IEEE, 2004.

[43] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. volume 29, pages 682 – 692, 2013. Special Section: Recent Developments in High Performance Computing and Security.

[44] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Scientific workflow applications on amazon EC2. In *2009 5th IEEE International Conference on E-Science Workshops*, Dec. 2009.

[45] S. Kalayci, G. Dasgupta, L. Fong, O. Ezenwoye, , and S. Sadjadi. Distributed and adaptive execution of condor dagman workflows. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE'2010)*, July 2010.

[46] 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*, pages 342–349, 2004.

[47] 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*, 2002.

[48] D. Laforenza, R. Lombardo, M. Scarpellini, M. Serrano, F. Silvestri, and P. Faccioli. Biological experiments on the grid: A novel workflow management platform. In *20th IEEE International Symposium on Computer-Based Medical Systems (CBMS'07)*, pages 489–494. IEEE, 2007.

[49] Laser Interferometer Gravitational Wave Observatory (LIGO). `http://www.ligo.caltech.edu`.

[50] A. Lathers, M. Su, A. Kulungowski, A. Lin, G. Mehta, S. Peltier, E. Deelman, and M. Ellisman. Enabling parallel scientific applications with workflow tools. In *Challenges of Large Applications in Distributed Environments (CLADE 2006)*, 2006.

[51] H. Li, D. Groep, and L. Wolters. Efficient response time predictions by exploiting application and resource state similarities. In *The 6th IEEE/ACM International Workshop on Grid Computing*, page 8, Nov 2005.

[52] J. Lifflander, S. Krishnamoorthy, and L. V. Kale. Work stealing and persistence-based load balancers for iterative overdecomposed applications. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, June 2012.

[53] Q. Liu and Y. Liao. Grouping-based fine-grained job scheduling in grid computing. In *First International Workshop on Education Technology and Computer Science*, Mar. 2009.

[54] X. Liu, J. Chen, K. Liu, and Y. Yang. Forecasting duration intervals of scientific workflow activities based on time-series patterns. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 23–30, 2008.

[55] Y. Ma, X. Zhang, and K. Lu. A graph distance based metric for data oriented workflow retrieval with variable time constraints. *Expert Syst. Appl.*, 41(4):1377–1388, Mar. 2014.

[56] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, and E. Field. SCEC CyberShake WorkflowsAutomating probabilistic seismic hazard analysis calculations. In I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 143–163. Springer, 2007.

[57] K. Maheshwari, A. Espinosa, D. S. Katz, M. Wilde, Z. Zhang, I. Foster, S. Callaghan, and P. Maechling. Job and data clustering for aggregate use of multiple production cyberinfrastructures. In *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date*, DIDC '12, pages 3–12, New York, NY, USA, 2012. ACM.

[58] R. Mats, G. Juve, K. Vahi, S. Callaghan, G. Mehta, and P. J. M. andEwa Deelman. Enabling large-scale scientific workflows on petascale resources using mpi master/worker. In *Proceedings of the 1st conference of the Extreme Science and Engineering Discovery Environment*, July 2012.

[59] S. R. McConnel, D. P. Siewiorek, and M. M. Tsao. The measurement and analysis of transient errors in digital computer systems. In *Proc. 9th Int. Symp. Fault-Tolerant Computing*, pages 67–70, 1979.

[60] N. Muthuvelu, I. Chai, E. Chikkannan, and R. Buyya. On-line task granularity adaptation for dynamic grid applications. In *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 266–277. 2010.

[61] N. Muthuvelu, I. Chai, and C. Eswaran. An adaptive and parameterized job grouping algorithm for scheduling grid jobs. In *10th International Conference on Advanced Communication Technology (ICACT 2008)*, volume 2, pages 975 –980, 2008.

[62] N. Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio, and R. Buyya. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In *Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, 2005.

[63] N. Muthuvelu, C. Vecchiola, I. Chai, E. Chikkannan, and R. Buyya. Task granularity policies for deploying bag-of-task applications on global grids. *Future Generation Computer Systems*, 29(1):170 – 181, 2013. ¡ce:title¿Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures¡/ce:title¿.

[64] W. K. Ng, T. Ang, T. Ling, and C. Liew. Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. *Malaysian Journal of Computer Science*, 19(2):117–126, 2006.

[65] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. UCSB Computer Science Technical Report 2008-10, 2008.

[66] 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*, 20(17):3045–3054, Nov. 2004.

[67] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. *Why do internet services fail and what can be done about it?* Computer Science Division, University of California, 2002.

[68] A. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 351–359, 30 2010-Dec. 3.

[69] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, and A. Iosup. Workflow monitoring and analysis tool for ASKALON. In *Grid and Services Evolution*. 2009.

[70] P.-O. Ostberg and E. Elmroth. Mediation of service overhead in service-oriented grid architectures. In *12th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 9–18, 2011.

[71] S.-M. Park and M. Humphrey. Data throttling for data-intensive workflows. In *IEEE Intl. Symposium on Parallel and Distributed Processing*, Apr. 2008.

[72] K. Plankensteiner, R. Prodan, T. Fahringer, A. Kertész, and P. Kacsuk. Fault detection, prevention and recovery in current grid workflow systems. In *Grid and Services Evolution*, pages 1–13. Springer, 2009.

[73] R. Prodan. Online analysis and runtime steering of dynamic workflows in the askalon grid environment. In *The Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 389 –400, May 2007.

[74] R. Prodan and T. Fabringer. Overhead analysis of scientific workflows in grid environments. In *IEEE Transactions in Parallel and Distributed System*, volume 19, Mar. 2008.

[75] R. Rodriguez, R. Tolosana-Calasanz, and O. Rana. Automating data-throttling analysis for data-intensive workflows. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pages 310–317, 2012.

[76] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proceedings of the International Conference on Dependable Systems and Networks*, July 2004.

[77] R. Sakellariou, H. Zhao, and E. Deelman. Mapping Workflows on Grid Resources: Experiments with the Montage Workflow. In F. Desprez, V. Getov, T. Priol, and R. Yahyapour, editors, *Grids P2P and Services Computing*, pages 119–132. 2010.

[78] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Mehta, F. Silva, and K. Vahi. Failure prediction and localization in large scientific workflows. In *The 6th Workshop on Workflows in Suppporting of Large-Scale Science*, Nov. 2011.

[79] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2006.

[80] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bała, E. Del Grosso, M. Casalegno, et al. Chemomentum-unicore 6 based infrastructure for complex applications in science and technology. In *Euro-Par 2007 Workshops: Parallel Processing*, pages 82–93. Springer, 2008.

[81] S. Shankar and D. J. DeWitt. Data driven workflow planning in cluster management systems. In *Proceedings of the 16th international symposium on High performance distributed computing*, HPDC '07, pages 127–136, New York, NY, USA, 2007. ACM.

[82] G. Singh, M. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D. S. Katz, and G. Mehta. Workflow task clustering for best effort systems with pegasus. In *15th ACM Mardi Gras Conference*, 2008.

[83] SIPHT. `http://pegasus.isi.edu/applications/sipht`.

[84] O. Sonmez, H. Mohamed, and D. Epema. Communication-aware job placement policies for the koala grid scheduler. In *Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, pages 79–79. IEEE, 2006.

[85] 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*, HPDC '10, pages 49–60, New York, NY, USA, 2010. ACM.

[86] Southern California Earthquake Center (SCEC). `http://www.scec.org`.

[87] C. Stratan, A. Iosup, and D. H. Epema. A performance study of grid workflow engines. In *9th IEEE/ACM International Conference on Grid Computing*, pages 25–32. IEEE, 2008.

[88] X.-H. Sun and M. Wu. Grid harvest service: a system for long-term, application-level task scheduling. In *International Parallel and Distributed Processing Symposium (IPDPS)*, page 8, April 2003.

[89] D. Tang, R. K. Iyer, and S. S. Subramani. Failure analysis and modeling of a vaxcluster system. In *Proceedings of the International Symposium on Fault-tolerant computing*, 1990.

[90] I. Taylor, M. Shields, I. Wang, and A. Harrison. The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[91] The TeraGrid Project. `http://www.teragrid.org`.

[92] T. L. L. P. T.F. Ang, W.K. Ng and C. Liew. A bandwidth-aware job grouping-based scheduling on grid environment. *Information Technology Journal*, (8):372–377, 2009.

[93] R. Tolosana-Calasanz, M. Lackovic, O. F Rana, J. Á. Bañares, and D. Talia. Characterizing quality of resilience in scientific workflows. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 117–126. ACM, 2011.

[94] R. Tolosana-Calasanz, O. F. Rana, and J. A. Bañares. Automating performance analysis from taverna workflows. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, CBSE '08, pages 1–15, Berlin, Heidelberg, 2008. Springer-Verlag.

[95] L. Tomas, B. Caminero, and C. Carrion. Improving grid resource usage: Metrics for measuring fragmentation. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '12)*, pages 352–359, 2012.

[96] H. Topcuoglu, S. Hariri, and W. Min-You. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[97] USC Epigenome Center. `http://epigenome.usc.edu`.

[98] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. In *ACM SIGMOD Record*, volume 34, pages 56–62, Sept. 2005.

[99] H. Ying, G. Mingqiang, L. Xiangang, and L. Yong. A webgis load-balancing algorithm based on collaborative task clustering. *Environmental Science and Information Application Technology, International Conference on*, 3:736–739, 2009.

[100] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3–4), 2005.

[101] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computer Systems*, 26(8):1200–1214, 2010.

[102] X. Zhang, Y. Qu, and L. Xiao. Improving distributed workload performance by sharing both cpu and memory resources. In *Proceedings of 20th International Conference on Distributed Computing Systems, (ICDCS'2000*, pages 233–241, 2000.

[103] Y. Zhang, A. Mandal, C. Koelbel, and K. Cooper. Combined fault tolerance and scheduling techniques for workflow applications on computational grids. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pages 244–251. IEEE, 2009.

[104] Y. Zhang and M. S. Squillante. Performance implications of failures in large-scale cluster scheduling. In *The 10th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2004.

[105] H. Zhao and X. Li. Efficient grid task-bundle allocation using bargaining based self-adaptive auction. In *The 9th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2009.

[106] G. Zheng, A. Bhatele, E. Meneses, and L. V. Kalé. Periodic hierarchical load balancing for large supercomputers. *International Journal of High Performance Computing Applications*, 25(4):371–385, 2011.