EFFICIENT SCHEDULING TECHNIQUES AND SYSTEMS FOR GRID COMPUTING

By

JANG-UK IN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2006

This document is dedicated to the graduate students of the University of Florida.

ACKNOWLEDGMENTS

I thank my parents, wife and all the co-workers. Especially heartfelt thanks go to Dr. Sanjay Ranka, chair of my committee, for his advice and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

EFFICIENT SCHEDULING TECHNIQUES AND ALGORITHMS FOR GRID COMPUTING

By

Jang-uk In

August 2006

Chair: Sanjay Ranka
Major Department: Computer and Information Science and Engineering

This dissertation discusses policy-based scheduling techniques on heterogeneous resource for grid computing. The proposed scheduling algorithm has the following features, which can be utilized on the grid computing environment. First, the algorithm supports the resource usage constrained scheduling. A grid consists of the resources that are owned by decentralized institutions. Second, the algorithm performs the optimization-based scheduling. It provides an optimal solution to the grid resource allocation problem. Third, the algorithm assumes that a set of resources is distributed geographically and is heterogeneous in nature. Fourth, the scheduling dynamically adjusts to the grid status. It tracks the current workload of the resources. The performance of the proposed algorithm is evaluated with a set of predefined metrics. In addition to showing the simulation results for the out-performance of the policy-based scheduling, a set of experiments is performed on open science grid (OSG). In this dissertation we discuss a novel framework for policy based scheduling in resource

allocation of grid computing. The framework has several features. First, the scheduling strategy can control the request assignment to grid resources by adjusting resource usage accounts or request priorities. Second, efficient resource usage management is achieved by assigning usage quotas to intended users. Third, the scheduling method supports reservation based grid resource allocation. Fourth, the quality of service (QOS) feature allows special privileges to various classes of requests, users, groups, etc. This framework is incorporated as part of the SPHINX scheduling system that is currently under development at the University of Florida. Experimental results are provided to demonstrate the usefulness of the framework. A grid consists of high-end computational, storage, and network resources that, while known a priori, are dynamic with respect to activity and availability. Efficient scheduling of requests to use grid resources must adapt to this dynamic environment while meeting administrative policies. In this dissertation, we describe a framework called SPHINX that can administer grid policies and schedule complex and data intensive scientific applications. We present experimental results for several scheduling strategies that effectively utilize the monitoring and job-tracking information provided by SPHINX. These results demonstrate that SPHINX can effectively schedule work across a large number of distributed clusters that are owned by multiple units in a virtual organization in a fault-tolerant way in spite of the highly dynamic nature of the grid and complex policy issues. The novelty lies in the use of effective monitoring of resources and job execution tracking in making scheduling decisions and fault-tolerance – something which is missing in today's grid environments.

CHAPTER 1
INTRODUCTION

Grid computing is increasingly becoming a popular way of achieving high performance computing for many scientific and commercial applications. The realm of grid computing is not limited to the one of parallel computing or distributed computing, as it requires management of disparate resources and different policies over multiple organizations.

Our research studies grid computing and related technologies. We propose novel grid resource management middleware and efficient scheduling techniques. This chapter discusses grid computing issues and technologies. Specifically, we discuss the major difference between the new computing paradigm and existing parallel and distributed computing. We introduce new concepts and terminologies defined in grid computing. We then present the proposed scheduling system and technologies, and discuss how it affects and contributes to the computing community.

Grid Computing

Data generated by scientific applications are now routinely stored in large archives that are geographically distributed. Rather than observing the data directly, a scientist effectively peruses these data archives to find nuggets of information [1]. Typical searches require multiple weeks of computing time on a single workstation. The scientific applications that have these properties are discussed in detail in the upcoming sections of this chapter.

Grid computing has become a popular way of providing high performance computing for many data intensive, scientific applications. Grid computing allows a number of competitive and/or collaborative organizations to share mutual resources, including documents, software, computers, data and sensors and computationally intensive applications to seamlessly process data [2, 3]. The realm of grid computing is beyond parallel or distributed computing in terms of requiring the management of a large number of heterogeneous resources with varying, distinct policies controlled by multiple organizations.

Most scientific disciplines used to be either empirical or theoretical. In the past few decades, computational science has become a new branch in these disciplines. In the past computational science was limited to simulation of complex models. However in recent years it also encapsulates information management. This has happened because of the following trends: (1) Large amounts of data are available from scientific and medical equipment, (2) the cost of storage has decreased substantially, and (3) the development of Internet technologies allows the data to be accessible to any person at any location.

The applications developed by scientists on this data tend to be both computationally and data intensive. An execution may require tens of days on a single workstation. In many cases it would not be feasible to complete this execution on a single workstation due to extensive memory and storage requirements. Computational grid addresses these and many other issues by allowing a number of competitive and/or collaborative organizations to share resources in order to perform one or more tasks. The resources that can be shared include documents, software, computers, data and sensors. The grid is defined by its pioneers [4] as follows:

> The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organization. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data and other resources, as is required by a range of collaborative problem solving resource brokering strategies emerging in industry, science and engineering.

The owner of a resource can choose the amount, duration, and schedule of the resources available to different users (see Figure 1). These policies can vary over time, impacting the available resources for a given application. A core requirement for success of these environments will be a middleware that schedules different resources to maximize the overall efficiency of the system.

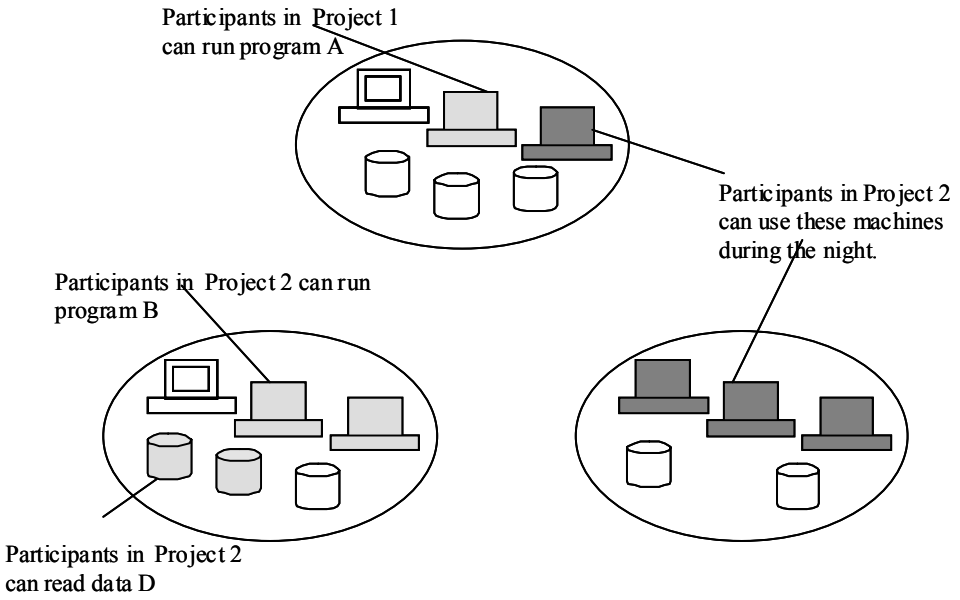


Figure 1-1. A grid with three subsystems. Each providing restricted access to a subset of applications.

Realizing the potential of grid computing requires the efficient utilization of resources. The execution of user applications must simultaneously satisfy both job execution constraints and system usage policies. Although many scheduling techniques for various computing systems exist [5-11], traditional scheduling systems are inappropriate for scheduling tasks onto grid resources for the following main reasons.

Although parallel or distributed systems address one or more of these characteristics, they do not address all of them in a cohesive manner for grids.

Virtual organization (VO) [12] is a group of consumers and producers united in their secure use of distributed high-end computational resources towards a common goal. Actual organizations, distributed nationwide or worldwide, participate in one or more VO's by sharing some or all of their resources. The grid resources in a VO are geographically distributed and heterogeneous in nature. These grid resources have decentralized ownership and different local scheduling policies dependent on their VO. The grid resources may participate in a VO in a non-dedicated way, which means the resources accept incoming requests from several different remote sources. The dynamic load and availability of the resources require mechanisms for discovering and characterizing their status continually.

The second major challenge in the grid-computing environment relates to the planning and scheduling of data analyses. The factors that guide the development of a plan include user requirements, global and local policy, and overall state. User requirements may include not only the virtual data request but also optimization criteria such as completion in the shortest time or usage of the fewest computing resources. Any plan is necessarily constrained by resource availability, and consequently, we must obtain all available state information. This complicates planning, as the global system state can be large and determining future system states can be difficult.

The complex interrelationships among different data representations (procedural vs. declarative), data locations (archived vs. cached), policies (local vs. global), and

computations (different user queries, background tasks, etc.) make planning and scheduling a challenging and rewarding problem.

New techniques are required for representing complex requests, for constructing request representations via the composition of representations for virtual data components, for representing and evaluating large numbers of alternative evaluation strategies, and for dealing with uncertainty in resource properties.

A virtual data grid must be able to allocate storage, computer, and network resources to requests in a fashion that satisfies global and local policies. Global policy includes community-wide policies governing how resources dedicated to a particular collaboration should be prioritized and allocated. Local policies are site-specific constraints governing when and how external users can use local resources and the conditions under which local use has priority over remote use. The execution of a plan will fail if it violates either global or local policy. Hence we require mechanisms for representing policies and new resource discovery techniques that can take into account policy information.

The purpose of planning and scheduling is to optimize the response to a query for virtual data given global and local policy constraints. Different optimization criteria may be applied to a PVDG request: minimize execution time, maximize reliability, minimize use of a particular resource, etc. For a given metric, optimization is driven by resource characteristics and availability.

The dynamic nature of the grid coupled with complex policy issues poses interesting challenges for harnessing the resources in an efficient manner. In our research, we study the key features of grid resource management systems and their performance on

Open Science Grid (OSG) [13], a worldwide consortium of university resources consisting of 2000+ CPUs.

Grid Resource Management Middleware: SPHINX

Efficient scheduling of requests to use grid resources must adapt to the dynamic grid computing environment while meeting administrative policies. Our research defines the necessary requirements of such a scheduler and proposes a framework called SPHINX. The scheduling middleware can administer grid policies, and schedule complex and data intensive scientific applications. The SPHINX design allows for a number of functional modules to flexibly plan and schedule workflows representing multiple applications on the grids. It also allows for performance evaluation of multiple algorithms for each functional module. We present early experimental results for SPHINX that effectively utilize other grid infrastructure such as workflow management systems and execution systems. These results demonstrate that SPHINX can effectively schedule work across a large number of distributed clusters that are owned by multiple units in a virtual organization. The results also show that SPHINX can overcome the highly dynamic nature of the grid and complex policy issues to utilize grid resources, which is an important requirement for executing large production jobs on the grid. These results show that SPHINX can effectively

- Reschedule jobs if one or more of the sites stops responding due to system downtime or slow response time.
- Improve total execution time of an application using information available from monitoring systems as well its own monitoring of job completion times.
- Manage policy constraints that limit the use of resources.

Virtual Data Toolkit (VDT) [13] supports execution of workflow graphs. SPHINX working with VDT is in the primary stages of exhibiting interactive remote data access,

demonstrating interactive workflow generation and collaborative data analysis using virtual data and data provenance. Also, any algorithms we develop will potentially be used by a wide user community of scientists and engineers. SPHINX is meant to be inherently customizable, serving as a modular "workbench" for CS researchers, a platform for easily exchanging planning modules and integrating diverse middleware technology. It will also deliver reliable and scalable software architecture for solving general-purpose distributed data intensive problems.

Efficient Scheduling Techniques

The dynamic and heterogeneous nature of the grid coupled with complex resource usage policy issues poses interesting challenges for harnessing the resources in an efficient manner. In our research, we present novel policy-based scheduling techniques and their performance on OSG. The execution and simulation results show that the proposed algorithm can effectively

1. Allocate grid resources to a set of applications under the constraints presented with resource usage policies.
2. Perform optimized scheduling on heterogeneous resources using an iterative approach and binary integer programming (BIP).
3. Improve the completion time of workflows in integration with job execution tracking modules of SPHINX scheduling middleware.

The proposed policy-based scheduling algorithm is different from the existing works from the following perspectives.

Policy constrained scheduling: The decentralized grid resource ownership restricts the resource usage of a workflow. The algorithm makes scheduling decisions based on resource usage constraints in a grid computing environment.

Optimized resource assignment: The proposed algorithm makes an optimal scheduling decision utilizing the Binary Integer Programming (BIP) model. The BIP

approach solves the scheduling problem to provide the best resource allocation to a set of workflows subject to constraints such as resource usage.

The scheduling on heterogeneous resources: The algorithm uses a novel mechanism to handle different computation times of a job on various resources. The algorithm iteratively modifies resource allocation decisions for better scheduling based on different computation times instead of taking a mean value over the time. This approach has also been applied to the Iterative list scheduling [1].

Dynamic scheduling: In order to handle the dynamically changing grid environments, the algorithm uses a dynamic scheduling scheme rather than a static scheduling approach. A scheduling module makes the resource allocation decision to a set of schedulable jobs. The status of a job is defined as schedulable when it satisfies the following two conditions.

- Precedence constraint: all the preceding jobs are finished, and the input data of the job is available locally or remotely.
- Scheduling priority constraint: A job is considered to have higher priority than others when the job is critical to complete the whole workflow for a better completion time.

Future scheduling: Resource allocation to a schedulable job impacts the workload on the selected resource. It also affects scheduling decisions of future schedulable jobs. The algorithm pre-schedules all the unready jobs to detect the impact of the current decision on the total workflow completion time.

When the scheduling algorithm is integrated with the SPHINX scheduling middleware, it performs efficient scheduling in the policy-constrained grid environment. The performance is demonstrated in the experimental section.

Table 1-1. The existing scheduling systems and the scheduling properties. This table shows the conventional scheduling systems and their scheduling property existence. The mark "v" means that a system has the property with the mark.

Systems	Adaptive scheduling	Co-allocation	Fault-tolerant	Policy-based	QoS support	Flexible interface
Nimrod-G		v			v	
Maui/Silver	v	v		v	v	
PBS		v			v	
EZ-grid		v		v		v
Prophet					v	
LSF		v	v	v	v	

Scheduling Systems

In this section we present the currently existing scheduling systems and their properties. Table 1-1 shows a set of scheduling systems and the given property existence. In the table, the mark "v" indicates that a system has the given property with the mark. In the table we specify a set of system properties. With adaptive scheduling we mean that the resource allocation decision is not finalized until the real job submission happens. The scheduling decision will change based on resource status and availability after the initial decision is made. Co-allocation means that a request may be allocated with several different resources. A real application requires different kind of resources such as CPU and storage. Co-allocation supporting scheduler allocates the required resources to the job. Fault-tolerant scheduling means that a job is rescheduled after its execution failure on a remote resource. In a dynamic grid environment execution failure is more likely to happen fairly often. The scheduling system is required to monitor job execution and reschedule it. Policy-based scheduling supports the heterogeneous resource ownership in grid computing. This topic is discussed in detail in the following section. Quality of service (QOS) is presented with deadline and other application requirements. A

scheduling system should make resource allocation decisions concerning the requirement. An ideal system provides a flexible interface to other modules such as monitoring and scheduling to allow the replacement of existing modules in the system with other customized modules.

The Portable Batch System (PBS)

Portable Batch System is a batch job and computer system resource management package designed to support queuing and execution of batch jobs on heterogeneous clusters of resources. PBS offers several scheduling systems to support various resource allocation methods; such as Round Robin, First In First Out (FIFO), Load Balancing, Priority-based and Dedicated Times [15]. The PBS configuration consists of several modules, the PBS client, server, scheduler and job execution clusters which run the PBS MOM daemon. In the PBS system a job is submitted along with a resource specification on one of the front-ends, handed to the server, scheduled, run by the MOMs in the execution clusters, and has output placed back on the front end [16]. PBS works quite well for handling batch processing.

However, as mentioned in the previous section, grid computing requires much more delicate resource management and refined request scheduling in a dynamically changing heterogeneous environment. The proposed resource allocation strategy achieves solutions to the issues by importing the concept of policy- and reservation-based scheduling for Quality of Service (QOS). The proposed scheduler also supports fully interactive request submissions for negotiating the level of QOS requirements according to the current and estimated near future grid weather after the user makes a submission.

Maui

Maui is an advanced job scheduler for use on clusters and supercomputers. It is an optimized and configurable tool capable of supporting a large array of scheduling policies, dynamic priorities and extensive reservations. The Maui scheduler can act as a "policy engine," which allows site administrators control over when and how resources are allocated to jobs [17]. The policies serve to control how and when jobs start. They include job prioritization, fairness policies and scheduling policies. The Quality of Service (QOS) feature allows a site to grant special privileges to particular users by providing additional resources, access to special capabilities and improved job prioritization. Maui also provides an advanced reservation infrastructure allowing sites to control exactly when, how and by whom resources are used. Every reservation consists of three major components, a set of resources, a timeframe and an access control list. The scheduler makes certain that the access control list is not violated during the reservation's timeframe on the resources listed [18,19]. Even though Maui is a highly optimized and configurable scheduler capable of supporting scheduling policies, extensive reservations and dynamic priorities, it has limitations in scheduling distributed workloads to be executed across independent resources in a grid.

A grid scheduling system must support global optimization in addition to a local best scheduling. The proposed scheduling framework supports the global optimization as well as local best fit by considering resource usage reservations and QOS requirements in the scheduling. The hierarchical architecture view of a grid in policy enforcement makes it possible to implement extensive and scalable resource allocation in the proposed scheduler.

LSF

LSF is a suite of application resource management products that schedule, monitor, and analyze the workload for a network of computers. LSF supports sequential and parallel applications running as interactive and batch jobs. The LSF package includes LSF Batch, LSF JobScheduler, LSF MultiCluster, LSF Make and LSF Analyzer all running on top of the LSF Base system. LSF is a loosely coupled cluster solution for heterogeneous systems. There are several scheduling strategies available in LSF. They include Job Priority Based Scheduling, Deadline Constraints Scheduling, Exclusive Scheduling, Preemptive Scheduling and Fairshare Scheduling. Multiple LSF scheduling policies can co-exist in the same system [20].

Even though LSF supports several different scheduling strategies, most of them do not provide enough ways for users to specify requirement and preference in resource allocation. The proposed scheduling strategy supports user interaction in resource allocation decisions by allowing QOS specification.

EZ-Grid

EZ-Grid is used to promote efficient job execution and controlled resource sharing across sites. It provides the policy framework to help resource providers and administrators enforce fine-grained usage policies based on authorization for the uses of their resources [21]. The framework automates policy-based authorization or access control and accounting for job execution in computing grids.

A major difference between the policy engine and our proposed framework is that our framework utilizes a hierarchically defined policy along three dimensions consisting of resource providers, request submitters and time, and uses submitters' Quality of Service requirements for resource allocation.

Resource Broker

The Resource Broker (RB) from the European Data Grid project provides a matchmaking service for individual jobs: given a job description file, it finds the resources that best match the users' request [22, 23]. The RB makes the scheduling decision based only on the information of individual user authentication and individual job execution requirements

Current plans suggest supporting different scheduling strategies. Our work goes beyond this by specifically accounting for VO policy constraints and VO-wide optimization of throughput via constraint matchmaking and intelligent scheduling algorithms. In addition, the proposed scheduler is designed to provide estimates of execution time so that the user may determine if a request fits within the user's deadlines. Finally, by considering the DAG as a whole, the middleware will be able to intelligently pre-determine any necessary data staging.

Pegasus

Pegasus [24] is a configurable system that can map and execute DAGs on a grid. Currently, Pegasus has two configurations. The first is integrated with the Chimera Virtual Data System. The Pegasus system receives an abstract DAG file from Chimera. Pegasus uses these dependencies to develop a concrete DAG by making use of two catalogs, the replica catalog that provides a list of existing data components, and a transformation catalog that stores a list of available executable components. With information from these catalogs, the Pegasus system maps the input abstract job descriptions onto grid resources. Then it adds additional jobs to provide the necessary data movement between dependent jobs. This final concrete plan is submitted to the grid execution system, DAGMan, which manages its execution.

In its second configuration, the Pegasus system performs both the abstract and concrete planning simultaneously and independently of Chimera. This use of Pegasus takes a metadata description of the user's required output products. It then uses AI planning techniques to choose a series of data movement and job execution stages that aims to optimally produce the desired output. The result of the AI planning process is a concrete plan (similar to the concrete plan in the first configuration) that is submitted to DAGMan for execution.

The framework presented in this document is distinct from the Pegasus work in many ways. For example, instead of optimizing plans benefiting individual users, the proposed framework, SPHINX allows for globally optimized plans benefiting the VO as a whole. In addition, Pegasus currently provides advanced forward planning of static workflows. The work presented in this document is designed to dynamically plan workflows by modifying groups of jobs within a DAG (sub-DAGs) and, depending on the nature of the grid, controlling the release of those sub-DAGs to execution systems such as Condor-G/DAGMan.

SPHINX is meant to be inherently customizable, serving as a modular "workbench" for CS researchers, a platform for easily exchanging planning modules and integrating diverse middleware technology. As a result, by including Pegasus planning modules in the SPHINX server, the resulting scheduling system would be enhanced by taking full advantage of knowledge management and AI planning, provided by Pegasus, while providing the flexible dynamic workflow and just-in-time job planning provided by SPHINX.

Condor

The proposed scheduling system, SPHINX utilizes the stable execution control and maintenance provided by the Condor system [25, 26]. The Condor Team continues to develop Condor-G and DAGMan. Recently, to improve its just-in-time planning ability, DAGMan has been extended to provide a call-out to a customizable, external procedure just before job execution. This call-out functionality allows a remote procedure to modify the job description file and alter where and how the job will be executed. SPHINX envisages using the call-out feature in DAGMan for just-in-time error recovery and corrective just-in-time planning. However, as DAGMan and SPHINX increase in functionality, DAGMan itself could become a scheduling client and communicate through this and other callouts to the scheduling server directly.

PROPHET

Prophet is a system that automatically schedules data parallel Single Process Multiple Data (SPMD) programs in workstation networks [27]. In particular, Prophet uses application and resource information to select the appropriate type and number of workstations, divide the application into component tasks, distribute data across workstations, and assign tasks to workstations. To this end, Prophet automates the scheduling process for SPMD applications to obtain reduced completion time. In addition, Prophet uses network resource information and application information to guide the scheduling process. Finally, Prophet is unique in that it addresses the problems of workstation selection, partitioning and placement together. The SPHINX system provides functionality for scheduling jobs from multiple users concurrently based on the policy and priorities of these jobs in a dynamically changing resource environment.

Data Analysis Systems

There are many other international Grid projects underway in other scientific communities. These can be categorized as integrated Grid systems, core and user-level middleware, and application-driven efforts. Some of these are customized for the special requirements of the HEP community. Others do not accommodate the data intensive nature of the HEP Grids and focus upon the computational aspect of Grid computing.

EGEE [28] middleware, called gLite [29], is a service-oriented architecture. The gLite Grid services aim to facilitate interoperability among Grid services and frameworks like JClarens and allow compliance with standards, such as OGSA [30], which are also based on the SOA principles.

Globus [31] provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. Globus provides basic services and capabilities that are required to construct a computational Grid. Globus is constructed as a layered architecture upon which the higher-level JClarens Grid services can be built.

Legion [32] is an object-based "meta-system" that provides a software infrastructure so that a system of heterogeneous, geographically distributed, high-performance machines can interact seamlessly. Several of the aims and goals of both projects are similar but compared to JClarens the set of methods of an object in Legion are described using Interface Definition Language.

The Gridbus [32] toolkit project is engaged in the design and development of cluster and Grid middleware technologies for service-oriented computing. It uses Globus libraries and is aimed at data intensive sciences and these features make Gridbus conceptually equivalent to JClarens.

UNICORE [33] provides a uniform interface for job preparation, and seamless and secure access to computing resources. Distributed applications within UNICORE are defined as multipart applications where the different parts may run on different computer systems asynchronously like the GAE services, or they can be sequentially synchronized.

NASA's IPG [34] is a network of high performance computers, data storage devices, scientific instruments, and advanced user interfaces. Due to its Data centric nature and OGSA compliance, IPG services can potentially interoperate with GAE services.

WebFlow [35], a framework for wide-area distributed computing, is based on a mesh of Java-enhanced Apache web servers, running servlets that manage and coordinate distributed computation and it is architecturally closer to JClarens .

The NetSolve [36] system is based around loosely coupled, distributed systems, connected via a LAN or WAN. Netsolve clients can be written in multiple languages as in JClarens and server can use any scientific package to provide its computational software.

The Gateway system offers a programming paradigm implemented over a virtual web of accessible resources [37]. Although it provides portal behavior like JClarens and is based on SOA, its design is not intended to support data intensive applications.

The GridLab [38] will produce a set of Grid services and toolkits providing capabilities such as dynamic resource brokering, monitoring, data management, security, information, adaptive services and more. GAE Services can access and interoperate with GridLab services due to its SOA based nature.

The Internet computing projects, such as SETI@Home [39] and Distributed.Net [40], which build Grids by linking many end-user PCs across the internet, are primarily number crunching projects that lack the large data management features of HEP Grids.

The Open Grid Services Architecture (OGSA) framework, the Globus-IBM vision for the convergence of web services and Grid computing, has been taken over by Web Services Resource Framework (WSRF) [41]. WSRF is inspired by the work of the Global Grid Forum's Open Grid Services Infrastructure (OGSI) [42]. The developers of the Clarens Web Services Framework are closely following these developments.

Scheduling Algorithms

This section discusses several existing scheduling algorithms. Although the referenced algorithms work well in the traditional high performance computing environment, they do not perform in a satisfactory manner with the characteristics of grids discussed in the previous section.

Iterative List Scheduling

This work [54] introduces an iterative list-scheduling algorithm to deal with scheduling on heterogeneous computing systems. The main idea in this iterative scheduling algorithm is to improve the quality of the schedule in an iterative manner using results from previous iterations. Although the algorithm can potentially produce shorter schedule length it does not support resource usage policies. It is a static scheduling algorithm, which assumes an unchanged or stable computing environment. In the dynamic and policy constrained grid computing environment the algorithm may not perform as the simulated results show in the dissertation.

Dynamic Critical Path Scheduling

Authors [55] propose a static scheduling algorithm for allocating task graphs to fully connected multiprocessors. It minimizes make-span subject to precedence constraint, which is determined by the critical path of the task graph. The homogeneous CPU-based scheduling algorithm assumes that the scheduler could manage the scheduling priority of jobs in a processor. This may not be true in a grid environment in which the resources have decentralized ownership and different local scheduling policies dependent on their VO.

Reliability Cost Driven Scheduling

The work [56] describes a two-phase scheme to determine a scheduling of tasks with precedence constraints that employs a reliability measure as one of the objectives in a real-time and heterogeneous distributed system. The static algorithm schedules real-time tasks for maximized reliability. The utility function of the algorithm finds a processor with the earliest start time for jobs in an application. In the presence of the policy constraint the algorithm may not be able to find the proper resource allocation to the application.

Heterogeneous Earliest Finish Time Scheduling

The algorithm [57] focuses on the appropriate selection of the weight for the nodes and edges of a directed acyclic graph, and experiments with a number of different schemes for computing these weights. The proposed technique uses the mean value approach to find the length of the produced schedule. Instead of the mean value approach our proposed algorithm uses the iterative approach to heterogeneous resources. The experimental results compare the two schemes. The off-line and priority-based scheduling may not be feasible in the grid-computing environment.

Dynamic Level Scheduling

This scheduling algorithm [58] match a job and a processor in an exhaustive way. The job is on the critical path of a directed acyclic graph (DAG), and the job starts on the processor at the earliest time. The algorithm uses the mean value approach on a heterogeneous CPU resource environment. The static and mean value-based scheduling may not produce a good scheduling result in policy-based grid computing.

Optimal Assignment with Sequential Search

The authors [59] describe two algorithms based on the A* technique. The first is a sequential algorithm that reduces the search space. The second proposes to lower time complexity, by running the assignment algorithm in parallel, and achieves significant speedup. The exhaustive and sequential search for the optimal assignment may not be feasible for a large tree search space even though their modified algorithm generates random solutions and prunes the tree. Our proposed algorithm performs optimal assignment in a different scheme. We utilize a sub-tree and iterative concepts instead of considering the whole tree and all heterogeneous resources.

Contributions

In this section we describe the main contribution of research to the scheduling system and techniques in Grid computing.

- Policy-driven request planning and scheduling of computational resources: We define and implement the mechanisms for representing and enforcing both local and global policy constraints. A grid scheduler properly needs to be able to allocate the resources to requests in a fashion that satisfies global and local constraints. Global constraints include community-wide policies governing how resources dedicated to a particular collaboration should be prioritized and allocated; local constraints include site-specific policies governing when external users can use local resources. We develop mechanisms for representing and enforcing constraints. We also develop policy-aware scheduling middleware and algorithms. We develop a novel framework for policy based scheduling in resource allocation of grid computing. The framework has several features. First, the scheduling

strategy can control the request assignment to grid resources by adjusting resource usage accounts or request priorities. Second, efficient resource usage management is achieved by assigning usage quotas to intended users. Third, the scheduling method supports reservation based grid resource allocation. Fourth, Quality of Service (QOS) feature allows special privileges to various classes of requests, users, groups, etc.

- A fault-tolerant scheduling system in a dynamic Grid environment: Two major characteristics of Grid computing we can point out are de-centralized resource ownership and dynamic availability of resource. Shared by multiple Virtual Organizations, a Grid is meant to be an environment to mutually share resources among organizations. The composition of the grid is not homogeneous, to say the least. This makes it difficult to guarantee expected performance in any given execution environment. Another important factor is that due to the dynamic availability of resources, the presence of 'unplanned downtimes' of certain resources in the Grid makes scheduling decision non-trivial as a job planned on a site may never be completes. These reasons make it very cumbersome for an application user to effectively use a grid. An application user usually throttles the jobs across the grid. The decision of how many jobs to send to a site is usually based on some static information like the number of CPUs available on the sites. However, the site with more CPUs might already be overloaded or this particular production manager (or his VO proxy, to be precise) might have a relegated priority at that remote site. As a result jobs might get delayed or even fail to execute. In such events, the application user has to re-submit the failed jobs. An experienced user may rely on his/her past experience and submit jobs to sites which have been more reliable in the past. However, the site which was working well before may have its own priority work to be done this time, thus temporarily relegating this user's priority. The proposed scheduling middleware, named SPHINX, will effectively overcome the highly dynamic nature of the grid to harness grid resources. The system is equipped with advanced job execution tracking module and incorporated with other monitoring systems to maintain the information of data and resource availability.

- Scheduling workflow model: An application scientist typically solves his problem as a series of transformations. Each transformation may require one or more inputs and may generate one or more outputs. The inputs and outputs are predominantly files. The sequence of transformations required to solve a problem can be effectively modeled as a Directed Acyclic Graph (DAG) for many practical applications of interest that the proposal is targeting. Most existing scheduling systems assume that a user application consists of a single job. It means that there is no precedent relationship on a workflow. In the scheduling system and algorithm development we use different types of workflows and resources. The research focuses on single task workflows initially. They are the simplest workflows that are interactive or batch type. Then, we extend the research to simple DAG-based workflows, which will reflect a variety of real world scenarios. Finally, our scheduling algorithms deal with multiple DAGs in achieving objective functions simultaneously. The experiments of the algorithms are performed in

heterogeneous resource environment. A resource type such as a CPU has a set of resources with different performance.

- In order to test and evaluate the proposed data analysis prototype we deploy the framework across a grid testbed named Grid2003/Open Science Grid which consists of more than 25 sites providing more than 2000 CPUs, and exhibit remote data access, workflow generation and collaborative data analysis using virtual data and data provenance, as well as showing non-trivial examples of policy based scheduling of requests in a resource constrained grid environment.

Outline

The dissertation is organized in the following manner. We discuss policy-based scheduling for single applications in Chapter 2. The chapter presents a policy-based novel-scheduling framework for obtaining a sub optimal scheduling solution on Grid resources. Specially, we also discuss resource allocation in the multi-dimensional policy space. We discuss the workflow centric scheduling middleware, SPHINX, with simulation and experiment results in Chapter 3. In that chapter we introduce the core features of SPHINX architecture, and discuss the distributed data analysis utilizing SPHINX and fault tolerant scheduling. It also discusses incorporating the scheduling system with other services such as JClarens and MonALISA. In Chapter 4 we present policy-based scheduling algorithms and supporting infrastructures for scheduling single and multiple workflows. We discuss scheduling techniques with awareness of given resource usage constraints and completion deadline. We present simulations and experiment results on Open Science Grid (OSG). We conclude the dissertation in Chapter 5 with future research plans.

CHAPTER 2
GRID POLICY FRAMEWORK

In this chapter we discuss a novel framework for policy based scheduling in resource allocation of grid computing. The framework has several features. First, the scheduling strategy can control the request assignment to grid resources by adjusting resource usage accounts or request priorities. Second, Efficient resource usage management is achieved by assigning usage quotas to intended users. Third, the scheduling method supports reservation based grid resource allocation. Fourth, Quality of Service (QOS) feature allows special privileges to various classes of requests, users, groups, etc. This framework is incorporated as part of the SPHINX scheduling system that is discussed in the next chapter. A set of experimental results is provided to demonstrate the usefulness of the framework.

Petascale Virtual Data Grids (PVDG's) will need to be able to satisfy all of the job requests by allocating storage, computer and network resources in a fashion that satisfies both global and local constraints. Global constraints include community-wide policies governing how resources should be prioritized and allocated. Local constraints include site-specific control as to when external users are allowed use local resources. As such, PVDG computing requires mechanisms for representing and enforcing policy-based scheduling techniques.

Policies, including authentication, authorization, and application constraints are important factors for maintaining resource ownership and security. The set of possible constraints on job execution can be various, and change significantly over time. These

constraints may include different values for each job, for example RAM requirements or connectivity needed, or constraints that are static for a specific job type, such as the operating system or architecture. Policies may include any information that should be specified to ensure that a job is matched to appropriate resources [60, 61].

The proposed Policy based scheduling framework can achieve local and global optimisation in the resource allocation by providing the following several features:

1. Control the request assignment to grid resources by adjusting resource usage accounts or request priorities. This feature allows the scheduler balance workloads across resources in a grid resulting in better overall utilization and turn around time.

2. Support reservation based grid resource allocation. The scheduling concept makes it possible for the scheduler to assign multiple dependent requests in an optimally synchronized manner. To support the feature the resources that are participating in the scheduling should allow request to be executed with a reserved amount of resource usage in a specific time. In addition to the resource side reservation the request must be completed within the reservation in terms of usage duration and amount.

3. Allow variable Quality of Service (QOS) privileges to various classes of requests, users, groups, etc. In addition to a QOS in a basic level a request submitter can make a specific QOS request according to the privileges assigned to the class to which the submitter or the request belongs. The request submitter passes a request to the scheduler with the QOS specification. A resource allocation considering the QOS feature should be interactive between the submitter and the scheduler. They communicate each other to adjust the QOS requirements.

Managing resource usage account prevents resource cycle waste and resource over usage. Resource providers assign usage quotas to intended resource users. The scheduler monitors resource usage by keeping track of the quota change. It saves resource cycles by assigning more requests to the idle resources, while reducing the number of requests assigned to the over used resources.

Key Features

The proposed policy based scheduling framework can achieve local and global optimization in the resource allocation by providing the following several features:

Control the request assignment to grid resources by adjusting resource usage accounts or request priorities. This feature allows the scheduler balance workloads across resources in a grid resulting in better overall utilization and turn around time. Heterogeneous grid resources have different capabilities for executing submitted requests. Statically adjusting resource usage quotas of users can support workload balance across the resources. Also a scheduler considers dynamically changing workload on each machine when the scheduler makes resource allocation decision. The updated workload information is available from a resource-monitoring module in real time. The proposed framework incorporates the quota and monitoring information into resource allocation decision. The information is formulated in LP functions. We will discuss the feature in detail in the next section.

Support reservation based grid resource allocation. The suggested framework makes it possible for a scheduler to assign multiple dependent requests to resources in an optimally synchronized manner. This feature is made possible due to the reservation driven scheduling. To support the feature the resources that are available in the scheduling should guarantee requests to be executed with a reserved amount of resource usage in a specific time period. The feature also supports global Quality of Service (QOS) satisfaction with a workload consisting of several dependent tasks. In order to make advanced reservation a scheduler needs to estimate request execution time and resource usage amount. A user can provide the information when she/he submits a request. In another way a scheduler can make the estimation based on functions of a

prediction module such as benchmarking or historical execution records. In the proposed framework optimisation functions make resource allocation considering the advanced reservation with policy and anticipated execution information.

Allow variable Quality of Service (QOS) privileges to various classes of requests, users, groups, etc. A user submits a request with its QOS requirement or preference. The QOS information may contain resource reservation specific requirement such as amount of resource (CPU, storage, etc) and a period of time. A user can also require a specific deadline of request execution or the best effort execution. The QOS privileges are different among the user classes. A class of users may have privileges for requiring more QOS then other groups do. A scheduler considering the QOS feature should interact with a request submitter for adjusting QOS achievement level. In the proposed system we assume that users specify the amount of required resource usages. The framework makes scheduling decision using optimisation functions for satisfying given QOS's and policy constraints.

Support interactive scheduling with request submitters. A scheduler interacts with the submitters in order to negotiate QOS achievement level. In the cases where the scheduler can't achieve the service level that a submitter specifies, or the scheduler can suggest different QOS specification for better performance of request execution the scheduler sends the submitter the alternatives to the original QOS requirement. After receiving the alternatives the submitter considers them, and sends back her/his decision to the scheduler. The decision may include accepting or denying the alternatives, or requesting other scheduling effort with different QOS requirement.

Managing resource usage account prevents resource cycle waste and resource over usage. In addition to the workload balance resource management system can control resource usage by adjusting usage quotas to intended resource users. The scheduler monitors resource usage by keeping track of the quota change. It saves resource cycles by assigning more requests to the idle resources, while reducing the number of requests assigned to the over used resources.

Policy Space

Grid computing requires collaborative resource sharing within a Virtual Organization (VO) and between different VOs. Resource providers and request submitters who participate within a VO share resources by defining *how* resource usage takes place in terms of *where what*, *who*, and *when* it is allowed. Accordingly, we assume that policies may be represented in a three (plus one) dimensional space consisting of *resource provider* (and *property*), *request submitter*, and *time*. We further assume that quota limits are descriptive enough to express how a resource may be used.

By exploiting the relational character of policy attributes, a policy description space may be conveniently represented as a hierarchical tree. Indeed, the heterogeneity of the underlying systems, the difficulty of obtaining and maintaining global state information, and the complexity of the overall task all suggest a hierarchical approach to resource allocation. Further, such a hierarchical approach allows for a dynamic and flexible environment in which to administer policies.

Three of the dimensions in the policy space, consisting of resource provider, request submitter and time, are modelled as hierarchical categorical policy attributes expressed in terms of quotas. An extra dimension, resource property, is modelled as a simple categorical attribute to the hierarchical resource provider dimension.

Administrators, resource providers or requesters who participate in a VO then define resource usage policies (in terms of quotas) involving various levels of this hierarchical space. In the following sections we discuss the hierarchical representation of each dimension.

Resource Provider

A resource provider is defined as an entity which shares some particular physical resource within the context of a grid. Physical resources participating in a grid are often naturally organised into hierarchical groups consisting of both physical and logical views. In the example shown in Figure 2-1, a hypothetical grid consists of many domains each containing one or many clusters of machines. The representation in Sphinx is generalised such that arbitrary levels in the resource provider hierarchy may be added to maintain scalability. For example, the resource provider domain level might represent a set of gatekeepers at particular sites, or it might represent a sub-grid, itself containing local domains. It is common to assume that each compute-cluster has a local resource scheduling system (such as Condor, PBS, or LSF to name a few) to serve remote requests, but there may be compute-clusters which do expose individual machines to the grid for direct access. The hierarchical dimension (categorical tree) representing resource providers is given the symbol $\boldsymbol{R}_{H,\ provider}$.

Sphinx performs global scheduling to allocate remote resources for requests across an entire grid. As such, Sphinx specifies resource allocation to the resource provider level in which either a local resource scheduler exists or a sub-grid scheduler exists.

Resource Property

Each resource provider is augmented by a list of non-hierarchical properties, such as CPU, memory, storage space, bandwidth, etc. At any level in the resource provider

hierarchy, quota limits for each resource property are appropriately aggregated. The categorical list representing resource properties is given the symbol $\boldsymbol{R_{property}}$.

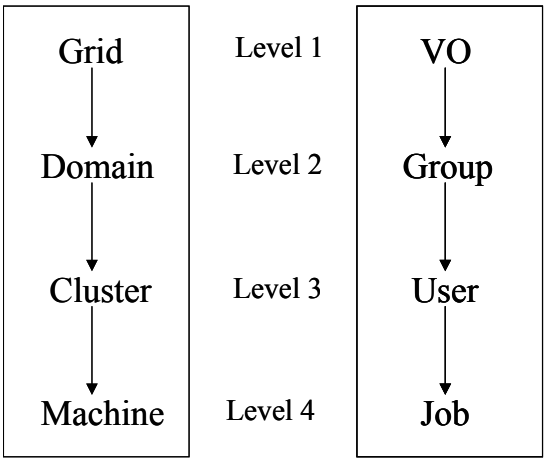


Figure 2-1. Examples of resource provider and request submitter hierarchies request submitter

A request submitter is defined as an entity, which consumes resources within the context of a grid. As in the case of resource providers, request submitters are naturally structured in a hierarchical manner. In the example shown in Figure 2-1, a typical VO might be described by several groups of users, each with possible proxies (e.g. jobs) acting on their behalf. Again, the representation in Sphinx is generalised so that arbitrary levels in the request submitter hierarchy may be added to maintain scalability or convenient logical views of a VO. For instance, a group may consist of individual grid users (each with a particular certificate) or other sub-groups, each containing sub-sub-groups or users. In analogy with the resource provider hierarchy, the deepest levels in the request submitter hierarchy represent entities possessing a grid certificate or proxy. In general, a request for grid resources may be submitted (by a sanctioned user with a certificate) from any level. This enables the pooling of account quotas to service high priority requests emanating from any particular level in the request submitter hierarchy

(such as a group or even the VO itself). The hierarchal dimension (categorical tree) representing request submitter is given the symbol $\boldsymbol{S_H}$.

Time

Time is used to allow for possible dynamism in policies in which, for example, maximum quota limits may change in a well- and pre-defined way (e.g. periodic) for request submitters. In addition, in order to provide a quality of service, the scheduler will need to plan and test, at various time frames in the future, possible resource allocation and de-allocation strategies. From a scheduling point of view, recent times are more important than times in the far future. Hence, Sphinx models the hierarchical nature of time by forming time frames using an adaptive logarithmic mesh, allowing for more finely or coarsely grained description of polices, depending on present or future scheduling requirements. The categorical list representing time is given the symbol **T**.

Three-dimensional Policy Space

As the policy space of quota limits defined above is comprised of categorical trees or lists, we construct a hierarchical policy tensor by taking the triple cross product of each hierarchical dimension:

$$\boldsymbol{Q_H = R_H \times S_H \times T}$$

where $\boldsymbol{R_H = R_{H,\,provider} \times R_{property}}$ represents resource providers (with property attributes), $\boldsymbol{S_H}$ represents request submitters, and $\boldsymbol{T}$ time. Each element of $\boldsymbol{Q_H}$ represents an allotted quota limit. There are several ways to extract meaningful policy views of such a space. For example, $\boldsymbol{Q_H}$ represents a "volume" of all possible quotas from all hierarchical levels of providers applied to all hierarchical levels of submitters and using all time scales. However, in the context of scheduling, a description of policies corresponding to particular aggregate-levels of providers, submitters, and times is often

more germane than using all possible hierarchical views. One simple example is that of a grid site (resource-provider) which routes all incoming jobs to a compute-cluster via a local scheduler. In such a situation, enforcing fine-grained policies for each machine is not possible using a grid-wide scheduler (such as Sphinx), but enforcing site-aggregate polices is possible.

Hence, one is more often interested in a particular view which collapses parts of the hierarchical policy space "volume" onto a sub-space, or aggregate "surface," containing high-level or fine-grained policy descriptions, depending on the need. In this spirit, we first determine the tree level of interest for each branch (thereby defining leaf-nodes of differing granularity) of each dimension and then form the resulting triple cross product of leaf-nodes:

$$\boldsymbol{Q_L = R_L \times S_L \times T}$$

Such a construction of QL , which is not unique, essentially allows for a flexible and efficient description of global and local polices. Various policy definitions are possible from the combination of leaf nodes in different levels of Resource-, Entity- and Time- policy hierarchy trees. Possible combinations from the trees make hierarchical policies in the three dimensions; Resource, Entity and Time. From the Figure 2-2 hierarchy trees, the combination (u2, r2, t3) means that there is a policy for User_1 on Cluster_1 in the time unit of Week. For example, User_1 can use one week of CPU time of machines on Cluster_1 during the month of July. (u5, r1, t2) means that Group_2 can use resources of Domain_1 for three months in a year. (u1, r5, t1) and (u1, r1, t2) defines policies for VO_1 on Domain_1 and 2 of Grid_1 in the time units of year and month respectively.

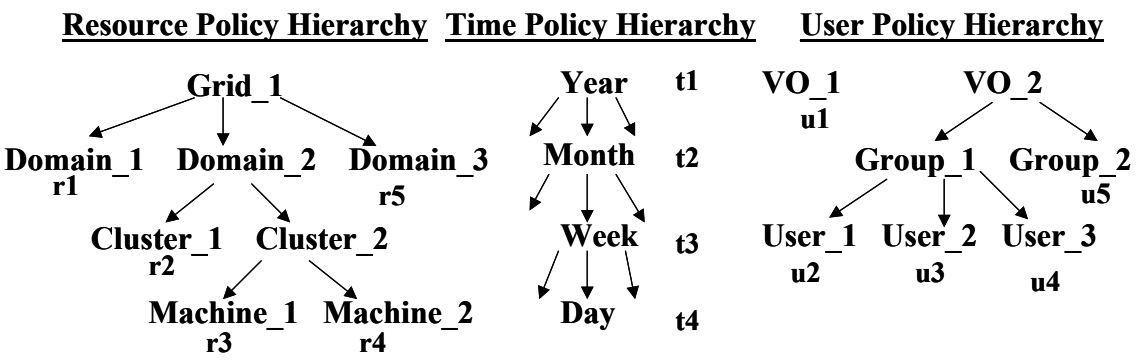


Figure 2-2. Hierarchical policy definition example

A Solution Strategy for Policy-based Scheduling

Given that a leaf-node tensor of policies has been defined as described in the previous section, we derive and apply an accounting tensor $\boldsymbol{A}$ which constrains a search for an optimal solution $\boldsymbol{x}$ for allocating resources in support of the requirements from a particular request $\boldsymbol{b}$ using the well known method of Linear Programming (LP). Specifically, we seek to minimize an objective function f (representing some heuristic knowledge of the grid):

$$\min[\, f(\boldsymbol{x}) \,]$$

subject to

$$\langle \boldsymbol{A}(t) , \boldsymbol{x} \rangle \quad \geq \quad \boldsymbol{b}$$

where $\langle \boldsymbol{A}(t) , \boldsymbol{x} \rangle$ denotes the inner product of $\boldsymbol{A}(t)$ with $\boldsymbol{x}$.

Model Parameters

Definition: Let $\boldsymbol{Q}$ be a tensor representing a leaf-node view of allotted quota limits (including pre-defined changes over time), $\boldsymbol{U}(t)$ be a tensor representing the amount of quota actually used at current time t. We then define $\boldsymbol{A}(t)$ to be a tensor representing the amount of quota still available at current time t by $\boldsymbol{A}(t) = \boldsymbol{Q} - \boldsymbol{U}(t)$ and subject to the

constraint that the remaining available quota always lie within the maximum allotted quota:

$\forall ijkl\ (\ 0 < A_{ijkl}(t) < Q_{ijkl}\ )$

where i indexes all resource properties, j indexes all (leaf-node) resource providers, k indexes all (leaf-node) request submitters and, l (logarithmically) indexes time. Note that t and T refer to different uses of time. The functional dependence of U(t) and A(t) on current time t explicitly recognizes the fact that U(t) and A(t) are updated in real-time according to actual resource usage monitoring information. This is distinguished from the T dependence of Q, for example, which is not updated in real-time but which does define quota limits and their pre-defined (e.g. possibly periodic) future variation by indexing T relative to the present. In particular, Q_{ijkl} , represents present quota limits for l = 0 and future quota limits for l > 0.

Definition: Let $\boldsymbol{W(t)}$ be a tensor representing the normalised (current and forecasted) grid weather such that $W(t)_{ijk}$ = *(amount of resource in use)/(resource size)* where *i* indexes all resource properties, *j* indexes all (leaf-node) resource providers, and *k* (logarithmically) indexes future steps in time from the present ($k = 0$). The estimated impact on grid weather from future resource allocation may be recursively obtained from current grid weather, $W(t)_{ij0}$, by the relation $W(t)_{ij(k+1)} = W(t)_{ijk} + \delta W(t)_{ijk}$, where $\delta W(t)_{ijk}$ represents any future allocations (or de-allocations) of resources which are to be scheduled on property *i* of resource *j* during the k^{th} interval of time away from the present.

Definition: Let $\boldsymbol{b}$ be a tensor representing the required amount of resource requested where, b_{ij} is indexed by i for all resource properties and j for all (leaf-node) request submitters.

Definition: The *Atomic Job Criterion (AJC),* for some particular resource provider J, request submitter K, and time frame L, is defined to be

$$(\exists i \;\; A_{iJKL}(t) \;<\; b_{iK}) \Rightarrow (\forall i \;\; x_{iJKL} \equiv 0)$$

and states that if there exists a resource property i which can not satisfy the requested resource amount, then the corresponding resource provider J is removed from the space of feasible resource providers during the particular time frame L. This condition reduces the space of resource providers to include only those sites that are capable of accommodating the full request (i.e. the ability to satisfy all requirements declared in $\boldsymbol{b}$). For jobs that are not atomic, for example split-able jobs suited to data parallelism, one would not impose such a stringent constraint as the AJC. In this chapter, we will only consider jobs that cannot be split into smaller sub-jobs. .

We wish to find a solution tensor $\boldsymbol{x} = [x_{ijkl}]$ which provides the optimal location to service the request within the reduced space of feasible providers where, i indexes resource properties; j indexes (leaf-node) resource providers; k indexes (leaf-node) request submitters; and l (logarithmically) indexes time.

Choice of an Objective Function and Optimization Metric

Several sophisticated objective functions and heuristic algorithms will be explored in the future. The simple choice of objective functions here is one in which the preferred

location to service the request is that which least impacts both grid weather and the request submitter's account usage [1]:

$$\min[\, f_{KL}(\boldsymbol{x}) \,] = \min[\, \Sigma_{ijkl}\, W_{jil}\, x_{ijkl}\, \delta_{lL}\, \delta_{kK} \,]$$

subject to: $\Sigma_{jl}\, A_{iljk}\, x_{ijlk} \geq b_{ik}$

where δ_{mn} is the Kronecker delta function (which is unity if $m = n$ and zero otherwise), K corresponds to the particular request submitter, and L is some particular (logarithmic) time view corresponding to possible variation in quota limits. Such a simple strategy only provides "greedy" scheduling decisions for request submitter K within a certain time frame L, but does attempt to improve the flexibility of future scheduling choices by disfavouring resource providers in which the remaining quota for submitter K would be largely consumed.

It may be shown that, for the case in which policies are constant for all times l, the above simple objective function $f_{KL}(\boldsymbol{x})$ is minimised when

$x_{ijKL} = 0$ (non-optimal provider of a resource property)

$= b_{iK}/A_{ijKL}$ (unique optimal provider of a resource property)

$= (b_{iK}/A_{ijKL})/N$ (N-identical optimal providers of a resource property)

Hence, non-zero entries of x_{ijkl} are interpreted as representing the fraction of quota to be used from the remaining available quota for resource property i at provider j for request submitter k during time interval l ; $\boldsymbol{x}$ itself represents the solution which minimally impacts grid weather and which minimally uses the remaining available quota for request submitter K. While using the objective function f_{KL}, as defined above, one is

[1] Please note that in the rest of this chapter, we have suppressed the functional time dependent notation of $\boldsymbol{U}(t)$, $\boldsymbol{A}(t)$, $\boldsymbol{W}(t)$ for the sake of clarity. It is understood, however, that $\boldsymbol{U}$, $\boldsymbol{A}$, and $\boldsymbol{W}$ are updated in real-time according to present conditions.

nearly assured of finding a unique resource provider favouring *any* resource property *individually*, one is unfortunately not assured of finding a unique resource provider which is favoured for *all* resource properties *together*.

One effective way to distinguish between resource providers *j*, which already optimally satisfy at least one of the requirements from request-submitter *K*, is to define a secondary optimisation metric based upon quota-usage for all resource properties at a particular resource provider. That is, for every resource provider *j* containing at least one non-zero resource property entry in the LP solution vector $\boldsymbol{x}_{jKL}$, calculate the minimum length:

$$\forall \boldsymbol{x}_{jKL} \neq \mathbf{0} \left(\min_j \left[\text{sqrt}\{ \Sigma_i (b_{iK}/A_{ijKL})^2 \} \right] \right)$$

This simply states that the favoured resource provider is that particular *j* which minimises the *overall* use of quotas, considering *all* required properties corresponding to request-submitter *K* (and at time window *L*). Used in connection with 0, this algorithm chooses a unique resource provider (up to multiple sites with identical lengths) from the set of resource providers with favoured properties. Finally, if there are multiple resource providers, each with identical and minimal overall quota-usage, a unique solution is made by randomly choosing one of the providers.

Quality of Service Constraints

Quality of service (QoS) constraints are supplied by a request submitter *K* in addition to, and in support of, a particular request requirement tensor $\boldsymbol{b}$ and may be expressed (for resource property *i* at resource provider *j*) in the form:

$$z_{ijK} < f_{ijK} < Z_{ijK}$$

where z_{ijK} (Z_{ijK}) is a lower (upper) limit and f_{ijK} is some function of $\boldsymbol{Q}$, $\boldsymbol{U}$, $\boldsymbol{W}$, $\boldsymbol{x}$, and/or $\boldsymbol{b}$.

In this chapter, we consider only a "greedy" realisation of quality of service in the sense that a quality of service is guaranteed to a request submitter by only considering that requester's individual desires, but disregarding other unscheduled requests in the system. This implies a time ordering and prioritization of requests. A more "socialised" quality of service that is realised by simultaneously considering the requester's individual desires as well as the overall cost (i.e. the impact placed on all other unscheduled requests currently in the system) will appear in a forth-coming chapter.

One simple example of a quality of service constraint that we investigate here is that of a submitter supplied deadline for request completion, or end date D_E. Let us assume that the element b_{0K} in the request requirement tensor $\boldsymbol{b}$ represents a usage requirement on CPU-time (e.g. in SI2000-hours) from request submitter K. Then let $C_j(b_{0K})$ represent the estimated wall clock completion time on resource j. In order to meet the desired quality of service at any particular resource j, the request must begin to be serviced on or before the start date $D_S = D_E - C_j(b_{0k})$. Such a request for quality of service may be interpreted in the context of 0 as, $z_{0jK} = \text{date}[\, l \,] < D_E - C_j(b_{0K}) = f_{0jK}$, where j represents a (leaf-node) resource, l represents (logarithmic) planning steps in time away from the present ($l = 0$), and date[l] converts discrete, relative time steps in l to an absolute date. By determining the latest possible start date over all resource providers j,

$\text{date}[\, P \,] = \max_j [\, D_E - C_j(b_{0k}) \,]$

one defines a feasible period of time (relative to now), $l < P$, in which the request may be serviced and still meet the specified QoS. Such a QoS constraint may be simply

imposed by restricting the sum over all future time intervals l to just that of the feasible time period $l < P$:

$$\min[\, f_{KL}(\mathbf{x}) \,] = \min[\, \Sigma_{ijk(l<P)}\, W_{jil}\, x_{ijkl}\, \delta_{lL}\, \delta_{kK} \,]$$

subject to: $\Sigma_{j(l<P)}\, A(t)_{iljk}\, x_{ijlk} \geq b_{ik}$

If a solution to this Linear Programming model is found, then it is the one which chooses the best provider to service the request from among the feasible space of resource providers and within the feasible time period, guaranteeing a simple QoS. It may not be possible to guarantee such a QoS, in which case the request submitter must either modify the requested requirements in $\boldsymbol{b}$ or the desired completion date D_E.

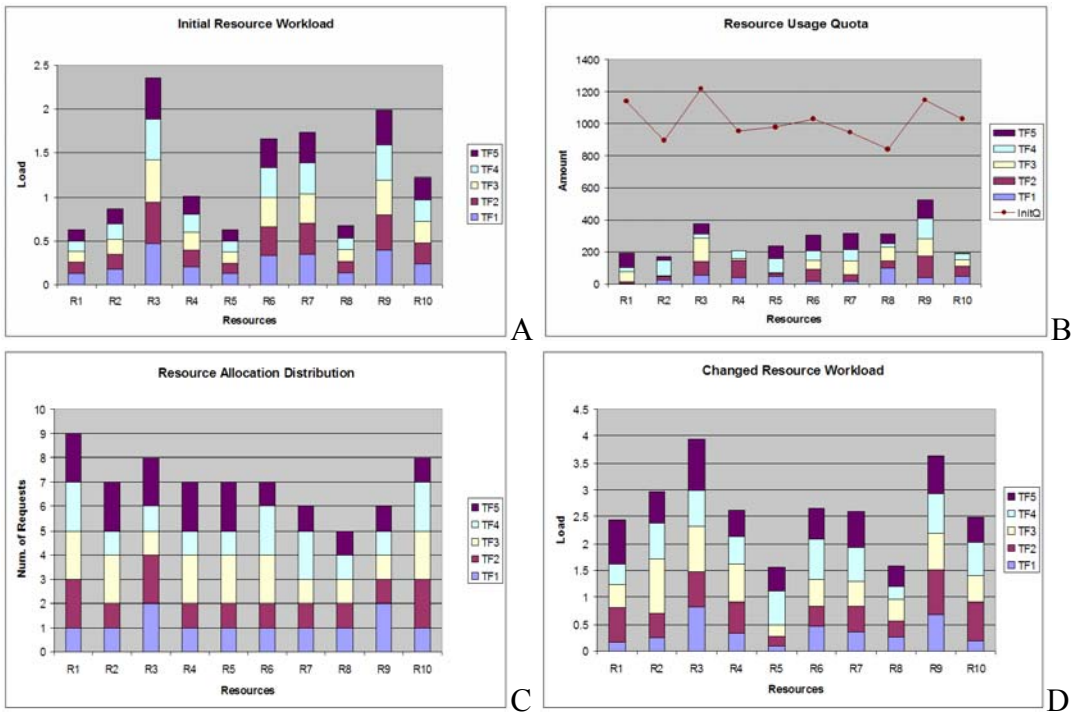


Figure 2-3. Policy based scheduling simulation results (In the legends of the graphs TFi means the i-th time frame, and InitQ represents the initially given resource usage quotas for a request submitter) A) Initial resource workload B) Resource usage quota C) Resource allocation distribution D) Changed resource workload.

Simulation Results

The policy based scheduling that has been described in mathematical formats is simulated in MATLAB. The simulation assumes ten resources are available for the request assignment by the scheduling strategy.

The first graph in Figure 2-3 shows the workload that each of the ten resources has in five time frames before any resource assignment for a request is made. The workload information on a resource in a time frame can be obtained from the estimated load status on the resource with information such as the current grid weather and request execution history in a Grid. Resources with ID 3 and 9 have higher initial workload than other resource, while resource with ID 1, 5 and 8 have fewer loads than the others in the beginning of the series of resource allocation. The second graph presents resource usage quota change. The dotted line in the graph shows the initial usage quotas that are available to a request submitter before the request submission, while the columns represent the quotas that are remained on the resources after the resource assignment. The third graph shows the request distribution among the resources after the policy-based scheduler completes the resource allocation for 70 requests. Each column in the graph represents the number of requests assigned to a resource in the time frames. The fourth graph shows the workload distribution on the resources after the scheduler completes the resource allocation.

These results show that the resource allocation depends not only on the workload but also on the resource usage quota for a request submitter. The policy-based scheduler achieves the resource assignment optimisation by considering the evenly consumed usage quota and the workload balance among the resources. For example, we can see that resource with ID 1 has been assigned to the largest number of requests in the simulation

because it has less overall workload for all the time frames, and a resource submitter is given high initial usage quota on the resources. In the case of request assignment to the resource with ID 8, even though the overall workload on the resource is less than on other resources, small number of requests is assigned to it because the quota for the submitter on the resource is less than on other resources.

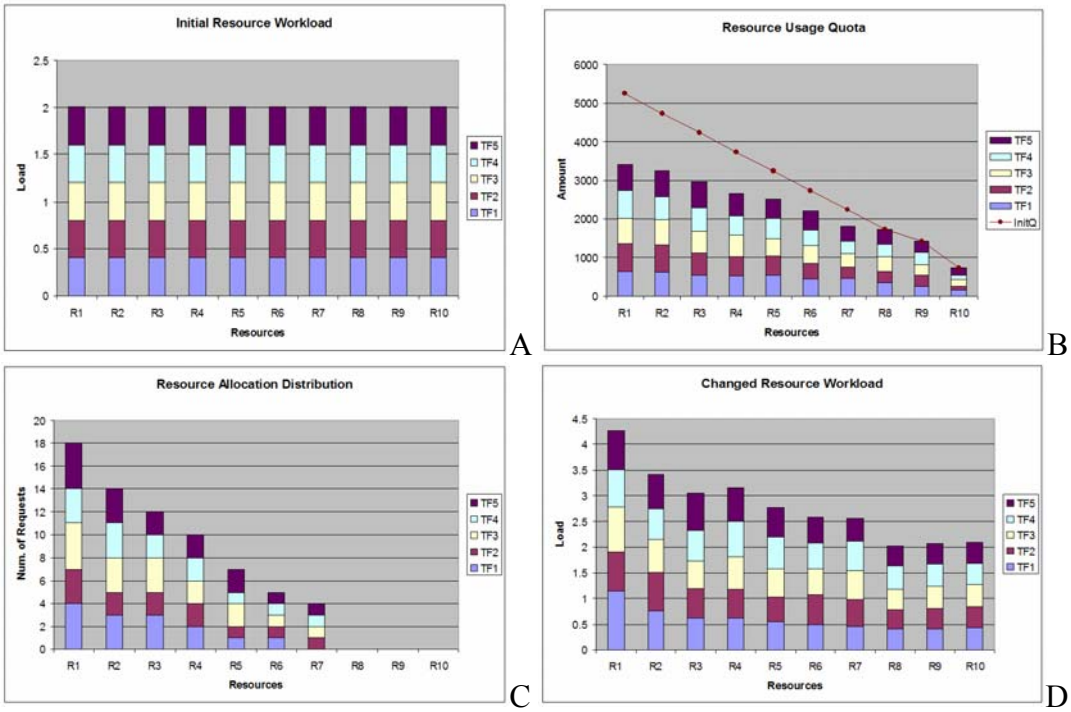


Figure 2-4. Policy based scheduling simulation results with highly biased resource usage quotas (In the legends of the graphs TF*i* means the *i*-th time frame, and InitQ represents the initially given resource usage quotas for a request submitter). A) Evenly distributed workload on the resources. B) Highly biased resource usage quotas. C) Resource allocation distribution. D) Changed resource workload.

A resource provider who manages the resource usage policies for the request submitters may change the policies to control the resource usage. For instance, the provider can increase the usage quota on the less loaded resources to make it more utilized, while decreasing the quota on the over loaded resources to prevent the scheduler from assigning additional requests to the resources. From the results, the quota on the

resource with ID 3 should be decreased to make the workload reduced in the future time frames. As an opposite case, the quota on the resource with ID 8 should be increased because the resource has been less loaded than the others. Increasing the quota causes the scheduler to assign larger number of requests to the resource.

The results in the Figures 2-4 and 2-5 show how the resource usage quotas and the resource workload distribution affect the resource allocation. Figure 2-4a shows the case that the workloads are evenly distributed in the five time frames before a policy-based scheduler starts a series of resource allocation. Figure 2-4b shows that the resource usage quotas for a request submitter are highly biased, in the sense that Resource with ID 1 provides the highest quota, whereas Resource with ID 10 allows the submitter use the resource with the smallest amount quota. Given the initial conditions the scheduler allocates the resources to requests following the biased quota allowance as seen in Figure 2-4c. Because the given workloads are same on the resources in all the time frames the unevenly distributed quotas only affect the resource allocation distribution. The Figure 2-4d presents changed resource workload after the completed resource allocation.

Figure 2-5 presents a case when the resource workloads are highly biased, while the resource usage quotas are same on resource in all the time frames at the time of resource allocation. With the initially given resource condition the request assignment is also highly biased following the resource workload distribution.

The results presented above, shows that a policy based scheduler or resource providers can control the workload on Grid resources by properly assigning resource usage quotas for requests submitters with the consideration of the load statuses that the

resources have been exposed. The scheduler can also achieve the load balanced request assignment on the resources utilizing the resource usage quota.

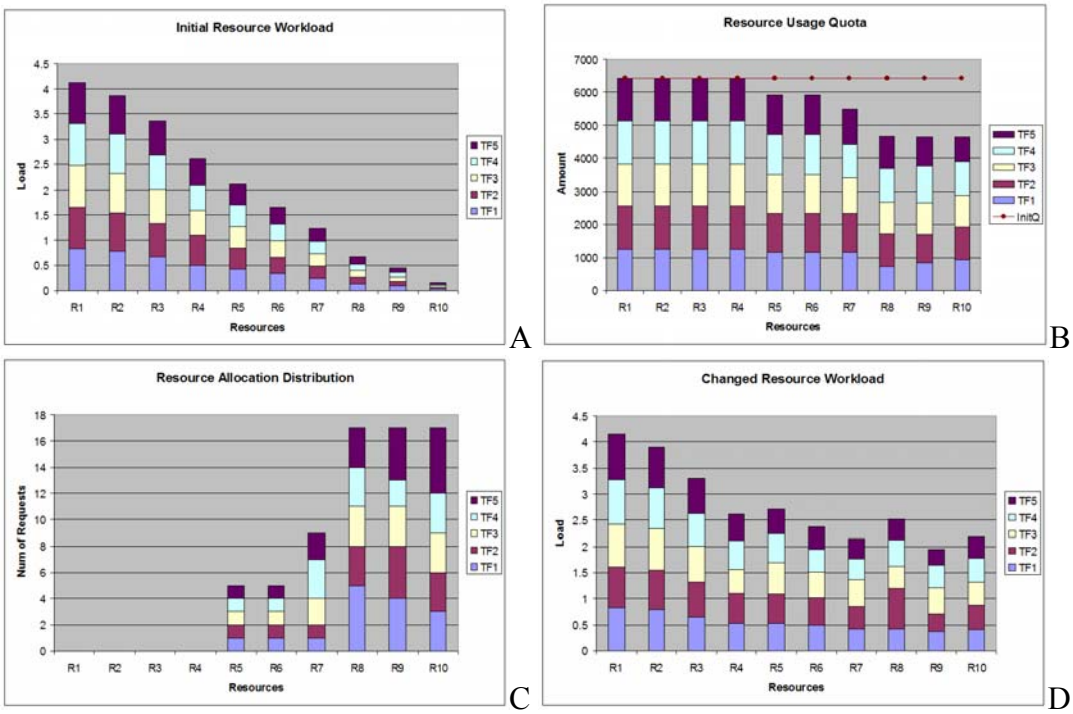


Figure 2-5. Policy based scheduling simulation results with highly biased workload on resources (In the legends of the graphs TF*i* means the *i*-th time frame, and InitQ represents the initially given resource usage quotas for a request submitter). A) Highly biased workload on the resources. B) Resource usage quota change. C) Resource allocation distribution. D) Changed resource workload.

Future Works

We have developed a novel framework for policy based scheduling and quality of service in grid computing. This framework uses a unique hierarchical representation of the key variables required for scheduling for effective and flexible representation. It also allows for a linear programming based solution strategy that is extensible to the inclusion of additional constraints and variables. Linear programming methods are well studied in the literature and several packages are available to provide fast and effective solutions based on the type and number of variables involved. Initial experimental results

demonstrate the usefulness of the framework and linear programming based solution methods for effective scheduling in multiple situations for policy and quality of service requirements. We are currently integrating a scheduling engine based on this framework into SPHINX and will present experimental results of the above policy framework on actual grids as well as the execution overhead of the solution strategy in the final version of the chapter.

CHAPTER 3
SPHINX: POLICY-BASED WORKFLOW SCHEDULING

A grid consists of high-end computational, storage, and network resources that, while known a priori, are dynamic with respect to activity and availability. Efficient scheduling of requests to use grid resources must adapt to this dynamic environment while meeting administrative policies.

In this section, first I discuss the necessary requirements of a grid scheduler. Then I present a scheduling framework called SPHINX that incorporates these unique grid characteristics, and implements the policy-based scheduling technique. The section also presents methods for integrating this framework with related infrastructure for workflow management and execution. I present early experimental results for SPHINX that effectively utilizes other grid infrastructure such as workflow management systems and execution systems. These results demonstrate that SPHINX can effectively schedule work across a large number of distributed clusters that are owned by multiple units in a virtual organization.

Requirements of a Grid-scheduling Infrastructure

A grid is a unique computing environment. To efficiently schedule jobs, a grid scheduling system must have access to important information about the grid environment and its dynamic usage. Additionally, the scheduling system must meet certain fault tolerance and customizability requirements. This section outlines the different types of information the scheduling framework must utilize and the requirements a scheduler must satisfy.

Information Requirements

A core requirement for scheduling in the dynamic grid environment is to successfully map tasks onto dynamically changing resource environment while maximizing the overall efficiency of the system. The scheduling algorithm that performs this mapping must consider several factors when making its decision. Seven factors significantly affect this scheduling decision:

Execution time estimation. Because of the heterogeneous nature of grid resources, their real execution performance differs from the optimal performance characterized by analytic benchmarking [6]. However, the real execution time can be effectively estimated, even on heterogeneous grid resources, by statistically analyzing the performance during the past executions [62]. Along with the past execution information, several methods such as statistical analysis, neural networks or data mining, can be used to estimate the execution time of a task [63].

Usage policies. Policies, including authentication, authorization, and application constraints are important factors for maintaining resource ownership and security. The set of possible constraints on job execution can be various and can change significantly over time. These constraints can include different values for each job, for example RAM requirements or connectivity needed, or constraints that are static for a specific job type, such as the operating system or architecture. Policies may include any information that should be specified to ensure that a job is matched to appropriate resources [60, 61].

Grid weather. The scheduling system must keep track of dynamically changed load and availability of grid resources. In addition, faults and failures of grid resources are certain to occur. The state of all critical grid components must be monitored, and the information should be available to the scheduling system.

Resource descriptions. Due to the heterogeneous nature of the grid, descriptions of grid resource properties are vital. Such descriptions include configuration information such as pre-installed application software, execution environment information such as paths to local scratch spaces, as well as hardware information. In addition, such grid resources may often only be available to users at an aggregate or logical level, hiding the actual, physical resource used to execute a job. Hence, the ability to categorize as well as to both finely and coarsely describe grid resource properties is important.

Replica management. The scheduling system must arrange for the necessary input data of any task to be present at its execution site. Individual data locations can have different performance characteristics and access control policies. A grid replica management service must discover these characteristics and provide a list of the available replicas for the scheduler to make a replica selection.

Past and future dependencies of the application. Grid task submission is often expressed as a set of dependent subtasks and modeled as a Directed Acyclic Graph (DAG). In this case, the subtasks are represented by nodes in a graph, and the dependencies are by branches. When allocating resources to the subtasks, inter-task dependencies affect the required data movement among resources. Utilization of this dependency information can generate a provably optimal scheme for communicating shared data among subtasks [64].

Global job descriptions. A grid is primarily a collaborative system. It is not always ideal within the grid to schedule a particular task or group of tasks for maximum efficiency. A good balance needs to be struck between the requirements of each user and

the overall efficiency. It is important for a grid scheduling system to have a global list of pending jobs so that it can optimize scheduling to minimize this global cost.

System Requirements

While the kinds of information above should be available to the system for efficient grid scheduling, the following requirements must be satisfied in order to provide efficient scheduling services to a grid Virtual Organization (VO) community.

Distributed, fault-tolerant scheduling. Clearly, scheduling is a critical function of the grid middleware. Without a working scheduling system (human or otherwise), all processing on the grid would quickly cease. Thus, any scheduling infrastructure must be strongly fault-tolerant and recoverable in the inevitable case of failures. This need for fault tolerance consequently gives rise to a need for a distributed scheduling system. Centralized scheduling leaves the grid system prone to a single point of failure. Distributing the scheduling functionality between several agents is essential to providing the required fault tolerance.

Customizability. Within the grid, many different VOs will interact within the grid environment and each of these VOs will have different application requirements. The scheduling system must be customizable enough to allow each organization with the flexibility to optimize the system for their particular needs.

Extensibility. The architecture of the scheduling system should be extensible to allow for the inclusion of higher level modules into the framework. Higher level modules could help map domain specific queries onto more generic scheduling problems and map domain specific constraints onto generic scheduling constraints.

Interoperability with other scheduling systems. Any single scheduling system is unlikely to provide a unique solution for all VOs. In order to allow cooperation at the

level of VOs, for example in a hierarchy of VOs or among VO peers, the scheduling system within any single VO should be able to route jobs to or accept jobs from external VOs subject to policy and grid information constraints. Inter-VO cooperation is an architectural choice reflecting a tradeoff between synchronization and flexibility of low level middleware choices and configurations across large organizations.

Quality of service. Multiple qualities of service may be desirable as there are potentially different types of users. There are users who are running small jobs that care about quick turnaround time or interactive behavior from the underlying system. On the other hand, large production runs may be acceptably executed as batch jobs. Users may put deadlines by which submitted jobs should be completed. The scheduling system should be able to provide these differential QoS features for the administrator/users.

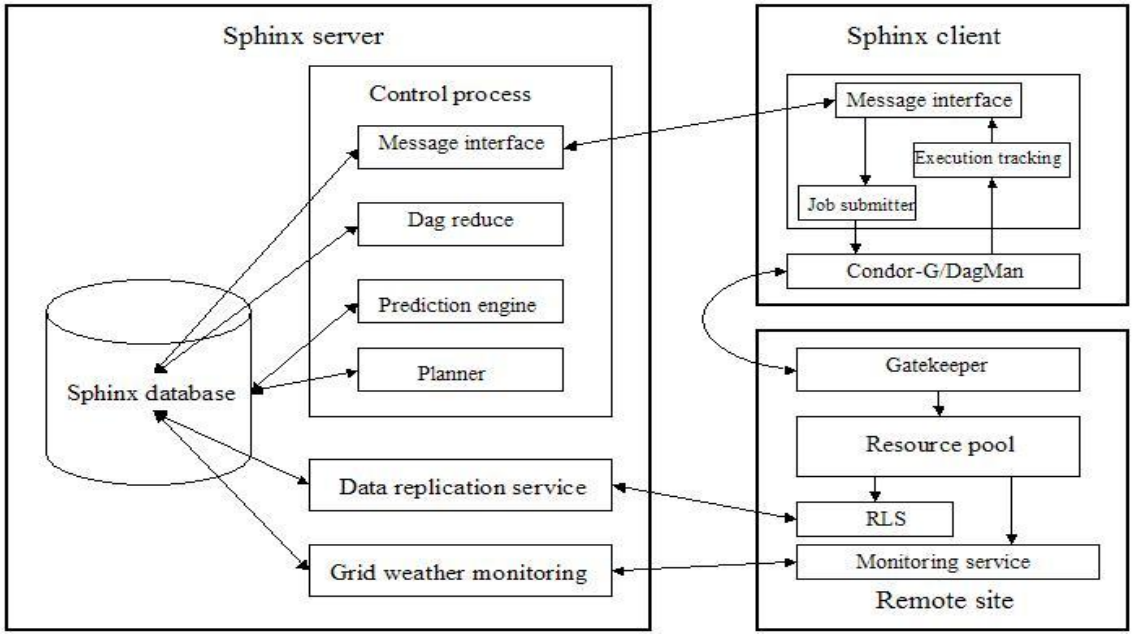


Figure 3-1. Sphinx scheduling system architecture

Highlights of SPHINX Architecture

The scheduling system, SPHINX is a novel scheduling middleware in dynamically changing and heterogeneous grid environment. In addition to the infrastructure requirements for grid scheduling described in the previous section, SPHINX focuses on several key functionalities in its architecture shown in Figure 3- for efficient and fault tolerant scheduling service.

Easily accessible system. SPHINX is modeled with an agent-based scheduling system consisting of two parties, the client and the server. The separation supports easy system accessibility and adaptability. The client is a lightweight portable scheduling agent that represents the server for processing scheduling requests. It provides an abstract layer to the service, and supports a customized interface to accommodate user specific functionalities.

Automated procedure and modulated architecture. SPHINX consists of multiple modules that perform a series of refinement on a scheduling request. The procedure begins from a 'start', and the final state should be annotated with 'finished', which indicates the resource allocation has been made to a request. Each module takes a request, and changes its state according to the functionality of itself. The system easily modifies or extends the scheduling automation by making necessary changes to a module without affecting the logical structure of other modules.

Table 3.1 shows all the states for jobs and dags defined in the current SPHINX prototype.

Robust and recoverable system. The SPHINX server adopts database infrastructure to manage scheduling procedure. Database tables support inter-process communication among scheduling modules in the system. A module reads scheduling

state of a request from the tables, edits the state, and writes the modification to the tables. It also supports fault tolerance by making the system easily recoverable from internal component failure.

User interactive system. SPHINX supports user interaction to its resource allocation procedure. A user submits a request with quality of service (QOS) requirement. The requirement may specify resource usage amount and period. It is challenging to be able to satisfy the specification in a dynamically changing grid environment. SPHINX facilities the scheduling decision by negotiating QOS achievement with a user.

Table 3-1. Finite automation of SPHINX scheduling status management.

DAG States	**Description**
Unreduced	The DAG has yet to be processed by the DAG reducer.
Unpredicted	The completion time of the DAG has not been estimated by the prediction engine.
Unaccepted	The completion time has been estimated for this DAG, and the server is waiting for the client to accept this estimation for final processing.
Unfinished	The DAG has begun processing but has not completed.
Remove	All jobs within this DAG have completed and can be removed from the queue during the next cleanup cycle.
Job States	Description
Unpredicted	The completion time of this job has not been estimated by the prediction engine.
Unaccepted	The DAG containing this job has not been accepted for final execution by the client.
Unplanned	The job has been accepted for scheduling by this server, but the planner has not created an execution plan for this job.
Unsent	The job is planned, but has not been sent to the client for execution.
Unfinished	The job is running on a remote machine and must be tracked by the tracking system.
Remove	The job has finished execution and no longer must be tracked by the tracking system. If the parent DAG for this job is also finished, the job may be removed from the tracking system during the next clean up phase.

Platform independent interoperable system. A scheduling system should expect the interaction with systems on various kinds of platforms in a heterogeneous environment. SPHINX adapts communication protocols based on XML such as SOAP and XML-RPC to satisfy the requirement. Specially, it uses the communication protocol named Clarens [65] for incorporating the concept of grid security.

Table 3-2. SPHINX client functionalities for interactive job scheduling and execution tracking

Functions	SPHINX API's	Parameters
Execution request	job_submit (String file_loc, String sender) //Client submits this request to SPHINX client	file_loc: abstract dag file location sender: request sender information
Scheduling request	send_request (String dagXML, String msgType, String sender) //SPHINX client sends this request to server	dagXML: dag in XML format msgType: request type sender: request sender information
Admission control	send_msg (String msgXML, String msgType) send_msg (String msgXML, String msgType, String sender) //SPHINX and user interact for resource allocation	msgXML: message in XML format msgType: message type sender: message sender info.

Submission request	createSubmission (String jobInfo)
	//SPHINX client create a submission file.
	submit_job (String rescAlloc, String submitter)
	//SPHINX client send the file to DAGMan/Condor-G
		JobInfo: scheduling decision information
		rescAlloc: job submission file
		Submitter: job submitter information

Execution tracking	updateStatus (int jobId) String status = getJobStatus (int jobId) //Update the status of the job with the information //from a grid resource management service	jobId: ID of a job that is currently running on a grid resource. The ID is assigned by the grid resource management system. status: the status of job, which the grid resource management system provides.

SPHINX Client

SPHINX client interacts with both the scheduling server that allocates resources for task execution and a grid resource management system such as DAGMan/Condor-G [4]. To begin the scheduling procedure, a user passes execution request to SPHINX client. The request is in the form of an abstract DAG that is produced by a workflow planner

such as the Chimera Virtual Data System [66]. The abstract plan describes the logical I/O dependencies within a group of jobs. The client sends scheduling request to the server with a message containing the DAG and client information. After receiving resource allocation decision from the server, the client creates an appropriate request submission file according to the decision. The client submits the file to the grid resource management system.

In order to achieve a user's quality of service (QoS) requirement SPHINX implements interactive resource allocation. The client as a scheduling agent negotiates QoS satisfaction level with the user. SPHINX presents the user resource allocation decision such as estimated execution time, resource reservation period and amount according to the current grid resource status. Then the user should decide acceptance of the suggestion. A basic QoS negotiation feature is developed in the current system, while we develop detailed and sophisticated version.

The tracking module in the client keeps track of execution status of submitted jobs. If the execution is held or killed on remote sites, then the client reports the status change to the server, and requests re-planning of the killed or held jobs. The client also sends the job cancellation message to the remote sites on which the held jobs are located. Table 3.2 shows the functionalities and SPHINX API's.

SPHINX Server

SPHINX server as a major component of the system performs several functions. The current version of the server supports the following functions. First, it decides how best to allocate those resources to complete the requests. Second, it maintains catalogs of data, executables and their replicas. Third, it provides estimates for the completion time of the requests on these resources. Fourth, the server monitors the status of its resources.

SPHINX server completes resource allocation procedure by changing the status of requests through the predefined states in Table 3.1.

As mentioned in the previous section, each of the functions in SPHINX is developed in modulated fashion. Each module performs its corresponding function to a DAG or a job, and changes the state to the next according to the predefined order of states. We discuss each of the modules in detail in the following statements. Table 3.3 shows SPHINX server functions described in this section.

Table 3-3. SPHINX server functions for resource allocation

Modules	SPHINX API's	Parameters
Message Handling Module	String incMsg = inc_msg_wrapper () out_msg_wrapper (String msg) msg_parsing (String msg) msg_send (String msg, String msgType String dest, String sender) //These functions are to send, receive and parse messages. //The message handling module is a gateway to SPHINX server	incMsg: incoming message msg: message in XML format msgType: message type dest: receiver information sender: sender information
DAG Reducer	dag_reducing (int dagId) //For each of all the jobs in the dag, call replica management //service to check if all the outputs of the job exist. If exist, //then reduce the job and all the precedence of the job.	dagId: ID of a dag in dag table. The state of the dag is unreduced
Prediction Engine	String est_info = exec_prediction(int jobId) //This function provides estimated information such as //execution time, resource usage (CPU, storage etc.)	est_info: estimated data in XML string format jobId: ID of a job in job table
Planner	Planning (int jobId, String strategy) //It is to allocate resources to jobs according to scheduling //strategy.	JobId: ID of job to be planned strategy: scheduling algorithm

Control Process. The main function of the control process is to launch the necessary server-side service modules to process resource allocation to a request. Stateful entities such as DAG's and Jobs are operated and modified by scheduling modules. This architecture for the SPHINX Server, in which the control process awakens modules for processing stateful entities, provides an extensible and easily configurable

system for future work. Figure 3- shows overall structure of control process in SPHINX. The controller checks the state of jobs that are currently in the procedure of scheduling. If the controller finds a job in one of the states, then it invokes a corresponding service module to handle the job.

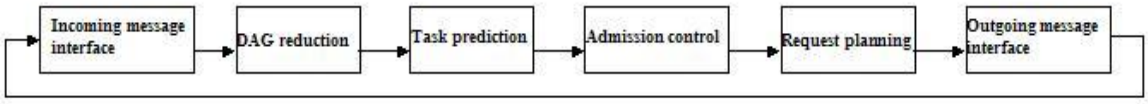


Figure 3-2. Overall structure of control process.

Message Handling Module. The message handling function is to provide a layer of abstraction between the internal representation of the scheduling procedure and the external processes. Additionally, the function is responsible for maintaining a list of all the currently connected clients, for ensuring that the list is kept accurate and for directing I/O from the various internal components to these various clients. The server maintains database tables for storing incoming and outgoing messages. Control process invokes incoming or outgoing message interfaces to the tables for retrieving, parsing and sending the messages.

DAG Reducer. The DAG reducer reads an incoming DAG, and eliminates previously completed jobs in the DAG. Such jobs can be identified with the use of a replica catalog. The DAG reducer simply checks for the existence of the output files of each job, and if they all exist, the job and all precedence of the job can be deleted. The reducer consults replica location service for the existence and location of the data.

Prediction Engine. The prediction engine will provide estimates of resource use. It estimates the resource requirements of the job based upon historical information, size of input if any, and/or user provided requirements. In the first implementation, this is

constrained to overall execution time by application and site. A simple average and variance calculation is used to provide an initial estimation scheme; this method could be made more intelligent and robust in future implementations. When the prediction engine is called, it selects a DAG for prediction, estimates the completion time of each job, and finally from this data, estimates the total completion time of the DAG.

Planner. The planner module creates an execution plan of the job whose input data are available. According to the data dependency a job should wait until all the precedent finish to generate output files. The execution plan includes several steps:

4. Choose a set of jobs that are ready for execution according to the input data availability.
5. Decide the optimal resources for the job execution. The planner makes resource allocation decision for each of the ready jobs. The scheduling is based on resource status and usage policy information, job execution prediction, and I/O dependency information described in Section 3.
6. Decide whether it is necessary to transfer input files to the execution site. If necessary, choose the optimal transfer source for the input files.
7. Decide whether the output files must be copied to persistent storage. If necessary, arrange for those transfers.

After the execution plan for a job has been created, the planner creates an outgoing message with the planning information, and passes the message to a message-handling module.

Tracking Module. The job-tracking module is responsible for keeping the tracking/prediction database on the SPHINX server current. In this first version, the primary functions of the tracking module is to check for job completion, update the job state in the tracking/prediction database, and when the job completes, add the execution time to the job's historical data in the prediction data tables. In later versions, additional functions can be added. For example, the tracking module could also track job resource

use in real time. It could then enforce resource usage policies by killing and re-queuing jobs that overstep estimated bounds. Of course, the tracking module can also monitor a host of additional prediction information and record it in the prediction tables.

Table 3-4. SPHINX API's for accessing data replicas through RLS service. In the table PFN or pfn represents physical file name, and lfn means logical file name.

API's	Parameters
Vector pfns = getPFN (String lfn) //It returns PFN mappings for the given lfn.	pfns: a list of physical file names lfn: logical file name

CreateMapping (String lfn, Sttring pfn)
//It creates mapping for the given lfn and
//pfn in the RSL service database.
	lfn: logical file name
	pfn: physical file name

Data Replication Service

The data replication service is designed to provide efficient replica management. SPHINX provides an interface to the service. The Globus Replica Location Service (RLS) [67] provides both replica information and index servers in a hierarchal fashion. In addition, GridFTP [68] is Grid Security Infrastructure (GSI) enabled FTP protocol that provides necessary security and file transfer functions. Initial implementations of SPHINX will both make use of RLS and GridFTP for replica and data management.

In this initial implementation, the detection and replication algorithms are based on a memory paging technique. Once a hot spot has been identified, an appropriate replication site is chosen; initially, the site will be chosen at random. As future versions are developed, both the method of hot spot identification and replication location selection will be modified using improved replica management systems as well as better algorithms. Table 3.4 shows SPHINX API's for accessing replica information through RLS.

Grid Monitoring Interface

The resource allocation decision made by the planner and the replication site selection by the data replication service depends on the information provided through the monitoring interface of SPHINX. As such, the interface provides a buffer between external monitoring services (such as MDS, GEMS, VO-Ganglia, MonALISA, and Hawkeye [69,70,71]) and the SPHINX scheduling system. In order to accommodate the wide-variety of grid monitoring services, the interface is developed as an SDK so that specific implementations are easily constructed. We use an interface to MonALISA for accessing resource monitoring information (Table 3.5).

Table 3-5. Database table schemas for accessing resource-monitoring information through MonALISA

Tables	Fields	Function
SITE_FARM_MAP	site_name varchar(255) not null farm_name varchar(100) not null	This table stores mappings from grid resource IP to MonALISA farm name.
ML_SNAPSHOT	site_name varchar(255) not null mfunction varchar(100) not null value double	This table stores the latest monitoring value for the given site and function.
ML_SNAPSHOT_P2P	site_source varchar(255) site_dest varchar(255) mfunctions varchar(100) value double	This table stores the latest monitoring value for the given function between the given two sites.

Grid monitoring service will be used to track resource-use parameters including CPU load, disk usage, and bandwidth; however, in addition, a grid monitoring service could possibly also collect policy information provided from each site, including resource cost functions and VO resource use limits. Additional interface modules will be developed to gather VO-centric policy information, which may be published and maintained by a VO in a centralized repository.

External databases. Replica information is likely to be tied closely to a particular VO, and it is necessary to incorporate external replica services. Two external

catalogs are considered in this chapter: the transformation catalog (TC) and the replica catalog (RC). Because the functionality is similar, both the RC and the TC may use similar or identical technology. The TC maintains information on transformations (these should be provided by workflow management systems such as Chimera). The TC maps a logical executable name to one or many physical executable names on particular grid resources. The RC maintains information on data produced within, or uploaded to a grid. The RC maps a logical file name to one or many physical file names on particular grid resources.

Internal database. The prediction engine queries the Prediction Tables (PT) to get estimated information for job execution, including execution time, CPU and disk usage, and bandwidth. In the PT, a vector of input parameters and a site with a vector of benchmarks represents a job. Thus, for every execution of every set of input parameters (job) on every set of benchmarks (site), the resources consumed by the job are recorded. By querying this database of historical processing information, the prediction engine can estimate the time required to execute a given job on a given set of resources.

The Job Tables (JT) maintains a persistent queue of submitted jobs. It monitors their progress, providing information on the status and location of job execution. In addition, all DAG (or job) state information in the scheduling system is incorporated into the job database. The tracking system within the scheduling server accesses the grid-monitoring interface and maintains the information in the JT.

Relationship with Other Grid Research

In this section, we discuss how SPHINX system research and development (R&D) interact with other ongoing R&D works.

Grid Information Services

Several systems are available for gathering information including resources access policies, resource characteristics and status, and required application environments. The current version of Globus Toolkit includes the Monitoring and Discovering Service (MDS) that can, in combination with other tools such as Ganglia, MonALISA and GEMS, gather this and other grid information. This system or others can be easily connected to the SPHINX system using the Grid Monitoring SDK.

Replica and Data Management Services

The Globus Replica Location Service (RLS) [67] provides both replica information and index servers in a hierarchal fashion. In addition, GridFTP is Grid Security Infrastructure (GSI) enabled FTP protocol that provides necessary security and file transfer functions. Initial implementations of SPHINX will both make use of RLS and GridFTP for replica and data management.

The Network STorage (NeST) service provides a common abstracted layer for data access in grid environments [72]. In addition, the Stork Data Placement (DaP) scheduler manages and monitors data placement jobs in grid environments [73]. As the SPHINX work develops, both NeST and Stork can be incorporated into the framework to provide robust, transparent management of data movement.

Job Submission Services

The GriPhyN VDT includes Condor-G as a grid submission and execution service. Condor-G uses the Globus GRAM API to interact with Globus resources [74], and provides many execution management functions, including restarting failed jobs, gathering execution log information, and tracking job progress. In addition, DAGMan is a job execution engine that manages the execution of a group of dependent jobs

expressed as a DAG. By leveraging the DAGMan job submission protocol, the scheduling system can dynamically alter the granularity of the job submission and planning to most efficiently process the jobs in its queue. For instance, if the prediction information indicates that a large group of fast jobs is waiting in the queue and the planner can assume the grid status will remain relatively stable during their execution, it can make a full-ahead plan for the whole group, construct a DAGMan submission DAG and pass the entire structure to DAGMan to manage. Conversely, if there are many long running jobs in the queue, the planner is able to release them one at a time to DAGMan to take full advantage of "just-in-time" scheduling.

Virtual Data Services

The SPHINX scheduler is currently designed to process abstract DAGs, as provided by the Chimera Virtual Data System, for grid execution. In particular, this work is fully integrated with the Chimera Virtual Data and Transformation Catalogs. These integration points provide SPHINX with the ability to flexibly and dynamically schedule large DAG workflows.

Future Planners and Schedulers

One of the main purposes of SPHINX research and development is to develop a robust and complete scheduling framework where further research and development can continue. Thus, it is essential that this initial development work should provide a feature for integrating new planning technology to the framework easily. Such integration is provided in two ways. First, the current SPHINX is developed in a modularized fashion. Modules in the system are developed, and tested independently. A single current SPHINX component can be exported into other systems, or new modules can be plugged into SPHINX system easily. Second, the planning module within the scheduling server

interacts only with the internal database API. Through this interface, any planning or strategy module can be easily added to the scheduling server. A collection of such planning modules could be provided, and the scheduling server chooses the most optimal based on the composition of the input tasks.

Table 3-6. Grid sites that are used in the experiment. CalTech represents California Institute of Technology, UFL does University of Florida, and UCSD does University of California at San Diego.

Site Name	Site Address	# of Processors	Processor Type
CalTech	citcms.cacr.caltech.edu	Four	Dual
UFL	ufloridadgt.phys.ufl.edu	Three	Dual
UFL	ufloridaigt.phys.ufl.edu	Nine	Dual
UCSD	uscmstb0.ucsd.edu	Three	Singular

Experiments and Results

The aim of the experiments we performed was to test the effectiveness of the scheduler as compared to the way things are done today on the Grid. We also compared and contrasted the different scheduling algorithms. There were two different features of grid systems that we evaluated:

Importance of feedback information: The "feedback" provides execution status information of previously submitted jobs on grid sites. The scheduling algorithms can utilize this information to determine a set of reliable sites to schedule jobs. Sites having more number of cancelled jobs than completed jobs are marked unreliable. This strategy is not used in the algorithms marked as without-feedback.

Importance of Monitoring Information: The monitoring systems provide performance of different components (especially compute resources) on the grid. This information is updated frequently and can be potentially used for effective scheduling. We wanted to determine the value of this information in effective scheduling.

Scheduling Algorithms

Round robin scheduling algorithm tries to submit jobs in the order of sites in a given list. All sites are scheduled to execute jobs without considering the status of the sites. If some sites are not available to execute the planned jobs, then the jobs are cancelled, and planned onto the next site in the list.

Algorithm based on the number of CPUs with feedback utilizes resource-scheduling information of previously submitted jobs in a local SPHINX server. After determining load rate with the following formula for each site it plans jobs to the least loaded sites.

$$\text{rate}_i = \frac{(\text{planned\_jobs}_i + \text{unfinished\_jobs}_i)}{\text{CPUs}_i}$$

where

rate_i : the load rate on site i

planned_jobs_i : the number of planned jobs to site i

unfinished_jobs_i : the number of running jobs on site i

CPUs_i : the number of CPUs on site i

Queue length based scheduling algorithm makes the scheduling decision based on the lengths of the job queues at the remote sites provided by a monitoring module. This algorithm utilizes the feedback information to determine the job execution site. The scheduler selects a site with the smallest load rate according to the following formula.

$$\text{rate}_i = \frac{(\text{queued\_jobs}_i + \text{running\_jobs}_i + \text{planned\_jobs}_i)}{\text{CPUs}_i}$$

where

rate_i : the load rate on site i

queued_jobs_i : the number of waiting jobs on site i

running_jobs_i : the number of jobs which are currently assigned to CPUs on site i

planned_jobs_i : the number of planned jobs to site i on a local scheduler

CPUs_i : the number of CPUs on site i

Job completion time-based scheduling algorithm utilizes the job completion rate information passed on by the job tracker module from the client to the server. In the absence of the job completion rate information, SPHINX schedules jobs based on round robin technique until it has that information for the remote sites. Thus, it uses a hybrid approach to compensate for unavailability of information. The site having the minimum job completion rate is chosen for the next schedulable job.

$$\underset{1 \le i \le n,\ A_i \neq 0}{Minimum} \left(\frac{Avg\_comp_i}{\sum_{k=1}^{n} Avg\_comp_k} \times A_i \right), \quad A_i = \begin{cases} 1, & \text{if site } i \text{ is available} \\ 0, & otherwise \end{cases}$$

where

n : the number of grid sites

Avg_comp_i : the average job completion time on site i

A_i is computed using the feedback information

Policy-constrained scheduling puts resource usage constraints on each of the algorithms. For example, the next formula shows a revised round robin scheduling algorithm with resource usage constraints.

site s

such that

$\text{quota}_{i,s} \geq \text{required}_{i,s}$, for property i

where

$\text{quota}_{i,s}$: usage quota of property i on site s given to a user

$\text{required}_{i,s}$: required amount of property i on site s specified by a user

Test-bed and Test Procedure

We use Grid3 as the test-bed for this experiment. Grid 3 currently has more than 25 sites across the US and Korea which collectively provide more than 2000 CPUs. The resources are used by 7 different scientific applications, including 3 high energy physics simulations and 4 data analyses in high energy physics, bio-chemistry, astrophysics and astronomy.

It is critical to test the performance of scheduling algorithms in the same grid environment because the resource availability changes dynamically. Each of these scheduling algorithms was executed on multiple instances of SPHINX servers multiple number of times that were running concurrently to compare pair-wise or group-wise performance. These servers were started at the same time so that they can compete for the same set of grid resources. We felt this was the fairest way to compare the performance of different algorithms in a dynamically changing environment. Table 3.7 shows the configuration of the machines that we set up SPHINX clients and servers.

Table 3-7. SPHINX server configurations

Name	CPU (Mhz)	RAM(Mb)	OS
dimitri.dnsalias.net	2x800	512	RH 7.3
Julian.dnsalias.net	2x3000	2000	Fedora Core 2

However, the availability and performance of each site grid changes dynamically. This fact makes the number of sites available during different experiments to vary. Each scheduling algorithm also chooses different set of sites for submitting jobs according to

its planning decision. The performance comparisons below represent our best case efforts to eliminate these effects by executing these in a pair-wise or group-wise approach described above.

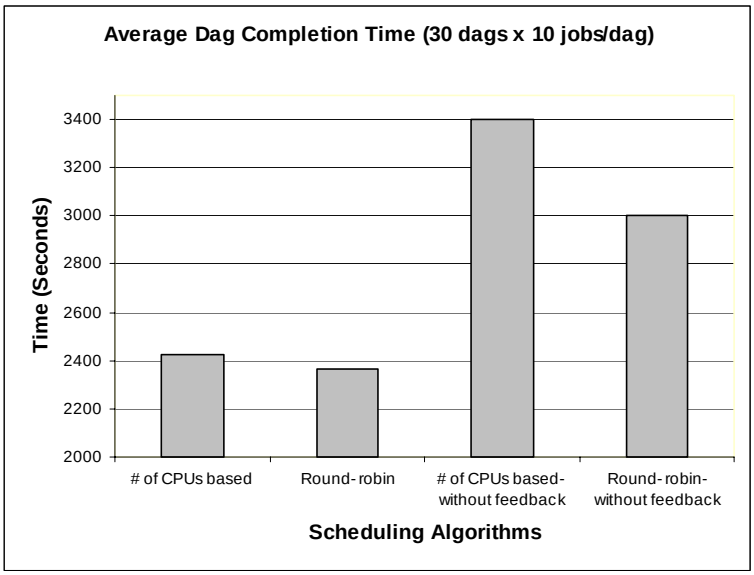


Figure 3-3. Effect of utilization of feedback information.

Performance Evaluation of Scheduling Algorithms

In order to compare the performance of the algorithms, we submit 30, 60 and 120 DAGs, each of which has 10 jobs in random structure. The job simulates a simple execution that takes two or three input files, spends one minute before generating an output file. The size of output file is different for each job, and the file is located on the execution site by default. Including the time to transfer remotely located input files onto the site it is expected that each job will take about three or four minutes to be completed. The depth of the DAGs is up to five, while the maximum number of independent jobs in a level is four or five.

Effect of Feedback Information

We start by studying the impact of using feedback information. We compare the average DAG completion times for the *round-robin* and *#-of-CPUs-based scheduling*

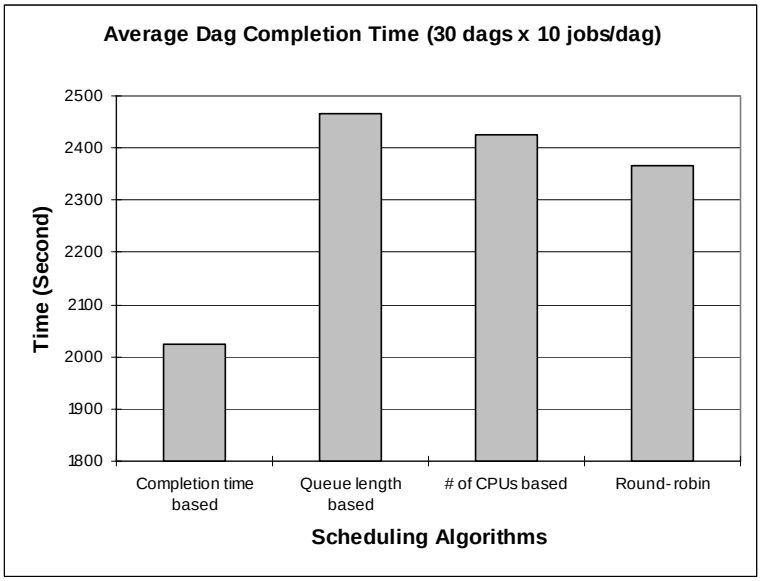
algorithms with and without feedback. The graph shown in Figure 3-3 plots the average DAG completion time for four algorithms – Round robin with feedback, round-robin without feedback, Number-of-CPUs-based scheduling algorithm with feedback and Number-of-CPUs-based scheduling algorithm without feedback. Feedback is basically used for flagging an execution site as faulty. This is done considering the number of completed and cancelled jobs on that site as reported to SPHINX server by the job tracker(s).

The feedback information includes the status of job execution such as hanged, killed, or completed on execution sites. A scheduler should be able to utilize this information to determine a set of available sites to schedule jobs. Without this feedback the scheduler keeps submitting jobs to unreliable sites resulting in many rescheduling jobs getting cancelled. As shown in the figure the average DAG completion time with the feedback information is less than the other case by about 20~29%.

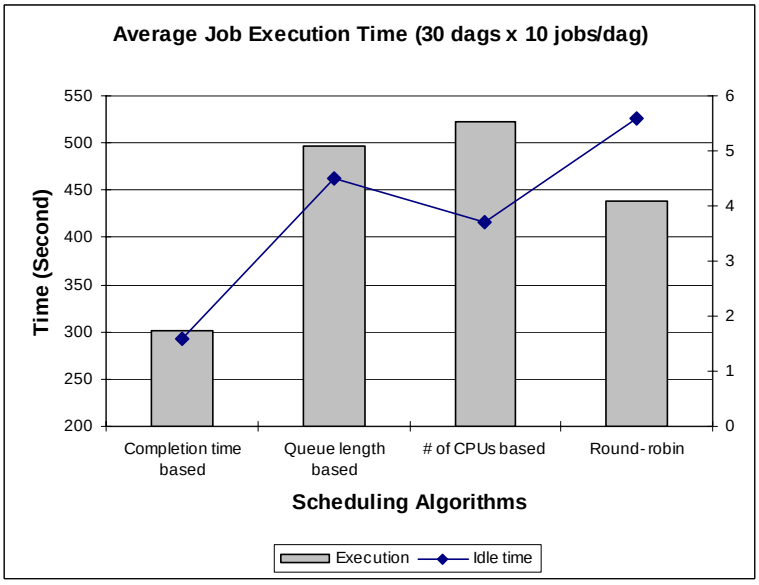
Round-robin scheduling without feedback is basically what a grid user would use for executing his jobs on the grid. This experiment basically proves how the feedback information in critical. It also brings to the fore the faultiness of the grid environment and how fault-tolerance can be achieved using SPHINX.

Comparison of Different Scheduling Algorithms with Feedback

The aim of the experiment is to demonstrate the dynamic nature of the grid and to evaluate which monitored parameter works best for making scheduling decisions.



A

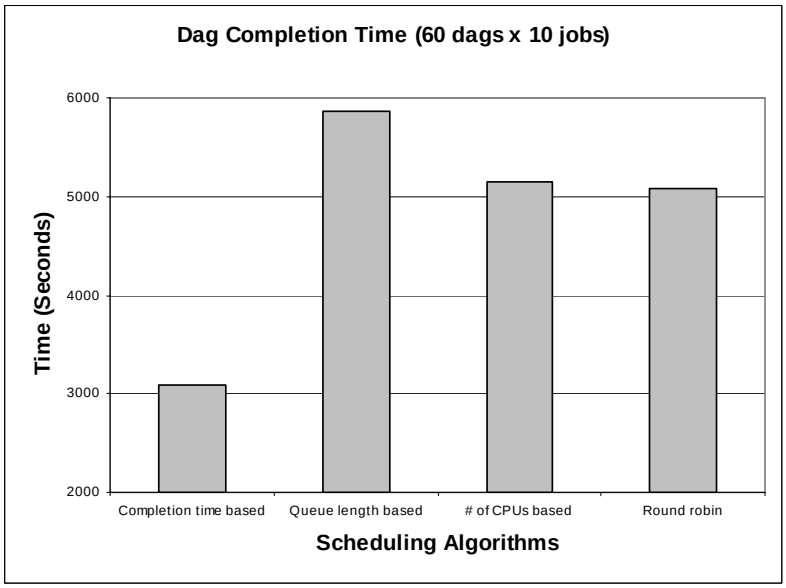


B

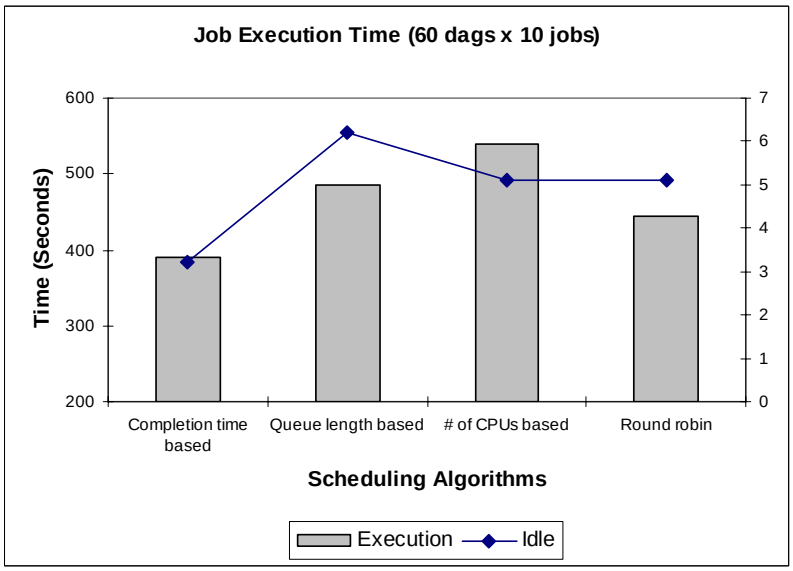
Figure 3-4. Performance of scheduling algorithms with 300 jobs and without any policy constraints (A) & (B). The Round-robin algorithm distributes the jobs equally among all available sites; the number-of-CPUs-based algorithm considers the relative number of CPUs on that site (static); the Queue-length based approach considers the job queue information (dynamic) from the monitoring system; while the Completion time based strategy uses the average job completion rates to select the execution site.

Figure 3-4(a) shows the performance of four different algorithms. Completion time-based scheduling algorithm (hybrid) performs better than other cases by about 17% in terms of average DAG completion time. This is because the scheduling algorithm schedules jobs to the sites that complete assigned jobs faster than other sites. Figure 3-4(b) presents job execution and idle time information. Jobs scheduled by completion time algorithms are executed faster than ones by other algorithms by about 50%, and execution waiting or idle time is less than by about 60 %.

The same experiment was repeated with 600 and then 1200 jobs instead of just 300 jobs. Figure 3-5 gives the result for the 600 jobs experiment. Here, we actually observe that the Job completion rate based approach performs comparatively better than other algorithms as compared to the 300 jobs experiment. Here the performance of the Job completion time based approach is from ~33% to ~50% better than other scheduling strategies. This is because the algorithm gets smarter as the scheduling processes – with more reliable job completion time information - and makes the planning decision for jobs effectively using the wider knowledge base.

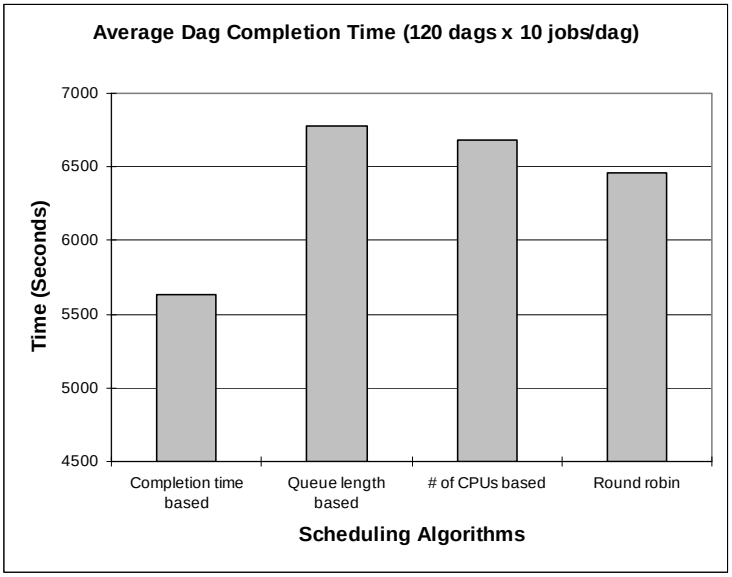


A

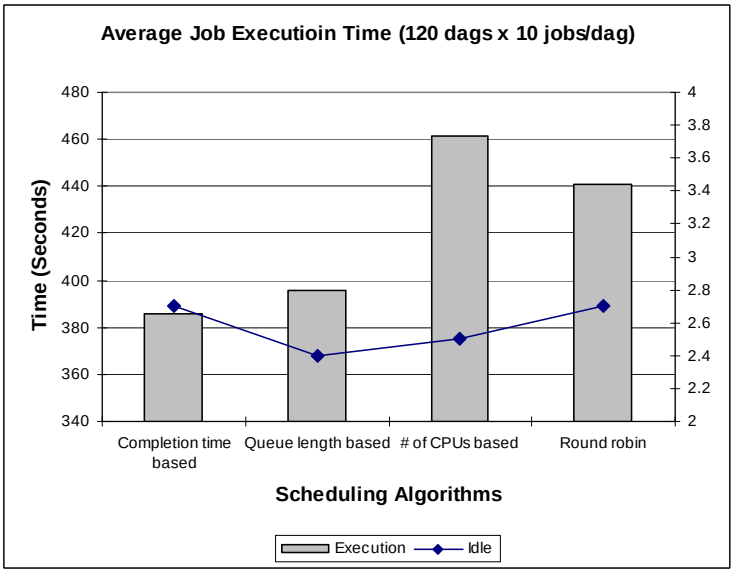


B

Figure 3-5. Performance of scheduling algorithms with 600 jobs and without any policy constraints (A) The average DAG completion time for each of the algorithms, and (B) The average job execution time and the idle time for each of the algorithms.



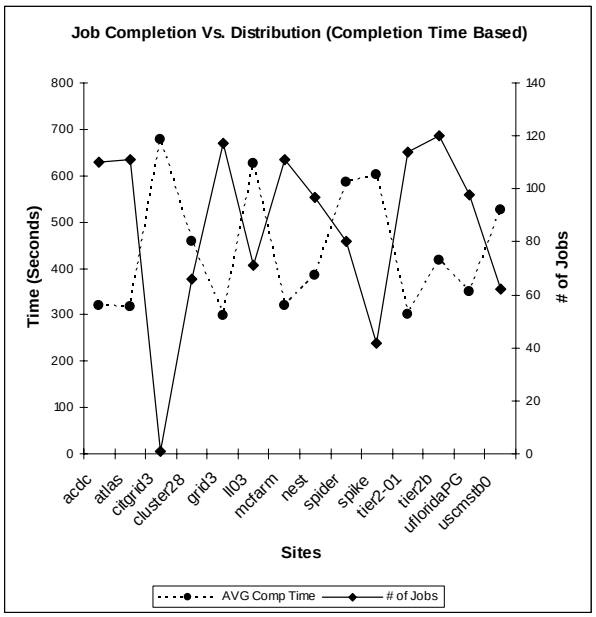
A



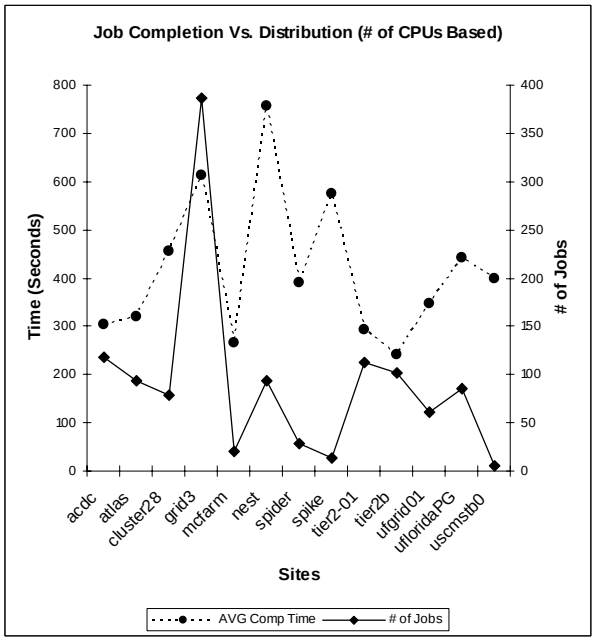
B

Figure 3-6. Performance of scheduling algorithms with 1200 jobs and without any policy constraints (A) & (B). The results follow the trend same as the 300 and 600 jobs experiments, thus exhibiting scalability.

Here it is worth noting that the absolute average DAG completion times for the 2 experiments should not and cannot be compared as the average load on the grid is different during these experiments.



A



B

Figure 3-7. Site-wise distribution of completed jobs vs. avg. job completion time (A) & (B). In the Job completion time based approach (a), the number of jobs scheduled on a site is inversely proportional to its average job completion time.

Figure 3-7(A) verifies that the *job completion rate based approach* indeed scheduled more jobs are to sites having least average-job-completion-time and vice-

versa. Other algorithms do not follow the trend e.g. *number of CPU based algorithm* listed here in Figure 3-7(B).

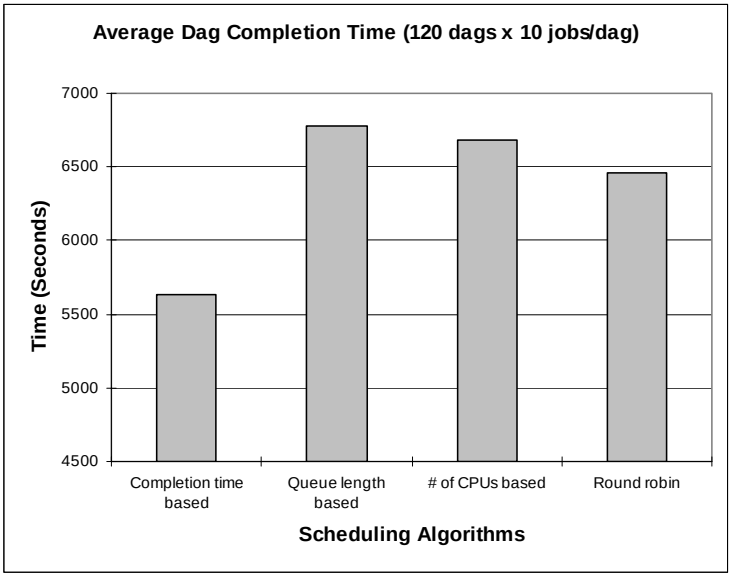
The result shows that simple workload information such as the number of CPUs and running jobs (as used in Round-Robin and simple Load-Balancing techniques) is not good enough to estimate the performance of dynamic and customized grid resources. The job completion rate approach keeps track of the job completion time, and utilizes the information to estimate the near future execution environment on the grid sites. This seems to be a much better predictor of actual performance on the different sites.

As monitoring systems mature and if the local sites make their performance measures more transparent and accurate the effectiveness of monitoring information in effective scheduling may improve. However, the data provided by extant monitoring systems and sites does not seem to be very useful.

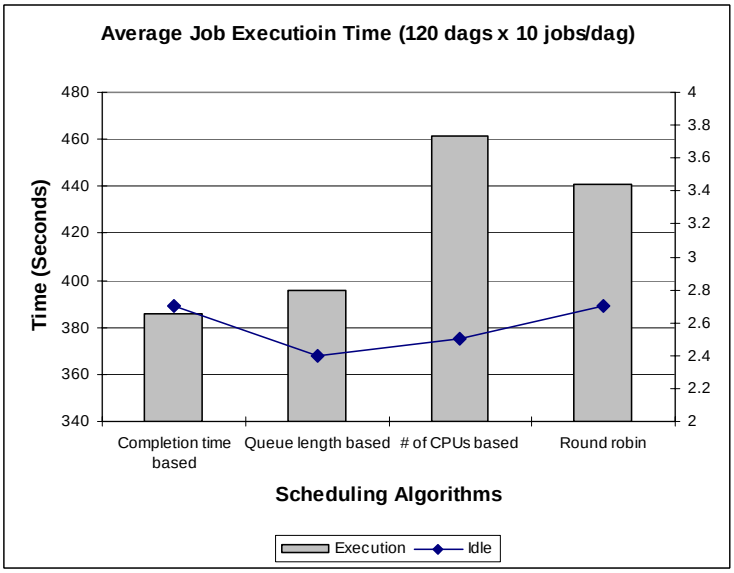
Effects of Policy Constraints on the Scheduling Algorithms

Figure 3.8 (A) and (B) show the performance of scheduling algorithms constrained by resource usage quota policy. A user's remaining usage quota defines the list of sites available to him for submitting jobs from which the scheduling algorithm recommends the execution site.

The results obtained are similar to those without policy. The results underline the ability of SPHINX to do policy-based scheduling. Even in the presence of policy constraints, SPHINX is able to get a scheduling efficiency similar to the one in a constraint-free grid environment.



A



B

Figure 3-8. Performance of policy based-scheduling algorithm (A) Average DAG completion time, and (B) Average Job Execution and Idle time. In each of the scheduling algorithms, policy constraints are applied to get the pool of feasible sites before using the scheduling strategy.

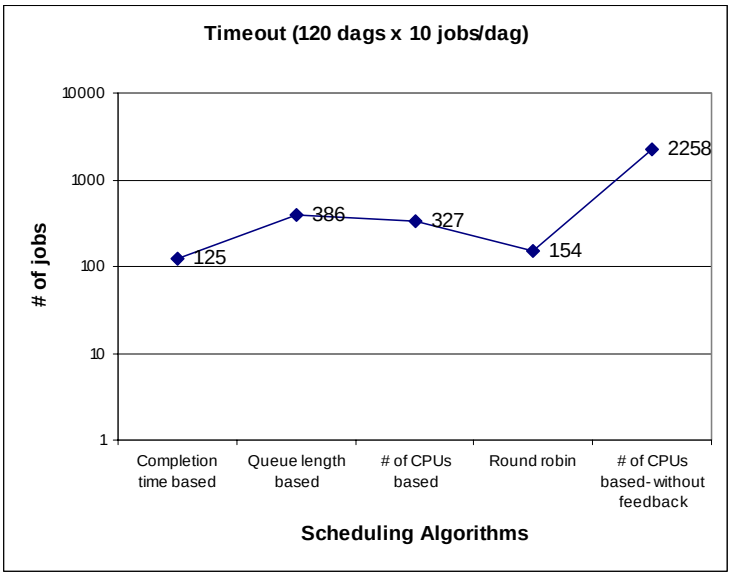


Figure 3-9. Number of timeouts in the different algorithms

Fault Tolerance and Scheduling Latency

Figure 3-9 gives the number of times jobs were rescheduled in each of the scheduling strategies. Note that without any feedback information, the number of resubmissions is very high (2258) as compared to 125 in the job completion rate based hybrid approach.

The graph in Figure 3-10 (A) presents scheduling latency of SPHINX scheduling system with different workload of submitted workflows at the same time. Scheduling latency is defined as the taken time from the job submission until the scheduling decision is made and the job is submitted to the targeted resource. Each line presents the different number of workflows which are submitted concurrently. I differentiate the job arrival rate per minute. SPHINX shows pretty stable performance until the workload of 13 jobs / min with different number of DAGs which are submitted concurrently.

The graph in Figure 3-10 (B) shows the scheduling latency of the cluster-based scheduling algorithm. The cluster size is set to three. It means that any three independently schedulable jobs in a workflow will be planned together in a single scheduling iteration. The scheduling algorithm performs multiple iterations of the job planning to determine the optimal resource allocation. The workflow arriving rate to a scheduling system is determined by the number of submitted workflows per minute. The workload is decided by the number of jobs of a workflow. The total number of submitted DAGs is 100. The algorithm shows good tolerance to the arriving rate up to nine workflows per minute with the different workloads such as 14, 12, 10 or 8 jobs.

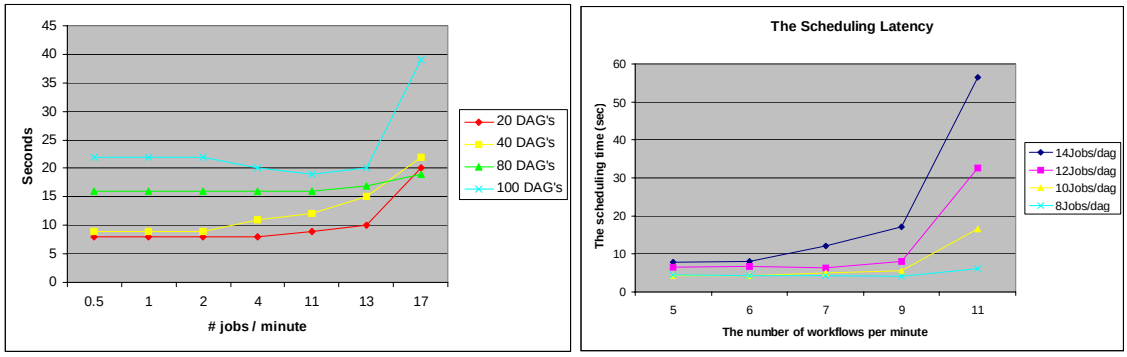


Figure 3-10. Sphinx scheduling latency: average scheduling latency for various number of DAG's (20, 40, 80 and 100) with different arrival rate per minute

Conclusion and Future Research

This chapter introduces techniques and infrastructure for fault-tolerant scheduling of jobs across the grid in a dynamic environment. In addition to the SPHINX architecture which is robust, recoverable, modular, re-configurable and fault-tolerant, the novel contributions of this chapter to the state-of-the-art is the effective use of monitored information in efficient scheduling without requirement of human interference in a highly dynamic grid environment. These results show that SPHINX can effectively

4. Reschedule jobs if one or more of the sites stops responding due to system downtime or slow response time.
5. Improve total execution time of an application using information available from monitoring systems as well its own monitoring of job completion times.
6. Manage policy constraints that limit the use of resources.
7. These results demonstrate the effectiveness of SPHINX in overcoming the highly dynamic nature of the grid and complex policy issues to harness grid resources, an important requirement for executing large production jobs on the grid.

We are investigating novel scheduling methods to reduce the turn around time. We are also developing methods to schedule jobs with variable Quality of Service requirements.

Latest updates on SPHINX are available at http://www.griphyn.org/sphinx

A novel grid scheduling framework for computing has been proposed in this chapter and an initial implementation presented. Resource scheduling is a critical issue in executing large-scale data intensive applications in a grid. Due to the characteristics of grid resources, we believe that traditional scheduling algorithms are not suitable for grid computing. This document outlines several important characteristics of a grid scheduling framework including execution time estimation, dynamic workflow planning, enforcement of policy and QoS requirements, VO-wide optimization of throughput, and a fully distributed, fault tolerant system.

Our proposed system, SPHINX, currently implements many of the characteristics outlined above and provides distinct functionalities, such as dynamic workflow planning and just-in-time scheduling in a grid environment. It can leverage existing monitoring and execution management systems. In addition, the highly customizable client-server framework can easily accommodate user specific functionality or integrate other scheduling algorithms, enhancing the resulting system. This is due to a flexible architecture that allows for the concurrent development of modules that can effectively

manipulate a common representation for the application workflows. The workflows are stored persistently in database using this representation allowing for development of a variety of reporting abilities for effective grid administration.

The development of SPHINX is still in progress, and we plan to include several additional core functionalities for grid scheduling. One such important functionality is that of estimating the resources required for task execution, enabling SPHINX to realistically allocate resources to a task. In addition, we plan to investigate scheduling and replication strategies that consider policy constraints and quality of service, and include them in SPHINX to improve scheduling accuracy and performance.

CHAPTER 4
POLICY-BASED SCHEDULING TECHNIQUES FOR WORKFLOWS

This chapter discusses policy-based scheduling techniques on heterogeneous resources for grid computing. The proposed scheduling algorithm has the following features, which can be utilized in grid computing environments. First, the algorithm supports the resource usage constrained scheduling. Second, the algorithm performs the optimization-based scheduling. It provides an optimal solution to the grid resource allocation problem. Third, the algorithm assumes that a set of resources is distributed geographically and is heterogeneous in nature. Fourth, the scheduling algorithm dynamically adjusts to the grid status by tracking the current workload of the resources. The performance of the proposed algorithm is evaluated with a set of predefined metrics. In addition to showing the simulation results for the out-performance of the policy-based scheduling, a set of experiment is performed on Open Science Grid (OSG).

Motivation.

Grid computing is recognized as one of the most powerful vehicles for high performance computing for data-intensive scientific applications. It has the following unique characteristics over the traditional parallel and distributed computing. First, grid resources are geographically distributed and heterogeneous in nature. Research and development organizations, distributed nationwide or worldwide, participate in one or more virtual organizations (VO's). A VO is a group of resource consumers and providers united in their secure use of distributed high-end computational resources towards a common goal. Second, these grid resources have decentralized ownership and different

local scheduling policies dependent on their VO. Third, the dynamic load and availability of the resources require mechanisms for discovering and characterizing their status continually.

The dynamic and heterogeneous nature of the grid coupled with complex resource usage policy issues poses interesting challenges for harnessing the resources in an efficient manner. In this paper, we present novel policy-based scheduling techniques and their performance on Open Science Grid (OSG), a worldwide consortium of university resources consisting of 2000+ CPUs. The execution and simulation results show that the proposed algorithm can effectively:

8. Allocate grid resources to a set of applications under the constraints presented with resource usage policies.
9. Perform optimized scheduling on heterogeneous resources using an iterative approach and binary integer programming (BIP).
10. Improve the completion time of workflows in integration with job execution tracking modules of SPHINX scheduling middleware.

Problem Definition and Related Works

An application scientist typically solves his problem as a series of transformations. Each transformation may require one or more inputs and may generate one or more outputs. The inputs and outputs are predominantly files. The sequence of transformations required to solve a problem can be effectively modeled as a Directed Acyclic Graph (DAG) for many practical applications of interest that the proposal is targeting.

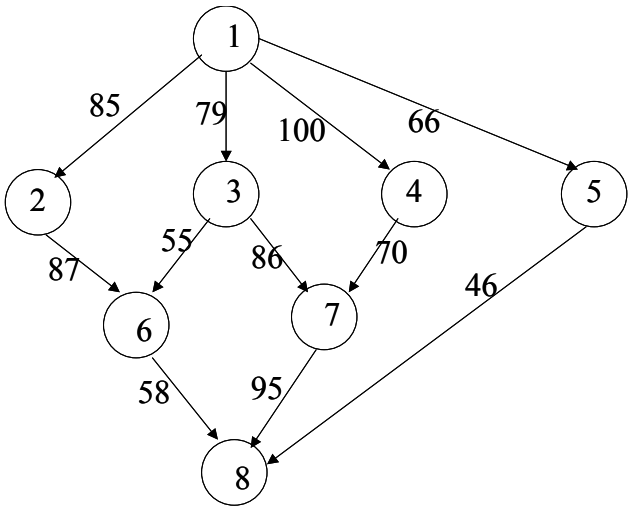


Figure 4-1. An example workflow in Directed Acyclic Graph (DAG). The figure shows a workflow consisting with eight jobs. The number of an edge represents a communication time. For simplicity we assume that the communication time is identical on different network. (Adapted from [54])

Figure 4-1 describes a DAG consisting of 8 tasks. It is useful to define an exit task – the completion of this task implies that the workflow is executed. Task 8 represents the exit task. A scheduling algorithm aims to minimize a workflow completion time obtained by the assignment of the tasks of the DAG to processors. A scheduling algorithm that is efficient and suitable in the target environment should be able to exploit the inherent heterogeneity in the processor and network resources. Most of the existing scheduling algorithms perform the mapping of tasks to processors in two stages:

Create a priority based ordering of the tasks. The priority of a task is based on its impact on total completion time. This requires determining the critical path, which in turn requires that the execution time of each task is available or can be estimated. The exact definition of critical path depends on the algorithm. Some algorithms use the longest path from the given task to calculate the critical path of a task. Others use the longest path from the start node to the given task to calculate its critical path.

Use the priority based ordering created in the previous step to map the tasks so that the total completion time is minimized. The process is performed one task at a time in an order based on the priority. It is incremental in nature. However, once a task is assigned to a given processor, it is not generally remapped.

The above approach has the following limitations for the target heterogeneous environment:

8. The amount of time required for a task is variable across different processors (due to heterogeneity). Thus estimating the priority based on cost of the critical path is difficult (this measure or related measures are used by most of the algorithms in determining a task's priority). Adaptations of these algorithms for heterogeneous processors use the average or median processing time of subsequent tasks to estimate the critical path. However, this may not be an accurate reflection of the actual execution task. In fact one processor may execute task A faster than Task B, while the reverse may be true for another processor – this may be due to differential amounts of memory, cache sizes, processor types, etc.

9. The tasks are assigned one at a time. Assuming that k processors are available at a given stage, this may not result in an optimal assignment. Clearly, one can call these algorithms k times sequentially to achieve the same goal. However, this may not result in the optimal allocation of the tasks on the k available processors. Mapping a large number of tasks simultaneously on available processors can allow for more efficient matching.

10. The tasks are assigned without any policy constraints. The policy constraints can restrict the subset of processors that can be assigned to a given task. This needs to be taken into account while making the scheduling decisions.

Past research on task scheduling in DAGs has mainly focused on algorithms for a homogeneous environment. Scheduling algorithms such as Dynamic Critical Path (DCP) algorithm [55] that show good performance in a homogeneous environment may not be efficient for a heterogeneous environment because the computation time of a task may be dependent on the processor to which the task is mapped. Several scheduling algorithms for a heterogeneous environment have been recently proposed. Most of them are based on static list scheduling heuristics to minimize the execution time of DAGs. Examples of

these algorithms include Dynamic Level Scheduling (DLS) algorithm [58] and Heterogeneous Earliest Finish Time (HEFT) algorithm [57].

The DLS algorithm selects a task to schedule and a processor where the task will be executed at each step. It has two features that can have an adverse impact on its performance. First, it uses the median of processing time across all the processors for a given task to determine a critical task. Secondly, it uses the earliest start time to select a processor for a task to be scheduled. These may not be effective for a heterogeneous environment. For instance, suppose that processor A and processor B are the only available processors for the assignment of task i. Assume that processor A becomes free slightly earlier than processor B (based on the mapping of tasks so far). Then task i is assigned to processor A, since processor A can execute it earlier than processor B. However, if processor B can finish task i earlier than processor A, the selection of processor B for the task should result in a better mapping. Also, the time required to execute the scheduling algorithm (cost of scheduling) is relatively high as the priorities of the remaining tasks need to be recalculated at each step of the iteration. The number of iterations is proportional to the total number of tasks

The HEFT algorithm reduces the cost of scheduling by using pre-calculated priorities of tasks in scheduling. The priority of each task is computed using an upward rank, which is also used in our proposed algorithm. Also, it employs finding the earliest finish time for the selection of a processor, which is shown to be more suitable for a heterogeneous environment. Although it has been shown to have good performance in the experiments presented in [57], it can be improved by using better estimations of the critical path. The Iterative list scheduling [54] improves the quality of the schedule in an

iterative manner using results from previous iterations. It only assigns one task at a time and does not support resource usage policies. It is a static scheduling algorithm, which assumes an unchanged or stable computing environment. In the dynamic and policy constrained grid computing environment the algorithm may not perform well – this is supported by the simulation results presented in this paper.

We address the above issues in the proposed algorithm and demonstrate that it is effective in satisfying the workflow completion deadline. The proposed algorithm consists of three main steps:

- Selection of tasks (or a task)
- Selection of processors for the selected tasks
- Assignment of selected tasks to selected processors based on policy constraints.

A subset of independent tasks with similar priority is selected for simultaneous scheduling. The tasks in the selected subset are scheduled optimally (i.e., to minimize completion time) based on policy constraints. Further, the scheduling derived is iteratively refined using the mapping defined in the previous iteration to determine the cost of the critical path. This estimation, in general, should be better than using the average computation time on any processor. We utilize the resource usage reservation technique to schedule multiple DAG workflows. The technique facilitates the deadline satisfaction of workflow completion.

When the scheduling algorithm is integrated with the SPHINX scheduling middleware it performs efficient scheduling on the policy-constrained grid environment. The performance is demonstrated in the experimental section.

Scheduling Algorithm Features

The proposed policy-based scheduling algorithm is different from the existing works in the following perspectives.

Policy constrained scheduling: Decentralized grid resource ownership restricts the resource usage of a workflow. The algorithm makes scheduling decisions based on resource usage constraints in a grid-computing environment.

Optimized resource assignment: The proposed algorithm makes an optimal scheduling decision utilizing the Binary Integer Programming (BIP) model. The BIP approach solves the scheduling problem to provide the best resource allocation to a set of workflows subject to constraints such as resource usage.

The scheduling on heterogeneous resources: The algorithm uses a novel mechanism to handle different computation times of a job on various resources. The algorithm iteratively modifies resource allocation decisions for better scheduling based on the different computation times instead of taking a mean value of the time. This approach has also been applied to the Iterative list scheduling [54].

Dynamic scheduling: In order to encounter a dynamically changing grid environment the algorithm uses a dynamic scheduling scheme rather than a static scheduling approach. A scheduling module makes the resource allocation decision for a set of schedulable jobs. The status of a job is defined as schedulable when it satisfies the following two conditions.

Precedence constraint: all the precedent jobs are finished, and the input data of the job is available locally or remotely.

Scheduling priority constraint: A job is considered to have higher priority than others when the job is critical to complete the whole workflow for a better completion time.

Future scheduling: The resource allocation to a schedulable job impacts the workload on the selected resource. It also affects the scheduling decision of future schedulable jobs. The algorithm pre-schedules all the unready jobs to detect the impact of the current decision on the total workflow completion time.

When the scheduling algorithm is integrated with the SPHINX scheduling middleware it performs efficient scheduling on the policy-constrained grid environment. The performance is demonstrated in the experimental section.

Notation and Variable Definition

In this section we define the notations and variables that are used in the proposed scheduling algorithm.

$comp_{ij}$: Computation time of job i on processor j ---(1)

$comm_{pj}$: Communication time from processor p to j ---(2)

The computation or execution time of a job on a processor ($comp_{ij}$) is not identical among a set of processors in a heterogeneous resource environment. The algorithm sets an initial execution time of a job with the mean value of the different times on a set of available processors. The time is updated with an execution time on a specific processor as the algorithm changes the scheduling decision based on the total workflow completion time. The data transfer or communication time between any two processors ($comm_{pj}$) is also different in the environment.

$prec_i$: A set of the preceding jobs of job i ---(3)

$succ_i$: A set of the succeeding jobs of job i ---(4)

An application or workflow is in the format of directed acyclic graph (DAG). Each job *i* has a set of preceding ($prec_i$) and succeeding ($comm_i$) jobs in a DAG. The dependency is represented by the input/output file relationship.

$Avail_{ij}$: The available time of processor j for job i ---(5)

EST_{ij} : Earliest Start Time of job i on processor j ---(6)

$$EST_{ij} = Max\{Avail_{ij}, Max_{k \in prec_i}\{EFT_{kp} + comm_{pj}\}\}$$

EFT_{ij} : Earliest Finish Time of job i on processor j ---(7)

$$EFT_{ij} = EST_{ij} + comp_{ij}$$

$compLen_i$: Workflow completion length from job i ---(8)

$$compLen_i = comp_{ip_i} + Max_{k \in succ_i}(comm_{p_i p_k} + compLen_k)$$

The algorithm keeps track of the availability of a processor ($Avail_{ij}$) to execute a job. We assume a processing model which all the jobs in a processor queue should be completed before a new job gets started. We assume a non-preemptive model. On the grid scheduling middleware, SPHINX obtains the information from a grid monitoring system such as MonALISA or GEMS. The algorithm computes the earliest start time of a job on each processor (EST_{ij}). A job can start its execution on a processor only after satisfying two conditions; first, a processor should be available ($Avail_{ij}$) to execute the job. Second, all the preceding jobs should be completed ($Max_{k \in prec_i}\{EFT_{kp} + comm_{pj}\}$) on the same processor or other processors. The earliest finish time of a job on a processor (EFT_{ij}) is defined by the earliest start time (EST_{ij}) and the job completion or execution time on the processor ($comp_{ij}$). The workflow completion time from a job to the end of a DAG ($compLen_i$) is defined recursively from the bottom to the job *i*. The value is used to decide the critical path in the workflow. The makespan of the critical path is used as a criterion to terminate the algorithm. The algorithm completes the scheduling when there is no improvement in the DAG completion time.

$Deadline_{d(i)}$: The workflow completion deadline of dag d to which job i belongs
$estCompTime_{d(i)}$: The estimated workflow completion time of dag d to which job i belongs

We also define the deadline and the estimated completion time of a workflow. They are used when we devise a profit function for the multiple workflow scheduling. The scheduling algorithm assumes that the workflow deadline ($Deadline_{d(i)}$) is submitted by a user, or is generated by a scheduling system. The scheduling algorithm computes the estimated workflow completion time ($estCompTime_{d(i)}$) with considering the current workload on Grid resources. The time is changed in the iterations of the scheduling due to the dynamic Grid resource status.

ExecTime								
	J1	J2	J3	J4	J5	J6	J7	J8
P1	70	68	78	89	30	66	25	94
P2	84	49	96	26	88	86	21	36
Policy								
I	J1	J2	J3	J4	J5	J6	J7	J8
P1	v	v	v	v		v		v
P2	v		v		v	v	v	v
Job	J1	J2	J3	J4	J5	J6	J7	J8
AvgExecTime	77	68	87	89	88	76	21	65
compLen	517	354	354	340	199	199	181	65
Prioritization	J1	J3	J2	J4	J6	J5	J7	J8
Assignment	P1	P2	P1	P1	P1	P2	P2	P1

Figure 4-2. An example for job prioritization and processor assignment for the workflow in Figure 1. The example presents a procedure to prioritize a set of jobs on the workflow from Figure 1 based on the scheduling function (P_{ij}). It also shows the processor assignment to the jobs based on the earliest finish time of a job on a processor (EFT_{ij}).

Figure 4-2 shows an example of the procedure to assign jobs in a workflow to a set of processors (P1 and P2) based on the workflow prioritization techniques described in this section. It presents the heterogeneous execution time of the jobs on each of the

processors (ExecTime). Each job such as J1, J2 or J8 has different execution time on the processors, P1 and P2. The figure also shows the resource usage restriction of the jobs. The mark (v) indicates that a processor allows a job to be executed on a corresponding processor. Otherwise, a job can't be run on a processor. In the first iteration of the resource allocation procedure the algorithm uses the average execution time (AvgExecTime) from the different time on the processors. In the next iterations the algorithm uses a specific execution time on a selected processor in the previous iteration. The algorithm computes the critical path length from each job to the bottom job of the workflow. The critical path length is calculated on the base of the predefined calculation for $compLen_i$ on (8). The values of the critical path length are used to prioritize the set of jobs in non-descending order (Prioritization). After sorting the jobs, the algorithm allocates a set of available processors to the job subject to the resource usage constraints. The assignment is determined with an optimization model that is discussed in the next sections.

Optimization Model

We devise a Binary Integer Programming (BIP) model to find an optimal solution to the scheduling problem on the proposed algorithm. In this section, we first define two scheduling profit functions for single workflow scheduling and multiple workflow scheduling respectively. Next, we discuss the optimization model utilizing the profit functions for the scheduling problem.

Profit Function for Single Workflow Scheduling

$Scheduling\ profit(p_{ij})$: the profit when job *i* is assigned to processor *j*

$$p_{ij} = \frac{compLen_i}{EFT_{ij}} \quad \text{---(9)}$$

where

$$EFT_{ij} > 0$$

The scheduling profit function for the single workflow scheduling intends to generate higher profit value when a job on the critical path of a DAG is scheduled to a processor which is able to complete the job in an earliest possible time. $CompLen_i$ is calculated by computing the critical path value from the job to the bottom job of the DAG. We can assume that a job with higher critical path value is more urgent to complete the DAG as soon as possible. For a job *i,* the function tries to find a processor with the smallest completion time of the job. As described above, the earliest completion time of a job on a processor is determined by computing the processor available time and estimated execution time of the job. The objective function of the optimization model utilizes the profit function to select a proper processor for a job in the schedulable job list.

Profit Function for Multiple Workflow Scheduling

$$p(i,j) = \frac{compLen_i}{EFT_{ij}} \times \frac{1}{deadline_{d(i)} - estCompTime_{d(i)}}$$

where

$$if\ deadline_{d(i)} - estCompTime_{d(i)} \le 0\ \ then$$

$$deadline_{d(i)} - estCompTime_{d(i)} = 1$$

$$EFT_{ij} > 0$$

The profit function, *p(i,j)* is defined for the multiple workflow scheduling. Given a job *i* and a processor *j,* the function *p* computes a scheduling profit of the assignment of the job onto the processor. A scheduling system assumes that workflow *d* is submitted

with a predefined deadline ($deadline_{d(i)}$) where job *j* belongs to workflow *d*. The deadlines of the given multiple workflows may be different. The profit function is used to prioritize a set of independent jobs in terms of the critical path value of the job *i* in a workflow *d* ($compLen_i/EFT_{ij}$) and the remaining time to the deadline ($deadline_{d(i)}$-$estCompTime_{d(i)}$). The scheduling algorithm gives higher priority to the workflow which is urgent to meet its deadline. The urgency is defined with the value of the remaining time. Due to the difference in the deadlines of the multiple workflows some jobs in a workflow may be more urgent to meet its deadline than other jobs in another workflow. The profit function maximizes the profit value by assigning higher profit to the job that is more critical to meet the workflow deadline.

Objective Function and Constraints

The scheduling algorithm determines the resource allocation to a set of independent jobs in the Binary Integer Programming (BIP) optimization model. It tries to assign the jobs to a set of processors, which will provide the jobs with the earliest finish time subject to the current resource load and the assignment constraints. The order of assignment to the jobs is decided by the urgency to meet the workflow deadline and the criticality to finish the workflow in short required time utilizing the profit functions and the job prioritization procedure of the algorithm. The assignment is limited by a set of constraints such as the resource usage constraint. The BIP optimization model is devised in the following format. The objective function ($Max\sum_i\sum_j p_{ij}\cdot x_{ij}$) is to maximize the profit function values for a set of jobs and processors. The profit functions for the single or multiple workflow scheduling are defined in the previous section. The objective function is constrained by the several constraints

$$Max \sum_i \sum_j p_{ij} \cdot x_{ij}$$

st.

$b_{ij} x_{ij} \le q_{ij}$ for each job i and processor j (Policy)

$\sum_j x_{ij} = 1$ for each job i (Assignment)

$\sum_i x_{ij} \le t_j$ for each processor j (Load)

$x_{ij} = 0 \; or \; 1$ (Binary)

where

b_{ij} : resource usage requirement of job i on processor j

q_{ij} : resource usage quota of job i on processor j

t_j : the limit of assigned jobs on processor j

The resource usage constraint ($b_{ij} x_{ij} \le q_{ij}$) is described with the two values, quota (q_{ij}) and requirement (b_{ij}) for a job i and a processor j. The quantitative model is flexible to express the policy of various resource types. The assignment constraint ($\sum_j x_{ij} = 1$) makes sure that a job is not divided or assigned onto more than two processors. Each processor shouldn't be loaded with a set of assigned jobs over the predefined quota (t_j) ($\sum_i x_{ij} \le t_j$). Currently the load is defined with the number of assigned jobs. The BIP model ($x_{ij} = 0 \; or \; 1$) is implemented with available BIP solvers. The optimization guarantees the optimal processor assignment to a set of jobs for the minimum resource requirement time.

Policy-based Scheduling Algorithm and SPHINX

In this section we discuss the policy-based scheduling algorithm in detail. This section also discusses the integration of the algorithm onto the grid scheduling middleware, SPHINX to run test applications on Open Science Grid.

Iterative Policy-based Scheduling Algorithm

The proposed scheduling algorithm uses the iterative approach to improve a resource allocation decision on heterogeneous Grid resource environment. It also makes an optimal scheduling decision by solving the scheduling problem modeled in Binary Integer Programming (BIP). Using the BIP model a set of independent jobs are scheduled at the same time, so that the scheduling is able to be optimal under the resource usage constraint environment. The algorithm uses the mean value approach to make an initial scheduling decision. In other words, the scheduling decision in the first scheduling iteration is made by the mean value of the job execution time on each processor. The scheduling algorithm then modifies the initial scheduling in an iterative way. In each iteration the execution time of a job is changed with a specific value on a selected processor in the previous iteration as the iterative scheduling proceeds. The iteration is terminated when there is no improvement in DAG completion time. The algorithm (Figure 4-3) is presented with detailed description below.

A workflow formatted in a DAG consists of several jobs. The algorithm generates a list with unscheduled jobs in the DAG on time *t* (1). Upon the time when the workflow is submitted the list consists of all the jobs on the workflow. While a subset of the jobs is completed the size of the list is decreased with a smaller number of jobs than the number of jobs in the initial scheduling time. The algorithm initially sets the job execution time with the mean value on heterogeneous resources (2). As the execution time on the processors is different for each job the mean value of the execution time may be different for each job in the list. In the iterative scheduling approach the best DAG completion time (*BestSL*) in a series of iterations is selected, and the scheduling decision on the corresponding iteration is applied to the resource allocation (3, 4).

Construct a job list, $L_{unscheduled}$ with unscheduled jobs in a workflow --- (1)
for each job i in $L_{unscheduled}$
    Set $exec_i$ with a mean value of the execution time on processors --- (2)
end for
Initialize *BestSL* and *SL* with a very large number --- (3)
While $SL \leq BestSL$ do --- (4)
    $BestSL = SL$
    for each job i in $L_{unscheduled}$
        Compute $compLen_i$ --- (5)
    end for
    Sort $L_{unscheduled}$ by the non-increasing order of $compLen_i$ --- (6)
    Dynamically cluster $L_{unscheduled}$ on the base of job dependency --- (7)
    for each sub list, $L_{schedulable}$
        for each job i in $L_{schedulable}$
            for each processor j in the processor list, $L_{processors}$
                Compute EFT_{ij} --- (8)
            end for
        end for
        Call $optSolver(L_{schedulable}, L_{processors,} compLen_i, EFT_{ij})$ --- (9)
        Assign each job i in $L_{schedulable}$ onto a selected processor --- (10)
        Update each processor workload and available time --- (11)
    end for
    Set *SL* with EFT_{ep}, where EFT_{ep} is *EFT* of an exit job e on processor p_e --- (12)
    for each job i in $L_{unscheduled}$
        Set $exec_i$ with the execution time on p_i --- (13)
    end for
end while

Figure 4-3. The iteration policy-based scheduling algorithm on heterogeneous resources. The algorithm utilizes an iterative scheduling scheme to deal with heterogeneous resources. It also performs an optimized scheduling by solving the policy-based scheduling problem that is modeled in Binary Integer Programming (BIP).

The scheduling length (*SL*) in each iteration represents the DAG completion time with the resource allocation decision made in the scheduling iteration, and is compared with the best scheduling length (*BestSL*) computed so far. The critical path length of each job in the list ($compLen_i$) is computed in (5) as it is defined in the previous section. Based on the values of the critical path length the jobs in the list are sorted in the non-decreasing order. The order of the jobs in the sorted list represents the criticality of the jobs to complete the workflow (6). After sorting the jobs the list is clustered with a set of subsets (7). The size of each subset should be dynamic depending on the preceding relationship among the jobs in the list. Given the cluster size k (>= 1) the clustering is performs based on the given size k and the preceding relationship. In other words, the size of a cluster is less than or equal to k, and the jobs in a cluster should be independent. The algorithm formulates the optimization-scheduling problem with the subset of jobs. Before calling the optimization solver function the algorithm computes the earliest completion time (EFT_{ij}) of each job on each processor (8) as the time is defined in the previous section. The optimization function is called with the parameters, a scheduling cluster, a set of processors, $compLen_i$ and EFT_{ij} (9). The optimization function is defined with the BIP model which is described in the previous section. The function returns a selected processor for each job in the cluster. The jobs are assigned to the processors (10). After the assignment the workload on each processor should be updated with the assigned job information such as the execution finish time of the assigned jobs (11). This information is used in the next scheduling iteration, and affects the assignment decision in the future scheduling. For the workload information the algorithm maintains the number of jobs and the next available time on each processor. The next available time of

a processor defines the time when the processor finishes the currently running job, and is ready for executing the next job.

In a scheduling iteration the algorithm computes workflow completion time resulting from the processor allocation decision to the schedulable job list (12). The completion time is defined with the longest job completion time in the workflow. In other words, it is defined with the earliest finish time of the exit job in the workflow. The algorithm updates the job execution time with the time on a selected processor in this iteration (13). In the next iteration the execution time of a job is deal with the time set in this step.

Scheduling Algorithm on SPHINX

We implement the proposed scheduling algorithm, and integrate it into the grid scheduling middleware, SPHINX. SPHINX performs dynamic scheduling with respect to the frequently changed load and availability of the grid resources. The scheduling system is integrated with a grid resource monitoring system such as MonALISA or GEMS to maintain grid resource status and workload information. The SPHINX job monitoring module keeps track of the job execution status on the sites.

After making the scheduling decisions about a workflow based on the proposed scheduling algorithm, SPHINX makes the practical resource allocation only to a set of schedulable jobs in the workflow. A schedulable job has the entire parent jobs completed and the inputs available in a grid network. The job also has high priority to complete the workflow in the sorted job list in the algorithm. SPHINX also uses the processor status monitoring information to organize a set of available processors. The scheduling algorithm uses the set of processors to allocate resources to the jobs.

Experiment and Simulation Results

The performance of the proposed policy-based scheduling algorithm is evaluated with the simulation and the execution on the Open Science Grid (OSG). In this section we discuss the performance of a set of test applications to compare the algorithm with other scheduling algorithms that use the mean value approach to the heterogeneous resource environment.

Network Configuration and Test Application

OSG is a grid-computing infrastructure that supports scientific computing. It consists of more than 25 sites, and collectively provides more than 2000 CPUs. The resources are used by 7 different scientific applications, including 3 high energy physics simulations and 4 data analyses in high-energy physics, biochemistry, astrophysics and astronomy.

In order to compare the performance of the algorithms, we generate a set of test workflows in directed acyclic graph (DAG) format. The workflow simulates a simple application that takes input files, and generates an output file. The size of the output file is different for each job, and the file is located on the execution site by default. The structure of a DAG, such as the depth and width, is set with different values. We set two kinds of parameters for the simulation: system parameters and workflow parameters. The system parameters consist of the number of processors and the resource usage constraints on the processors. For the simulation, we set the number of processors to 20, and the constraints are set with 10%, 50% and 90%. 10% of resource constraint means that the resource usage is limited to 90% of the total resources, while 90% represents that only 10% of the total resources are available to execute a user's jobs.

The workflow parameters of the simulation describe a workflow in DAG format. They consist of the number of jobs, the height of a DAG, the number of jobs in each level, the amount of input data for each job, the communication to computation ratio (CCR) and the request types. The request types are used to represent the workflow types. All the jobs in a DAG may request the same amount of CPU resources, while the jobs may require different job processing time.

We generate a virtual workflow pool with the workflow parameters. Each DAG in the pool has different workflow related parameters representing different workflows.

List Scheduling with the Mean Value Approach

In the experiment the list-scheduling algorithm is compared with the policy-based scheduling algorithm in their performance of DAG completion time. The list scheduling uses the mean value approach to decide the execution time of a job on heterogeneous resources.

The list-scheduling algorithm sorts a set of jobs in a workflow in the non-descending order of the workflow completion time. The workflow completion time is calculated from each of the jobs to the end of the workflow. In order words, the algorithm gives higher scheduling priority to a job, which affects the workflow completion time in a critical manner. The algorithm schedules the jobs in the sorted list one by one, and assigns a job onto the processor on which the job can finish as soon as possible. The resource usage policies constrain the assignments.

Taking the mean value for the job execution time might not reflect the actual job execution on the heterogeneous resources. It results in a non-optimal scheduling for a workflow in the list-scheduling approach. Policy constraints might drive the resource assignment in the list scheduling into an unreasonable scheduling decision. It is mainly

caused by the fact that scheduling the jobs in the sorted list one by one does not allow a chance to consider the policy constraints among multiple jobs. The scheme may result in a long DAG completion time by allocating a job to the wrong processor in terms of the earliest finish time of a job (EFT_{ij}).

The workload-based algorithm considers the number of CPUs on a grid site and the current workload on the site. The workload is determined by maintaining the previous assignment information on a local scheduling system. The workload of a site consists of the number of jobs planned on the site and the number of jobs currently being executed on the site.

$$(num\_planned_i + num\_submitted_i)/num\_CPU_i$$

where

$num\_planned_i$: the number of planned jobs on site i
$num\_submitted_i$: the number of submitted jobs on site i
num_CPU_i: the number of CPUs on site i

The Simulated Performance Evaluation with Single DAG

The first simulation evaluates the performance of the algorithms when the resource usage constraint is different. The constraint is defined with the ratios of the limited number of processors to the total number of processors. The policy constraint limits the job execution only on the available set of processors.

The graphs in Figure 4-4 show the average DAG completion time with different constraints. Each graph shows the completion time when the number of tried DAGs is 20, 200, 500 or 700. As we have mentioned in the previous section two kinds of parameters are defined in the simulation, system-related and workflow-related parameter.

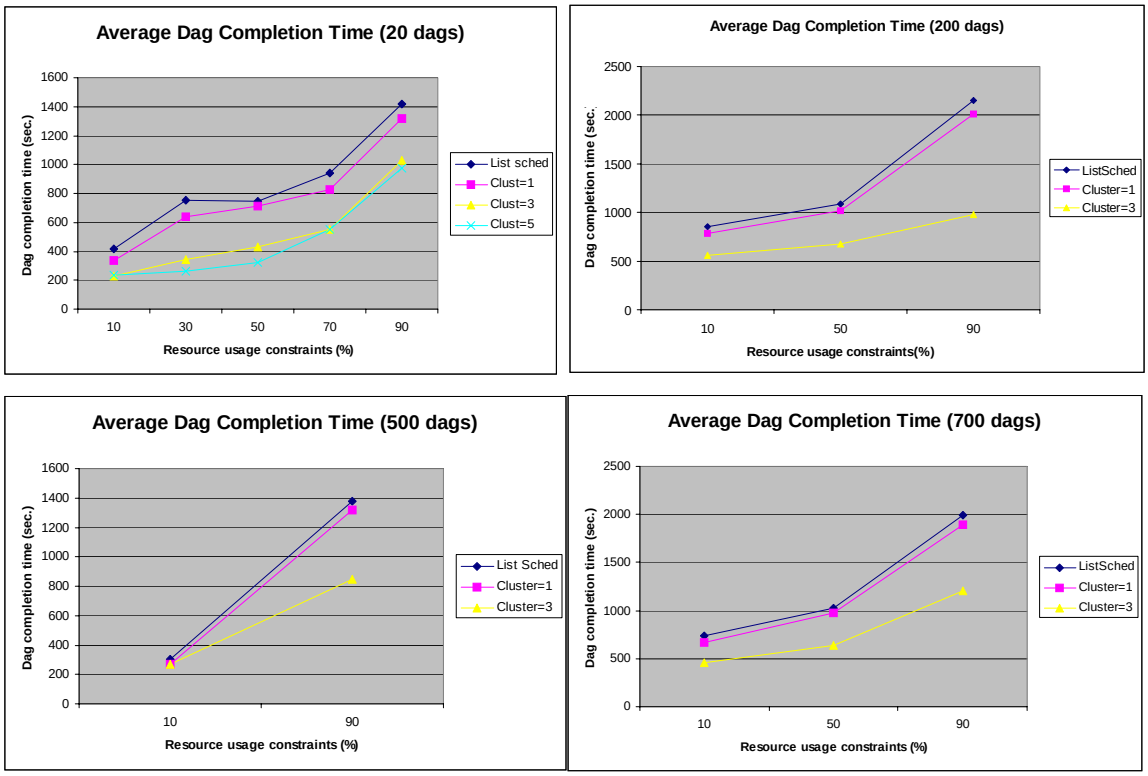


Figure 4-4. The constraint and clustering effect on DAG completion. The graphs show the DAG completion time when the resource usage constraints are different. The constraints are defined by the ratio of the available processor to the total processors. The constraint specifies a set of available processors on which a job is allowed to run.

The system-related parameter includes the number of processors and the resource usage constraint. In the simulation on Figure 4-4 we use 20 processors, and the constraint is changed from 10% to 90% increased by 20 %. The workflow-related parameters includes the number of jobs which is 10, 20, 40 or 80, the number of levels of a DAG which is 2, 4, 8 or 10, the link density which is 10% ~ 100%, the CCR which is 1, 4 or 8.

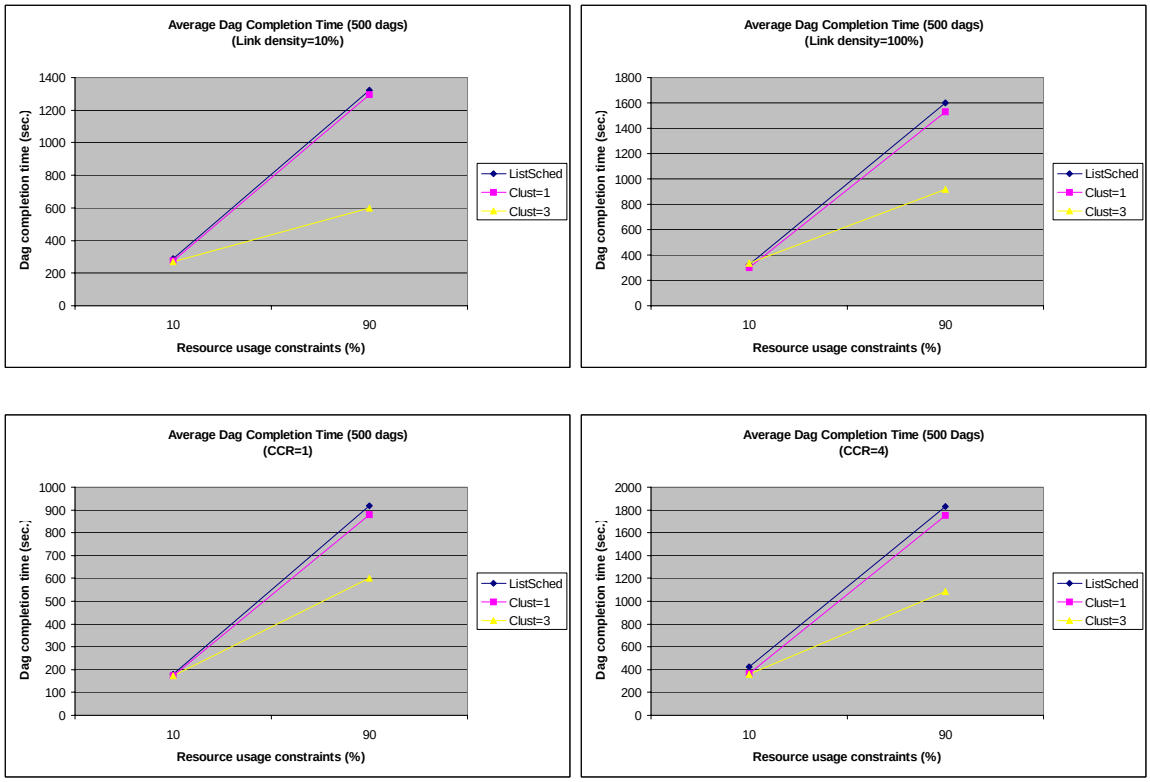


Figure 4-5.Average DAG completion time with 500 DAGs when the two workflow parameters are different. The link density is 10% or 100%. The CCR is 1 or 4. The resource usage constraint is 10% or 90%

From the DAG pool we select the different number of tried DAGs for each of the simulation results. We choose 20, 200, 500 or 700. We believe that the large number of tried DAGs gives more stable results in the average workflow completion time. In the next sets of simulations we focus the simulation with 500 and 700 DAGs because the results are more consistent with a large number of tried DAGs.

For the results shown in Figure 4-5 we simulate the scheduling when two DAG properties, CCR and the link density are controlled with the following specific numbers. The CCR is 1 or 4, and the link density is 10% or 100%. The other workflow and system properties are randomly chosen with the following range.

- The number of jobs per DAG: 10, 20, 40 or 80
- The number of levels of a DAG (height): 1, 2 or 10

- The number of processors: 20
- The resource usage constraints: 10% or 90%

In this simulation the types of workflows are defined with the two features. The first is the communication to computation rate (CCR) and the second is the link density. Based on the CCR the communication time is defined by

Communication time = (Computation time * CCR) / Communication rate

We simulate two different kinds of DAGs in terms of CCR. With a large CCR a DAG is more communication oriented than with a small CCR.

Another parameter to specify a workflow is link density. The link density is defined with the number of input data from the jobs in the parent level to a job in a child level. It determines the relationship between jobs within a DAG. With the link density equal to 20% a job takes input data from 20 % of the jobs in parent levels.

The first two graphs in Figure 4-5 show the performance of the algorithms when the link density between jobs is different. The number of inputs of each job defines the link density. In the experiment the number of outputs is set to one. As the number of inputs increases the DAG completion time is increased. One of the main reasons is that the job takes more time to be ready to run on a processor with multiple inputs than with a small number of inputs.

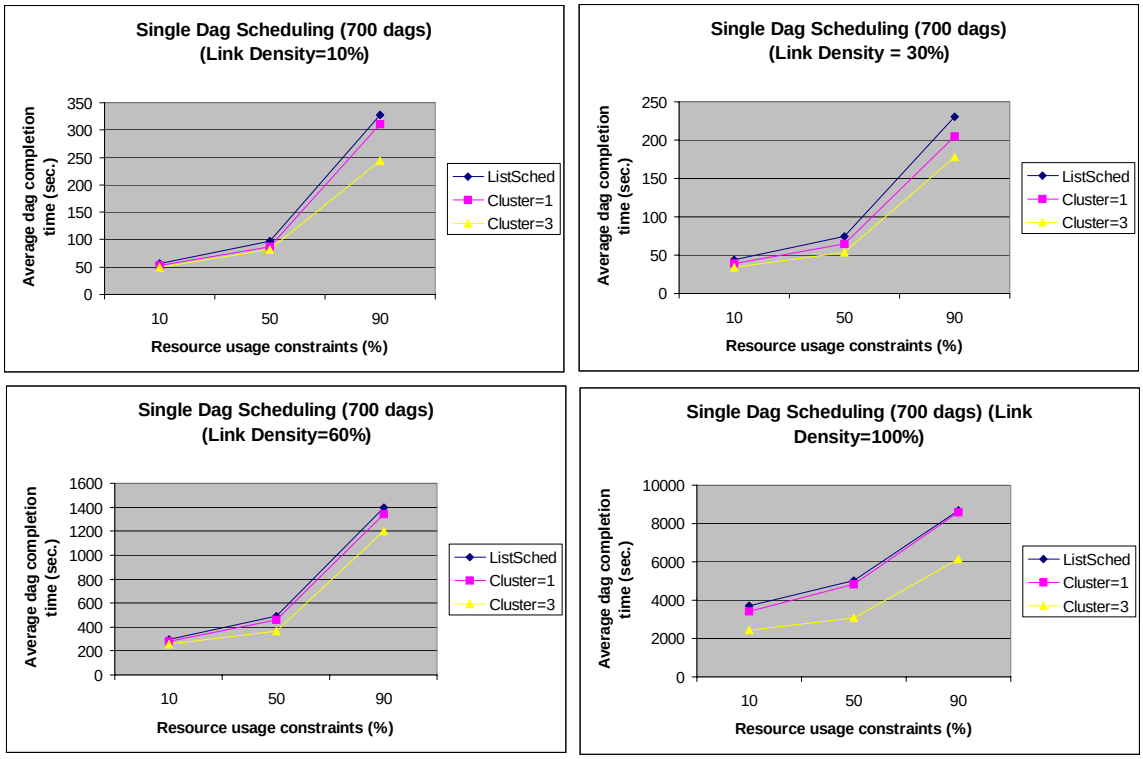


Figure 4-6: Average DAG completion time with the different scheduling algorithms such as the list scheduling and the cluster-based scheduling. The number of tried DAGs is 700. The results show the performance of the algorithms when the list density is changed with 10%, 30%, 60% and 100%.

The second two graphs in Figure 4-5 show the effect of the CCR on the DAG completion time. With communication-oriented workflow with CCR equal to 4 the policy-based scheduling shows good tolerance of the constraints compared to the list scheduling. The DAG completion time with the cluster-based scheduling is more stable than the list scheduling when resource usage constraints change from 10% to 90%. The policy-based scheduling shows consistent performance with various types of jobs. It means that the performance of the scheduling algorithm is stable with the different types of workflows such as the communication-oriented and the computation-oriented.

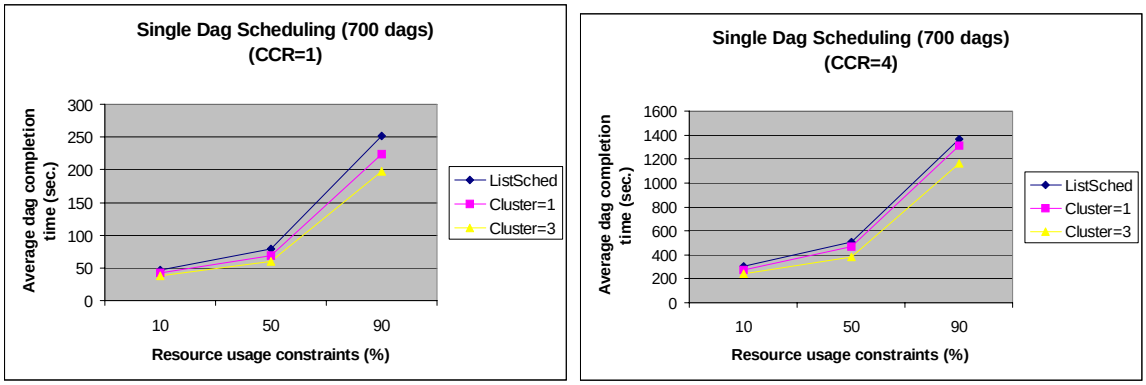


Figure 4-7. Average DAG completion time with the different scheduling algorithms such as the list scheduling and the cluster-based scheduling. The number of tried DAGs is 700. The results show the performance of the algorithms when the communication to computation ratio (CCR) is changed from 1 to 4.

The simulation results in the Figure 4-6 show the performance of the different scheduling algorithms when the link density is changed with 10%, 30%, 60% and 100%. The number of tried DAGs is 700 in the simulation. With the large number of DAGs the simulation gives more stable and reliable results. As we saw from the result with the 500 DAG simulation the cluster-based scheduling gives better scheduling results than the list scheduling in the 700 DAG simulation. The other system and workflow properties for the simulation are the same as the property values in the 500 DAG simulation.

For the simulation results in Figure 4-7 we perform the simulation for the two different communication to computation rates (CCR); 1 and 4. We compute the average completion time of the 700 DAGs for the different resource usage policy constraints, 10%, 50% and 90% when the scheduling algorithms are different; the list scheduling, the clustering with the cluster size equal to 1 and the clustering with the cluster size equal to 3. We maintain the same value range for the other system and workflow related properties with the values for the 500 DAG simulation.

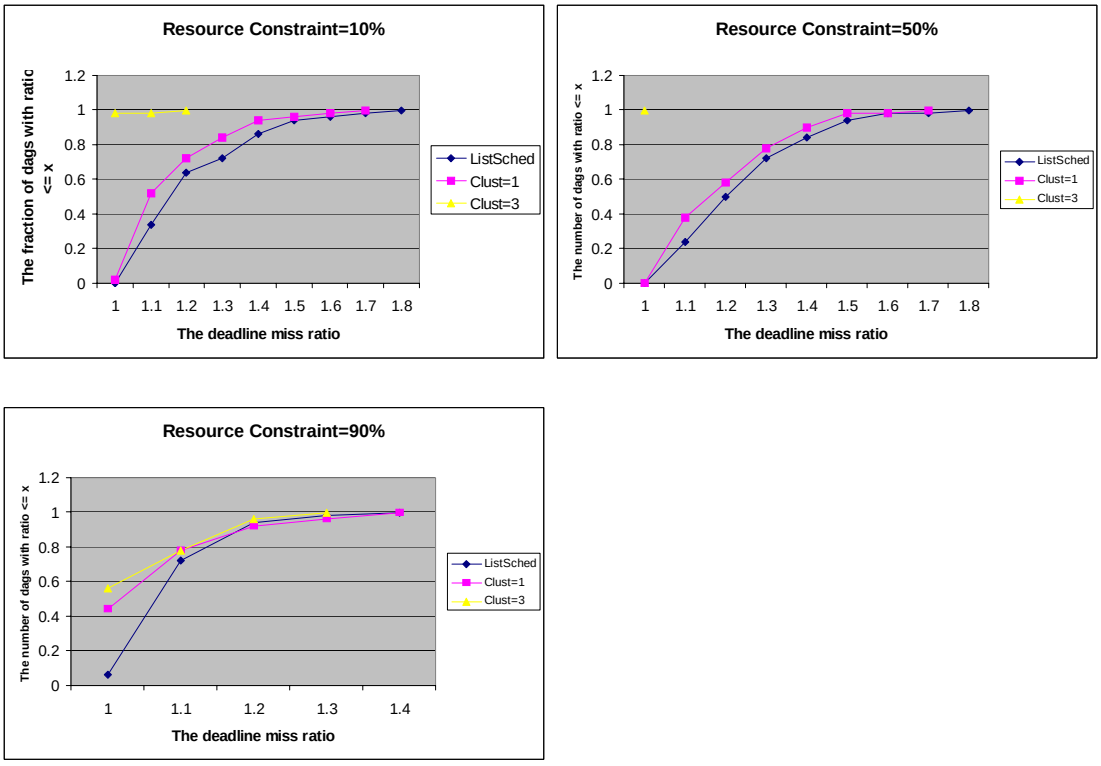


Figure 4-8. The scheduling performance of the different scheduling algorithms such as the list scheduling and the list-based scheduling with multiple workflows. The set of workflows for the simulation is 20, each of which consists of 50 DAGs. The deadline miss ratio is defined with the DAG completion time divided by the best DAG completion time. The best DAG completion time is given with the smallest DAG completion time under the given condition with the system and workflow related properties.

The Simulated Performance Evaluation with Multiple DAGs

In this section we simulate the performance of the scheduling algorithms with multiple workflows. The workflows are randomly selected from the DAG pool which contains DAGs with different values for the system and workflow-related parameters such as the number of jobs in a DAG, the number of input data of a job, the number of output data of a job, the height of a DAG. and The communication to computation ratio (CCR). The ranges of values for the parameters are defined in the following list.

- The number of jobs in a DAG: 10, 20, 40, 60 or 80.
- The link density: 10%, 20%, … 100%
- The number of output: 1

- The height of a DAG: 1, 2, …, 10.
- The CCR: 1, 2, 4, 8 or 16.
- The number of processors: 20
- The resource usage constraints: 10%, 50% or 90%

We define a performance metric of the simulation. The deadline miss ratio is defined with the ratio of a DAG completion time to the deadline. We simulate the performance of a DAG with three different algorithms. We take the ratio of a DAG completion time with an algorithm to the deadline as the deadline miss ratio. The function below shows the metric to evaluate the performance of the scheduling algorithm. The deadline miss ratio is defined with two factors; the workflow completion time and the given deadline of workflow completion. The workflow completion time is determined with the largest job completion time in a DAG.

$$\frac{compTime_d}{deadline_d}$$

where

$compTime_d$: The execution end time - the execution start time

$deadline_d$: The completion deadline of dag d

The completion deadline of a workflow is set with an admission value and an estimated completion time of the workflow. The admission value is used to control the deadline miss rate. With a lower admission value the deadline becomes tighter, resulting in a higher miss rate. With a higher admission value the deadline miss rate becomes lower. The estimated workflow completion time is computed based on the estimated current load on grid resources, the load factor of the workflow and the constraints of the workflow execution. In our simulation we set $\boldsymbol{a}$ equal to 1, which means that the deadline is equal to the estimated DAG completion time on a given resource load.

$$\text{Deadline} = (1 + \boldsymbol{a}) * \boldsymbol{EstComp}$$

Where

a: the admission value

EstComp: the estimated DAG completion time

To implement the simulation setup we devised a table showing the dynamic resource load information. This table represents the best DAG completion time on the current resource load. In our simulation setup the completion time is to be the deadline as described above. The main purpose of the resource load table is to keep track of the current resource load and to decide estimated workflow completion time under the given load. With multiple workflows we also use the table to make future resource usage reservations. Given sequential workflows, a previous workflow has higher priority to use a given resource set than a later workflow. A given resource load from a workflow prevents a usage from the following workflows. The table is used to maintain the resource usage reservation information.

The graphs in Figure 4-8 present the performance of the scheduling algorithms with multiple DAGs. The set of workflows for the simulation is 20, each of which consists of 50 DAGs. The graphs show the fraction of DAGs to the total 50 DAGs which has a deadline miss ratio less than the given value in X. Most of the 50 DAGs complete the jobs with less than a 1.2 deadline miss ratio when the resource usage constraints are 10% and 50%. Both scheduling results with a resource usage constraint equal to 90% are similar because the intensive constraint the cluster-based scheduling has not much chance to make the best scheduling decision for a job. In other words, the list scheduling result is not much different from the result of the cluster-based scheduling due to the large constraint with 90%.

The Performance Evaluation with Single DAG on OSG

In addition to the performance simulation we present the experiment to show the performance evaluation on OSG. The experiment presents the performance of the cluster-based scheduling algorithm, and compares it with the list-scheduling algorithm and other simple scheduling algorithms described in the previous section. In this experiment a resource represents a grid site on OSG. Specially, the experiment uses the CPU resource to run a set of workflow. Therefore, a resource indicates the CPU resource in a grid site in this section.

For a experiment with single DAG scheduling we use the following system and workflow-related parameter setup.

- The number of DAGs: 150
- The number of jobs per DAG: 4, 8, 16 or 32
- The height of DAG: 1, 2, 4 or 8
- The communication delay: 1
- The link density: 10%, 50% or 100%
- The number of output: 1

The Test Application

In the experiment we use Condor-G job specification language to format an application. We use the Condor and Globus infrastructure to submit jobs to a remote processor and to maintain the processor load information in OSG. The job represented in the Condor job specification language takes a set of predefined input data, which may be located in a local execution site or a remote site. After taking all the input data the job is executed for the predefined number of minute to generate a text file with the size of 1KB. The test file is an output of the job, and is located on the execution site. A set of input data is specified in the job description language, and transmitted from a location where the files are located to the job execution site. In this experiment we facilitate the Sphinx

job execution monitor. It will keep track of the job execution statue information on the Grid sites. If a job fails on a remote site then the monitor will detect the failure, and reschedule the job keeping the failed information on its local database. A site with a large number of failure is eliminated from the list of available sites for job execution.

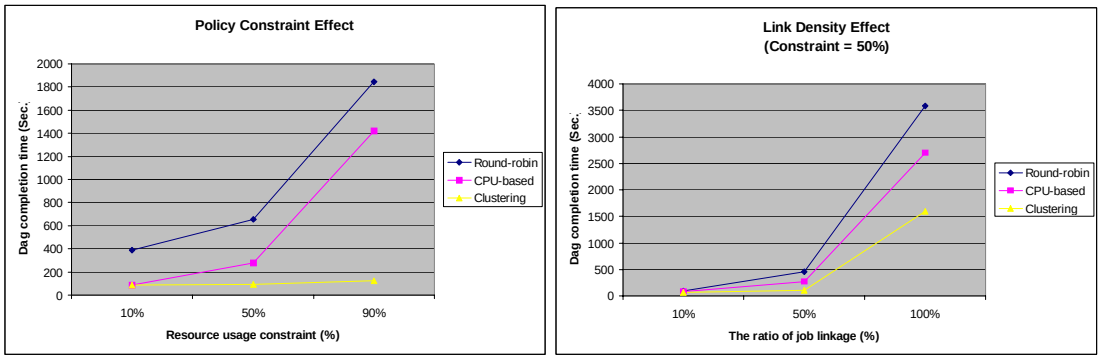


Figure 4-9. The scheduling performance evaluation of the scheduling algorithms such as the Round-robin, the CPU-based and the cluster-based with a single workflow scheduling. The resource usage constraints are changed with the different value of the constraints for 10%, 50% and 90%.

The first experiment result in Figure 4-9 presents the average DAG completion time with the different scheduling algorithms such as the Round-robin, CPU-based and cluster-based algorithms for the different resource usage constraints with 10%, 50% and 90%. The second graph shows the results with a different link density setup when the resource constraint is set to 50%. In the experiment the link density is changed with the values of 10%, 50% and 100%.

The average DAG completion time with the cluster-based scheduling with the cluster size equal to 3 is stable when the constraint is increased, while the completion time is dramatically increased with the Round-robin and the CPU-based when the constraint get intensive in the first graph.

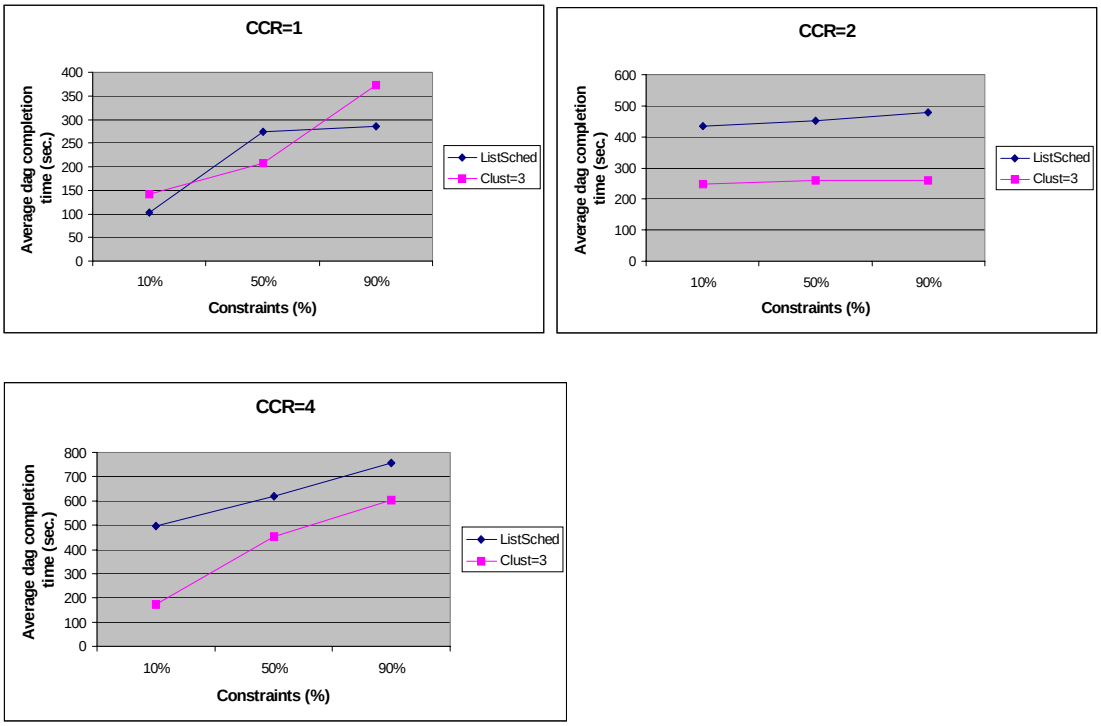


Figure 4-10. The scheduling performance comparison between the list scheduling and the cluster-based scheduling when the CCR is changed with the values of 1, 2 and 4. The resource usage constraint is 10%, 50% or 90%.

We also compare the scheduling performance of the cluster-based scheduling algorithm against the list-scheduling algorithm. The workflow types used in the experiment are categorized with the different CCR's and the link densities. The graphs in Figure 4-10 show the average DAG completion time when the CCR is different with the values of 1, 2 and 4. With the large CCR's the cluster-based scheduling outperforms the list-scheduling algorithm, while there is no much different in the performance with the CCR equal to 1. The results show that the cluster-based scheduling algorithm works better than the list-scheduling algorithm with the communication-oriented workflow, while the list-based scheduling outperforms the cluster-based scheduling with the computation-based workflows.

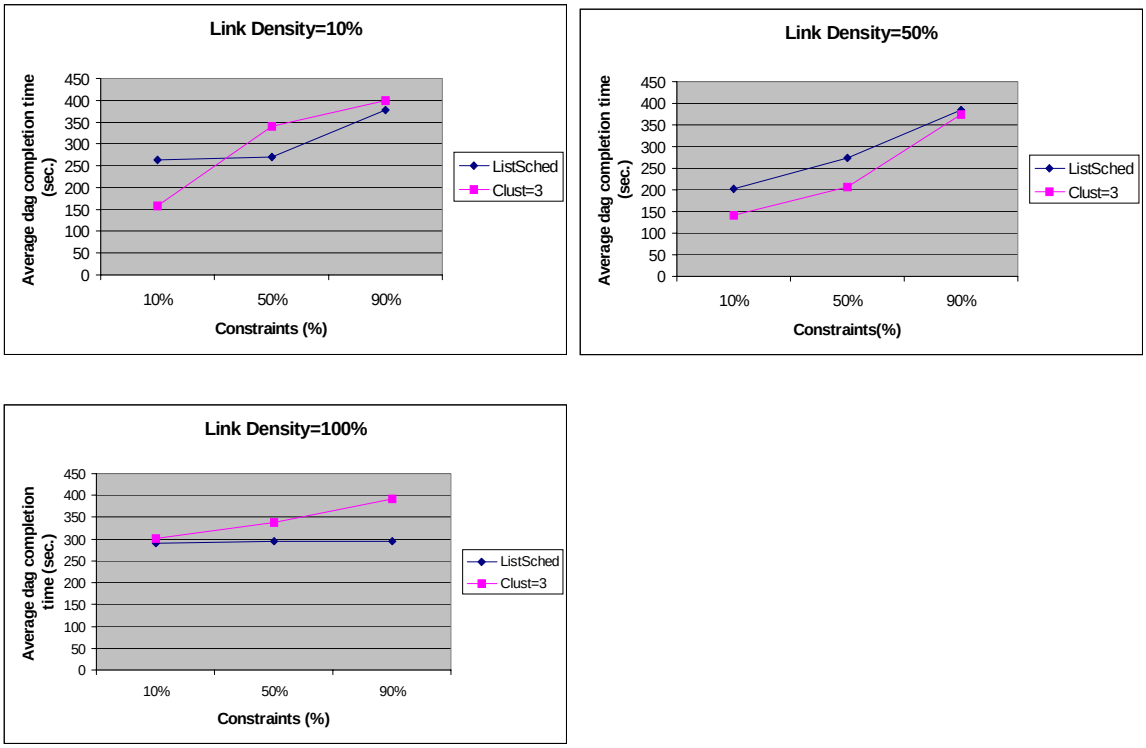


Figure 4-11. The scheduling performance comparison between the list scheduling and the cluster-based scheduling when the link density is changed with the different values of 10%, 50% and 100%. The resource usage constraint is 10%, 50% or 90%.

The graphs in Figure 4-11 show the average DAG completion time with the different link densities. The link density is defined with:

$$\text{Link density}_i = \text{parentJobs}_i / \text{upperjobs}_i * 100$$

where

parentJobs_i: The number of parent jobs of job i
upperjobs_i: The number of jobs in the upper level of job i

The graphs show the scheduling performance of the two algorithms when the resource usage constraints are changed with the values of 10%, 50% and 90%. There is no large performance difference between the list-scheduling algorithm and the cluster-based scheduling algorithm.

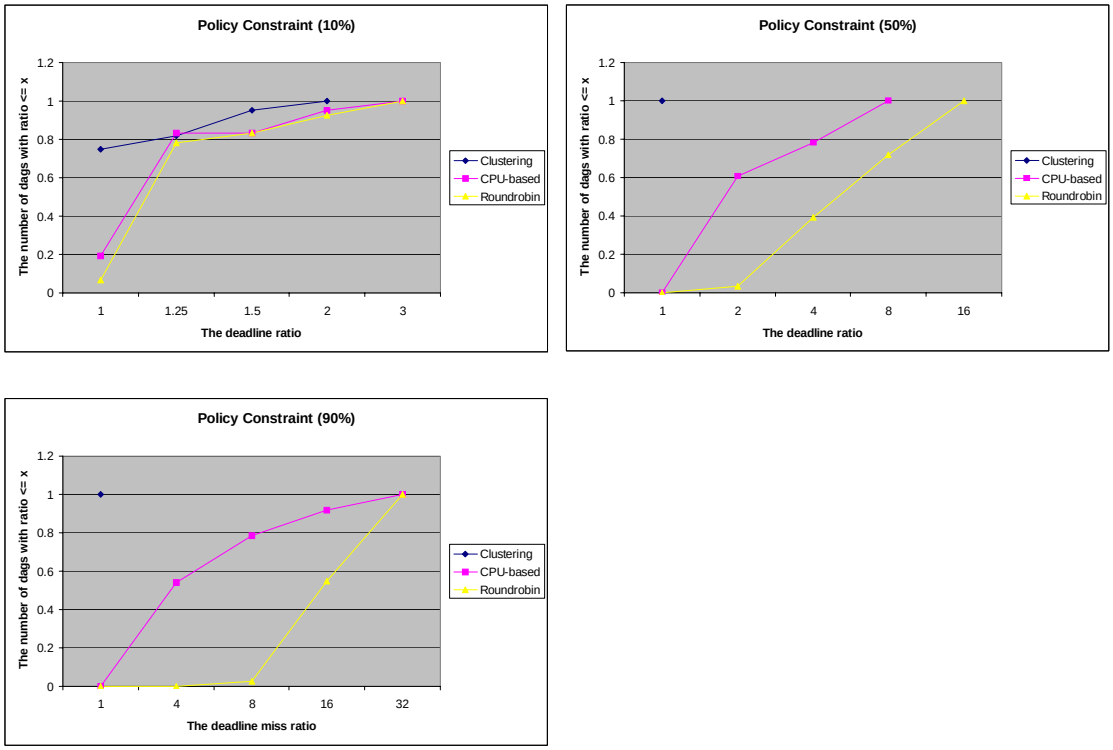


Figure 4-12. The performance of the multiple DAG scheduling with the simple scheduling algorithms (Round-robin and CPU-based) and the cluster-based scheduling algorithm. The graphs show the fraction of DAGs which meet the deadline with the miss ration equal to 1 or the fraction of DAGs which miss the deadline with the ratio greater than 1. Each graph shows the result when the resource usage policy is 10%, 50% or 90%.

The Performance Evaluation with Multiple DAGs on OSG

With the next sets of experiments we perform the multiple DAG scheduling experiment on OSG. We compare and analysis the performance of the scheduling algorithms such as the cluster-based scheduling, the list-scheduling, the Round-robin and the CPU-based. In order to specify the workflow type we give control to the workflow-related parameters such as the link density and the CCR.

In order to make the experiment results reliable and stable we repeat the experiment 10 times. In each try the set of multiple DAGs consists of 50 DAGs which are randomly

selected from the DAG pool. The ranges of the values of the system-related and workflow-related parameters are defined in the following list.

- The number of data sets: 10
- The number of DAGs per set: 50
- The number of jobs per DAG: 4, 8, 16 or 32
- The height of a DAG: 2, 4 or 8
- The communication delay: 1
- The link density: 10%, 50% or 100%
- The number of outputs: 1

The graphs in Figure 4-12 show the performance of the multiple DAG scheduling with the simple scheduling algorithms (Round-robin and CPU-based) and the cluster-based scheduling algorithm. The graphs show the fraction of DAGs which meet the deadline with the miss ration equal to 1 or the fraction of DAGs which miss the deadline with the ratio greater than 1. Each graph shows the result when the resource usage policy is 10%, 50% or 90%.

The fraction of DAGs which misses the deadline is dramatically smaller with the cluster-based scheduling than with the simple scheduling such as the Round-robin and the CPU-based when the resource usage constraints are 50% and 90%. The performance on the constraint equal to 10% is similar between the two scheduling types. With the loosen resource usage constraint the simple algorithms are smart enough to choose right processors to finish jobs, and also with many available processors to execute jobs the performance is not much sensitive to the scheduling strategy.

The experiment results on Figure 4-14 present the performance of the multiple DAG scheduling when the workflow types are different with the several link densities. As we have mentioned in the previous section the large value of the link density the jobs within a workflow are linked intensively and it make the DAG completion taking longer

time than the DAG with the loosely connected jobs because the start time of a highly linked job may be delayed with the waiting time for the parent job completion.

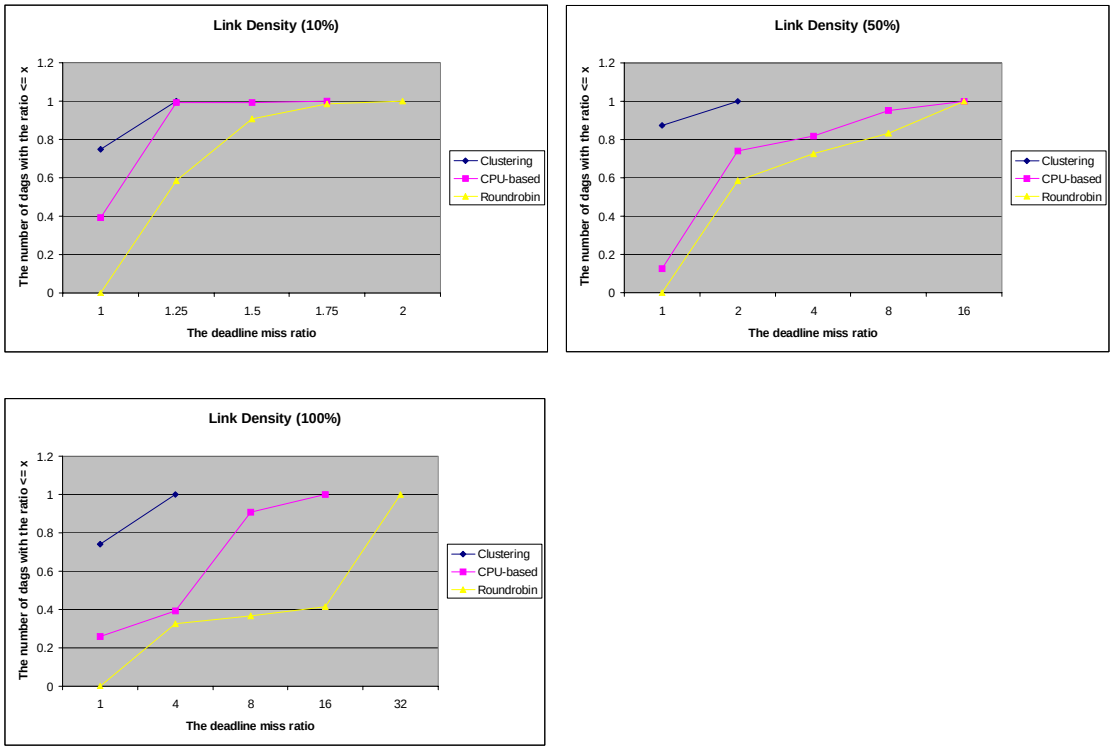


Figure 4-13. The performance of the multiple DAG scheduling with the simple scheduling algorithms (Round-robin and CPU-based) and the cluster-based scheduling algorithm. The graphs show the fraction of DAGs which meet the deadline with the miss ration equal to 1 or the fraction of DAGs which miss the deadline with the ratio greater than 1. Each graph shows the result when the link density is 10%, 50% or 100%, while the resource usage constraint is same with 50%.

The graphs in Figure 4-13 shows The performance of the multiple DAG scheduling with the simple scheduling algorithms (Round-robin and CPU-based) and the cluster-based scheduling algorithm. The graphs show the fraction of DAGs which meet the deadline with the miss ration equal to 1 or the fraction of DAGs which miss the deadline with the ratio greater than 1. Each graph shows the result when the link density is 10%, 50% or 100%, while the resource usage constraint is same with 50%.

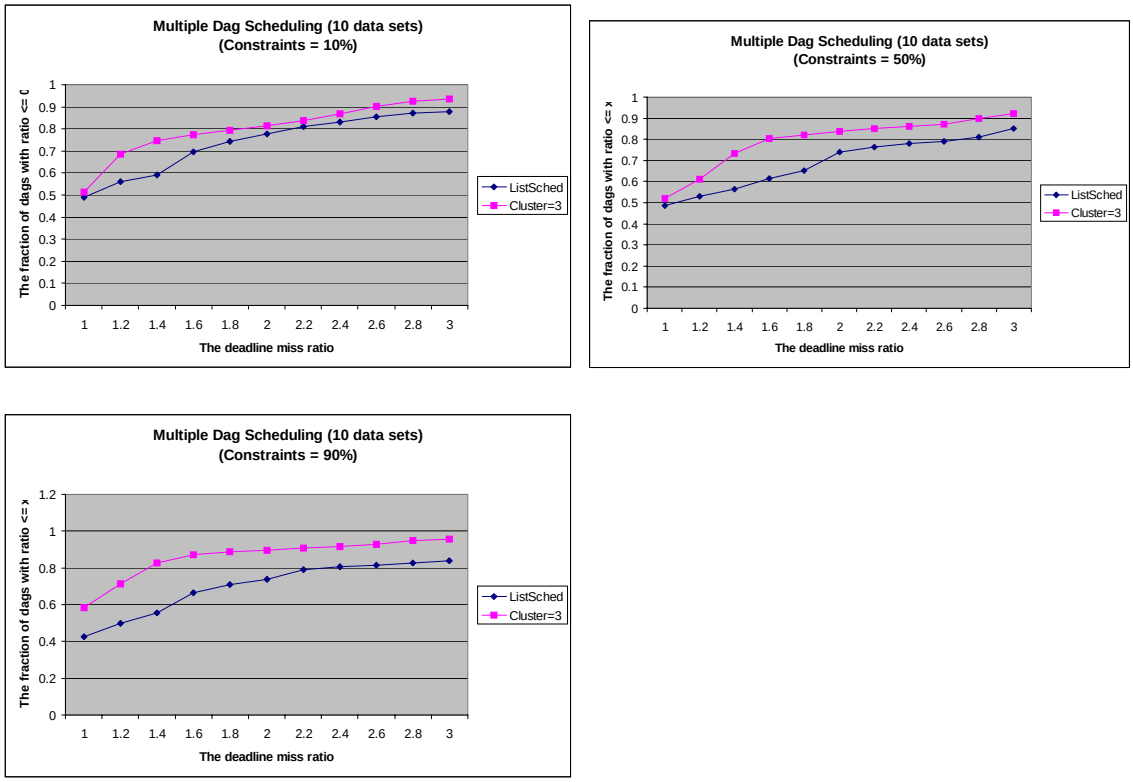


Figure 4-14. The performance of the multiple DAG scheduling algorithms such as the list-scheduling and cluster-based scheduling. The graphs show the fraction of DAGs which meet the deadline (the deadline miss ratio = 1) or miss the deadline (the ratio > 1). Each of the graphs shows the performance when the resource usage constraint is 10%, 50% or 100%.

With the DAGs on the high link density the cluster-based scheduling shows the better performance than the simple scheduling in terms of the deadline miss ratio. The performance difference between the scheduling algorithms with the loosely connected DAGs (link density equal to 10%) is not significant because the type of workflow can be scheduled with any scheduling strategy to complete in the reasonable completion time (1<= deadline miss ratio <=2).

We also show the performance comparison between the cluster-based scheduling and the list-scheduling. The graphs in Figure 4-14 present the performance of the multiple DAG scheduling algorithms such as the list-scheduling and cluster-based

scheduling. The graphs show the fraction of DAGs which meet the deadline (the deadline miss ratio = 1) or miss the deadline (the ratio > 1). Each of the graphs shows the performance when the resource usage constraint is 10%, 50% or 100%.

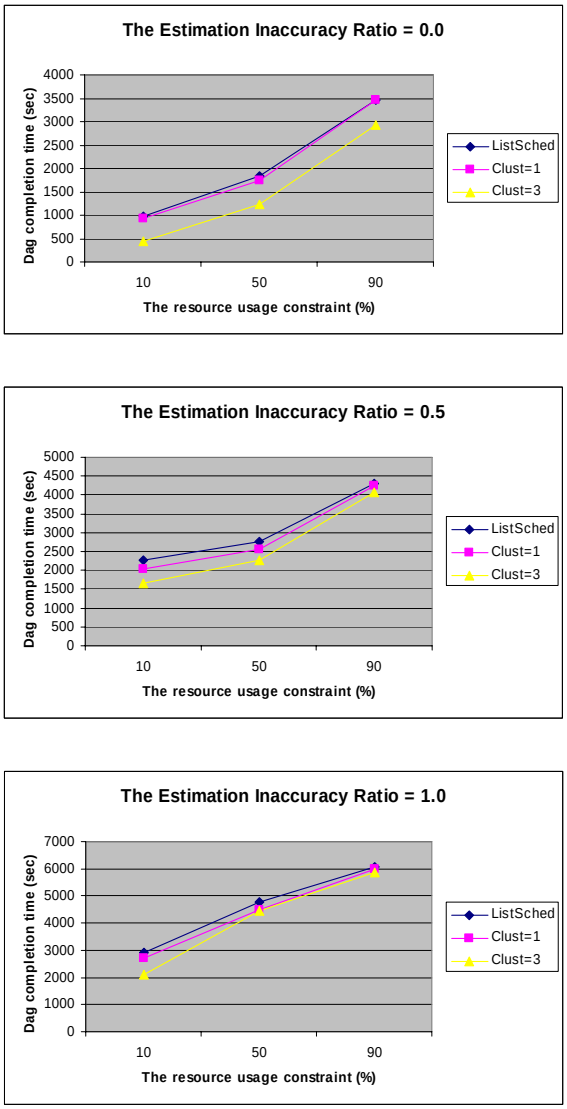


Figure 4-15. The single dag scheduling sensitivity to the job execution time estimation. The graphs shows the average dag completion time for the three different scheduling algorithms such as list scheduling, clusterSize=1 and clusterSize=3. The graphs show the completion time for the different estimation inaccuracy ratios when the resource usage constraint is changed with 10, 50 and 90 %.

The Algorithm Sensitivity to the Estimated Job Execution Time

The execution time of a job on the heterogeneous resource environment is hard to estimate and is very critical to the performance of scheduling algorithms. The execution time of a job on the processors should be estimated properly on a scheduling decision procedure. The simulation results in the previous section show the performance of different algorithms when the estimates are precise. In this section, we compare the relative performance of the algorithms when the estimates may be inaccurate.

We setup a function to simulate an estimated job execution time. Given an execution time of the job on a heterogeneous site and a ratio of the estimation inaccuracy the function generates a job execution time. The inaccuracy ratio determines the difference between the execution time used by the algorithm and the actual estimated time used during the simulation. For example, given that the execution time is 100 seconds and the inaccuracy ratio is 0.3 the estimated execution time is 70 or 130 seconds. The function definition is given such that:

$$estimatedTime_{ij} = execTime_{ij} + random(1,-1) \times execTime_{ij} \times rat$$

where

$estimatedTime_{ij}$: Estimated execution time of job i on processor j

$execTime_{ij}$: Given execution time of job i on processor j

$random(1,-1)$: A random selection between 1 and -1

$0 \le rat \le 1$: The ratio of the estimation inaccuracy

We simulated the above algorithms to investigate the performance of the scheduling algorithms for a range of accuracy ratios of 0.0, 0.5 and 1.0. We perform the simulation for single dag scheduling as well as multiple dag scheduling. The graphs in Figure 4-15 show the average dag completion time when we use single dag scheduling

algorithms. We choose the estimation inaccuracy ratio for the values of 0.0, 0.5 and 1.0. In order to compare the robustness of the proposed scheduling algorithms we use three different scheduling algorithms: ClusterSize=3 or ClusterSize=1, and ListSched.

The inconsistency ratio represents the ratio of the estimated time to the given execution time. For example, the value 0.5 means that the estimated time is different from the given time by 50%, while 0 means that the estimated time is same with the given time. We plot the dag completion times with different resource usage constraints. The usage constraints are changed with the values of 10, 50 and 90%.

The graphs show the affect of the inaccuracy to the performance of a scheduling algorithm. The clustering based algorithm with the cluster size equal to 3 outperforms the other algorithms when the estimated time is equal to the given time, in other words, the inaccuracy ratio is 0.0. Although the clustering algorithm with cluster size 3 is better than the other two algorithms, the performance improvements decrease as the inaccuracy ratio increases to 1.0. Also, the absolute performance of all the algorithms decreases when the estimation times are not accurate. The results show the negative impact of the inaccuracy of execution time estimation. It also, shows that the clustering based approach is relatively better even when the estimates are not accurate.

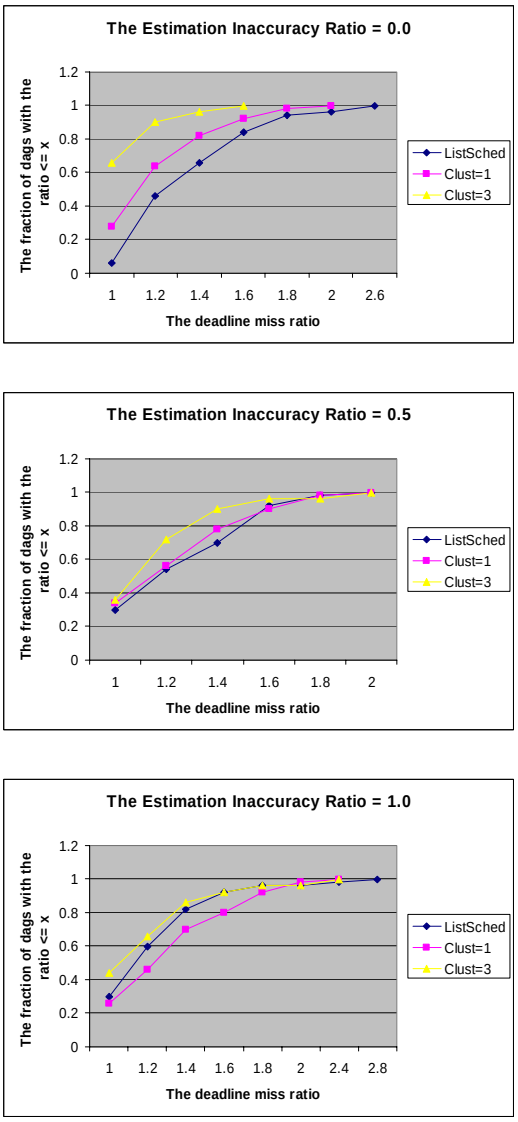


Figure 4-16. The multiple dag scheduling sensitivity to the job execution time estimation. The graphs shows the average ratio of the dags with the deadline miss ratio equal to or less than a given value for the three different scheduling algorithms such as list scheduling, clusterSize=1 and clusterSize=3. The graphs show the ratio for the different estimation inaccuracy ratios when the deadline miss ratio is different from 1 to 2.6.

We also perform the performance sensitivity test with the multiple dag scheduling algorithms. The simulation results in Figure 4-16 show the ratios of dags to a given get of dags, of which the deadline miss ratio is equal to or less than a given value. The estimation inaccuracy ratio values are set to 0.0, 0.5 and 1.0. We use 20 sets of 50 dags

that are randomly selected with different CCR, the number of jobs, and dag structure. The resource usage constraint is set to 50%. The graph with the inaccuracy ratio equal to 0 shows that the cluster-based scheduling algorithm with the cluster size equal to 3 outperforms the other algorithm in terms of the deadline miss ratios. More than 60% of the given sets of dags are completed within the deadline with the algorithm, while the other algorithms do not perform as good as the cluster-based scheduling. However, the performance difference is lower when the estimation inaccuracy ratio is 0.5 and 1.0. Also, less than 40% of the total dags meet their deadline with any algorithms when the estimation is not accurate. Thus, the absolute performance of all the algorithms decreases when the estimation times are not accurate. It is important to note the clustering based approach is still better than list scheduling even when the inaccuracy ratio is 1.0. This demonstrates the robustness of this approach.

The above simulation results of single DAG and multiple DAGs also show the critical importance of the job execution estimation to the algorithm performance. Accurate estimation leads to better absolute and relative performance of intelligent scheduling algorithms.

Conclusions

In this paper we introduce a novel policy-based scheduling algorithm. It allocates grid resources to an application under the constraints presented with resource usage policies. It performs optimized scheduling on heterogeneous resources using an iterative approach and binary integer programming (BIP). The algorithm improves the completion time of an application in integration with job execution tracking and history modules of SPHINX scheduling middleware. The implementation of the algorithms in SPHINX

makes it possible to schedule jobs onto OSG according to policy-based scheduling decisions.

CHAPTER 5
CONCLUSIONS

In this chapter we conclude the dissertation, and discuss future works. We discuss the current research issues on Grid computing, and out research hypothesis to the problems. We overview our solutions to the research issues. We also present future research works to improve the current techniques and software.

Executing data intensive applications on large grids requires allocating dynamically changing storage, computer and network resources in a fashion that satisfies both global and local constraints. Global constraints include community-wide policies governing how resources should be prioritized and allocated. Local constraints include site-specific control as to when external users are allowed use local resources.

We will develop algorithms and software that uses heterogeneous and dynamically changing resource status information to execute workflows in grid environment. One of the main focuses of this research is to develop algorithms that are able to gracefully handle the impact of latency of monitoring (time between collecting information from each sensor to time when this information is available in a collective form to the decision making process) on the quality of workflow scheduling in an adaptive resource environment.

The algorithms are also able to resolve resource-scheduling issues utilizing efficient operational research methodologies for optimization subject to resource usage policy constraints. Achieving global optimization of workflow execution has been researched in high performance application development for many years, while it is still

open issue. Especially in grid computing environment in which heterogeneous and autonomous resources and requests have various usage and execution requirement it is critical to satisfy overall quality of service (QOS) such as workflow completion deadline.

The research also focuses on developing a novel-scheduling framework for achieving globally optimized resource allocation. The proposed strategy satisfies global QOS requirement as well as performs locally best-effort scheduling. The resource allocation is derived by resource usage policies for intended users and on-time scheduling decision to reflect dynamically changing grid environment.

We tackle the issue of fault-tolerant scheduling by utilizing job execution tracking and rescheduling functionality in the system. Another issue that we tackle is distributed resource management. The research tries to develop a framework distributed based on resource content information that is characterized by system prosperities.

Policies, including authentication, authorization, and application constraints are important factors for maintaining resource ownership and security. The set of possible constraints on job execution can be various, and change significantly over time. Policies may include any information that should be specified to ensure that a job is matched to appropriate resources. This research develops a novel framework for policy-based scheduling. The study investigates efficient resource usage management and prioritization for the scheduling strategies.

In order to develop reliable and scalable software for solving general-purpose distributed data intensive problems the research investigates possible architectures that combine existing grid services. We will deploy the prototype across Grid testbeds to demonstrate the efficiency of the system. It will exhibits interactive remote data access,

interactive workflow generation and collaborative data analysis using virtual data and data provenance, as well as showing non-trivial examples of policy-based scheduling of requests in a resource constrained grid environment.

The development of scheduling algorithms and software that effectively use monitoring information along with information about the jobs already scheduled can lead to providing reasonable guarantees for completing the workflows within deadlines even though the available resources may change over time. A key portion of our research will be to understand and model the rate of change of the total resources availability and understand its impact on the completion times.

We also argue that a scheduling system based on resource usage policies and request preference such as deadline is sufficient to satisfy specific scheduling requests for QOS and to achieve workload balance across a grid. In the research we investigate efficient architectures of distributed resource management framework to coordinate access to shared resources among autonomous instances. QOS is classified with two general dimensions; soft QOS and hard QOS. Each dimension has multiple properties such as resource usage amount or workflow execution deadline.

We assume that resource usage accounts or request priorities adjustment can control the request assignment to grid resources. This feature allows the scheduler balance workloads across grid resources resulting in better overall utilization and turn around time. In addition to the workload balance a resource management system can manage resource usage by adjusting usage quotas to intended resource users. The scheduler monitors resource usage by keeping track of the quota change. It saves

resource cycles by assigning more requests to the idle resources, while reducing the number of requests assigned to the overly used resources.

We also argue that the implementation of a prototype of distributed high-level services supporting grid-enable data analysis within research communities can lead to satisfy the need to rapidly access and analyze massive data from globally dispersed scientists in hundreds of collaborative teams. The research investigates the associated complex behavior of such an end-to-end system. In particular, the prototype integrates several existing grid services for the distributed data analysis.

We develop performance metrics to demonstrate the application's usefulness. Our scheduling algorithms are implemented onto the application, and show impact of the monitoring information onto the performance. We perform simulation to understand the impact of monitoring information and its latency on effective scheduling in adaptive resource environment.

The simple schemes will include round robin, guided self-scheduling etc. Our goal would demonstrate that the newly developed algorithms have significantly superior performance in terms of meeting deadlines and other performance measures as compared to simple approaches.

The research also presents simulation to demonstrate the efficiency of proposed distributed scheduling architecture. The simulation shows the performance of request scheduling on the three different network topologies; content-based, geography-based and centralized scheduling networks. The content-based network is constructed according to our research arguments, while the geography-based is a traditional

distributed network based on resources' geographical distance. In the centralized service a single server takes charge of scheduling requests to all the resources in a grid.

In the simulation and experiments we use different types of workflows and resources. The research focuses on single task workflows initially. They are simplest workflows that are interactive or batch type. Then, we extend the research to simple DAG-based workflows, which will reflect a variety of real world scenarios. Finally, our scheduling algorithms deal with multiple DAGs in achieving an objective function simultaneously. The experiments of the algorithms are performed in heterogeneous resource environment. A resource type such as CPU, storage or network has a set of resources with different performance.

In order to test and evaluate the proposed data analysis prototype we deploy the framework across a grid testbed named Grid2003 which consists of more than 25 sites providing more than 2000 CPUs, and exhibit remote data access, workflow generation and collaborative data analysis using virtual data and data provenance, as well as showing non-trivial examples of policy based scheduling of requests in a resource constrained grid environment.

LIST OF REFERENCES

1. J. Gary, A. Szalay, The World Wide Telescope: An Archetype for Online Science. Microsoft Research Technical Report, 75(4), 6, August 2001.

2. P. Avery, I. Foster, The GriPhyN Project: Towards Petascale Virtual-Data Grids. The 2000 NSF Information and Technology Research Program, Arlington, VA, 2000.

3. I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2150, 2001.

4. J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of Tenth International Symposium on High Performance Distributed Computing, 7-9, San Francisco, CA, 2001.

5. A. Gerasoulis, and T. Yang, On the Granularity and Clustering of Directed Acyclic Task Graphs, IEEE Trans. Parallel and Distributed Systems 5(9), 951-967, 1994.

6. A. Ghafoor, and J. Yang, A Distributed Heterogeneous Supercomputing Management System, Computer, 26(6), 78-86, June 1993.

7. G. Karypis, and V. Kumar, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, Technical Report, Department of Computer Science, University of Minnesota, 1995.

8. M. Kaddoura, and S. Ranka. Runtime Support for Parallelization of Data-Parallel Applications on Adaptive and Nonuniform Environments, Journal of Parallel and Distributed Computing, Special Issue on Workstation Clusters and Network-based Computing, 163-168, June 1997.

9. Y. Kwok, and I. Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, 31(4), 406-471 December 1999.

10. S. Ranka, M. Kaddoura, A. Wang, and G. C. Fox, Heterogeneous Computing on Scalable Heterogeneous Systems, in Proceedings of the SC93 Conference, 763-764, Phoenix, AZ, November 1993.

11. T. Yang, and A. Gerasoulis, DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors, IEEE Trans. Parallel and Distributed Systems, 5(9), 951-967, 1994.

12. I. Foster, The Grid2003 Production Grid: Principles and Practice, in Proceedings of 13th IEEE International Symposium on High-Performance Distributed Computing, 236-245, Honolulu, HI, 2004.

13. Open Science Grid, May 1st, 2006, http://www.opensciencegrid.org/index.php?option=com_frontpage&elMenu=Home Date Last Visited: March 6, 2006.

14. Virtual Data Toolkit, May, 2004, http://www.cs.wisc.edu/vdt/documentation.html. Date Last Visited: March 6, 2006.

15. A. Bayucan, Globus-enabled PBS: the PBS-Globus Interface, Globus retreat 2000, Pittsburgh, PA, August 2000

16. S. Bittner, Selecting and Implementing the PBS Scheduler on an SGI Onyx2/Origin 2000, Cray User's Group, May 1999

17. Maui Scheduler Documentation, Maui High Performance Computing Center, 1999 http://www.hpc2n.umu.se/doc/maui/index.html Date Last Visited: March 6, 2006.

18. Q. Snell, M. Clement, D. Jackson, C. Gregory, The Performance Impact of Advance Reservation Meta-scheduling, IPDPS 2000 Scheduling Workshop, 83-85, Cancun, Mexico, May, 2000

19. Maui Scheduler Administrator's Guide Version 3.2, Supercluster Research and Development Group, http://www.supercluster.org/mauidocs/mauiadmin.shtml Date Last Visited: March 6, 2006.

20. LSF Administrator's Guide, Version 4.1, Platform Computing Corporation, February 2001

21. B. Sundaran, B. M. Chapman, Policy Engine: A Framework for Authorization, Accounting Policy Specification and Evaluation in Grids, 2nd International Conference on Grid Computing, Nov. 2001

22. B. Segal, Grid Computing: The European Data Project. In IEEE Nuclear Science Symposium, Lyon, France, October 2000.

23. M. Ruda, Integrating GRID Tools to Build a Computing Resource Broker: Activities of DataGrid WP1. CHEP 2001, Beijing, September 2001.

24. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, Pegasus: Planning for Execution in Grids. Technical Report GriPhyN-2002-20, November 2002.

25. D. Thain, T. Tannenbaum, M. Livny, Condor and the Grid, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003.

26. J. Basney, M. Livny, Deploying a High Throughput Computing Cluster, High Performance Cluster Computing, Rajkumar Buyya, Editor, Vol. 1, Chapter 5, Prentice Hall PTR, May 1999.

27. J. Weismann, Prophet: Automated Scheduling of SPMD Programs in Workstation Networks, Concurrency: Practice and Experience, 11(6), 301-321, May, 1999.

28. Enabling Grids for E-science, April 14th, 2006, http://egee-intranet.web.cern.ch/egee-intranet/index.htm Date Last Visited: March 6, 2006.

29. Lightweight Middleware for Grid Computing, April 14th, 2006, http://glite.web.cern.ch/glite/ Date Last Visited: March 6, 2006.

30. Toward Open Grid Services Architecture, http://www.globus.org/ogsa Date Last Visited: March 6, 2006.

31. I. Foster, C. Kesselman Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications 1997; 11(2):115–128.

32. R. Buyya , The Gridbus Toolkit: Enabling Grid Computing and Business, http://www.gridbus.org, Date Last Visited: March 6, 2006.

33. J. Almond , D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce, Future Generation Computer Systems 1999; **15**:539–548.

34. W. Johnston, D. Gannon, B. Nitzberg, Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid, Eighth IEEE International Symposium on High Performance Distributed Computing, Redondo Beach, CA, August 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999.

35. E. Akarsu, G. Fox, W. Furmanski, T. Haupt, WebFlow—High-level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing. SC98: High Performance Networking and Computing, Orlando, FL,1998.

36. H. Casanova, J. Dongarra , NetSolve: A Network Server for Solving Computational Science Problems. International Journal of Supercomputing Applications and High Performance Computing, 11(3), 1997

37. E. Akarsu, G. Fox, T. Haupt, A. Kalinichenko, K. Kim , P. Sheethaalnath, C.Youn, Using Gateway System to Provide a Desktop Access to High Performance Computational Resources. The 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), Redondo Beach, CA, August 1999.

38. Gridlab. March 6th, 2006, http://gridlab.org Date Last Visited: March 6, 2006.

39. SETI@Home. http://setiathome.ssl.berkeley.edu/ Date Last Visited: March 6, 2006.

40. Distributed.Net. January 24th 2006, http://www.distributed.net/ Date Last Visited: March 6, 2006.

41. The WS-Resource Framework. http://www.globus.org/wsrf/ Date Last Visited: March 6, 2006.

42. Global Grid Forum. http://www.ggf.org/ogsi-wg Date Last Visited: March 6, 2006.

43. D. Doval, D. O'Mahony, Overlay Networks, A Scalable Alternative for P2P, IEEE Internet Computing, August 2003.

44. S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A Scalable Content-Addressable Network, ACM SIGCOMM, 2001.

45. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishman, Chord: A Scalable Peer-to-peer Loopup Service for Internet Applications, ACM SIGCOMM, 2001.

46. B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical report, U. C. Berkeley, 2001

47. A. Crespo, H. Garcia-Molina, Semantic Overlay Networks for P2P Systems, Technical report, Stanford University, Jan. 2003.

48. W. Hoschek, A Unified Peer-to-Peer Database Framework for Scalable Service and Resource Discovery, Proc. of the International IEEE/ACM Workshop on Grid Computing, Baltimore, USA, Nov. 2002. Springer Verlag.

49. D. Bradley, Condor-G Matchmaking in USCMS, Condor technical report, University of Wisconsin, Nov. 2003

50. A. Carzaniga, A.L. Wolf, Content-based Networking: A New Communication Infrastructure, NSF Workshop on an infrastructure for Mobile and Wireless Systems, Scottsdale, AZ, October, 2001

51. A. Carzaniga, M.J. Rutherford, A.L. Wolf, A Routing Scheme for Content-based Networking, Proceedings of IEEE INFOCOMM 2004, Hong Kong China, March, 2004.

52. R. Chand, P. Felber, A Scalable Protocol for Content-based Routing in Overlay Networks, Proceedings of the IEEE International Symposium on Network Computing and Applications, Cambridge, MA, April, 2003.

53. M. Aron, D. Sanders, P. Druschel, W. Zwaenepoel, Scalable Content-aware Request Distribution in Cluster-based Network Servers, Proceedings of the 2000 Annual Usenix Technical Conference, San Diego, CA, June, 2000.

54. G.Q. Liu, K.L. Poh, and M. Xie, Iterative List Scheduling for Heterogeneous Computing, Journal of Parallel and Distributed Computing, 65(5), 654-665, 2005.

55. Y. K. Kwok, I. Ahmad, Dynamic Critical-path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, 7(5), 506-521, 1996.

56. X. Qin, H. Jiang, Reliability-driven Scheduling for Real-time Tasks with Precedence Constraints in Heterogeneous Distributed Systems, In Proceedings of the International Conference Parallel and Distributed Computing and Systems 2000 (PDCS 2000), Las Vegas, USA, November 6-9, 2000.

57. H. Zhao, and R. Sakellariou, An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm, Euro-Par 2003, LNCS 2790, Springer 2003.

58. G. C. Sih, and E.A. Lee, Dynamic-level Scheduling for Heterogeneous Processor Networks. In Proceedings of the Second IEEE Symposium on Parallel and Distributed Systems, 1990.

59. M. Kafil, and I. Ahmad, Optimal Task Assignment in Heterogeneous Distributed Computing Systems, Concurrency, IEEE 6(3), 42-50, 1998.

60. R. Ramen, M. Livny, M. Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL

61. J. M. Schopf, Ten Actions When Superscheduling. Scheduling Working Group, Informational Category, Global Grid Forum.

62. Y. A. Li, J.K. Antonio, H. J. Siegel, M. Tan, D.W. Watson, Determining the Execution Time Distribution for a Data Parallel Program in a Heterogeneous Computing Environment. Journal of Parallel and Distributed Computing 44 (1), 35-52, 1997.

63. M. Iverson, F. Ozguner, L. C. Potter, Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. Proceedings of 8th Heterogeneous Computing workshop (HCW), Apr. 1999, San Juan, Puerto Rico.

64. M. Tan, H. J. Siegel, J. K. Antonio, Y. A. Li, Minimizing the Application Execution Time Through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System. IEEE Transactions on parallel and distributed system, 8(8), August 1997.

65. Conrad Steenberg, Clalens, http://clarens.sourceforge.net Date Last Visited: March 6, 2006.

66. I. Foster, J. Voeckler, M. Wilde, Y. Zhao, Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. The 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), 2002.

67. A. Chervenak, Giggle: A Framework for Constructing Scalable Replica Location Services. To appear in Proceedings of SC2002 Conference, November 2002.

68. T. Sandholm, J. Gawor, Globus Toolkit 3 Core – Agrid Service Container Framework, http://www-unix.globus.org/toolkit/documentation.html Date Last Visited: March 6th, 2006.

69. P. Raman, A. George, M. Radlinski, R. Subramaniyan, GEMS: Gossip-Enabled Monitoring Service for Heterogeneous Distributed Systems. GriPhyN technical Report 2002-19, Dec 17, 2002

70. T. Tannenbaum, Hawkeye, Condor Week, Paradyn, March 2002.

71. X. Zhang, J. Freschl, and J. Schopf. A Performance Study of Monitoring and Information Services for Distributed Systems. Proceedings of HPDC, August 2003.

72. J. Bent, Flexibility, Manageability, and Performance in a Grid Storage Appliance, Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing Edinburgh, Scotland, July 2002.

73. T. Kosar, M. Livny, Stork Data Placement (DaP) Scheduler, http://www.cs.wisc.edu/condor/stork/ Date Last Visited: March 6, 2006.

74. J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC 10) IEEE Press, 5:237-246, 2002.

75. F. Rademakers, R. Brun, ROOT: An Object-Oriented Data Analysis Framework, Linux Journal, 51, July 1998

76. H.B. Newman, I.C. Legrand, P. Galvez, R. Voicu, C. Cirstoiu, MonALISA: A Distributed Monitoring Service Architecture, CHEP 2003, La Jola, California, March 2003

77. J. Blythe, E. Deelman, Y. Gil, The Role of Planning in Grid Computing, To appear in Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS), Trent, Italy, June 9-13, 2003.

78. R. Ramen, M. Livny, M. Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July 28-31, 1998

79. K. Ranganathan, I. Foster, Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. International Symposium for High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 2002.

80. M. Carman, F. Zini, L. Serafini, Towards an Economy-Based Optimization of File Access and Replication on a Data Grid, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 340-345, Berlin, Germany, May 2002

BIOGRAPHICAL SKETCH

Jang-uk In obtained his Master of Science in computer science on December 1999. He obtained his Bachelor of Science in computer science from Hannam University, Taejon , South Korea, on February 1997.