MULTI-LEVEL ADAPTATION FOR PERFORMABILITY IN DYNAMIC WEB SERVICE WORKFLOWS

Lavanya Ramakrishnan

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Department of Computer Science

Indiana University

June 2009

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Dennis Gannon, Ph.D.
(Principal Advisor)

Randall Bramley, Ph.D.

Steven Johnson, Ph.D.

Beth Plale, Ph.D.

June 3, 2009

Daniel A. Reed, Ph.D.

Acknowledgements

I have had the support of mentors, collaborators, family and friends in this incredible journey towards the doctoral degree.

I would not have completed this work without Jaidev, my best friend and life partner. Jaidev pushed me to go back to school, provided feedback on my work, shared my obscure geek excitement, and helped maintain a work-life balance.

Both our families have been supportive, patient and helpful as I have progressed through graduate school. My parents inculcated a strong interest in math, science and learning and provided a strong foundation in questioning, reasoning and decision-making; all important tools for research. Growing up, I was inspired by my grandparents who had values and ideas well beyond their times and were very supportive of my various endeavors over the years. I am very fortunate to be able to enjoy the love and support of my paternal grandmother to this day. Much of my interest in computer science started when my brother, Bharath taught me to write BASIC programs when I was around ten and for that I will ever be grateful. Bharath and his wife, Chitra have also provided unconditional support on numerous occasions. Jaidev's parents' annual visits and the family reunions with them, my sister-in-law, Jaee and her husband, Anirudha have always been a welcome change. My nephew, Dhruv and niece, Uma constantly remind me of the simple joys of life with

their curiosity and excitement.

Dennis Gannon encouraged me to come back to school for the doctoral degree and made it possible for me to work remotely. He provided an ideal research environment - encouraging me to pursue independent research while being available when I needed guidance. Beth Plale has been a great mentor and collaborator, providing key insights into technical as well presentation aspects of this research, and has always been available for discussions. I am thankful to Dan Reed for providing me great opportunities at RENCI and for nudging me to explore topics that were outside my comfort zone which has helped make this work stronger. Randall Bramley and Steve Johnson have provided helpful pointers and suggestions throughout my graduate school years.

I have been lucky to have numerous collaborators through different projects. I am thankful to Jeff Chase for our long discussions about research (and life). Kelvin K. Droegemeier patiently explained various concepts of mesoscale meteorology and Rich Wolski, Charles Koelbel and (late) Ken Kennedy provided key insights on resource management for workflows in grid environments and made it possible for me to collaborate with the VGrADS team.

Laura Grit, T. Mark Huang, Adriana Iamnitchi, David Irwin, Yang-Suk Kee, Anirban Mandal, Daniel Nurmi, Graziano Obertelli, Kiran Thyagaraja, Asim YarKhan, Aydan Yumerefendi and Dmitrii Zagorodnov provided critical pieces of software infrastructure for the GROC and LEAD-VGrADS activities. I am also thankful to past and present Extreme Lab members, colleagues at MCNC and RENCI for numerous productive discussions. Other team members of LEAD, Bioportal, VGrADS, SCOOP projects also helped in numerous ways. My research was funded through various National Science Foundation (NSF) programs including the Middleware initiative (GROC), ITR (LEAD and VGrADS) and SDCI (workflow emulator) and TeraGrid allocations.

The workflow survey was a critical piece of the puzzle and I am grateful to the people who contributed including Suresh Marru, Brian Blanton, Howard Lander, Steve Thorpe, Jeffrey Tilson,

Sriram Krishnan, Luca Clementi, Ravi Madduri, Wei Tan, Cem Onyuksel, Yogesh Simmhan, Sudharshan Vazhkudai, Vickie Lynch. Access to large scale system infrastructure at various sites has been important for validating my hypothesis. I am thankful to system administrators and support staff at Duke University, University of North Carolina - Chapel Hill/RENCI, Univ. of California-Santa Barbara, Univ of Houston, Univ of Tennesse - Knoxville, Univ of California - Santa Barbara and TeraGrid sites. Brad Viviano also taught me a number of things about system administration and managing clusters that has influenced some key design decisions.

My friend, Anjali, no matter which part of the world life takes her, provides support that I can't begin to describe. I have also been lucky to have many friends who have been part of my support system while I lived in Bloomington, Durham and Sunnyvale. They provided me with necessary distractions in the form of meals, coffee, movies and hikes.

Abstract

Large scale computations from various scientific endeavors are composed as workflows that access shared data and high performance systems. Similarly, business applications in cloud computing systems use distributed infrastructure as part of mainstream business models. Recent advances in grid and cloud computing provide tools to monitor and manage execution. However they do not not provide predictable bounds on the Quality of Service (QoS) that can be expected in such variable multi-user distributed environments. Understanding the dynamic properties of resources and coordinated control of resources and workflows is critical especially for deadline-sensitive workflows such as weather prediction.

In this dissertation we revisit the software stack that supports the multi-tier services and propose and evaluate the WORDS (Workflow ORchestrator for Distributed Systems) architecture that abstracts the differences between specific resource models and provides a clear separation of concerns between the resource-level and application-level tools. In the context of the WORDS architecture we explore interfaces and mechanisms necessary for providing predictable quality of service to web service workflows with time and accuracy constraints.

We make the following four primary contributions. First, we propose a resource abstraction across grid and cloud resource control mechanisms that enables higher-levels tools to abstract the

differences between systems. Second, we propose a probabilistic Quality of Service (QoS) model that enables providers to quantify the variation in resource availability; both for resource procurement due to competition and for the duration of the resource request from failures at various levels. Third, we use performability analysis through a Markov Reward Model to quantify the loss in performance and study the impact on cost due to availability variations. Finally, we propose a multi-phase orchestration approach that balances performance, reliability and cost considerations for a set of workflows.

Contents

List of Tables

List of Figures

1

Introduction

In the last few years we have seen the emergence of multi-core processors, virtualization technologies and web services that have revolutionized the computing models in use. Increasingly, distributed resources and data are shared across virtual communities and used to solve scientific and business problems. More recently, companies are leveraging distributed computing infrastructures through utility, grid and cloud computing as an integral part of mainstream business models. The high performance computing domain at supercomputing centers has seen a similar trend in supporting scientific applications such as drug discovery [12], cancer research [29], weather modeling and prediction [54], earthquake engineering [120] through deployments of large-scale distributed infrastructure such as TeraGrid [179] and Open Science Grid [132]. These trends are changing the software services and the interaction of higher-level software with distributed systems.

The advent of the internet has also resulted in the emergence of sophisticated end-user tools such as web interfaces and portals that enable the end-user to access distributed information and resources. Workflow tools have emerged at the application later that allow users and businesses to compose work units as a sequence of automated or semi-automated operations. Workflow tools have been used to model business processes i.e., to automate information and task sharing among

individuals of a company or business partners. More recently workflows and workflow tools have become an integral part of cyberinfrastructure [10, 49]. Workflow tools allow a scientist to compose and manage complex scientific distributed computation and data in distributed resource environments.

Workflow tools in distributed cyberinfrastruture support basic resource interaction functions and provide limited Quality of Service (QoS) guarantees or failure recovery. Workflows and distributed system infrastructures have evolved in largely isolated environments and only have a weak interaction model available today. The new resource models ushered in by cloud computing models introduce additional challenges in assuring QoS since their dynamic characteristics exacerbate performance and reliability behavior of underlying hardware resources. In addition, workflow tools are increasingly used for applications with more complex and diverse requirements operating in highly distributed environments that have large real-time variability. For example, scientific explorations often have uncertainties that need to be resolved during runtime either through user intervention or other rule-based mechanisms. Thus, while a general structure of the workflow is known, the exact structure of the workflow is often determined during execution. Workflows with timeliness requirements such as weather prediction, economic forecasting, hurricane track forecasting require coordination and adaptation of the resources at runtime to meet their QoS requirements. For instance, the weather prediction workflows often have stringent deadlines and arriving data determines the execution plan and hence resource requirements. Cyberinfrastructure supporting such applications needs to have proactive planning and dynamic adaptation.

Distributed architectures are moving towards a web service oriented framework [5, 64]. A service oriented framework helps define standard modular interfaces to functionality while allowing separation of concerns and policies that may be tied with distinct applications or institutions. The service oriented architecture also gives rise to a multi-layered system with the workflow, service

and resource layer each implementing distinct policies. Thus there is a need for interaction mechanisms between application layer tools and the resources for better management of the cyberinfrastructure to meet the needs of the user.

We develop WORDS (**W**orkflow **Or**chestrator for **D**istributed **S**ystems) in the context of dynamic web service workflows. The multi-level system takes a holistic view of the resource and user space and defines the interaction between them. This thesis addresses the issues of being able to provide predictable quality of service for scientific workflows in the presence of variability in the underlying system characteristics. We use the weather prediction workflows from the Linked Environments for Atmospheric Discovery (LEAD) project as the primary use case due to its timeliness and accuracy constraints.

We discuss the hypothesis and contributions of this research in Section 1.1. We provide an overview of WORDS in Section 1.2. We present an overview of the constraint space in WORDS in Section 1.3 and the resource abstraction in WORDS in 1.4. We discuss the necessity of joint analysis of performance and dependability in distributed environments in Section 1.5. We discuss the development of container based provisioning approach in leased environments in Section 1.6. We provide an overview of workflow orchestration approach for deadline sensitive workflows in dynamic and environments (Section 1.7).

1.1 Thesis Hypothesis and Contributions

As distributed environments are used to solve larger and more complex scientific and business problems from different domains, we need dynamic and adaptive elements in application tools. Higher-level user tools such as workflow engines and portal frameworks need the ability to express and enforce user specified constraints and changes. The system as a whole needs to be

flexible, resilient to both resource variability in terms of performance and reliability, as well as able to meet the end-user's dynamic requests. The dynamic aspects of next-generation workflows that are targeted to be run in the distributed environment require us to study the closer coordination of user requirements with the resource management layer. Based on the following requirements, the hypothesis is:

It is possible to design a multi-level adaptation architecture to meet performability (i.e., both performance and reliability) guarantees to support dynamic changes in workflow and resource behavior

The following are the contributions towards the above stated hypothesis:

- the WORDS system architecture in the context of a service oriented architecture that defines the interactions required between application and resource layers,
- probabilistic resource abstractions that allow higher level application techniques to be shielded from specific resource models and yet account for the variability in policy and runtime behavior,
- a constraint model that defines the conditions and expectations a user can specify on single or multiple workflows and extensions to a resource specification language to specify reliability attributes,
- performability as a metric for capturing the multi-dimensional behavior of resources in terms of performance, reliability and cost and study its impact on scheduling and fault tolerance strategies,
- workflow orchestration for deadline-senstive workflows that leverages workflow characteristics and resource behavior in the context of multi-user dynamic environments.

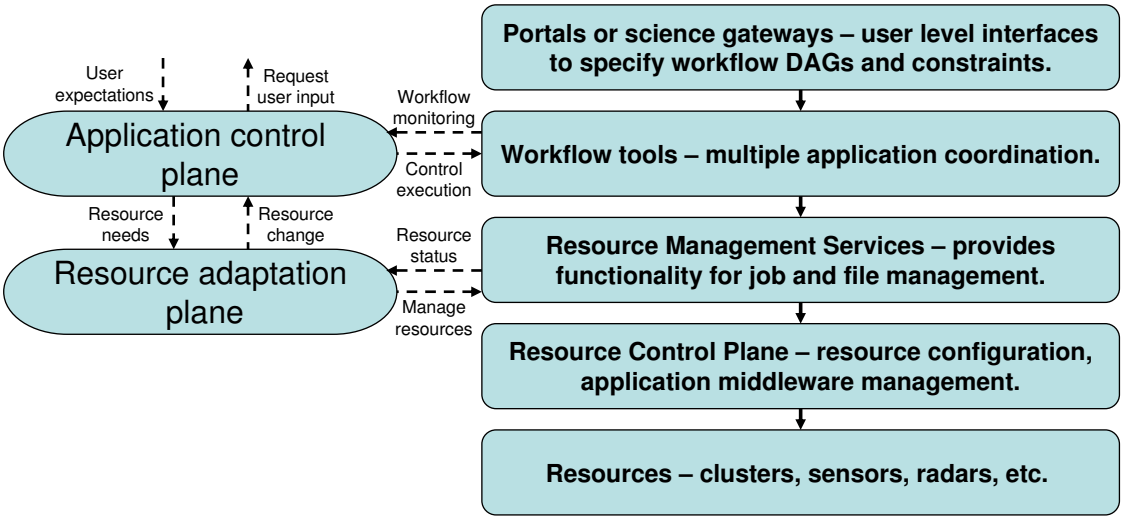


Figure 1.1: Multi-level Adaptation. The two level adaptation plane that works in conjunction with the active software hierarchy. The adaptataion plane is orthogonal to the main execution software stack. The two level plan also facilitates a clear separation of concerns between the application and resource layers.

1.2 Workflow ORchestrator for Distributed Systems (WORDS)

Distributed resource systems today are composed of a hierarchy of software systems, composed of (a) a resource management middleware that interacts with the underlying cluster, HPC resources, instruments, sensors etc. and (b) an application tools layer consisting of scientific codes, portal, workflow tools, web services. Dynamic changes from the user and/or the application impact the system top-down whereas changes in the resources - performance, reliability and availability impact the system bottom-up. To support a dynamic environment, we need *multi-level adaptation* that supports local changes while balancing the global state of the system. *Adaptation* refers to (a) coordinated planning of resources and services with workflow characteristics to meet the needs of the user, (b) ability of the workflow to adjust to resources' performance, reliability, availability variations, (c) ability to react to application, workflow or other user-initiated changes.

Figure 1.1 shows the interaction of the hierarchical adaptation planes with the software hierarchy of distributed middleware today. Users interact with web interfaces or portal environments that in turn interact with workflow tools to manage the user's execution environment. The workflows use resources where access is mediated through the resource management services. The adaptation plane is orthogonal to the main execution path and hidden from the user. A critical function of the application control plane is planning, monitoring and remediation. The resource adaptation plane manages resource interactions including resource acquisitions, monitoring and resource related remediation in consultation with the application control plane. These interactions between the end-user, multiple levels of software infrastructure and the underlying resources is a critical component for next generation *dynamic* cyberinfrastructure. The components of the WORDS system are detailed in Chapter 5.

1.3 Constraint Model for Workflows and Resource Requirements

Today's tools allow users to specify some higher level quantitative resource requirements (e.g., number of processors, processor type, etc) and the work unit dependencies through directed acyclic graphs (DAG). However in a competitive multiple service provider community, ushered in by grid and cloud computing business models, we require multiple levels of constraint specifications. First, the user must be able to provide some higher level constraints and conditions (e.g., budget, priorities, etc) on the work units to guide workflow orchestration decisions. Second, at the resource level it is important to be able to capture the qualitative resource behavior that the workflow orchestration can in turn use to meet the user's requirements.

Application level tools today have limited support to specify constraints and QoS requirements on the workflow. Scientific users want to specify various constraints on the workflow such as time

or resource constraints or priorities on parts of the workflow or relationships between different workflows. Workflow standards such as WS-BPEL [202] and languages provided by tools [36, 48, 107, 174] are not rich enough to support such constraints. We explore the constraint model that drives the resource plane interactions. The workflow constraint model is a limited set, focused on resource needs for scientific workflows, and provides a strong foundation for expansion for other needs.

Today's resource management tools provide mechanisms to select and discover resources and to manage applications' QoS requirements [205]. To guide lower level performability based workflow planning, next generation resource management tools need to provide user-level interfaces that allow users to specify performance and reliability requirements for the application. The virtual grid description language [90] is a hierarchical language for resource abstractions that allows users to specify qualitative resource performance specifications. We propose an extension to the existing virtual grid description language that enables users to specify availability requirements guided by resource cost models and budget of the user.

1.4 Resource Abstractions

Different resource models such as grid, utility or cloud computing provide standard interfaces for interaction with different types of services including job management, data transfer and other resource management functions like resource discovery. The application middleware available today is largely focused on task coordination and placement of tasks on resources based on performance. However the new resource models introduce additional burdens on the middleware: resource types are virtualized making it hard if not impossible to predict accurate execution times of applications on specific resources; additional services (e.g., advanced reservation) come at an

extra cost requiring cost considerations in the scheduling mechanism; resource arrival and departure times vary. Thus the current approach is severely limiting as we move to next generation infrastructure (e.g., cloud computing [9], GENI [69]) where business and service models are closely associated with the underlying architecture of these networked systems. End consumers have a choice of multiple service-providers and associated cost and value models that will need to be exposed at the resource level and leveraged appropriately at the application tools layer in concert with user requirements.

We explore the interaction of user level information and choices with resource management models through a resource abstraction. We propose a probabilistic QoS abstraction that allows us to capture the variability in distributed environments, with respect to resource allocation decisions as well as runtime hardware and software failures. The communication flow, through the resource abstraction, between the user's requirements and the resource model's ability to meet the requirements allows consumers to exercise various cost-benefit variations. The abstraction also gives providers the chance to provide alternatives that closely match the user's requirements. This abstraction is essential for next-generation data centers as they balance needs of different user groups with the variability in underlying resource hardware. This abstraction also gives users the ability to compare and contrast QoS capabilities of various resource providers, and thus is a foundation to a dynamic competitive market driven by user preferences.

1.5 Performability Modeling

Scientific applications have diverse performance and reliability requirements that are often difficult to satisfy, given the variability of underlying resources. Moreover, as grid and web services

continue to evolve, rapidly changing software stacks with specific configuration and service reliability challenges exacerbate application execution times and failures. Availability variations in these systems can be from hardware failures, dynamic performance variability, failures of software services or bugs in the software (data from LEAD Production workflows available in Appendix B). Today, resource selection decisions are typically made using simple resource status and performance models, despite frequent component failures [93, 205]. Simplistically, a resource can be considered to be in one of two states, either "fully-operational" or "failed." However, the diversity and the complexity of distributed environments makes graceful performance degradation in the presence of failures critical. Availability can vary due to failure of one or more critical services, load on one or more resource components, recovery from a failure, etc. These variations manifest as a loss in performance that can result in increased application execution times or as a complete failure that require rescheduling. Thus in addition to handling failures, workflow orchestration needs to account for possible loss in Quality of Service (QoS) from resource availability variations in any planning strategies

J. Meyer introduced the concept of performability [116] evaluation as a mechanism to combine performance and availability analysis when considering resource behavior. We use performability, as a composite measure of a resource's performance and dependability i.e., a measure of the system's performance in the event of failures. We present a qualitative model to capture and analyze the effect of resource reliability on application performance and cost models. Specifically, we addresses the following research challenges: (a) managing potentially conflicting resource selection goals such as performance and reliability in workflow planning (b) determining appropriate fault tolerance strategies based on application and resource characteristics.

1.6 Container Provisioning

The increasing separation between resource providers and consumers in today's grid and cloud computing environments makes resource control very important and difficult. We propose and evaluate a lease-based hosting architecture as a viable mechanism for resource providers and consumers to manage a dynamic shared pool of resources. The approach illustrates the dynamic assignment of shared pools of computing resources to hosted environments. It shows how to extend grid management services to use a dynamic leasing service to acquire computational resources and integrate them into application environments in response to changing demand. In our prototype, each user or group runs a private grid based on an instance of the Globus Toolkit (GT4) middleware running within a network of virtual machines at the provider sites. Each site controls a dynamic assignment of its local cluster resources to the locally hosted grid points of presence.

1.7 Workflow Orchestration

Workflow planning techniques so far have focused on performance based resource selection and mapping in conjunction with run-time systems handling failures and variations. However in a multi-user environment where multiple workflows are submitted by each user, priorities, budget and other constraints need to be accounted for across the workflows. Current workflow scheduling and planning methods prove to be insufficient for the additional requirements of next generation workflows.

In addition, as these workflows run in resource environments such as grid and cloud computing, it is vital to account for the diversity and variability of the resources in planning techniques. When resources are procured across a group of users or workloads, workflow planning needs to be preceded by a separate resource acquisition phase and followed up with adaptation as resource

properties change during workflow execution. We use the term *workflow orchestration* to refer to a holistic, coordinated, dynamic and adaptive approach to workflow planning that works with user requirements and variable resource characteristics while being shielded from specific resource policy or systems. In this context, workflow orchestration handles planning and execution management that includes resource acquisition, workflow scheduling and real-time adaptation. We design, develop and evaluate workflow orchestration techniques for deadline-sensitive workflows.

1.8 Thesis Outline

The rest of the thesis is organized as follows. We illustrates requirements of next-generation scientific workflows through examples from various domains in Chapter 2. Next, we discuss the characteristics of distributed systems that cater to the needs of these workflows (Chapter 3). In Chapter 4, we present related work and discuss the current status of resource management systems and workflow tools used in distributed environments. We present the WORDS system, discuss the concepts and define related terminology in Chapter 5. We present a performability model to capture performance and availability variations of distributed resources (Chapter 7). We discuss the resource abstractions and the application interactions with the resource model in Chapter 8. We detail the workflow orchestration in Chapters 9 and 10. We present our conclusions and future directions in Chapter 11.

2

Understanding Workflow Requirements Through Examples

Workflows and workflow concepts have been used to model a repeatable sequence of tasks or operations in different domains including the scheduling of manufacturing operations, inventory management, etc. The advent of internet and web services has seen the adoption of workflows as a means for business process management [175] and as an integral component of cyberinfrastructure for scientific experiments [10, 49].

Workflow tools allow users to compose and manage complex distributed computation and data in distributed resource environments. Workflows have different resource requirements and constraints associated with them. For example, application workflows with stringent deadline driven requirements such as weather prediction, economic forecasting are now increasingly run in distributed resource environments.

In this chapter, we discuss workflow examples from different domains: bioinformatics and biomedicine, weather and ocean modeling, astronomy, etc. These examples have been obtained by talking to domain scientists and computer scientists who composed or run these workflows.

Each of these workflows has been modeled using different workflow tools and sometimes the flow is even managed through scripts. For each workflow we specify the running time of applications and input and output data sizes associated with each task node. Running time of applications and data sizes for a workflow depend on a number of factors including user inputs, specific resource characteristics and run-time resource availability variations [98]. Thus our numbers are approximate estimates for typical input data sets that are representative of the general characteristics of the workflow.

The rest of the chapter is organized as follows. We present an overview of the projects that were part of our survey in Section 2.1. The weather and ocean modeling workflows are detailed in Section 2.2. Next, in Section 2.3, we describe the bioinformatics and biomedicine workflows. We describe the astronomy and neutron science and computer science examples in Sections 2.4 and 2.5. We discuss the use case scenarios and the characteristics of the workflow in Section 2.6 and finally summarize our survey in Section 2.7.

2.1 Overview

Table 2.1 presents an overview of the survey that included workflows from diverse scientific domains and cyberinfrastruture projects. In the following sections, we provide a brief description of the project, workflow and usage model of the workflows as available today. For each workflow we specify the running time of applications and input and output data sizes associated with each task node. Running time of applications and data sizes for a workflow depend on a number of factors including user inputs, specific resource characteristics and run-time resource availability variations [98]. Thus our numbers are approximate estimates for typical input data sets that are representative of the general characteristics of the workflow. For each of the workflows, we also

Domain	Project	Website	Tool
Weather and Ocean Modeling	Linked Environments for Atmospheric Discovery (LEAD), TeraGrid Science Gateway	http://portal.lead.project.org	xbaya, GPEL, Apache ODE
	Southeastern Coastal Ocean Observing and Prediction Program (SCOOP)	http://www.renci.org/focusareas/disaster/scoop.php	[Scripts]
	North Carolina Floodplain Mapping Program		[Scripts]
Bioinformatics and Biomedical	North Carolina Bioportal, TeraGrid Bioportal Science Gateway	http://www.renci.org/focusareas/biosciences/motif.php	Taverna
	MotifNetwork	http://www.motifnetwork.org/	Taverna
	National Biomedical Computation Resource (NBCR), Avian Flu Grid, Pacific Rim Application and Grid Middleware Assembly	http://nbcr.sdsc.edu/ http://gemstone.mozdev.org http://www.pragma-grid.net/ http://avianflugrid.pragma-grid.net/ http://mgltools.scripps.edu/	Kepler, Gemstone, [Scripts] and Vision
	cancer Biomedical Informatics Grid (caBIG)	http://www.cagrid.org/	Taverna
Astronomy	Pan-STARRS	http://pan-starrs.ifa.hawaii.edu/public/, http://www.ps1sc.org/	
Neutron Science	Spallation Neutron Source (SNS), Neutron Science TeraGrid Gateway (NSTG)	http://neutrons.ornl.gov/	

Table 2.1: Workflow Survey Project Overview. We surveyed workflows from different domains including weather and ocean modeling, bioinformatics and biomedical workflows, astronomy and neutron science. The table shows the project information and tools used by the scientists.

provide a DAG representation of the workflow annotated with computation and data sizes.

2.2 Weather and Ocean Modeling Workflows

In the last few years the world has seen a number of severe natural disasters such as hurricanes, tornadoes, floods, etc. The models used to study weather and ocean phenomenon use real-time observational data in conjunction with a number of parameters that are varied to study the possible scenarios for prediction. In addition the models must be run in a timely manner and information disseminated to disaster response agencies. This creates the need for *large scale modeling* in the areas of meteorology and ocean sciences, coupled with an *integrated environment* for analysis, prediction and information dissemination. A number of cyberinfrastructure projects are building tools and constructing workflows to facilitate next-generation weather and ocean modeling science.

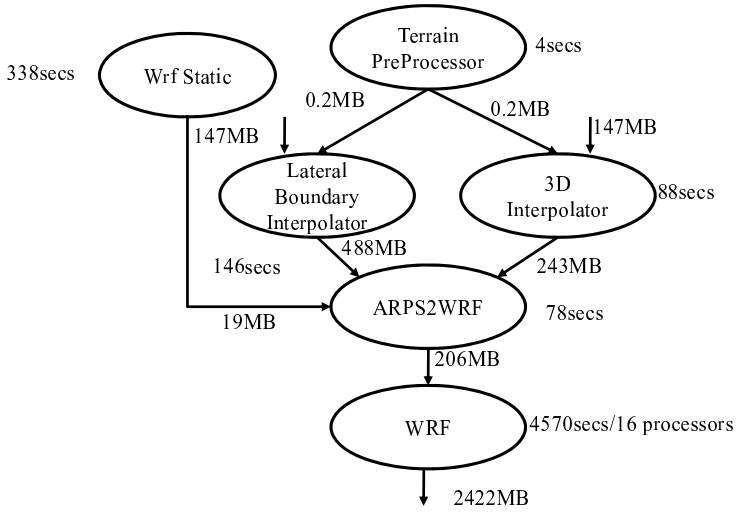


Figure 2.1: LEAD North American Mesoscale (NAM) Initialized Forecast Workflow. The workflow processes terrain and observation data to produce weather forecasts.

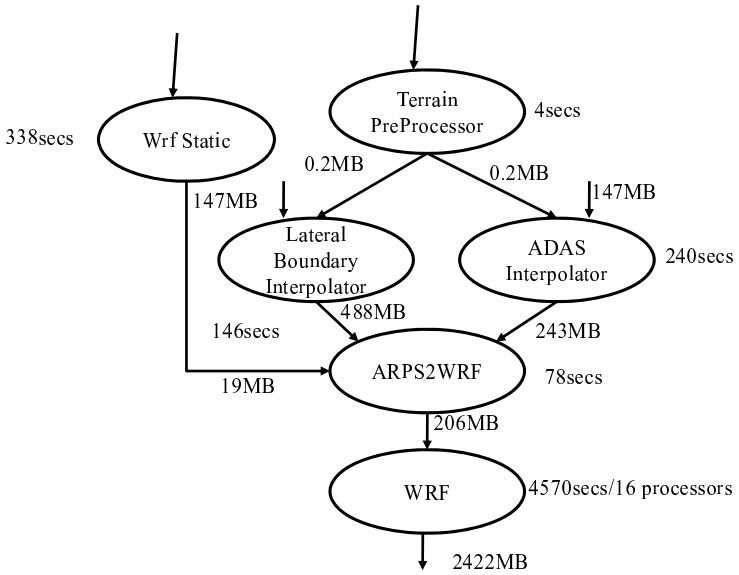


Figure 2.2: LEAD ARPS Data Analysis System(ADAS) Initialized Forecast Workflow. The workflow processes terrain and observation data to produce weather forecasts.

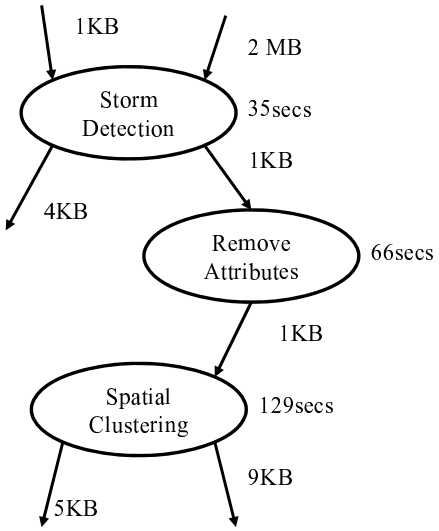


Figure 2.3: LEAD Data Mining Workflow. The workflow processes weather data to identify regions where weather phenomenon is present.

2.2.1 Mesoscale Meteorology

The Linked Environments for Atmospheric Discovery (LEAD) [54] is a cyberinfrastructure project that supports mesoscale meteorology. The infrastructure of LEAD needs to support real-time dynamic, adaptive response to severe weather. A LEAD service workflow has constraints on execution time and accuracy due to weather prediction deadlines. The typical inputs to a workflow of this type are streaming sensor data [135] that must be pre-processed and then used to launch an ensemble of weather models. The model outputs are processed by a data mining component that determines whether some ensemble set members must be repeated to realize statistical bounds on prediction uncertainty. Figures 2.1, 2.2 and 2.3 show the workflows available through the LEAD portal that include weather forecasting and data mining workflows [102]. Each workflow task is annotated with computation time and the edges of the directed acyclic graph (DAG), that represent the data dependencies between the tasks, are annotated with file sizes. The weather forecasting workflows are largely similar and vary only in their preprocessing or initialization step. While the data mining workflow can be run separately today, it can trigger forecast workflows or steer remote radars for additional localized data in regions of interest [135].

2.2.2 Storm Surge Modeling

Southeastern Universities Research Association's (SURA) Southeastern Coastal Ocean Observing and Prediction (SCOOP) program is a distributed project that is creating an open-access grid environment for the southeastern coastal zone to help integrate regional coastal observing and modeling systems [159, 138].

Storm surge modeling requires assembling input meteorological and other data sets, running

models, processing the output and distributing the resulting information. In terms of modes of operation, most meteorological and ocean models can be run in hindcast or forecast modes. The hindcast mode, initiated by a user, is used as an after fact of a major storm or hurricane, for post-analysis or risk assessment. The forecast mode is used for prediction to guide evacuation or operational decisions [138] and is driven by real-time data streams. Often it is necessary to run the model with different forcing conditions to analyze forecast accuracy. This results in a large number of parallel model runs, creating an ensemble of forecasts. Figure 2.4 shows a five member ensemble run of tidal and storm-surge ADCIRC model. ADCIRC is a finite element model that is parallelized using Message Passing Interface (MPI) [108]. For increased accuracy of forecast the number of concurrent model runs is increased. The workflow has a predominantly parallel structure and the results are merged in the final step.

The SCOOP ADCIRC workflows are launched according to the six hour synoptic forecast cycle used by the National Weather Service and the National Centers for Environmental Prediction (NCEP). NCEP computes an atmospheric analysis and forecast four times per day at six hour intervals. Each of the member runs i.e. each branch of the workflow gets triggered when wind files arrive through Local Data Manager (LDM) [190], an event-driven data distribution system that selects, captures, manages and distributes meteorological data products. The outputs from the individual runs are synthesized to generate the workflow output that is then distributed through LDM.

In the system today each arriving ensemble member is handled separately through a set of scripts and Java code [138]. The resource selection approach [99] makes a real-time decision for each model run and uses knowledge of scheduled runs to load-balance across available systems. However this approach does not have any means of guaranteeing desired QoS in terms of completion time.

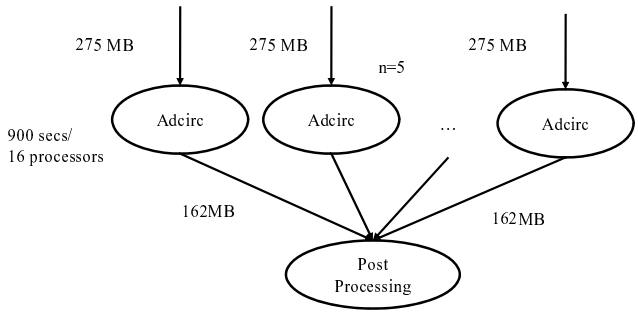


Figure 2.4: SCOOP Workflow. The arriving wind data triggers ADCIRC that is used for storm-surge prediction during hurricane season.

2.2.3 Floodplain Mapping

The North Carolina Floodplain Mapping Program [123, 19] is focused on developing accurate simulation of storm surges in the coastal areas of North Carolina. The deployed system today consists of a four-model system that consists of the Hurricane Boundary Layer (HBL) model for winds, WaveWatch III and SWAN for ocean and near-shore wind waves, and ADCIRC for storm surge. The models require good coverage of the parameter space describing tropical storm characteristics in a given region for accurate flood plain mapping and analysis. Figure 2.5 shows the dynamic portion of the workflow. Forcing winds for the model runs are calculated by the Hurricane Boundary Layer (HBL) model that serve as inputs to the workflow. The HBL model is run on a local commodity linux cluster. Computational and storage requirements for these workflows are fairly large requiring careful resource planning. An instance of this workflow is expected to run for over a day. The rest of the workflow today runs on RENCI's Bluegene system [149].

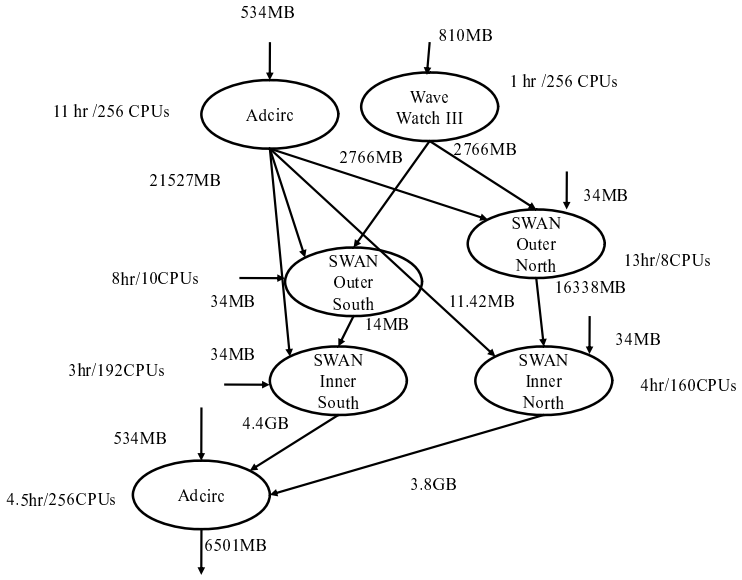


Figure 2.5: NCFS workflow. A multitude of models are run to model the storm surges in the coastal areas of North Carolina.

2.3 Bioinformatics and Biomedical workflows

The last few years have seen large scale investments in cyberinfrastructure to facilitate bioinformatics and biomedical research. The infrastructure allows users to access databases and web services through workflow tools and/or portal environments. We surveyed three major projects - North Carolina Bioportal, cancer Biomedical Informatics Grid (caBIG), and National Biomedical Computational Resource (NBCR) to understand the needs of this class of workflows. Significant number of these workflows involve small computation but involve access to large-scale databases that need to be preinstalled on available resources. While the typical use cases of today have input data sizes in the order of megabytes, it is anticipated that in the future data sizes might scale to gigabytes.

2.3.1 Glimmer

The North Carolina Bioportal and The TeraGrid Bioportal Science Gateway [142] provides access to about 140 bioinformatics applications and a number of databases. Researches and educators use the applications interactively for correlation, exploratory genetic analysis, etc. The Glimmer workflow is one such example workflow that is used to find genes in microbial DNA (Figure 2.6). The Glimmer workflow is sequential and light on both compute and data.

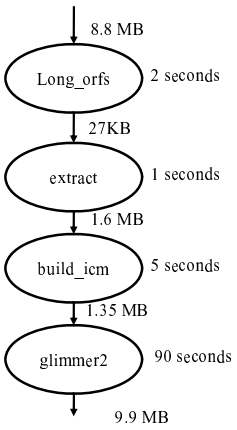


Figure 2.6: Glimmer Workflow. A simple workflow used in educational context to find genes in microbial DNA.

2.3.2 Gene2Life

Let us consider the Gene2Life workflow used for molecular biology analysis. This workflow takes an input DNA sequence, searches databases to find genes matching the sequence. It globally aligns the results and attempts to correlate the results based on organism and function. Figure 2.7 depicts the steps of the workflow and the corresponding output at each stage. In this workflow the user provides a sequence that can be a nucleotide or an amino acid. The input sequence performs two parallel BLAST [4] searches, against the nucleotide and protein databases respectively. The

results of the searches are parsed to determine the number of identified sequences that satisfy the selection criteria. The outputs trigger the launch of ClustalW, a bioinformatics application that is used for the global alignment process to identify relationships. These outputs are then passed through parsimony programs for analysis. The two applications that may be available for such analysis are dnapars and protpars. In the last step of the workflow plots are generated to visualize the relationships, using an application called drawgram. This workflow has two parallel flows.

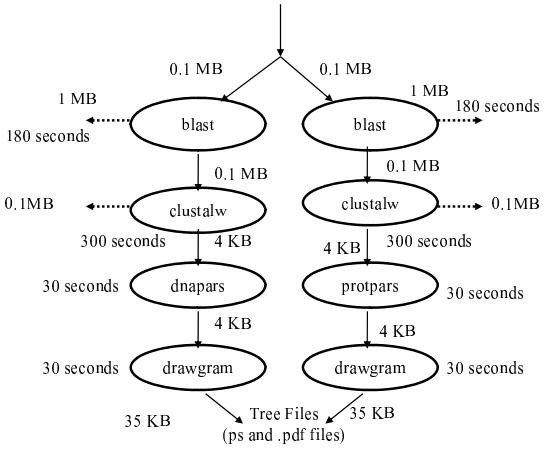


Figure 2.7: Gene2Life Workflow. The workflow is used for molecular biology analysis of input sequences. The dotted arrows show the intermediate products from this workflow that are required by the user and/or might be used to drive other scientific processes.

2.3.3 Motif Network

The MotifNetwork project [184, 185] is building a software environment to provide access to domain analysis of genome sized collections of input sequences. The MotifNetwork workflow (Figure 2.8) is computationally intensive. The first stage of the workflow assembles input data and processes the data that is then fed into Interproscan service. The concurrent executions of InterProScan is handled through Taverna and scripts. The results of the domain "scanning" step are passed to an MPI code for the determination of domain architectures. The motif workflow has a

parallel split and merge paradigm where preprocessing spawns a set of parallel tasks that operate on subsets of the data. Finally, the results from the parallel tasks are merged and feed into the multi-processor application.

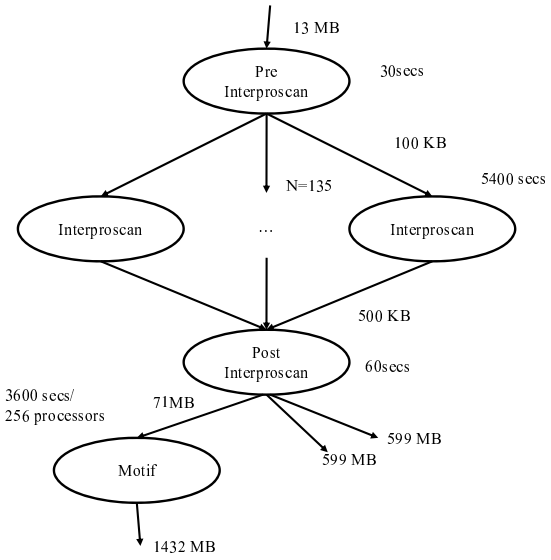


Figure 2.8: Motif Workflow. A workflow used for motif/domain analysis of genome sized collections of input sequences.

2.3.4 MEME-MAST

The goal of National Biomedical Computation Resource(NBCR) is to facilitate biomedical research by harnessing advanced computational and information technologies. The MEME-MAST (Figure 2.9) workflow deployed using Kepler [3, 107] allows users to discover signals or motifs in DNA or protein sequences and then search the sequence databases for the recognized motifs. This is a simple workflow often used for demonstration purposes. The workflow is a sequential workflow similar to Glimmer.

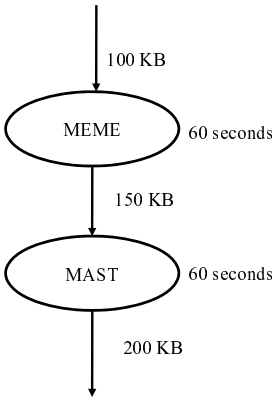


Figure 2.9: MEME-MAST Workflow. A simple demonstration workflow used to discover signals in DNA sequences.

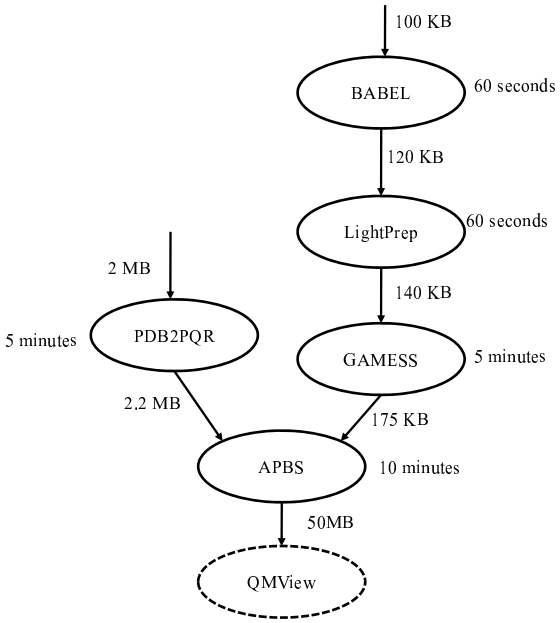


Figure 2.10: Molecular Sciences Workflow. The workflow is used to study atomic structures of proteins and ligands.

2.3.5 Molecular Sciences

An important process in the drug-design process is understanding the three-dimensional atomic structures of proteins and ligands. The Gemstone project, a client interface to a set of computational chemistry and biochemistry tools, provides the NBCR community access to a set of tools that allows users to analyze and visualize atomic structures. Figure 2.10 shows an example molecular science workflow. The workflow in its current incarnation runs in an interactive mode where each step of the workflow is manually launched by the user once the previous workflow task finishes. The first few steps of the workflow involve downloading the desired protein and ligand from the Protein Data Bank (PDB) database and converting it to a desired format. Concurrent preprocessing is done on the ligand using the Babel and LigPrep services. Finally GAMESS and APBS are used to analyze the ligand and protein. The results are finally visualized using the QMView which is done as an offline process. First few steps have small data and small compute and finally produce megabytes of data.

2.3.6 Avian Flu

The Avian Flu Grid project is developing a global infrastructure for the study of Avian Flu as an infectious agent and as a pandemic threat. Figure 2.11 shows a workflow that is used in drug design. It is used to understand the mechanism of host selectivity and drug resistance. The workflow has a number of small preprocessing steps followed by a final step where up to 1000 parallel tasks are spawned. The data products from this workflow are small.

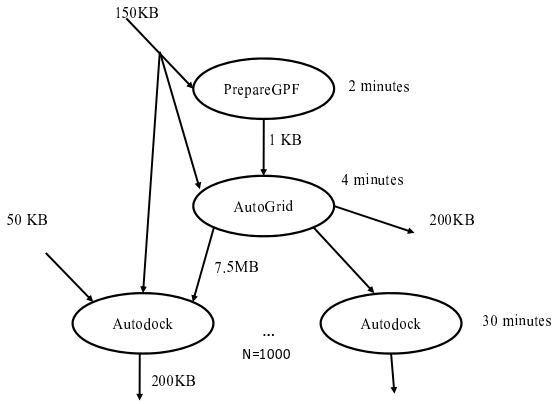


Figure 2.11: Avian Flu Workflow. A workflow used in drug design to study the interaction of drugs with the environment.

2.3.7 caDSR

The cancer Biomedical Informatics Grid(caBIG) is a virtual infrastructure that connects scientists with data and tools towards a federated cancer research environment. Figure 2.12 shows a workflow using the caDSR (Cancer Data Standards Repository) and EVS (Enterprise Vocabulary Services) services [28] to find all the concepts related to a given context. The caDSR service is used to define and manage standardized metadata descriptors for cancer research data. EVS in turn facilitates terminology standardization across the biomedical community. This workflow is predominantly a query type workflow and the compute time is very small in the order of seconds.

2.4 Astronomy and Neutron Science Workflows

In this section, we consider scientific workflow examples from the astronomy and neutron science community.

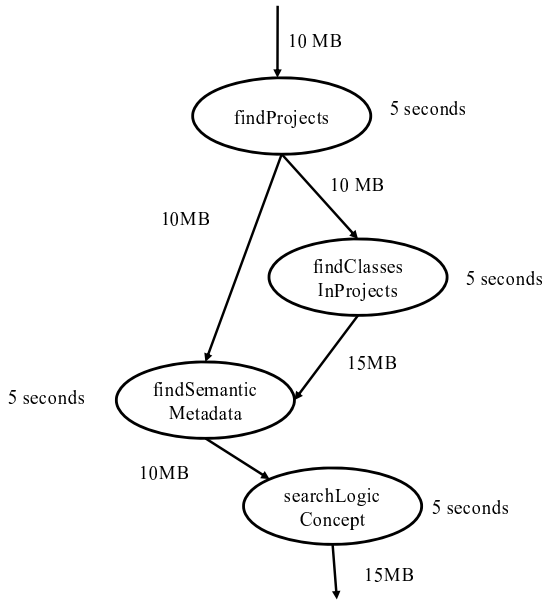


Figure 2.12: Cancer Data Standards Repository Workflow. A workflow used to query concepts related to an input context.

2.4.1 Pan-STARR

The goal of the Pan-STARRS's (Panoramic Survey Telescope And Rapid Response System) project [57] is a continuous survey of the entire sky. The data collected by the currently deployed prototype telescope 'PS1' will be used to detect hazardous objects in the Solar System, and other astronomical studies including cosmology and Solar System astronomy. The astronomy data from Pan-STARRS is managed by the teams at John Hopkins University and Microsoft Research through two workflows. The first PSLoad workflow (Figure 2.13) stages incoming data files from the telescope pipeline and loads them into individual relational databases each night. Periodically the online production databases that can be queried by the scientists, are updated with the databases collected over the week by the PSMerge workflow (Figure 2.14). The infrastructure to support the PS1 telescope data is still under development. Both the Pan-STARRS workflows are data intensive but require coordination and orchestration of resources to ensure reliability and integrity of the

data products. The workflows have a high degree of parallelism achieved by working on small subsets of the data.

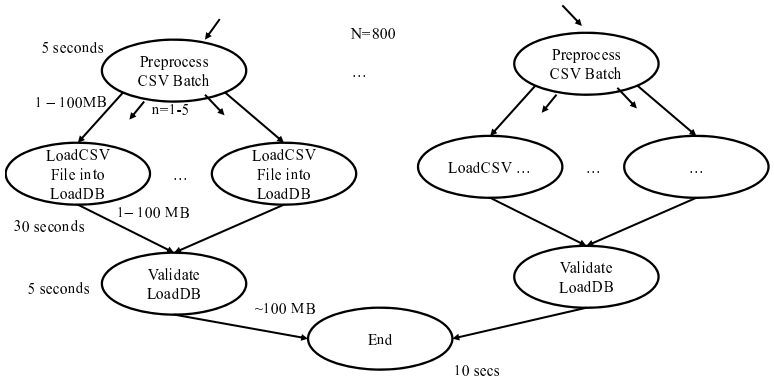


Figure 2.13: PSLoad Workflow. Data arriving from the PS1 telescope is processed and staged in relational databases each night.

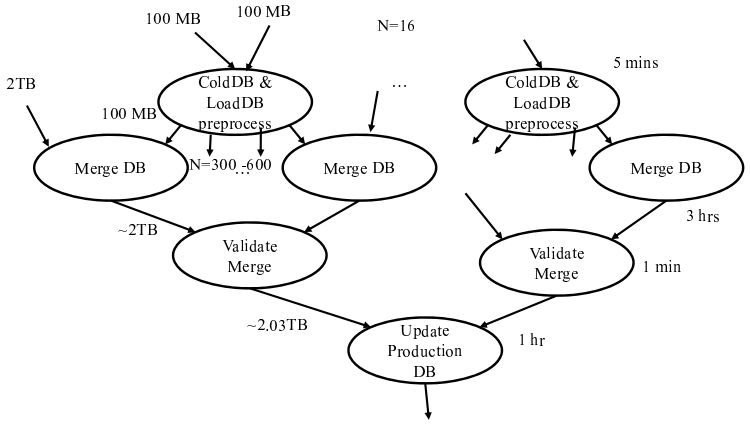


Figure 2.14: PSMerge Workflow. Each week, the production databases that astronomers query are updated with the new data staged during the week.

2.4.2 McStas workflow

Neutron science research enables study of structure and dynamics of molecules that constitute materials. Neutron Source at Oak Ridge National Laboratory connect large neutron science facilities that contain instruments with computational resources such as the TeraGrid [109]. The Neutron

Science TeraGrid Gateway enables virtual neutron scattering experiments. These experiments simulate a beam line and enables experiment planning and experimental analysis. Figure 2.15 shows a virtual neutron scattering workflow using McStas, VASP, and nMoldyn. VASP and nMoldyn are used for molecular dynamics calculations and McStas is used for neutron ray-trace simulations. The workflow is computationally intensive and currently runs on ORNL supercomputing resources and TeraGrid resources. The initial steps of the workflow run for a number of days and are then followed by an additional compute intensive step. The workflow is sequential and has small data products.

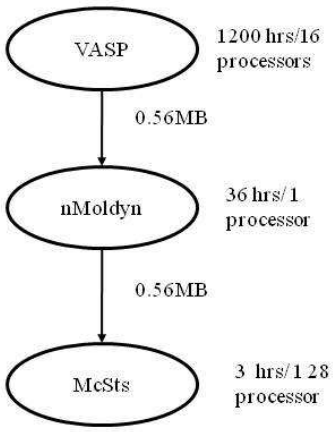


Figure 2.15: McStats Workflow. This workflow is used for Neutron ray-trace simulations.

2.5 Computer Science Examples

Workflow tools are increasingly being used in different scenarios both in scientific as well as business processes. In addition programming constructs such as map and reduce facilitate problems to be composed as distinct work units with stated dependencies. In this section, we explore some examples that illustrate workflows whose users are often computer scientists or programmers.

2.5.1 Animation

Rendering computer animation frames is fairly time consuming. Distributed rendering on multiple processors has been known to provide significant speedups over running on a single processor [39]. The animation workflow is based on distributed rendering that is commonly used today for frame generation. The animation workflow has map-reduce style programming model where work is distributed and the results are gathered and synthesized for the final result. The computational and data sizes are rough numbers used for illustration [35, 206].

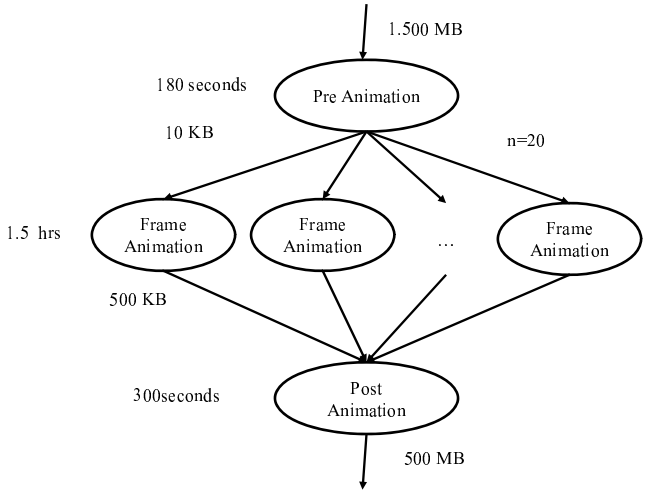


Figure 2.16: Animation Workflow. The rendering work is distributed across a multitude of nodes.

2.5.2 Performance Measurement

Applications running in distributed environments like Grid and cloud computing resources often experience significant changes in performance. Benchmarking and performance experiments are often critical in these environments to determine the best binary for a given set of resources. Tilson et al. [186] describe a way to use workflow tools to facilitate the benchmarking of a large number of variable parameters including compiler, link and runtime flags (Figure 2.17).

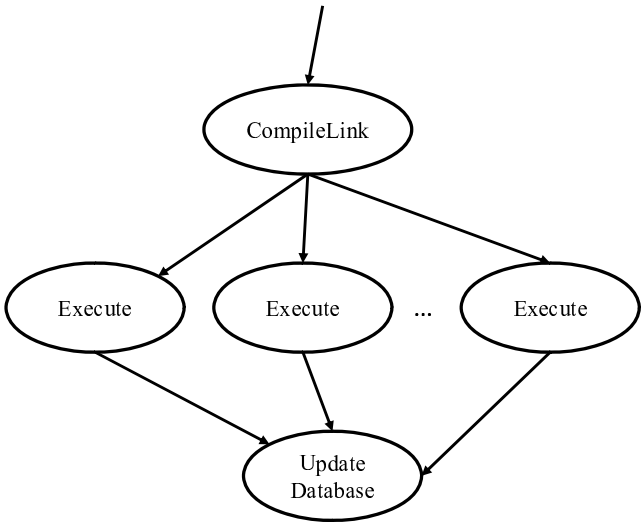


Figure 2.17: Performance Measurement Workflow. The workflow is used for benchmarking applications with various compiler, link and runtime flags.

2.5.3 Load Balancing

Recent computing models have resulted in application middleware investigating mechanisms to dynamically manage the resource pool. Cloud computing services such as Amazon EC2 [5] allow users and applications to increase their resource pool on increased load and decrease the number of resources when the load drops. When considering the load from different users or applications that use a defined resource pool we can consider the entire load managed by the middleware to be a "workflow of workflows" where the task dependency is based on number of concurrent resources available. For example if there are four independent tasks(Figure 2.18) and just one resource the workflow would be a simple sequential workflow. However if there were two resources available, two tasks would run and then subsequently the remaining two tasks would run. Similarly if three resources were available, three tasks would initially execute in parallel. A similar strategy would be followed for workflows where in addition to the workflow dependencies, execution dependency is created between two tasks that need to run on the same resource (shown by dotted lines). In

Figure 2.18 three workflows are scheduled on three processors. In this case the head nodes of the workflow are scheduled on the workflows. Subsequently, the two parallel tasks from workflow *a* is scheduled with one of the parallel tasks from workflow *b*. In this case, there is an execution dependency between workflow *b*'s second task and the first task from workflow *c*.

In a more general case consider a cloud computing application that procures more resources as the load increases and reduce the number of resources as the load decreases. Thus the resources procured or allotted can be represented as a workflow task graph where each node in the graph represents the resource slot (Figure 2.19).

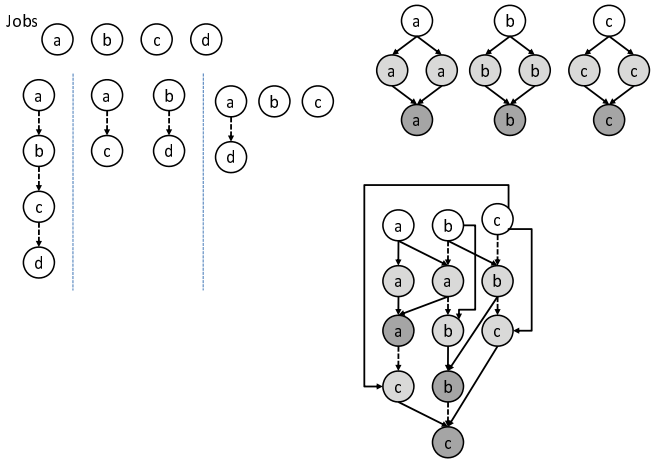


Figure 2.18: Load balancing workflow. When jobs or workflows are scheduled on resources, a dependency is created from the resource availability constraint. In the left side of the figure, we show how jobs a, b, c, d are scheduled on one, two or three processors. When scheduled on one processor, the jobs get mapped sequential resulting in a virtual dependency where job b must wait for job a to finish. Similarly for workflows, if we were to schedule them on three processors, in addition to their workflow task dependency, their execution dependency is determined by the execution of one or more of the tasks from other workflows.

2.6 Discussion

In this chapter, we have presented a number of workflows from different domains. The workflows have varying requirements and constraints. In this section we provide a high level discussion

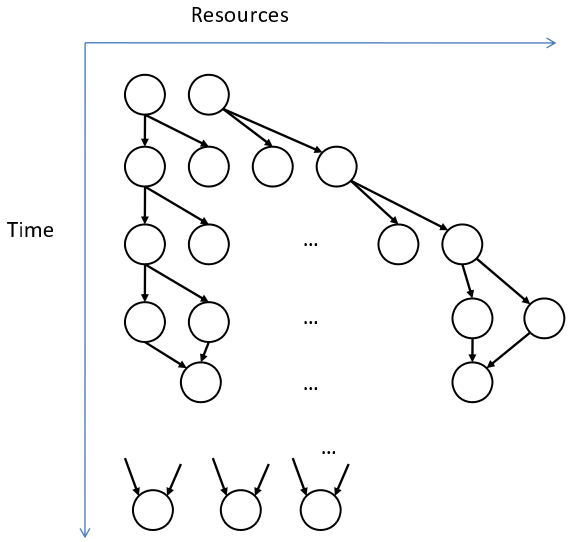


Figure 2.19: Resource profile as a workflow. A dynamic application manager might procure resources as load increases and release resources as load falls below a threshold. The resource profile over time can be represented as a workflow structure.

on use case scenarios and workflow characteristics. Additionally, the workflow examples demonstrate the required support in next-generation workflow and resource management tools to support dynamic and cloud computing environments.

2.6.1 Use case scenarios

It is often important to understand the use case scenarios for the workflows. Workflows are used in a number of different scenarios - a new workflow may be initiated in response to dynamic data or a number of workflows may be launched as part of an educational workshop. In addition, the user may want to *specify constraints* to adjust the number of worklows to run based on resource availability [143].

User-initiated workflows. The typical mode of usage of science cyberinfrastructure is where a user logs into the portal and launches a workflow for some analysis. The user selects a pre-composed

workflow and supplies the necessary data for the run. In this scenario, we need mechanisms to procure resources and enable workflow execution, provide recovery mechanisms from persistent and transient service failures and adapt to resource availability or recover from resource failures during workflow execution. The user may also want the ability to pause the workflow at the occurrence of a predefined event, inspect intermediate data and make changes during workflow execution. The LEAD (Section 2.2), bioinformatics and biomedicine (Section 2.3) workflows are all user-initiated workflows through portal environments.

Workflow priorities Let us consider a scenario of an educational workshop with multiple competing users. Resources are typically reserved for this event through out-of-band mechanisms for advanced reservation. In this scenario resource allocation needs to be based on existing load on the machines, resource availability, the user priorities and workflow load. The bounded set of resources available to the workshop needs to be proportionally shared among the workflow users. If there is a weather event during the workshop, resources need to be reallocated and conflicting events need some arbitration. The LEAD (Section 2.2), bioinformatics and biomedical (Section 2.3) workflows are also used in education workshops with competing user needs.

Dynamic Event. A number of scientific workflows get triggered by newly arriving data. Multiple dynamic events and their scale need priorities between users for appropriate allocation of limited available resources. Resources must be allocated to meet deadlines. Additionally, to ensure successful completion of tasks, we might need to replicate some of the workflow tasks for increased fault tolerance. It is possible that with advance notice of upcoming weather events, we may want to anticipate the need for resources and try to procure them in advance. The weather forecasting, storm surge modeling (Figure 2.4), flood-plain mapping (Figure 2.5) and the astronomy workflows (Figures 2.13 and 2.14) are launched with the arrival of data.

Advanced User Workflow Alternatives and Constraints. An advanced user may to provide a set of

constraints (e.g. time deadline) on a workflow. Scientific processes such as weather prediction, financial forecasting have a number of parameters and computing an exact result is often impossible. There is a need to run multiple workflows (i.e. *workflow sets*) that need to be scheduled together. It is often necessary to run a minimal number of the workflows for confidence in the result. Thus for workflow sets, users specify that they minimally require M out of N workflows to complete by the deadline. Thus in the weather forecasting workflow, the user specifies that fewer parallel ensemble members could be run to get a quicker result. Alternatively the user may be willing to sacrifice forecast resolution to get some early results which then define the rest of the workflow.

These scenarios illustrate the need for an adaptation framework that implements *online planning and control of workflows* to assess resource needs, proactively adapt to failures and workflow needs based on priorities and policies specified by the user.

2.6.2 Workflow Types

The workflows described in this chapter vary significantly in their computational and data requirements. A number of the bioinformatics workflows often have tasks that are based on querying large databases in order of minutes for the task execution. In other cases we see each of the tasks of a workflow require computation time on the order of hours or days on multiple processors. In some cases sub-parts of the workflow may also present different characteristics. In addition, the sizes of the intermediate data products also vary. Workflow management strategies for each of these workflows can vary and thus require the understanding of the workflow to apply appropriate techniques. We consider the characteristics that help classify workflow types in this section.

Structure. The size of the workflow is an important characteristic to determine resource requirements, etc. We consider the tasks of the workflow as its structural characteristic. The size of the workflows that are deployed today in most production environments are relatively small. The

largest workflows in our set contain hundreds of independent tasks. The Avian Flu (Figure 2.11) and PanSTARRS(Figures 2.13 and 2.14) workflows has over a thousand nodes but the computation at each node is expected to take only a few minutes to an hour. Scientists express a need to run larger sized workflows but are often limited by available resources or workflow tool features that are needed to support such large-scale workflows. Today, workflow tools have limited composition support for large workflows - ability to specify repeated tasks, display parts of a workflow, etc. In addition, they have little or no support to specify resource requirements, conditions or other constraints on part or the entire workflow. It is also often difficult in grid environments today to scale workloads up or down due to batch queue wait times and other factors. In addition to the total number of tasks in a workflow it is also important to consider the width and length of the workflows. The width of the workflow (i.e. maximum number of parallel branches) determines the concurrency possible and the length of the workflow characterizes the makespan (or turnaround time) of the workflow. We observe that in our workflow examples, the larger sized workflows such as the motif workflow (Figure 2.8) and the astronomy workflows (Figures 2.13 and 2.14) the width of the workflow is significantly larger than the length of the workflow.

Pattern. The workflows that we surveyed depict the basic control flow patterns such as sequence, parallel split, synchronization [192]. The parallel split-synchronization pattern has similarities to the map-reduce programming paradigm. A number of workflows divide the work units into distinct work units and the results are then combined - e.g. Animation (Figure 2.16), Motif workflow (Figure 2.8), Pan-STARRS workflows (Figures 2.13 and 2.14).

Computation. In addition to the structure and pattern of a workflow it is important to understand the computational requirements. In our workflow examples we observe that computational time required by the workflows can vary from a few seconds to several days. A number of the

bioinformatics workflows depend on querying large databases and have small compute times including the Glimmer workflow (Figure 2.6), Gene2Life (Figure 2.7), caDSR (Figure 2.12). Similarly the initial parts of the LEAD forecast workflow (Figures 2.1 and 2.2) and the LEAD data mining workflows (Figure 2.3) have small computational load. A number of the workflows including the forecasting parts of the LEAD workflow, Pan-STARRS workflows (Figures 2.13 and 2.14), SCOOP (Figure 2.4), SNS (Figure 2.15), Motif (Figure 2.8), NCFS (Figure 2.5) have medium to large sized compute requirements.

Data. The workflows are associated with different types of data including input data, backend databases, intermediate data products, output data products. A large number of the bioinformatics applications often have small input and small data products but often rely on huge backend databases that are queried as part of task execution. These workflows require that the databases be pre-installed on various sites and resource selection is often based on selecting the resources where the data is available. Workflows such as LEAD (Figures 2.1 and 2.2), SCOOP (Figure 2.4), NCFS (Figure 2.5) and Pan-STARRS workflows (Figures 2.13 and 2.14) have fairly large sized input, intermediate and output data products. The Glimmer workflow (Figure 2.6) has similar sized input and output data products but its intermediate data products are smaller. In today's production environments workflows often compress data products to reduce transfer times through intermediate scripts etc. When scheduling workflows on resources, a number of data issues need to be considered including the availability of the required data as well as the data transfer time of both input and output products.

The combination of the structural and pattern characteristics, the computational and data sizes helps in understanding the workflow requirements when making planning and adaptation decisions. We present a workflow analysis approach to determine workflow characteristics that affect resource planning and adaptation decisions in Chapter 9

2.6.3 Multiple workflows

The user interacts with applications through various portal and graphical interfaces for workflow tools. Workflow management techniques today are focused on managing single workflows in a distributed environment [112, 205]. However portal environments facilitate simultaneous multi-user access to the same workflows and underlying resources. In addition, a number of scientific explorations including the weather and ocean modeling workflows (Section 2.2) often require a large number of parallel runs to be launched to study different parameters to increase result accuracy.

Competing workflows. Portal and gateway environments allow a number of workflows from different users to be launched simultaneously. In such cases workflows from different users are often competing for the same resource. For example, a LEAD forecasting workflow will need to have higher priority than a workflow launched by a user in an educational workshop. Workflow management techniques needs to account for the different classes of workflow users when allocating resources.

Data sharing and reuse. When multiple workflows exist in the system, there is an opportunity to save computational time by reusing data products from identical executions [47]. However in these situations it is also important to manage data privacy concerns when managing data products from potentially competing workflows.

Workflow set. Scientists often conduct parametric or exploratory studies that involve launching multiple parallel workflows. The workflows might share data products between them or use the same set of resources. We use the term *workflow set* to refer to workflows that need to be scheduled together to meet their relationship constraint such as data dependencies or accuracy constraint. In addition, there are workflows from different users which have the same priority and similar

constraints requiring them to be scheduled such as to ensure fairness. There is limited capabilities to be able to ensure such policies in the workflow engines available today.

Thus we need tools and mechanisms to manage competing workflows or workflow sets in a system. Workflow tools will need to support the multiple workflow scenario or "workflow of workflows". In addition, as we move to more dynamic resource environments such as cloud systems, usage of tools such as the Dryad execution engine [84] or MapReduce [45] to manage multiple workflow execution must be studied.

2.6.4 Workflow Capabilities

Workflow tools have limited capabilities today to allow users to specify constraints and other expectations from their workflows. We outline the capabilities that users might need in workflow composition tools.

Exploratory. Scientific explorations often have uncertainties that need to be resolved during runtime. Input data sizes can vary largely affecting the characteristics of the workflow. In a number of explorations scientists and their workflows interact with real-time data collecting instruments such as the Large Hadron Collider (LHC) [100], sensors, radars [54, 135], etc. Thus in these cases while a general structure of the workflow is known, the exact characteristics of the workflow is determined during execution.

Interactive. Business workflows and scientific explorations often require a "human-in-the-loop" as part of the workflow. Workflow management techniques often have to consider sub-parts of the workflow for scheduling and adaptation.

Constraints. In addition to the workflow description, users often need to specify various constraints on the workflow. The weather and ocean modeling workflows (Section 2.2) are time-sensitive. The workflow results must be obtained in advance for weather response agencies to be take appropriate action. In addition the cost of resources (either allocation seconds on TeraGrid or real dollars on resources such as Amazon EC2) might be a consideration for the end user.

2.6.5 Resource coordination.

Scientific workflows largely run in batch queue based grid environments and business workflows run on monolithic corporate systems. However the advent of utility and cloud computing systems change the mode of operation of scientific processes. Cloud computing systems allows users to customize software environments allowing workflow tools to be able to manage application specific software and data on the resources. In addition procuring resources in advance for later workflow steps can be achieved with the new resource access mechanisms thus minimizing workflow makespan by reducing resource wait times. Thus new mechanisms are required in workflow and resource management tools.

2.7 Summary

In this chapter, we investigated workflows from various domains that have different structures, computational and data requirements. We summarize the results of the workflow survey and their characteristics in Table 2.2.

We consider the structural, computational and data aspects of the workflows. The total number of tasks and the number of parallel tasks are useful in understanding the structure of the workflow. The workflows in our survey vary from a handful of tasks to thousands of components. The

Workflow Name	Total no. of tasks	Max width	Max task processor width	Computation	Data sizes	Pattern
LEAD Weather Forecasting	6	3	16	hours	megabytes to gigabytes	Sequential
LEAD Data Mining	3	1	1	minutes	kilobytes	Sequential
Storm Surge	6	5	16	minutes-hours	megabytes	Parallel-merge
Flood-plain mapping	7	2	256	days	gigabytes	Mesh
Glimmer	4	1	1	minutes	megabytes	Sequential
Gene2Life	8	2	1	minutes	kilobytes to megabytes	Parallel
Motif	138	135	256	hours	megabytes to gigabytes	Parallel-split
MEME-MAST	2	1	1	minutes	kilobytes	Sequential
Molecular Sciences	6	2	1	minutes	megabytes	Parallel-merge
Avian Flu	$\sim$ 1000	1000	1	minutes	kilobytes to megabytes	Parallel-split
caDSR	4	1	1	seconds	megabytes	Sequential
PanSTARRS Load	$\sim$ 1600 - 41000	800 - 40000	1	minutes	megabytes	Parallel-split-merge
PanSTARRS Merge	$\sim$ 4900 - 9700	4800 - 9600	1	hours	gigabytes to terabytes	Parallel-split-merge
McStats	3	1	128	days	kilobytes to megabytes	Sequential

Table 2.2: Summary of Workflow Characteristics. The total number of tasks and the number of parallel tasks are useful in understanding the structure of the workflow. The maximum processor width of a task helps us understand the number of processors required simultaneously. The computation and data sizes shows a rough order of the time and the size of data products from this workflow. Each of the workflow may include one or more patterns. Our goal is to capture the dominant pattern seen in the workflow. Workflows are classified as Sequential (mostly tasks that follow one after the other), Parallel (multiple tasks run at the same time), Parallel-split(one task's output feeds to multiple tasks), Parallel-merge(multiple tasks merge into one task), Parallel-merge-split (both parallel-merge and parallel-split) and Mesh (where task dependencies are interleaved).

maximum processor width of a task helps us understand the number of processors required simultaneously. A number of the workflows are simple and usually require a single processor per task. However the motif, flood-plain mapping and McStats workflows often require multiple processors for parallel data processing either for an MPI style application or a number of parallel tasks.

The computation and data sizes shows a rough order of the time and the size of data products from this workflow. The majority of our workflows have about megabytes to gigabytes of data. However a few workflows such as PanSTARRS merging can result in large sized databases as outputs.

In addition, each of the workflow may include one or more patterns. Our goal is to capture the dominant pattern seen in the workflow. Workflows are classified according to their structural characteristics as:

- Sequential: consists of tasks that follow one after the other.
- Parallel: consists of multiple tasks that can be run at the same time.
- Parallel-split: one task's output feeds to multiple tasks.
- Parallel-merge: multiple tasks merge into one task.
- Parallel-merge-split: both parallel-merge and parallel-split.
- Mesh: task dependencies are interleaved.

Workflow vary significantly in their structure, user constraints associated with them and environments they run in. Additional mechanisms to understand the characteristics of the workflows and other capabilities and coordinate their execution with underlying resource layer is necessary for applying specific orchestration techniques in dynamic grid and cloud environments.

3

Distributed Systems

Scientific workflows have varied requirements that include access to distributed data sets and high performance computational resources (Chapter 2). High performance computing and storage systems deployed at supercomputing centers serve the needs of large scale science and engineering problems. The need to share data and resources across organizational boundaries resulted in the evolution of grid computing protocols. Grid deployments, such as TeraGrid [179], Open Science Grid [132], serve the needs of scientific communities. Similar distributed deployments have evolved in other environments as well. For example, PlanetLab [136] provides a research network that supports the research and development of new internet services. More recently, cloud computing has evolved to support mainstream business models on a pay-as-you-go model for storage and compute cycles.

These trends have resulted in a variety of protocols and access models that provide access to underlying resources. For example, scientific users are granted access to supercomputing resources through a competitive proposal review process and are allocated "service units" [176]. Users can use their service units by submitting jobs to a batch queue system, which executes the job on the user's behalf once enough resources become available. Cloud computing systems today grant users

access to resources and are charged for the computational and storage services they use [5]. Cloud systems, unlike batch systems, enable *explicit resource control*, i.e. users request specific quantities and types of resources at specific times. Yet users of both these systems cannot expect strong QoS assurances due to availability variations.

The computing and storage infrastructure landscape has been continuously changing in the last few years. End consumers have a choice of multiple resource providers, however the diversity in extant interfaces makes the task of comparing QoS capabilities extremely difficult. Thus there is need for closely examining the interaction between application middleware and resource-level software. We detail the characteristics of distributed systems in Section 3.1. We compare and contrast grid and cloud systems in greater detail in Section 3.2. This research is a result of collaboration with a number of grid and utility computing projects. We present an overview of these collaborative projects in Section 3.3 and finally summarize in Section 3.4.

3.1 Overview

We have seen parallel trends in the development and deployment of advanced IT infrastructure in the last decade. We have seen the deployment of large-scale government funded HPC environments at supercomputing centers that serve the needs of science and engineering problems. These HPC environments are coupled together with grid computing protocols that enable sharing of resources and data across organizational boundaries. Similarly, utility and cloud computing are ongoing efforts focused on packaging compute and storage resources as metered services that are available over the internet. We study the characteristics of these systems with respect to their resource management and QoS capabilities.

3.1.1 High Performance and Grid Computing

Traditional high performance supercomputing systems have batch queuing software such as Maui/PBS [115], Sun Grid Engine [73], PBS [133, 187], etc. These sites implement different policies for user job priority, backfill, and other job scheduling optimizations. In the batch model users specify a job duration and incur wait times since most of these systems are under provisioned.

Batch queue software manages the mapping of user workloads to resources through a space-sharing policy, where user jobs are granted exclusive access to their requested resources. These jobs can rarely be pre-empted or migrated. The amount of time an individual job will wait in the queue is difficult to predict at the time of job submission. This uncertainty comes from various features of the batch queue system. First, most batch queue schedulers have a FIFO queue at their core and sites often configure complex site policies that grant special priorities to individual users and/or groups that are not known to the end user. Also, users typically specify a maximum amount of time their job will execute, most jobs finish in much less time [52, 53]. Various tools have been developed to aid resource selection and workflow planning decisions based on queue wait time predictions and performance models [127, 205].

The traditional grid computing protocols provide an overlay atop these batch systems that are made accessible through standard web services interfaces. Globus, an open source toolkit, provides standard job submission and data transfer interfaces that allows applications to interact with multiple sites through a single interface.

3.1.2 Utility and Cloud Systems

On-demand or utility computing provides metered infrastructure and services akin to public utilities such as electricity, water, etc. There have been various research prototypes that implement

on-demand computing environments where resources are requested for a specified time period and obtained under concrete terms and conditions [136, 83]. Resource providers in this model are able to provide stronger guarantees through *explicit resource control* since the requests are bounded in both time and space [75]. However it is possible that over-subscription leads to lease requests being rejected when they are redeemed. Leases are also be granted for future time periods providing the advanced reservation capability similar to batch systems [139].

Cloud computing or Infrastructure as a Service(IaaS), a more recent trend, provides a relatively new resource model where multiple virtual servers hosted in data centers are used by individuals or groups through a paid subscription model. Amazon's EC2 system [5] is the most prevalent example in operation. At the time of writing, for more than 20 machine instances from Amazon's EC2 service, a user had to fill an online form that was processed out of band. Cloud computing provides an illusion of infinite computing resources available on demand, i.e.,in current cloud systems resources are accessible to the user almost instantly, with startup time of the instance and image imposing the only delays [9]. Compute resources in EC2 today are charged for the closest instance hour consumed i.e., if you use a resource for 10 minutes, you get charged for one hour. Thus the provisioning of EC2 resources could be considered to be "leases" which are available in one hour increments. This might be considered a simplifying assumption since EC2 allows a user to retain a resource for any number of hours whereas leasing systems typically provide stricter bounds on finish time. This model requires resource providers to provision resources such that all user requests can always be met. However, as cloud systems are configured to grant different service level agreements, providers will under-provision resources to increase profits and decrease idle time on resources requiring additional resource control policies.

3.2 Grids and Clouds: A Comparison

Grids and cloud systems vary widely in their specific mechanisms and protocols. In fact, the exact definition of what constitutes cloud systems is still being debated widely in the community. However, as cloud systems evolve and grid systems mature, there is a need to investigate the software stack running on these systems towards providing predictable quality of service for end users. In this section, we compare and contrast the different dimensions of these systems. Earlier efforts have summarized [9] and compared [66] various aspects of these systems. Our comparison is focused on studying the interaction of different aspects of the software stack for managing QoS. Additionally, specific related work is covered in Chapter 4.

3.2.1 Applications

Grid systems have been used for computational modeling and data analysis for large-scale science and engineering problems. National and regional grid deployments serve the needs of scientists from varied domains including bioinformatics and biomedicine [142], geology [70], earthquake engineering [120], astrophysics [132], weather [54] and storm-surge modeling [138]. These environments are used for scientific modeling and data analysis, educational purposes, etc (Chapter 2).

Cloud computing systems provide different levels of abstraction to the end-user. Services such as Amazon EC2 [5] provide web service interfaces to procure virtual machine interfaces that can then be customized by higher-level tools, services or end-users. In addition platforms and applications services are layered atop the hardware layer for specific purposes. Some current examples include Google AppEngine [72] provides a platform for web applications, Microsoft Azure [118]

provides Windows based internet services, Vertica [194] and Sonian [170] provides data warehousing facility and archive services over Amazon S3 respectively. Salesforce [156] provides Customer Relationship Management (CRM) software services.

3.2.2 User Roles

Grid and cloud services serve the needs of multiple user groups simultaneously, thus often being referred to as multitenant architectures. These environments have a hierarchy of user roles. Resource providers such as TeraGrid sites manage underlying hardware and infrastructure software such as batch queues and Globus services for job management and file transfer. Cloud or Infrastructure as a Service (IaaS) providers are the resource providers in cloud systems, that provide customizable virtual machines.

Software-as-a-Service(Saas) and Platform-as-a-service providers build specialized services on existing cloud systems catering to specific user groups. Similarly, programmers and IT personnel manage project-specific services such as eventing system, science gateways or portals, data services, application web services, etc in grid deployments today. In grid environments, programmers often work closely with scientific users and manage higher-level tools and application codes.

3.2.3 Programming Models

Scientific applications in grid environments are predominantly based on three execution models - Master-Worker, Divide and Conquer and Single-Program Multiple-Data (SPMD) [56]. Cloud computing applications are composed using the MapReduce programming model for processing large data set applications [45]. Dynamic workflows consist of elements composed from the basic execution models [20, 54]. We discuss the programming models and its impact on resource

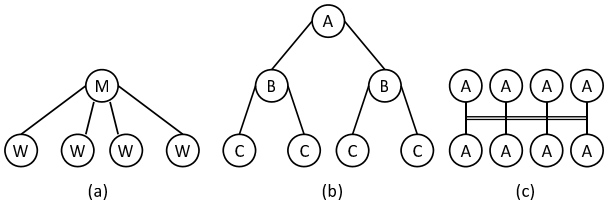


Figure 3.1: Scientific Application Programming Models (a) Master-Worker (b) Divide and Conquer (c) Single-Program Multiple-Data (SPMD)

management in this section.

Master-Worker. In the Master-Worker paradigm (shown in Figure 3.1(a)), the master decomposes the problem into small tasks and distributes these tasks for execution. Primary communication is between the master and the workers, as the master is responsible for collecting partial results to produce the final result. Depending on the master and workers' execution characteristics (e.g. long or short running), coupled with resource availability, one resource selection policy may choose a more reliable node to execute the master task and an appropriate fault tolerance strategy.

Divide and Conquer. Similarly, as seen in Figure 3.1(b), the Divide and Conquer strategy partitions the problem into two or more smaller problems that can be solved independently and combined. Each subtask may be further split into separate tasks. Unlike the Master-Worker model, the subtasks are interdependent. Hence the performance and reliability requirements (e.g. for the communication links) may vary significantly from the Master-Worker model.

SPMD. In the SPMD model (Figure 3.1(c)), each task executes common code on different data. Failure of one task adversely affects the entire application, requiring global coordination.

MapReduce. In the MapReduce programming model, the user expresses the computations as two functions: Map and Reduce. This programming model is similar to constructs in programming languages and database operations that splits the input data into disjoint units and processes them separately. The Map function splits the problem into smaller parts and distributes the problem on

separate processors. The Reduce function, in turn merges results from the distinct computations. This programming model facilitates parallelism enabling the use of distributed systems for processing. Google's implementation of MapReduce has support for backup operations that handle stragglers and failed operations on the systems.

Workflows. Finally, workflows allow applications to define data and condition dependent execution. The workflow itself is a hybrid of one or more execution models mentioned above. As we observed in Chapter 2 workflows have different structures that capture its degrees of parallelism that influence resource selection decisions.

3.2.4 Resource Procurement

Resource procurement is implicit in batch queue systems. Users submit a job description to the batch queue and the job then waits its turn to acquire resources. When the requested resources become available, the job starts executing. The job is killed if it exceeds the requested wall clock time. Thus users do not have specific bounds on when resource are available. The QBETS(Queue Bound Estimation from Time Series) [125] service provides the methodology for predicting bounds on the amount of queue wait times or the probability that a job will finish within a specified duration. Specialized queues or other policies are in place at sites to provide higher level of quality of service to specific user groups, e.g., urgent applications [14].

Resource procurement in leasing systems is explicit, i.e., users request fixed quantity of resources at a specific start time and for a specified duration. Leasing systems provide additional capabilities to extend or vacate leases [75, 83]. Users in cloud systems today request resources and pay for services as long as they are used.

Supercomputing centers today allow *explicit resource control* in the form of offline or online advanced reservation requests. These advanced reservations allow users to specify a fixed start time in the job description at a premium charge [76, 160]. The service units are charged irrespective of whether resources are actually used. While most job schedulers support advance reservations, this feature has typically been reserved for special users since it is expected to negatively impact both system utilization and regular batch job wait times [166, 169]. These pre-arranged agreements are also not effective as mechanisms for dealing with dynamic load conditions.

3.2.5 Data and Storage Management

In addition to access to computing resources, applications in both grid and cloud systems need access to backend databases, storage for intermediate and output data, access to archival services, etc. Grid sites often have shared file system (e.g., NFS, GPFS) on all the nodes. Data management across grid sites are handled through Globus based services such as GridFTP and Reliable File Transfer(RFT) and Replica Location Service(RLS). Storage, archiving of user data are handled on an individual basis by application services and/or user. Cloud systems provide different data management solutions. Companies such as Amazon Web Services [5] and Nirvanix [122] provide storage services accessible through web service interfaces. In addition, databases are accessible to cloud applications. In addition, Amazon also provides persistent block level storage volumes, called Elastic Block Store, accessible from Amazon EC2 instances.

3.2.6 Cost Models

The cost models for batch systems are expressed in terms of service units(SUs), where one SU originally represented one CPU-hour on an IA-64 cluster [178]. A normalization factor is used

based on benchmarking results to account for different machine configurations. The service unit model is used for calculating computing units, however more recently a similar metric is being used for access to high performance storage systems [177].

Cost models in cloud systems are based on a pay-as-you-go model. Systems such as Amazon Web Service [5] and Google AppEngine [72] charge for use of CPU for every hour. Storage and data transfers are also charged similarly.

3.2.7 Service Guarantees

Both grid and cloud systems have mechanisms to provide various levels of service guarantees at different places in the stack. Both grid and cloud systems have services that monitor resources and services [201, 82]. In cloud systems, some application level tools such as Google AppEngine [72], Apache Hadoop [7] have built in mechanisms to handle unreliable nodes. Internally applications use fault tolerance and recovery mechanisms such as automatic retry, replication, checkpoint-restart. Similar mechanisms are available at various levels in the grid software stack to monitor and manage failures [43, 79, 81, 124].

However both these systems undergo various availability variations including complete failures [9, 93]. Resource providers such as TeraGrid offer user a service unit credit back when resources undergo a complete failure. Similary, Amazon's user service agreement gives credit back if reliability falls below 99.9%. However, for predictable QoS, we need to account for these availability variations in planning and scheduling decisions.

3.2.8 Summary

Both grid and cloud systems today are composed of a complex hierarchy of resource and software systems that consist of scientific codes, portals, workflow tools, web services, resource management middleware and underlying clusters and distributed resources. These tools provide various capabilities with the goal of harnessing data and computational cycles distributed across various organizations to meet the needs of the users.

The complex set of interactions between the end-user, different layers of software infrastructure and the underlying resources is a critical component of next generation cyberinfrastructure. We summarize the software stack running atop these distributes systems as:

- A resource control plane that can manage QoS of the underlying resource as a commodity that can be specialized for the user's needs.
- A higher level services layer that can be used for coordination of the underlying resources for efficient and reliable execution.
- Application level tools like workflow tools, that can harness the distributed resources through the services layer to support user needs.
- End-user portal interfaces that allow the users to specify needs and constraints to interact with the underlying resource needs.

Grid and cloud computing protocols provide the substrate for wide-area resource access. However, there are common challenges across grid and cloud systems

- Grid and cloud systems have ad hoc resource interaction protocols. Application mechanisms such as workload planning are closely tied to specifics of the resource model. As grid and

cloud systems evolve to support mainstream business models and scientific processes it is necessary that algorithms and mechanisms in each layer of the software hierarchy can evolve independent of specific mechanisms and the interaction between the layers are well defined.

- Each of the systems has protocols to query, procure resource allocations. However there is limited ability to compare and contrast QoS capabilities of resource allocations both in terms of guarantees for receiving the allocation and runtime failures.

- Application level tools and programming models provide support for specifying application dependencies. There is limited support to allow users to specify dynamic user requirements and constraints that better represent the needs of the user.

- Application tools handle fault tolerance (i.e., minimization software and hardware failures affecting workflow completion) and recovery from failures at different levels in the hierarchy. However systems often suffer a number of availability variations that impact performance that are unaccounted by applications.

- Data centers have policies in place to serve multiple user groups that have varied requirements. But providing predictable quality of service, with differentiated service levels and cost structures, is still an open challenge

3.3 Collaborations

This work addresses the above challenges in the context of providing predictable quality of service for scientific workflows that use distributed resources. We revisit the software stack deployed in the context of distributed systems such as grid, utility and cloud computing and this has been possible due to various collaborative efforts. The three primary collaborative projects are:

- **Linked Environment for Atmospheric Discovery (LEAD).** is a cyberinfrastructure project for mesoscale meteorology. The deadline-sensitive workflows in LEAD provide the motivation for this thesis. This work builds upon the existing foundation provided by web services workflow framework in LEAD.

- **Virtual Grid Application Development Software (VGrADS).** explores the resource level interfaces required to facilitate grid programming and application development. The probabilistic model over batch systems uses the virtual advanced reservations from VGrADS. This thesis also demonstrates the interaction of the workflow orchestration component with the virtual grid execution system.

- **Open Resource Control Architecture.** is a leasing architecture for utility computing. We demonstrate the interaction of application-level components with the leasing core for scheduling scientific applications on leased virtual machines.

In this section we describe these projects and the resulting collaborative efforts in greater detail.

3.3.1 Linked Environment for Atmospheric Discovery (LEAD)

Linked Environments for Atmospheric Discovery (LEAD) [54] is an NSF funded project that is building a scalable national cyberinfrastructure for mesoscale meteorology. The goal of LEAD is to provide a service oriented dynamic adaptive workflow orchestration system. The LEAD cyberinfrastructure consists of a TeraGrid Science Gateway, i.e., portal, that provides an interface to interact with applications and resources. The user composed workflows access application and data web services to launch, monitor and manage user computations and data sets. The LEAD software stack has various monitoring [54] and fault-tolerance measures [87] to handle runtime failures.

The LEAD workflows and its requirements are the primary motivation for this work. The nature of weather modeling that imposes stricter constraints on time and accuracy make predictable quality of service from underlying resources more challenging. Events that cause adaptive behavior can occur at any level in the LEAD system - a new workflow might be initiated in response to a weather condition, there might be inefficiencies in an ongoing workflow execution at the middleware layer, or there might failures or performance variabilities in the system layer [140]. Thus the system needs to be able to adapt to *multi-level* changes while meeting the needs of the given workflow. Specifics of the LEAD workflows and its requirements were discussed in Chapter 2.

The WORDS architecture and various sub-components are more generally applicable to different scenarios accessing grid and cloud systems but were developed in the context of the LEAD cyberinfrastructure. The service oriented web services architecture in WORDS is inherited from the LEAD project. The LEAD production deployment consists of about thirty different persistent services. A subset of these services was used to prototype a system that demonstrate the effectiveness of the proposed research. These components are described in the context of the architecture (Chapter 5). In addition, an emulation environment (described in Appendix A) has been developed for a more thorough evaluation of the workflow planning and resource layer policies.

3.3.2 Virtual Grid Application Development Software (VGrADS)

The VGrADS projects explores the challenges with application development and management of performance for scientific applications that run in grid environments. The virtual grid execution system (vgES), the software environment developed by the VGrADS project, abstracts resources from grid and cloud systems and provides a uniform execution interface. The virtual grid abstraction enables applications to interact with local batch queue systems, advance reservations on TeraGrid resources, Amazon EC2 systems and local cloud sites running Eucalyptus, an open source

cloud software [129]. We describe here the relevant details of the VGrADS project:

Virtual Grid Execution System. The Virtual Grid Execution System (vgES) [90] provides an abstraction for dynamic grid applications to deal with complex resource environments. The virtual grid description language is a hierarchical language for resource abstractions that allows users to specify users to specify qualitative resource specifications [34]. This qualitative specification shields users from the complexity of the metrics of the underlying resources. The language supports two specific language constructs: associators for describing the relationship between the nodes and operators that describe the network bandwidth requirement between nodes or associators themselves. The virtual grid description language supports three high-level associators **LooseBagOf, TightBagOf, Cluster** to describe a set of processors with different connectivity. The language also has operators (**close, far, highBW, lowBW**) that can be used to describe the network connectivity between the high-level resources defined by the associators. The vgES provides an integrated resource selection and binding approach to resource allocation enabling higher tolerance to lower resource availability [91].

Virtual Advanced Reservations. The vgES system uses probabilistic reservations to provide guarantees on resource acquisition on grid systems using Virtual Advanced Reservation for Queues (VARQ) [126]. VARQ builds on queue wait time prediction techniques from QBETS [125] to give users the ability to request "virtual advanced reservations" i.e., a user can specify a fixed start time for the job. QBETS consumes historical resource request data and makes job completion probability predictions using statistical methods such as a clustering algorithm to categorize similar job requests, an on-line change point detection heuristic to detect abrupt variations in the data, and an empirical quantile prediction technique. Previous studies show that though the queue wait time experienced by jobs is highly variable, the upper bound predictions produced by QBETS are more

stable, often over days or weeks. Thus VARQ computes a *probability trajectory*, at 30 second intervals, between the time a user makes a reservation request and the specified deadline and uses the trajectory to find the latest point in time where a resource request can be submitted to meet a specified minimum success probability. Through this methodology, users obtain access to probabilistic or virtual *advanced reservations* that attempt to achieve some level of resource control over systems that provide little or no explicit resource control. The mechanism does have certain cost trade-offs; for example., a resource request can start earlier than the predicted start time, thus using additional resource allocation time.

There are numerous connections between this work and the VGrADS project as listed below:

- We explore reliability extensions required in resource request specifications in the context of virtual grid description language in vgES (Chapter 6).
- We explore probabilistic reservations as an overlay over existing systems to provide QoS guarantees. We use VARQ (Virtual Advanced Reservations for Queues) [126] based reservations to determine if effective workflow orchestration is possible without explicit resource control in batch systems. A virtual advanced reservations obtained through VARQ is an instance of the resource slot abstraction with probabilistic bounds on obtaining a slot of certain duration by a given time.
- The LEAD and VGrADS collaboration has resulted in the ability to orchestrate deadline-driven meteorological workflow sets atop distributed grid and cloud systems. The evolving integrated system has been demonstrated on the exhibition floor at the premier Supercomputing conference (SC) for the last three years. The workflow orchestration techniques that have driven this integrated environment are based on WORDS (Chapter 10)

3.3.3 Open Resource Control Architecture (ORCA)

Open Resource Control Architecture (ORCA) is an extensible architecture for on-demand networked computing infrastructure developed at Duke University. ORCA provides a resource control plane to manage a diverse computing environments on a common pool of hardware resources such as virtualized resources. Shirako [83] provides a substrate of actors that provide a leasing mechanism separating resource allocation policies from the management of the service or a resource. Shirako is a Java-based resource leasing core that is based on a common, extensible resource abstraction. Shirako contains an implementation of Cluster-On-Demand, which supports dynamic leasing of resources from cluster provider sites.

We explore the interfaces required by a resource coordinator that interacts with various grid and cloud systems. In Chapter 8 we use Shirako and COD to implement a dynamic resource control plane for Globus grids, based on Xen virtual machines.

3.4 Summary

We compared and contrasted grid and cloud systems and their resource control policies. The interaction between user and resource is critical to build a rich, flexible, dynamic and adaptive environment. This interaction is necessary for the application requirements to be coordinated with resource characteristics. Thus while advancing mechanisms and algorithms in each layer of the software hierarchy, it is important for the layers to interoperate and coordinate adaptation strategies to support dynamic workflows. We use the lessons learned from the different distributed systems available today to develop the WORDS architecture. The WORDS architecture provides a resource abstraction that addresses the separation of concern between resource and application layers while guiding their interaction.

4

Related Work

Grid computing [61, 62] concepts were first explored in the mid-90s in national laboratories and academic institutions. The first generation of grid technologies and research (e.g. Globus [60], Legion [74], Condor [103]) focused on the ability to harness distributed grid resources to run scientific applications. In early 2000s, grid computing attracted interest from industry. The collaboration between the grid community, largely composed of researchers in academic and national laboratories, with the business community led to a service-oriented grid architecture [64] embodied in OGSA [63] and subsequently in WSRF(Web Services Resource Framework) [130]. Most of the techniques for management of adaptation in grid environments are handled by resource management architectures. More recently cloud computing systems have evolved that support mainstream business models. We compared and contrasted the different interfaces provided by these systems in Chapter 3. In this chapter, we focus primarily on specific implementations of resource and workflow management that have common elements with this work.

Grid computing is increasingly used to deploy and run grid applications [16] in various scientific domains. Science gateways or portals or workflow tools [175] are used to handle the complex interactions of the applications and data and provide intuitive user interfaces. As workflow tools

are evolving, there is a need to understand the interactions between these tools and resource management architectures in meeting user's QoS requirements. In this chapter, we describe the related work in the area of resource management and workflow tools for managing the QoS guarantees for workflows.

4.1 Resource Management

There are various resource management systems [119] in the context of grid systems with different scheduling and adaptation techniques to meet the needs of the applications. Monitoring tools are used to evaluate system and application performance to aid in scheduling and rescheduling decisions.

4.1.1 Resource Selection and Meta-schedulers on the Grid

AppLeS [17] provides a framework for adaptive scheduling on the grid through distinct steps for resource discovery and selection, schedule generation and selection, application execution and schedule adaptation. AppLeS supports long-running grid applications by iteratively computing and implementing refined resource schedules. Various site selection policies and meta-schedulers such as Grid Service Broker, GridWay, Nimrod/G, etc [85, 26, 191, 193] are being explored in the context of the grid. These provide an interface for applications to submit jobs to multiple sites and use standard monitoring tools to collect monitoring information from different grid sites.

Silberstein [162] propose grid execution hierarchy and a scheduling algorithm that adapts to the multilevel queue feedback to manage mixed workloads. The resource management in Legion [32] provides a resource selection and policy framework that has a stronger support for local autonomy among member sites through its object oriented programming model. SPRUCE [14] provides

gateway extensions for urgent computing through the idea of right-of-way tokens.

While resource selection and mapping techniques aid scheduling, there is still a need to be able to represent and enforce user policies to guide the scheduling in real-time for user workflows.

4.1.2 Resource Provisioning

Several works have proposed resource reservations with bounded duration for the purpose of controlling service quality in a grid. Globus toolkit's GARA (General-purpose Architecture for Reservation and Allocation) proposes a QoS architecture that has support for advanced reservations, brokered co-reservations, and adaptation [65]. The architecture has three primary components - online control interfaces that allow applications or agents to modify resource characteristics, sensors that detect the need for adaptation and decision procedures that provide a policy framework. The prototype GARA implementation supports differentiated service mechanisms for coordinated management of high-end networked applications [59]. Other GARA extensions [42, 157] focus on co-allocation policy decisions in a bandwidth broker architecture when users attempt to make bandwidth reservations across administrative domains. Smith et al. [167] study the impact of resource reservations on scheduling through mean wait times of queued applications. The wait times of applications submitted to the queues increases when reservations are supported. The results also show that best performance is achieved when applications can be stopped and restarted, backfilling is used and accurate run-time predictions are used [168].

SNAP (Service Negotiation and Acquisition Protocol) [41] provides a generalized model in which resource interactions are mapped into a well-defined set of resource independent service level agreements. The SNAP protocols define negotiation protocols for three types of SLAs - resource acquisition agreements, task submission agreements, task-resource binding agreements.

Condor [181] is a a specialized workload system to manage compute-intensive jobs across clusters and idle desktop workstations. The ClassAd mechanism in Condor provides a flexible and expressive framework for matching job requirements with resource availability [144]. Gangmatching [145] and set matching [105] are extensions to the Condor matchmaking mechanisms that can handle multilateral and multiple-resource selections. The Condor-G [67] system leverages Globus and Condor to provide multi-domain resource discovery and scheduling.

Czakowski et al. [40] introduce the underlying concepts for an agreement based resource management. Agreements represent management policy in strict policy terms that can be asserted. Singh et al. [163] propose an agreement-based resource provisioning model that allows user mechanisms to discover a set of provisionable resources and a policy for pricing these resources that can then guide user scheduling decisions. The resource availability in this model is represented as a "slot" that is the number of resources available from a certain start time.

Several research efforts have proposed bounded resource units such as leases, slots, slices, advanced reservations, etc [41, 65, 83, 90]. These abstractions define properties for time and resource information but have little or no QoS information. The concept of decoupled resource selection and scheduling [205] and the slot abstractions[83, 163] has been discussed earlier.

Thus, while we need lower-level protocols and agreements for resource provisioning, we also need mechanisms to coordinate them with higher level user workflows.

4.1.3 Workflow Scheduling

Mandal et. al [112] propose a heuristic strategy using performance model based in-advance scheduling for optimal load-balancing on grid resources using the GrADS infrastructure [92]. Batch queue prediction has been used to predict queue wait times [21] and used in conjunction with the

performance model for workflow scheduling [127]. Huang and Chien [78] discuss using Virtual Grids to simplify the application scheduling process [90]. The scalability of scheduling algorithm is improved by pre-selecting a set of "good" resources for workflows.

Blythe et. al. [20] identify and evaluate two resource allocation strategies for workflows - task-based and workflow-based. The task-based algorithm greedily allocates tasks to resources. Workflow-based algorithms find an optimal allocation for the entire workflow and perform better for data-intensive applications.

Various DAG scheduling algorithms have been proposed for grid environments for optimizing makespan, meeting deadline and/or budget constraints or dealing with uncertainty [111, 155, 204]. The underlying assumption of all these algorithms is that resources are guaranteed to be available at a given time, whereas resource availability is highly variable. Deelman et al. [50] detail the computational and storage costs of running the Montage workflow on Amazon EC2 resources.

Heuristic techniques are often used to qualitatively select and map resources to available resource pools. Most use performance as criteria for resource selection. Reliability of resources as a metric coupled with performance has not been used for resource selection in the context of grid applications.

4.1.4 Economy Based Grid Resource Management

Grid environments enable resource providers to serve multiple user communities. As grids are increasingly used in mainstream, an economy based resource management is required for fair sharing and accountability. GRACE (GRid Architecture for Computation Economy) [25] proposes an infrastructure to enable flexible application scheduling using dynamic resource trading services. The Nimrod/G resource broker implements a scheduling mechanism that takes an application

deadline for a job and a resource access budget.

Wolski et al. [200] investigate G-commerce - computational economies for controlling resource allocation using commodities markets and auctions. The authors conclude that commodities markets are a natural choice for grid environments.

Economic models for the grid can truly work only when the user can evaluate his options in real-time or specify guidelines that capture his buying choices to service agents. This requires a two-way communication between resources and the user-space, that is missing in today's cyberinfrastructure.

4.1.5 Monitoring and Adaptation frameworks

Various monitoring tools including Network Weather Service [201] and CloudStatus [37] are used in grid and cloud deployments today. These tools provide a way to monitor resource status - availability and performance.

In addition, tools like Autopilot [147, 150, 195] and SvPablo [151] provide techniques in which applications and source code can be instrumented to collect performance data that can then be used to control or steer the applications. Autopilot provides a sensor-actuator framework to allow steering of grid applications. Autopilot demonstrates the use of a fuzzy logic decision procedure infrastructure to manage grid application performance variability during execution. These techniques are applied directly at the resource level and/or individual application level. The emergence of web services and workflow tools indicates the need for multi-level monitoring and adaptation techniques in addition to existing resource-level techniques.

The Grid Application Development Software (GrADS) project developed a framework that used the notion of a configurable object program that contains application source code, resource selection

and mapping strategies. The execution framework also provided a mechanism for contract monitoring using Autopilot for interrupting and remapping an application when performance falls below acceptable levels. The GrADS workflow scheduler [15] uses a performance-model based workflow scheduling, rescheduling by stop-restart and rescheduling by process swapping.

Weissman et al. [198] present a dynamic grid service architecture that supports dynamic service hosting and resource allocation, a dynamic leasing framework and a model for service robustness that can represent the sensitivity of the service to fluctuations in the environment. Previously, an architecture for adaptable software in grid environments has been proposed [24]. The paper also discusses a consistency model for components that encapsulates parallel codes.

Reflective middleware is organized as a set of collaborating components that can interact with traditional middleware and allow for customizing component behavior dynamically [96]. Reflective architectures also allow for fine-grain resource management through system meta-interfaces. DynamicTAO [97], OpenORB [18] and OiL (ORB *in Lua*) [110] are among the implementations of reflective middleware that allow dynamic replacement of application components.

4.1.6 Virtualization

The Virtual Grid Execution System (vgES) [90] provides an abstraction for dynamic grid applications to deal with complex resource environments. The vgES provides an integrated resource selection and binding approach to resource allocation allowing higher tolerance to lower resource availability [91]. The Cluster on Demand (COD) project [33] proposes the idea of "on-demand" workspaces configured to the application's specific requirements including possibly the operating system. Shirako [83] provides a substrate of actors that provide a leasing mechanism separating resource allocation policies from the management of the service or a resource.

Keahey et al [88] propose a virtual workspace abstraction for grid applications that can implemented using virtual machines and/or COD mechanism. A community broker is then used by the user or application to request and configure virtual workspaces.

Cloud computing interfaces today are based on virtualization with front-end web service interfaces. The most prominent cloud computing service provider today is Amazon [5]. Amazon provides compute power, storage and other services on a pay per usage model enabling other service providers to use the elastic IT infrastructure to host various web based applications or to use the infrastructure to handle peak or overflows from their regular systems. Eucalyptus [129] provides a software infrastructure that enables sites to setup "cloud computing" with EC2 compatible interfaces on their local infrastructure.

Virtualizations provided by these system greatly simplify the resource interaction. But we need higher-level services that can interact with these systems on behalf of users and their policies.

4.1.7 Fault Tolerance and Performability

Khalili et. al [93] measured the performance and reliability of production computation grids such as the TeraGrid [179] and GEON [70]. The results showed that the success rate for benchmark and application runs was between 55% and 80%. The performance variations was in the 50% range largely due to batch scheduler delays.

Reed et al. [146] present examples and data quantifying reliability of current systems and techniques to detect imminent failures in the environment. The paper also shows how intelligent and adaptive software can be used to react to failures and for efficient system use.

Weissman [197] describes a wide-area scheduler that supports two fault tolerance options for SPMD applications, a very common programming model for grid applications. The scheduler

supports replication and application-level checkpointing with performance models. Nurmi et al [124, 128] describes a model that combines historical measurements of resource availability with an checkpoint-recovery delay estimate to generate model-based checkpointing intervals to minimize overhead.

Hwang and Kesselman [79, 80] propose a flexible framework for fault tolerance in grid environments. The framework has a failure detection service that uses an event mechanism to monitor and detect failures. A flexible workflow failure handling framework built on top of the failure detection service applies task and workflow fault tolerance techniques. The workflow framework is evaluated with multiple failure recovery techniques including checkpoint-restart, replication and retrying.

Alonso et. al [2] discuss the limited fault tolerance capabilities in commercial workflow systems and specifically discuss increasing fault tolerance using exception handling and replication strategies for increased availability. Specifically, the authors discuss the idea of providing different availability levels such as critical, important and normal to workflows that determine the recovery criticality and fault tolerance strategy that should be used.

Performability is the joint treatment of performance and availability that started in the 1970s. The term was defined by J. Meyer as a composite measure of a system's performance and reliability and to qualify system performance in the event of failures [116]. Performability has been recognized as an important metric for grid environments which have a high amount of variability in reliability and performance [183, 182]. Performability analysis has been applied in the context of lower level computer networks and communication systems but has not been applied to study the higher level workflow behavior in distributed environments [77, 154].

4.2 Workflow Management

Workflow tools are increasingly becoming a critical component of cyberinfrastructure [49]. Workflow tools first evolved in the commercial sector to represent business transaction processes. More recently workflow tools are increasingly being applied to capture the scientific experimentation process in a grid environment. Some of the business techniques and tools such as WS-BPEL [202] have been applied to grid workflow tools. However scientific workflows differ largely from business workflows. Business workflows are fairly static and changes are fairly infrequent. Scientific workflows such as the ones discussed in Chapter 2 are more dynamic and require adaptation to the scientific exploration process (e.g. undecided workflow steps) as well as the adaptation to the underlying resource behavior [175]. The focus of QoS guarantees for business workflows are on usability and successful service completion.

In this section, we discuss the support for dynamic and adaptive workflows in business processes and also detail the scientific grid workflow tools and supported capabilities available today.

4.2.1 Dynamic and Adaptive Workflows in Business Processes

Workflows have been used to model business processes for a long time. There have been tools and reference models developed in the Workflow Management Coalition (WfMC) for a long time now [101]. The WfMC defines workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [199]. Business workflows that invoke multiple services over the internet have to accommodate for failures and a dynamic environment. They usually support ACID(Atomicity, Consistency, Isolation, and Durability) transaction processing to guarantee against failures.

One such system is eFlow [31], a system that supports composition and execution of composite services. eFlow supports definition of adaptive and dynamic service processes through dynamic service discovery during execution, supports the notion of multiservice nodes that can be used to invoke multiple parallel instances of the same type of service. eFlow also has support for dynamic service node creation that uses a generic service node in the representation that resolve to a specific instance at runtime. eFlow uses consistency rules to prevent run-time errors from modifications.

It is often necessary to deviate from the pre-planned process definition during workflow execution. For cases where it is not cost-effective to specify all possible changes in the workflow plan, ADEPTflex provides a way to modify the workflow execution at runtime [148]. It verifies correctness of dynamic changes and management of concurrent, temporary or permanent changes.

Koksal et al. [95] describe a component based workflow system that allow users to make dynamic modifications through a Dynamic Modification Tool that can be used to make permanent changes to the workflow definitions or temporary changes to instances of the workflow. Liu et al. [104] proposes a handover policy specification that allows users to specify policy on how workflow instances may change when a process definition changes.

The need to cooperate across loosely coupled business organizations has lead to the development of web-service oriented architectures. This has led to the evolution of high level workflow languages such as WS-BPEL (Web Services Business Process Execution Language), often referred to as BPEL that allow methods to define and support orchestration of fine grained loosely coupled processes. BPEL workflow engines provide fault tolerance support through dynamic binding of the services through look-up registries using the UDDI protocol [189].

Business workflow tools provide some level of fault tolerance and adaptability. These techniques cannot be directly applied to scientific workflow tools due to difference in workflow and resource characteristics.

4.2.2 Scientific Grid Workflow Tools

New workflow tools have been developed to represent and run scientific processes in a distributed grid environment. There are various workflow tools such as Kepler [3, 107], Taverna [131], Pegasus [46, 48], Triana [36], that allow users to compose their applications and services into a logical sequence. These tools are developed in the context of specific application domains and have various features to allow users to compose and interact with workflows through a graphical interface, provides seamless access to distributed data, resources and web services. Yu and Buyya provide a taxonomy for scientific workflow systems that classify systems based on four elements of a grid workflow systems - a) workflow design, b) workflow scheduling, c) fault tolerance and d) data movement [203]. We provide here a summary of the features provided by the primary workflow systems in the context of dynamic and adaptive workflows, resource management and fault tolerance.

Kepler [3, 107] builds on Ptolemy II [22, 23], a java based component assembly framework, that uses actor oriented design that emphasizes concurrency and communication between components. Kepler provides uniform mechanism for reporting errors but does not have any adaptation components at this time.

Pegasus [48, 71] provides a planning system for use in grid environments. Pegasus integrates an AI planning system to generate a concrete workflow plan from an abstract description from the user that is then submitted to a Condor Directed Acyclic Graph Manager (DAGMan) [38]. The Condor DAGMan uses the graph representation to manage dependencies between jobs and hence acts as a meta-scheduler for Condor jobs. Pegasus allows constraints to be specified regarding feasible resources and data dependencies on input fields. Pegasus also applies optimization techniques such as node aggregation, data product reuse. Deelman et al. [46] discuss the higher level workflow management issues in grid environments. They discuss the tradeoffs of different scheduling and

planning techniques for workflows such as global and local decisions; full-plan-ahead, in-time local and in-time global scheduling. The authors also propose the idea of using multiple abstract and concrete workflows coupled with execution monitoring as techniques for fault tolerance planning.

Duan et al. [55] use the approach of workflow partitioning and optimization using a master-slave communication model in the ASKALON Workflow Enactment Engine. This configuration allows for a more scalable and fault tolerant coordination of workflows in a grid environment.

Triana [36] provides a graphical problem solving environment that allows users to compose workflows through a drag-and-drop interface. Triana has a simple XML language to describe the components and their interactions. Triana can use other external language representations such as WS-BPEL through pluggable language converters. Triana's workflow language has no explicit support for control constructs such as loops and branching. These are described by specific components that operate over a sub-workflow. Triana can distribute group tasks across multiple machines in a grid environment either in parallel or in a pipeline through distributed services. The Triana Controlling Service [174] allows the task-graph to be updated incrementally and also supports control commands that can be used to control functionality such as start/stop algorithm. etc. The gridMonSteer [196] is a simple non-intrusive monitoring and steering architecture that can work with Triana to allow scientists to interact with a workflow to receive intermediate results and interact with legacy applications running in grid environments. The gridMonSteer has an application wrapper that sends monitoring events to a controller with a standard interface. The controller itself then can be tailored for different types of requirements i.e., a generic application, visualization or a workflow controller. In the current implementation of the application controller, it receives input requests and output notifications from the application wrapper.

The Taverna workbench [131] developed in the context of ^{my}Grid enables the composition and

execution of workflows in the Life Sciences. Taverna has a XML based language - *Simplified conceptual workflow language* (Scufl) that allows users to link third party applications and web services into workflows. Scufl is a data flow centric language and provides implicit support for handling collections, control structures such as iterations. Taverna implements fault tolerance where it will retry failed service invocations a certain number of times with exponential back-off delays. Taverna does not differentiate between failure of services and failure of the underlying resources such as the network fabric. Taverna does not support automatic substitution of services since their scientific equivalence is hard to determine. Taverna allows users to specify alternate but identical services for any step in the workflow. Taverna uses UDDI registries, local disk scavenging for service discovery and provides user service selection tools such as FETA for semantic searching for candidate services.

The GPEL workflow engine [164, 165] uses WS-BPEL [202] to manage long running applications in a grid environment. The GPEL Engine provides the capabilities to control the workflow execution instance through a state document. This capability allows users to pause execution, replay workflow steps, etc. The LEAD system uses both GPEL based workflow engine and Apache ODE, which is an open source BPEL workflow engine [8].

Workflow tools today allow scientist to link complex scientific process through various XML languages and supporting graphical interfaces. They also provide some level of planing and optimization techniques and fault tolerance to react to changes in grid resource and services performance and reliability. The need for adaptation at the workflow engine to both system and user behavior has been recognized as a critical requirement to support next generation cyberinfrastructure science [49]. However, the tools available today do not handle dynamic elements of scientific workflows. The spectrum of QoS issues that arises from the interplay of user constraints and resource behavior has not been considered.

4.2.3 Workflow Constraints and Quality of Service

In addition to the workflow tools and systems discussed already, we discuss some related work to QoS, constraint specifications and exception handling in workflow systems.

Mangan and Sadiq [113] present a constraint approach to workflow process modeling using a relational model to capture flexible business processes. The approach uses a standard set of modeling constraints and process constraints to develop a workflow schema. Each constraint is considered as a composition of one or more elementary conditions. The two types of constraints are task conditions, for describing task dependencies, and instance data conditions, for complex constraints where a certain value must be satisfied. This constraint language provides the flexibility required in business processes. However the language is not rich enough to capture the uncertainties of the scientific exploration process and the constraints that are required for adaptation decisions.

The METEOR workflow system [30] implements a predictive quality of service model that accounts for task performance and reliability and using that to then compute workflow QoS metrics based on workflow patterns. Klingemann et al. [94] describe a technique for deriving a model of web service behavior in cross-organizational workflows from the externally observed service behavior. The approach is based on Markov chains that is constructed from the log of past executions of services. These models are based in a service oriented environment and cannot directly capture the complexity of distributed grid environments.

Aalst et al [192] and Russell et al. [121] describe in great detail the workflow control patterns that are encountered when modeling and analyzing workflows. The workflow patterns can largely influence resource management decisions (e.g., taking advantage of parallel tasks by coallocation). In addition, Russell et al. [153] investigate the workflow resource patterns which describe the interaction of workflow tasks with resources, where resources include humans. These patterns are

useful in specifying authorization, resource constraints between workflow tasks, etc. Occurrence and handling of workflow exceptions has been detailed earlier [152]. They use their analysis of eight workflow systems to propose a graphical, technology-independent language for exception handling strategies in workflows.

Workflow systems for grid environments need a rich language to support user constraints, ability to adapt to conflicting constraints across workflows and the environment, the ability to adapt to failures and exceptions through the system. We also need to be able to use workflow patterns and dependencies for appropriate resource allocation decisions.

4.3 Summary

The resource management tools available in the grid provide mechanisms to select and discover resource services to manage grid applications' QoS requirements. More recently scheduling techniques have been applied to optimize resource selection for a Directed Acyclic Graph (DAG). These techniques are harder to apply to dynamic and adaptive grid workflows. There is also limited availability of tools and techniques for workflows to dynamically change to resource availability.

While there is some support for dynamic and adaptive capabilities available in workflow tools, we need additional capabilities to allow us to express and enforce complex constraints of the application as well as the underlying resources, often high performance supercomputing centers in grid environments.

5

Workflow Orchestrator for Distributed Systems

In this chapter, we present the **W**orkflow **OR**chestrator for **D**istributed **S**ystems (WORDS) that facilitates the separation of concerns between resource and application layers for effective workflow orchestration. WORDS enables an holistic, coordinated, dynamic and adaptive approach to workflow management using user requirements and variable resource characteristics while being shielded from specific resource policy or systems. In the context of this system architecture we explore a standard set of interfaces and mechanisms required at the resource layer in grid and cloud systems to implement effective workflow orchestration for deadline-sensitive applications. WORDS has the following characteristics:

- It provides a separation of concerns between the resource and application level. This enables specific protocols to be implemented in each level of the software stack while not affecting the interaction protocols.
- The interaction between application and resource layer is through a powerful resource abstraction that shields the differences in grid and cloud protocols and represents resource

properties and its QoS capabilities.

- The orchestration system in WORDS is orthogonal to the main execution stack enabling higher degrees of planning and adaptation.

The rest of this chapter is organized as follows. We present the overview of the WORDS architecture in Section 5.1. We discuss the resource abstraction in WORDS that defines the interaction between the application and resource layers in Section 5.2. We provide an overview of the resource and application layers in Sections 5.3 and 5.4. User roles (Section 5.5) and terminology (Section 5.6)in the context of this architecture is detailed. Finally we summarize the research questions that we address in the context of this architecture (Section 5.7).

5.1 Overview

Figure 5.1 shows the WORDS system that introduces a separation between the resource layer and application layer. The orchestration system receives a specification of a workflow or as a set of workflows and user constraints. Each workflow in the set is represented as a directed acyclic graph (DAG). The constraints may include time, accuracy, etc. The workflow planner communicates user requirements to the resource coordinator which initiates resource procurement. The resource coordinator interacts with both grid and cloud sites through conventional scheduling mechanisms and interfaces. The resource coordinator can also work with other execution abstraction systems such as the virtual grid execution system [90] that mitigate the job and data management differences from each model (more in Chapter 10).

The resource coordinator interacts with various site specific resource control mechanisms and returns a Gantt chart to the application layer. A Gantt chart consists of a set of resource slots from

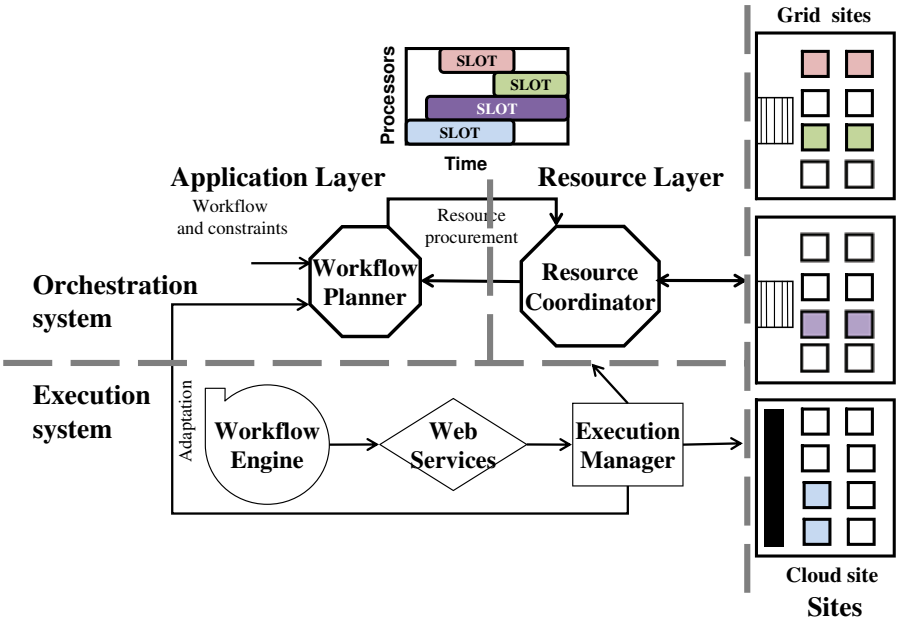


Figure 5.1: WORDS Architecture. The orchestration system introduces a clean separation of resource level and application-level functionality through a resource abstraction (slot). The workflow planner interacts with the resource coordinator to facilitate resource acquisition.

different sites and each slot is annotated with its associated properties. A *resource slot* is an abstract unit of a resource set on a site that has been assigned to the application or user by the resource layer with defined width (i.e., number of processors) and length (i.e., duration). The *resource slot* is central to our resource abstraction and may represent resources allocated to a job through the batch queue system or to a user in cloud systems or through advanced reservation or probabilistic mechanisms. Analogously, we use *abstract* resource slot as a representation to communicate resource requirements from the workflow planner to the resource coordinator. The abstract resource slot is not tied to a particular site or system and communicates the properties of the resources desired. Let us consider an example, for a workflow A, the workflow planner can request one resource slot of 16 processors for two hours with a minimum probability of 0.9 of getting the slot and a minimum probability of 0.99 that the slot will not fail. The resource coordinator in turn interacts with the sites and return a slot of 16 processors starting at 8 am and ending at 10 am with a 0.93 probability of

getting the slot and 0.99 probability that the slot will not fail.

The workflow planner determines a schedule by assigning tasks on the slots using criteria such as computational time, data transfers, success probabilities, cost, etc. Additionally the characteristics of the slots are used to determine appropriate fault tolerance strategies. This process of resource acquisition and task mapping is iterative with the goal of enhancing the schedule for some or all tasks in the workflow.

The execution system (bottom of Figure 5.1), consisting of the workflow engine and web services requires only minimal change to support the orchestration system. The execution manager handles slot-based resource functions such as job submissions in concert with the resource coordinator. The workflow planner cannot anticipate all runtime failures that might occur.The WORDS architecture provides resistance to runtime failures through the execution system that is responsible for detecting deviations from the original schedule or other failures. The execution manager invokes the orchestration components with updated DAG (e.g., the parts of the DAG that have not been run yet) and resource information to reevaluate if user constraints can still be met under changed circumstances.

The WORDS architecture provides a clear separation of functions between the resource and application layer for orchestration decisions. This separation of concerns allows the workflow planner to concentrate on user space and the resource coordinator handles site interaction. The WORDS architecture also has an hour-glass model similar to the Internet Protocol (IP) hour-glass model. It enables resource-layer and application-layer protocols and policies to evolve independently. The communication between the two layers is facilitated through the slot abstraction. Thus, the WORDS architecture provides a resource abstraction that the higher level workflow orchestration can use for planning workflows to meet user constraints atop dynamic distributed systems.

5.2 Resource Abstraction

Fundamentally, grid and cloud computing systems have different access models and policies. However there are also similarities - resources are assigned to jobs or leases for durations of time; resources are often provisioned across competing user groups and resource requests can fail; large scale systems also experience hardware and software failures. The resource abstraction needs to capture the various dimensions of resource property including cost, policy and variability associated with policy and hardware. The WORDS architecture is based on a least common denominator resource model that abstracts the specific properties of grid and cloud systems. The model captures the common minimal set of of properties across the systems that enables the higher-level workflow orchestration to provide effective QoS guarantees for deadline-sensitive workflows. The model does not capture additional resource properties that are provided by specific systems. The degree of effectiveness of workflow orchestration over each system varies based on specific resource control policies. For example, if the resource coordinator returns a set of slots from cloud systems that enable explicit resource control the workflow orchestration can provide higher-levels of QoS.

The resource abstraction in WORDS is powerful enough to abstract various systems and flexible to accommodate various dynamic environments. For example, if mechanisms are available in resource systems to support dynamic resizing of resource reservations or slots the resource coordinator can update the Gantt chart with the new information. The workflow orchestration then uses this information while being shielded from specific underlying implementations. Such dynamic adaptation might also require user/application intervention to determine trade-offs in cost and time. The WORDS architecture enables this interaction which is otherwise difficult if not impossible in today's systems.

The QoS model in the resource abstraction is probabilistic and captures the variability associated with resource procurement (i.e., advanced reservations, job submissions, etc) and failure characteristics during allotted duration. The probabilistic model is important since with or without explicit resource control, strong QoS guarantees cannot be made in distributed systems due to the variability and complexity of the underlying resources and policy. Explicit resource control through advanced reservations in batch systems or in cloud systems has certain impacts - resource providers need to over-provision resources for peak demand and there can be an adverse effect on system utilization and wait times [166, 169]. Probabilistic guarantees help resource providers manage the variability in QoS including unexpected load, utilization and other runtime factors.

We explore probabilistic resource procurement for enabling dynamic workflow orchestration over resources with little or no explicit resource control. As cloud systems advance, the same techniques can be applied to them since lease or cloud resource requests are similar to job requests with fixed time units [75]. In overbooked leasing systems we can calculate an equivalent probability using the number of resource lease requests that are overbooked. In addition to resource procurement, hardware and software services have failure characteristics. Thus we define QoS properties that captures the variability aspect of the resources. The probabilistic resource model allows providers to specify quantitative bounds on resource requests e.g., there is a 95% chance that a request for a three hour slot of 16 processors starting in one hour can be met and there is a 99% chance that resources will stay up during the required duration. The probabilistic resource model is presented in greater detail in Chapter 8.

5.3 Resource Layer

In the orchestration system in WORDS , the resource layer is responsible for interacting with different resource systems such as grid and cloud systems and shield higher-level tools from specific mechanisms or site policies. The resource layer queries resource status and availability (through the Resource Coordinator) and monitors execution for failures or changes in performance or reliability (through the Execution Manager) and feeds the information to the application layer. In the WORDS architecture the resource layer has the following key functions:

Resource procurement. A primary function of the resource layer is resource recruitment or procurement. The resource layer receives a set of resource requests, represented as abstract slot requests, from the application layer. The abstract slot requests guides resource procurement strategies across multiple grid and cloud sites.

Site monitoring. The resource layer additionally monitors the sites for resource changes such as resources becoming available, failures or performance fluctuations that can then drive additional workload scheduling or adaptation decisions.

Execution monitoring. Despite rigorous planning, failures and changes in the underlying resources tend to occur during workflow execution. In addition to monitoring the resources, the resource layer would also interface with application level execution systems to monitor execution of individual jobs or workloads on the mapped resources.

5.4 Application Layer

The goal of existing application tools such as workflow planners has largely been time optimizations based on task execution times and data transfer times. Workflow planners apply resource selection techniques [20] to map abstract workflows onto a concrete set of resources. Workflow tools also apply optimization techniques such as intermediate data product reuse, node aggregation, etc in the workflow planning stage [48]. However in addition to the workflow task-resource mapping based on performance it is often necessary to make adjustments in the plan based on resource availability and reliability characteristics in the workflow planning.

The application layer provides an effective adaptation framework that can react to resource behavior in conjunction with user specified constraints and application changes. In addition, the application layer in WORDS acts as a global coordinating adaptation agent allowing arbitration in the occurrence of conflicts. Specifically, the application layer in WORDS has the following functions:

Workflow admission. The workflow planner provides a two way communication between the user and the resource layer enabling users to interact with the system and adjust workload characteristics with availability. This interaction makes possible a joint decision between the system and end-user on workflow admission into the system prior to execution. WORDS enables the dynamic scientific process such as when users setup triggers to be notified when resources become available so they can pursue additional scientific explorations.

Resource procurement. The workflow planner uses the specified workload information to determine and drive the resource recruitment and selection choices in the resource layer. Workload characteristics such as sensitivity to time and cost considerations result in different policies that can be applied at this level. We consider some illustrative policies for deadline and cost sensitive

workloads (Chapters 9 and 10).

Execution plan. The workflow planner applies user constraints and available resources to develop an execution plan for the specified workload. The execution plan developed by the planner accounts for resource availability characteristics and includes fault tolerance strategies to enhance the success probability of completion. This plan is then used by the execution system to monitor and orchestrate the progress of the workflow.

Dynamic adaptation. The orchestration system handles adaptation changes and can react to changes in resources or user requirements and create a new execution plan. The application layer would also be responsible for arbitrating conflicting adaptation events from different sources.

We handle cases that require dynamic execution level control for deadline-sensitive workflows at the execution manager level. More generally, it is desirable to have a higher-level workflow controller component that manages execution systems like workflow engines. In that case, a workflow controller, in concert with the planner would be the final decision authority for making dynamic adaptation changes. Resource layer implements local policies to react to resource availability characteristics that are shielded from the application. However any significant event that results in a change in the workflow execution (e.g. rescheduling due to a resource failure) will go through the workflow controller. The workflow engine can also be configured to consult the workflow controller on significant events such as failures, etc. The workflow controller also would constantly monitor the system and initiate an adaptation of the workflow execution in consultation with the workflow planner. However workflow engines today have limited support for such dynamic control during execution, hence evaluation of this capability is outside the scope of this work.

The orchestration system in WORDS is the focus of this research. The orchestration system facilitates a higher degree of control in adaptation decisions by a clear separation of concerns between application and resource layer.

5.5 User Roles

Both grid and cloud systems today have different user roles such as end users, service providers, system administrators. Each of these users interact with the WORDS system.

Infrastructure providers. Grid and cloud systems alike have distinct infrastructure or resource providers that include personnel at infrastructure sites for managing the machines, network, operating system and resource-level services. The infrastructure providers manage the needs of competing user groups. The probabilistic QoS model in WORDS enables resource providers to quantify the uncertainty associated with allocation and failures.

Service Providers. Service providers operate on existing infrastructure to provide specialized services to the end-user. The clear separation of functions in WORDS between the resource and application layers allows a transparent resource acquisition process that can be reflected back to the user. In addition, this enhances accountability in service contracts between different user-levels.

Users. The end-user interacts with the WORDS architecture through user specialized interfaces such as web interfaces, that allows the user to specify workload description and user-level constraints. The WORDS architecture enables the end-user to specify constraints to guide scheduling and resource-level decisions. The WORDS architecture enables user-participation in adaptation decisions. Unless otherwise specified, user or scientific user is used to refer to this class of users.

5.6 Terminology

This research explores the WORDS architecture in the context of resource interfaces and workflow tools in grid and cloud systems. Components have evolved in separate and diverse worlds with different and sometimes overlapping terminology. In this section, we provide a clarification

terminology used in the context of WORDS and the rest of this dissertation.

Workflow and Task We use the term *workflow* to depict a sequence of operations, applications or service calls that have dependencies on their execution. The workflows may be BPEL based workflows that consist of a sequence of web service calls or a sequence of binary applications invoked through a script. We use the term *task* or *service* to refer to components of the workflow. All workflows in our system are represented as directed acyclic graphs (DAG). The vertices in our DAG represent tasks and edges represent data flow operations. We do not consider workflows or workloads that have loops or cycles in their representation.

Workflow Set. We consider the use case where workflows may have dependencies between them in Chapter 2. We use the term *workflow set* to indicate a collection of workflows that need to be considered for scheduling concurrently.

Constraint and Policy. We use the term *constraint* or user constraint to identify conditions that users specify on workflows or workflow sets. We use the term *policy* to indicate priorities and rules that service or infrastructure providers have in place to differentiate between users or groups of users.

Resource and Site. We use the term *resource* and *site* frequently. The term resource refers to a single node or a cluster. Site may be used in place of resource to indicate resources under one administrative unit. We specifically do not address federation of clusters within an organization. If an organization has multiple clusters, each addressable separately for purposes of resource interaction, then our system is oblivious to the relationship that exists between clusters. However if these clusters on a single site are connected by high-speed networks our planning algorithm will automatically select sites with proximity as part of the data transfer cost analysis.

5.7 Summary

The WORDS architecture supports the needs of workflow orchestration atop grid and cloud systems. The degree of effectiveness of workflow orchestration over each system varies based on specific resource control policies they implement, that is reflected in the resource abstraction. The goal of the architecture is the "separation of concerns" between the resource and application layers. The resource layer is responsible for resource acquisition and other specific resource management mechanisms whereas the application layer uses the abstracted *resource properties* to make orchestration decisions. The slot abstraction is the center of the interaction model between the two layers. The concept of decoupled resource selection and scheduling [205] and the slot abstractions[83, 163] has been discussed earlier. However the interaction and the interfaces between the application layer requirements and resource model variability and its impact on high-level workflow orchestration has not been studied before.

In the context of the WORDS architecture, we address the following research questions:

- Is a common abstraction possible that captures the different properties of grid and cloud systems and yet enables higher level systems to be shielded from specific system implementations?
- What information is required in next-generation data-center interfaces to improve support for dynamic adaptive workflows?
- Can users be allowed to express dynamic user and resource constraints?
- Is it possible to provide predictable quality of service atop systems that do not provide explicit resource control?

- How can workflows account for variability in performance, and reliability that are inherent to distributed large-scale systems?
- How can workflow sets be scheduled to meet multiple constraints such as deadline and accuracy? How can higher-level tools determine appropriate fault tolerance strategies with cost and other constraints?

6

Constraint Model

Dynamic scientific workflows (e.g. LEAD) require the ability to specify various criteria, such as time and accuracy, on the workflows and the relationship between workflows. Similarly the application level tools need the ability to express qualitative expectations on resources queried. In today's cyberinfrastructure environments there are mechanisms to monitor performance and ensure reliability, to select and discover resources and tools to express task dependencies. However there is little or no support available at the resource or user level that allow users to express requirements relevant to the application workload. Workflow planning mechanisms need to understand constraints or requirements that come from the scientific experiments to be able to trade-off various system characteristics. For example, a user with a time sensitive workflow may be willing to use more expensive resources and a scheduling strategy needs to take that into account.

In this chapter, we explore the constraint space at the user and resource layer to facilitate specifying dynamic user requirements. Specifically, we address the following issues.

- We explore the type of requirements that users would like to specify on workflows and workflow-sets. We use these requirements to define a constraint space that defines the scope of constraints a user can specify on various elements in the WORDS architecture.

- We explore the resource specifications that are needed for scientific application codes. The virtual grid description language (vgdl) allows users to specify qualitative performance requirements during resource selection. We propose extensions to *vgdl* that capture availability expectations from resources.

In this chapter, we address the interfaces necessary at the user level to specify constraints on the workflows and the ability to specify resource requirements at the resource level that can translate to user requirements to specific resource properties.

6.1 Workflow Constraints

In this section, we explore the type of constraints that a user likes to specify on a workflow or a workflow set. First we present some examples and then present the model.

6.1.1 Examples

We consider a user's workflow set $W = \{W_1, W_2, ..., W_9\}$ where workflow W_i is a description of a DAG. These workflows can run on any of the resources in the set $R = \{R_1, R_2, ..., R5\}$.

Deadline. A user specifies that the workflow set must complete by a deadline D from now. Additionally the user specifies that W_3 must finish by D_3 where $D_3 < D$.

Accuracy. The user specifies that in the given set at least four out of the nine workflows must complete by the specified deadline.

Workflow Importance. The user specifies that in the given set, W_1, W_8 and W_9 are most important followed by W_2 and W_7.

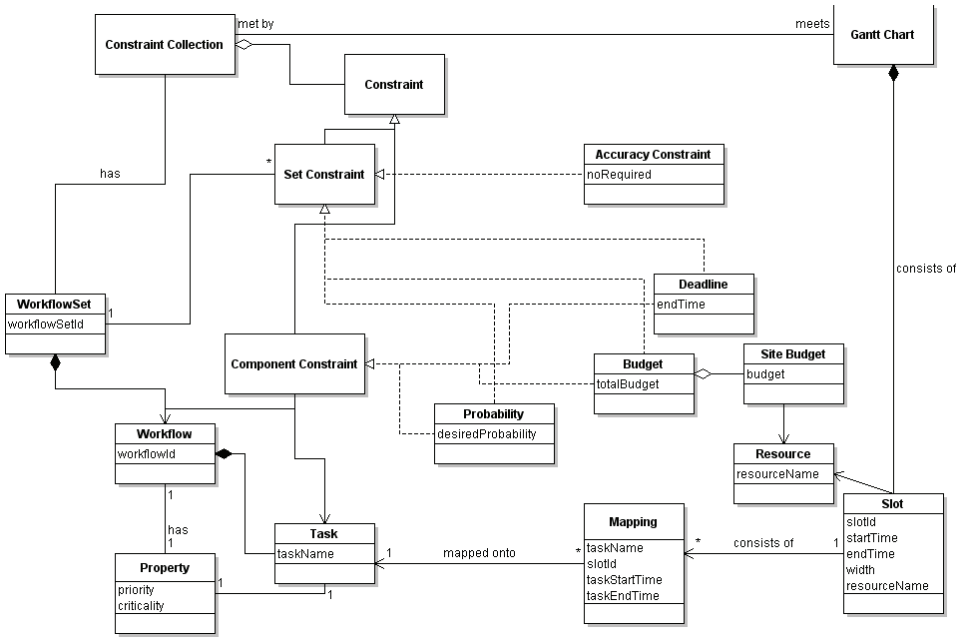


Figure 6.1: Workflow Constraint Model UML. The figure shows the various components in WORDS and the relations and constraints the user can specify on those components.

Budget. The user is willing to use 1000 computational units on each site.

Success Probability. Each task in the workflow that is scheduled must have a 50% chance of completing by the deadline.

6.1.2 Model

Figure 6.1 shows the conceptual view of the elements in the WORDS system and the constraints the user can specify on the artifacts. A *WorkflowSet* is composed of *Workflows*. Each workflow is composed of *Tasks*. A *Workflow* or *Task* can have a *Property* that captures the priority and criticality of the workflow or task. In our implementation, the criticality is the value assigned by the user and the priority is assigned by the system. Criticality captures the value of the entity to the user relative

to other entities that belong to the user. The priority is assigned by the system to indicate the user's priority relative to other users in the system. Priority and criticality values are expected to be tied with cost-models in the long-term to prevent misuse. A *WorkflowSet* is associated with a *Constraint Collection* that details the user requirements on the workflows. A *Constraint Collection* may have two types of user constraints, i.e., constraints on a workflow or task (*Component Constraint*) or on a set (*Set Constraint*). The constraint model is motivated by the LEAD workflows and address the following types of constraints.

- **Probability.** A user can specify a desired success probability of completion at either the task or workflow or workflow set level. This quantitative guarantee the user requires on a specific workflow.
- **Deadline.** Time-sensitive workflows such as weather prediction have deadline associated with them. Thus a user can specify a deadline on a workflow set, workflow or task that guides scheduling decisions.
- **Budget.** Distributed resource models are increasingly made available as metered services, with different costs associated with quality of service. Thus it is important to let users specify a budget that they are willing to expend on an experiment. A budget may be specified per site or a cumulative budget across all sites may be specified.
- **Accuracy.** In addition to the above constraints, a user can also specify an accuracy constraint for a workflow set. The accuracy constraint captures the minimum number of workflows in a set that must complete successfully to satisfy the science accuracy requirements (Chapter 2).

The goal of WORDS is to determine an execution plan that satisfies the set of constraints. The plan is represented in a Gantt Chart as a set of mappings of tasks on the resource slots.

6.1.3 Conflict Resolution

The constraints that users specify may have inconsistencies or result in conflicts. Generally, aggressive or invalid constraints such as a deadline or an accuracy constraint that can't be met are directly handled by the planning components in the system. These will result in the user being asked to modify constraints since a valid execution plan could not be determined. We ensure the following checks for resolution in the WORDS system.

- Property may be specified at various levels - task, workflow or workflow set. A component inherits the higher level's property. For example, if a criticality is not specified at the task level, it has the same criticality as its parent workflow.
- A deadline may be specified at the task, workflow or workflow set. If a deadline is not specified at the task level the workflow's deadline is used to determine the task deadline. If a workflow does not have a specified deadline, it has the same deadline as the workflow set it belongs to. If a task deadline is specified that is not consistent with the workflow set deadline, the orchestration system in WORDS will not be able to come up with a valid execution plan and return an error to the user.
- Our constraint model allows the user to specify a budget per site as well as a total budget. If the budget is specified for all sites and a total budget is specified by the user, the sum is verified. If individual site budget is not specified the total budget is considered to be uniformly distributed across the sites.
- Workflows in WORDS are ordered by their priority and criticality. User roles and associated cost models are envisioned to stop users from specifying aggressive criticality values. If priority and criticality are identical for two or more workflows, the deadline and submission time of the workflow is used for determining the order of the workflow.

6.2 Resource Request Specifications

In WORDS , the user constraints are received by the workflow planner that translates the requirements and drives the resource procurement through the coordinator. The resource coordinator queries different sites to request resources. The virtual grid execution system (described in detail in Chapter 3) allows high-level, qualitative performance requirements to be specified that guides resource selection. The language in vgES is a hierarchical language for resource abstractions that allows users to specify qualitative resource performance specifications. We provide an extension to the virtual description language that enables users to specify availability requirements.

A critical dimension to managing an application's reliability requirements is understanding its specific characteristics. We discuss the reliability requirements for grid applications with different execution models in Section 6.2.1. In addition, we illustrate the virtual grid extensions using two application examples - mpiBLAST [44] and the Weather Research and Forecasting (WRF) model [117].

6.2.1 Reliability Requirements of Scientific Applications

In Chapter 3, we identified that scientific codes are composed with common parallel programming model representations - (a) Master-Worker, (b) Divide and Conquer, (c) SPMD and (d) workflows.

In the master-worker paradigm, the master decomposes the problem into small tasks and distributes these tasks for execution. Primary communication is between the master and the workers, as the master is responsible for collecting partial results to produce the final result. Depending on the master and workers' execution characteristics (e.g. long or short running), coupled with resource availability, one resource selection policy could be to choose a more reliable node to execute

the master task and an appropriate fault tolerance strategy.

Similarly, the Divide and Conquer strategy partitions the problem into two or more smaller problems that can be solved independently and combined. Each subtask may be further split into separate tasks. Unlike the master-worker model, the subtasks are interdependent. Hence the performance and reliability requirements (e.g. for the communication links) might vary significantly from the master-worker model. In the SPMD model, each task executes common code on different data. Failure of one task adversely affects the entire application, requiring global coordination.

Finally, workflows allow applications to define data and condition dependent execution. The workflow itself is a hybrid of one or more execution models mentioned above. For workflows, assuring high reliability and high performance for the entire workflow duration can be very expensive. In such cases, the workflow planning software may request a combination of high reliability and lower reliability nodes to offset costs. The workflow planning strategy could then apply additional fault-tolerance mechanisms such as replication or checkpoint-restart to increase the success probability. To summarize, a cost-benefit analysis of application characteristics in concert with resource characteristics is required to determine an appropriate resource selection and corresponding fault tolerance strategy. Next, we present some examples of scientific codes and possible resource requests to satisfy their needs.

6.2.2 Examples

mpiBLAST. The Basic Local Alignment Search Tool (BLAST) [4] compares nucleotide or protein sequences and finds regions of similarity between them to detect functional and evolutionary relationships. The parallel version of BLAST, mpiBLAST, follows the master-worker execution model. Consider an mpiBLAST resource request for a master node connected to a set of worker nodes, each with at least 4 GB of memory. In the virtual grid description language (vgDL), this would be

specified as follows:

$mpiBLAST1 = MasterNode = \{memory >= 4GB, disk > 20GB\}$**highBW LooseBagOf** $< WorkerNode > [4:32]; WorkerNode = \{memory >= 4GB\}$

One fault tolerance strategy might require the network link between the master and the worker to have "good" reliability (Section 6.2.3). The modified vgDL might look like the following:

$mpiBLAST2 = MasterNode = \{memory >= 4GB, disk > 20GB\}$**(goodReliability AND highBW) LooseBagOf** $< WorkerNode > [4:32]; WorkerNode = \{memory >= 4GB\}$

In addition to the network being reliable, the request could also specify that the master node be highly reliable:

$mpiBLAST3 =$ **HighReliabilityBag** $< MasterNode >= \{memory >= 4GB, disk > 20GB\}$**(goodReliability AND highBW) LooseBagOf** $< WorkerNode > [4:32]; WorkerNode = \{memory >= 4GB\}; MasterNode = \{memory >= 4GB, disk > 20GB\}$

Weather Research and Forecast (WRF) Model The Weather Research and Forecasting (WRF) model [117] is a mesoscale numerical weather prediction system. The WRF model is an SPMD computation where geographic regions are modeled in parallel. For a simple WRF execution, the request might be for a cluster with 8 to 32 nodes, each with at least 4 GB of memory:

$wrf1 =$ **TightBagOf** $< CNode > [8:32]; CNode = \{memory >= 4GB\}$

We might require all the nodes and the network connecting them to be highly reliable since this is an SPMD computation. A modified request is shown below to request a HighReliabilityBag:

$wrf2 =$ **HighReliabilityBag** $< ManyNodes > [1:1]; ManyNodes =$ **TightBagOf** $< CNode > [8:32]; CNode = \{memory >= 4GB\}$

From these examples we see that applications can have varied reliability requirements based on their characteristics. Workflow planning components need higher-level interfaces to describe collective qualitative reliability requirements in the resource selection process. These requirements are based on application characteristics and other real-time constraints such as deadlines or budget. These user-specified attributes can then guide selection allowing the system to apply scheduling and adjust fault tolerance levels and expectations at run-time.

6.2.3 Reliability Specification

In this section, we discuss the extensions required to the virtual grid description language to support reliability specifications. We define a high-level qualitative reliability metric space that can be used to request resources. The qualitative levels are mapped to well-defined quantitative reliability levels in the virtual grid to enable runtime monitoring and adaptation. We define a five point qualitative reliability scale that maps to quantitative levels of availability as follows:

- Excellent (90-100 %)
- Good (80-89%)
- Satisfactory (70 - 79%)
- Fair (60-69%)
- Poor (59-0%)

We expect the exact definition of the levels to vary in specific deployment contexts and evolve with advances in underlying computer hardware architectures. These qualitative levels map directly to resource cost models enabling users or application level tools to trade-off resource quality

with cost considerations. This prevents users from requesting high reliability resources when it is not required. This situation is analogous to when users specify an expected wall clock time on their batch jobs today in HPC systems - specifying a longer than required wall clock time could result in being penalized with longer queue wait times and a short wall clock time can result in the job getting killed earlier.

We define a set of associators for collective node reliability. These associators map to the qualitative reliability set defined earlier

- **HighReliabilityBag.** A set of nodes with *Excellent* reliability.
- **GoodReliabilityBag.** A set of nodes with *Good* reliability.
- **MediumReliabilityBag.** A set of nodes with *Satisfactory* reliability.
- **LowReliabilityBag.** A set of nodes with *Fair* reliability.
- **PoorReliabilityBag.** A set of nodes with *Poor* reliability.

Similarly, we define operators for specifying network reliability levels.

We add the following operators that describe network link reliability and are mapped to similar quantitative levels as above:

- **highReliability.** A set of nodes with network link reliability that is *Excellent*.
- **goodReliability.** A set of nodes with network link reliability that is *Good*.
- **mediumReliability.** A set of nodes with network link reliability that is *Satifactory*.
- **lowReliability.** A set of nodes with network link reliability that is *Fair*.
- **poorReliability.** A set of nodes with network link reliability that is *Poor*.

Typically a reliability specification for components are expressed as a value for an associated interval with a desired confidence level (e.g., the disks on the head node of a cluster are 90% reliable between 10 pm and 4 am, with a 95% confidence interval). The vgDL's extensible attribute mechanism allows these to be additionally specified if required. These resource request specifications map to the reliability states that a resource encounters during its lifetime (Chapter 7).

6.3 Summary

In this chapter, we explored the user constraint model and resource reliability specifications to support dynamic scientific workflows. This space of requirements from user and resource properties provides the foundation for the orchestration to explore various trade-offs in application execution.

7

Performability Modeling

Workflows experience significant variations in performance and reliability during their real-time execution. In this chapter we explore performability, a measure of lost QoS due to reliability variations, as a basis for workflow scheduling and fault tolerance strategies. Workflow orchestration needs to account for user and application level requirements and consider the resource space behavior in real time. As explained earlier today's workflow and resource planning relies on resource status and performance models, despite frequent component failures [93, 205]. Thus resources are considered to be in one of two states, either "fully-operational" or "failed." Realistically, the availability of resources can vary greatly based on failure of one or more critical services, load on one or more resource components, recovery from a failure, etc. These variations manifest as a loss in application performance that can result in increased application execution times or as a complete failure that might require rescheduling. In addition, earlier studies show the variation of application performance on a resource over multiple executions [98]. Thus it is often impossible to predict accurately the exact running time of an application on a diverse set of resources. The prediction problem gets further exacerbated as resources are made available through virtualization and cloud computing. The advent of new technologies in recent years and the complexity of the

multi-level software stack makes the current day methodologies insufficient to make accurate decisions. Thus we need methodologies in application middleware that not only handle failures but also account for possible loss in Quality of Service (QoS) from resource availability variations in any planning strategies. For this, a resource provider needs to provide an assured level of service under a cost model even as there are performance and reliability variations in hardware.

Today's grids are composed of a conceptual resource hierarchy (e.g., individual systems, data systems and clusters) and a software hierarchy with a multiplicity of execution models (e.g., SPMD, parameter sweep, workflow). To build a resilient environment one needs a multi-level strategy that can detect and adapt to performance variations and failures at each level and across levels. As an example, consider the previously discussed meteorological application (Chapter 2) with constraints on execution time and accuracy due to weather prediction deadlines [54]. The inputs to a typical workflow of this type are streaming sensor data that must be pre-processed and then used to launch an ensemble of weather models. The model outputs are processed by a data mining component that determines whether some ensemble set members must be repeated to realize statistical bounds on prediction uncertainty. In this environment, both performance and reliability guarantees (i.e., the critical workflow elements must complete and must do so within the given time constraint) are essential. Thus the application and resource layer must interact and adjust strategies to balance and manage user expectations as discussed in Chapter 5. In this chapter, we propose and develop a performability model that enables workflow scheduling and planning to account for dynamic resource behavior.

We use performability [116], as a composite measure of a grid's performance and dependability (i.e., a measure of the system's performance in the event of failures) and present a qualitative model to capture and analyze the effect of resource reliability on application performance. We propose a

model to analyze performability as a metric for workflow planning (Section 7.1). We discuss performability modeling based workflow planning (scheduling and fault tolerance) in Section 7.2.1. We present an experimental evaluation of performance on real applications with induced availability variations, and an analytical evaluation of parameters affecting performability (Section 7.3). Subsequently in Chapters 9 and 10 we apply the performability modeling to workflow orchestration.

7.1 Degradaded Service Modeling

Grid systems are often able to survive the failure of one or more components and continue to provide service, but with reduced performance. The behavior and status of systems with multiple interacting components is typically captured using stochastic process modeling. J. Meyer introduced the concept of performability [116] evaluation as a mechanism to combine performance and availability analysis when considering resource behavior. In this context, performability is defined as "the probability that a system reaches an accomplished level y over a utilization interval (0,t)." Grid systems often have multiple hardware and software components or services that contribute to system state. The probability of staying in a certain state with respect to transition rates between states is used to quantify system performance and reliability.

Markov Reward Models (MRM) are typically used to model degradable systems and capture joint performance and system reliability. A Markov reward model consists of a Markov chain that describes a system's possible states and an associated reward function. By modeling the system as an MRM, we associate reliability levels with states in the Markov chain. The reward rates correspond to system performance in the different states allowing us to model the behavior and capture the probability of the system delivering performance at different availability levels. If detailed

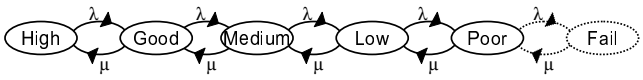


Figure 7.1: Resource Reliability Model. A Markov chain representing the five reliability states of the machines and the transitions between the states represent the failure and repair rates.

monitoring data was available from systems, we can classify the current state of the system and then use appropriate probabilities to determine its behavior during workflow execution.

We develop and describe the model for resource state reliability in Section 7.1.1 and apply reward rates to the state based on application performance and cost, developing our performability model in Section 7.1.2.

7.1.1 Resource State Reliability Model

In today's systems, quantitative reliability metrics are used (e.g. mean time to failure (MTTF), mean time to repair (MTTR) and mean time between failures (MTBF = MTTF +MTTR.) These reliability metrics are used for different system components, including storage, network and computing resources and often guide fault tolerance strategies. We use the qualitative five point reliability scale (described in Chapter 6) in conjunction with reliability metrics to assess system performability.

Resources can be in a variety of states based on the functioning of each component in the system - hardware and software. Accurately capturing each state for changes in behavior can be a tedious and complicated task. However, we seek the high-level system behavior in terms of delivered QoS. Distributed resources exhibit the Markov property - the next system reliability level depends only on the present state and is independent of previous states. For example, if a service resets from an error, and is restored to an operational state it does not matter if it has failed at some earlier point. Thus we define a Markov chain (Figure 7.1) representing the five reliability levels - High, Good, Medium, Low, Poor and the complete failure state "Fail." This model represents the reliability

states of a resource where each resource might be a composite object such as a cluster of nodes. In this analysis, we classify the reliability levels based on the cumulative resource reliability level that includes hardware and software components required for application execution. We use resource failure (λ) and repair (μ) rates that are inverses of the MTTF and MTTR respectively to model the transition rates from a state.

This qualitative model helps characterize the system with a Markov chain that can be easily modeled, avoiding the state space explosion problem, while capturing the salient aspects of system behavior [77]. In our model, the transitions occur between adjacent failure states, i.e., a series of failures will shift the system from the High to Good state and corresponding repairs will move the system back to the higher reliability state. Such a Markov chain, where transitions only occur between adjacent states (Figure 7.1), is defined as a birth-death process [154]. We define the Markov chain with uniform failure (λ) and repair rates (μ) that are consistent with the qualitative nature of the virtual grid system. These simplifying assumptions can be relaxed given more detailed performance data and models of cluster, grid and cloud resources.

The transient analysis of the Markov chain gives the instantaneous reliability of the system (i.e., the probability that the system is working at time t, regardless of the number of times it may have failed in the time (0,t)). For simplicity, we consider the steady state probability of occupancy in each state - the likelihood that the resources are collectively in one of the states shown in Figure 7.1. The steady state solution for a birth-death process is given by:

$$\pi_n = \rho^n \pi_o \tag{7.1}$$

$$\pi_0 = 1 - \rho \tag{7.2}$$

where ρ is the failure-to-repair ratio ($\rho = \lambda/\mu$) and n is the state identifier such that n=0,1, 2, 3, 4 represents "High", "Good", "Medium", "Low", "Poor" respectively. Thus π_0 is the steady state probability of being in the High reliability and π_4 is the steady state probability of being in the Poor state. Because resources are considered repairable, the system will be in an operational state if the repair rate exceeds its failure rate, i.e., $\lambda < \mu$ or $\rho < 1$, else the system's steady state would tend towards complete failure. From equations (7.1) and (7.2), we see that the steady state probabilities of being in the respective reliability states depend on the failure-to-repair ratio of the resources.

7.1.2 Performability Model

In this section, we extend the simple resource reliability model defined in the previous section to create a Markov Reward Model (MRM) that includes system performance and cost enabling a joint treatment of performance, cost and reliability. The MRM consists of a Markov chain shown in Figure 7.1 and an associated reward function that represents system performance. For each state $i \in S$, r_i represents the reward obtained for time spent in that state, measured either as reward obtained per unit time or reward obtained on transition from a state.

To measure overall system performability, we must measure the quantum of work achievable in a given interval or alternatively assess the rate the system can perform work, given the probabilities of being in different states. We measure performability as the accumulated reward rate over a specified time interval. If $Z(t)$ is the system reward rate at time t, the expected instantaneous reward rate is given by

$$E[Z(t)] = \Sigma r_i \pi_i(t) \tag{7.3}$$

where $i \in S$ and $\pi_i(t)$ is the probability of being in a particular state and r_i is the corresponding reward at time t.

However a resource might visit infinitely large number of states during execution, making it important to measure the expected steady-state reward rate of a given machine, which can be expressed as

$$E[Z] = \Sigma r_i \pi_i \tag{7.4}$$

Combining these results (Equations 7.2 and 7.4) the performability of a set of resources depends on the resources' failure-to-repair ratio (reliability ρ) and the rewards associated (r_i) with each state.

Performance. Let us assume that the application run time on a resource in High reliability state is T time units. Earlier work has shown that the performance variation follows a normal distribution [98]. Thus, we denote performance of the application in the other states as $(T + n_1x), (T + n_2x), (T+n_3x), (T+n_4x)$ units respectively. The parameters x, n_1, n_2, n_3, n_4 are performance degradation factors used to capture the increase in the application's execution time at lower reliability levels. The increase in time in each state is given by $n_i * x$. The parameter x is the constant degradation factor that is seen due to machine availability characteristics and is independent of the application characteristics. The parameters n_1, n_2, n_3, n_4 captures the time increase experienced by the application on the machine at degradation factor of x. The parameters n_1, n_2, n_3, n_4 have the unit of time. In cyberinfrastructure deployments, the performance degradation factors will be determined by historical information of the resource failure characteristics and benchmarking results of the application on the resource. The reward rate indicates the performance level of the system within the operation constraints (i.e. reliability level) of that state. Thus, we use the inverse of the time taken

by an application in a particular state to denote the reward level in the state. This enables us to capture the performance associated with various reliability levels.

If an application is in the "High" state the amount of work completed in unit time would be greater than the work completed in the "Fair" state. For a particular application running on a specific machine, the reward rate is the inverse of the running time in the state. If a machine is in the "Good" state throughout, the application would take $(T+n_1x)$ time units to finish or $1/(T+n_1x)$ work units would be completed per unit time. The expected reward rate is obtained by substituting the performance reward rates in equation 7.4:

$$E[Z_T] = \frac{1}{T}\pi_0 + \Sigma\frac{1}{T+n_ix}\pi_i \tag{7.5}$$

The inverse of the steady-state reward rate is the projected application execution time on degradable systems and is given by:

$$T_{projected} = 1/E[Z_T] \tag{7.6}$$

Thus knowing the failure-to-repair ratio of a machine and the application's behavior in different reliability levels, we can predict the execution time of an application that accounts for the reliability variations.

Cost Model. Today's grid and cloud systems have different cost models associated with them. On production grids (e.g., TeraGrid), users are allocated service units through an allocation review process and service units are deducted from the quota for storage and processor usage. Systems

such as Amazon EC2 charge users for data transfers and the instance hours used on the machines. Both systems provide fixed pricing models for a given machine to all end-users and complete failures are handled through a refund process. However, the scale of these systems require differentiated pricing models that account for dynamic service levels during execution. Thus cost models on degradable systems need to correspond to the system state. We assume that the cost-rate (i.e. cost per unit time) for a system in the High reliability state is given by c_0. Correspondingly the cost in the "Good", "Medium", "Low" and "Poor" states are given by c_1, c_2, c_3, c_4 units. Thus the expected steady-state cost rate can be given by:

$$E[Z_C] = \Sigma c_i \pi_i \tag{7.7}$$

where $i \in S$ and π_i is the steady state probability of being in a particular state and c_i is the corresponding cost rate in that state. Thus the expected steady-state total cost for an application is given by

$$TotalCost = \frac{1}{E[Z_T]} * E[Z_C] \tag{7.8}$$

An important question that arises in degradable systems is how does a resource provider set its pricing such that it accounts for the variability in the system while making the prices competitive. A consumer will use resources in a degradable state only if the total cost incurred by the application is equal or lower in the degradable state. Thus for a given application,

$$TotalCost_{Poor} \leq TotalCost_{Low}...$$
$$\leq TotalCost_{High}$$

where $TotalCost_{state}$ is the total cost incurred by application when a resource is in the given state during execution. Let us consider an example application with execution time T in the High reliability state. At a cost-rate of c_0 it would incur a total cost of $T * c_0$ in the High reliability state. The same application would take $(T + n_1x)$ in the Good reliability state and hence cost $(T + n_1x) * c_1$. Thus for the pricing to be competitive, the Good reliability state must be priced as shown:

$$TotalCost_{Good} \leq TotalCost_{High}$$
$$(T + n_1x) * c_1 \leq T * c_0$$
$$c_1 = costFactor_1 \frac{T * c_0}{T + n_1x}$$

Thus, more generally the pricing in a given state is given by:

$$c_i = costDegradation_i \frac{T * c_0}{T + n_ix} \tag{7.9}$$

where $costDegradation_i$ is the cost factor in state i. At $costDegradation_i = 1$ the total cost to run an application in state i is the same as the cost in the High reliability state even though the application might take longer to complete. A $costDegradation_i < 1$ gives the user an incentive to use the

Parameter	Machines			
	A	B	C	D
Application running time T	30 min	30 min	25 min	15 min
Failure-to-repair rate ρ	0.1	0.4	0.4	0.6
Perform. x=2	0.033	0.032	0.038	**0.055**
Perform. x=100	**0.031**	0.0224	0.027	0.029
Effective cost rate	0.99	0.93	0.93	0.82
Total Cost at x=2	0.50	0.49	0.48	0.25
Total Cost at x=100	0.54	0.69	0.58	0.46

Table 7.1: Performability Example. Table shows performability and cost for different performance model numbers and reliability characteristics where $n_1 = 1, n_2 = 2, n_3 = 3, n_4 = 4$

resources in degraded states. In deployments, a resource provider will use benchmark applications to measure the degraded performance in the different system states to set the appropriate pricing.

7.1.3 An Example

As an example, consider an application with different run times on different machines, as shown in Table 1. We use example values for the failure-to-repair-rate (ρ) and performance degradation factors (x, n_1, n_2, n_3, n_4) to study the variation in expected steady state reward rates. If we considered only performance, we would pick machine D as it completes the application most quickly. If we were to select a resource based on reliability, we would pick machine A, the one with the lowest failure-to-repair ratio.

Now let us calculate the performability for the application running on machine D at a degradation factor of $x = 2$ where $T = 15$, $\rho = 0.6$ and $n_1 = 1, n_2 = 2, n_3 = 3, n_4 = 4$ using equation 7.4

$$
\begin{aligned}
E[Z] &= \Sigma r_i \pi_i \\
&= r_0 * \pi_0 + r_1 * \pi_1 + r_2 * \pi_2 + r_3 * \pi_3 + r_4 * \pi_4 \\
&= \frac{1}{T} * (1-\rho) + \frac{1}{T + n_1 * x} * \rho * (1-\rho) + \frac{1}{T + n_2 * x} * \rho^2 * (1-\rho) \\
&\quad + \frac{1}{T + n_3 * x} * \rho^3 * (1-\rho) + \frac{1}{T + n_4 * x} * \rho^4 * (1-\rho) \\
&= \frac{1}{15} * (1-0.6) + \frac{1}{15 + 1 * 2} * 0.6 * (1-0.6) + \frac{1}{15 + 2 * 2} * (0.6)^2 * (1-0.6) \\
&\quad + \frac{1}{15 + 3 * 2} * (0.6)^3 * (1-0.6) + \frac{1}{15 + 4 * 2} * (0.6)^4 * (1-0.6) \\
&= 0.027 + 0.014 + 0.008 + 0.004 + 0.002 \\
&= 0.055
\end{aligned}
$$

For a low performance degradation factor (x=2), performance outweighs the importance of the reliability, making machine D superior. However, at higher performance degradation factor (x=100), machine A is a better choice than machine D.

Similarly we assign the cost rates in the different states to be $c_0 = 1, c_1 = 0.9, c_2 = 0.8, c_3 = 0.7, c_4 = 0.6$ units/hr. We calculate the steady-state effective cost rate (using equation 7.7) that assigns a basic rate for the resource accounting for its reliability characteristics. We see that machine A has the highest cost rate whereas machine D has the lowest cost rate. Using the cost-rate and the projected application running time we finally calculate the cost for the given application on the given resources. At lower degradation factor, the cost on machine D is significantly less than the cost on the other machines. However as the degradation factor increases, the cost increases since the applications are projected to take longer on the resource. At higher degradation factor of $x = 100$ the cost of application running on machine D is still the cheapest but the cost difference is minimal.

Thus, the combined analysis of performance and reliability and cost through the performability metric helps resource selection decisions by considering multiple dimensions of resource behavior.

7.2 Workflow Planning for Performability

Complex scientific applications in distributed environments are composed as workflows where each step is a parallel or sequential application with a specific programming model. These complex applications are often run over a distributed set of resources that are selected based on the performance of the applications on a resource and associated data movement costs [20]. Moreover, workflows in domains such as mesoscale meteorology [54] and storm surge modeling [138] are time sensitive and often require additional fault tolerance strategies to meet deadlines. Time sensitive workflows are dependent on both high performance and reliability, making performability analysis critical. The first stage in workflow planning is a resource selection based on performability characteristics and programming models. The second stage includes resource mapping to reduce the makespan or cumulative workflow execution time and applying fault tolerance strategies to increase the reliability. In this section, we discuss the implications of performability analysis on workflow scheduling and fault tolerance strategies in greater detail.

7.2.1 Programming Models

Grid applications have different programming models that affect resource selection and fault tolerance strategies (illustrated in Chapter 3). The effective system performability is the minimum performability of its individual components. For example, a Master-Worker programming model needs higher reliability for the master and the communication network connecting the master and worker. Typically in the master-worker programming model, the masters are longer lived than the

workers. The performability of the master can be considered to be the system performability for a master-worker application.

$$E_{(M-W)} = Minimum\,(E_{Master}, E_{Workers}, E_{Network}) = E_{Master} \tag{7.10}$$

when $T_{Master} >> T_{Workers}$ and $T_{Master} >> T_{Network}$ and T_{Master} and $T_{Workers}$ is the running time of the master and workers respectively, and $T_{Network}$ is the effective data transfer time on the network links.

The Divide and Conquer model is an extended case of the master-worker paradigm where each subtasks might spawn additional tasks. Typically, the higher the task is in the tree, the longer the running time and the more critical is its performability. Hence extending the model from the master-worker discussion, we can see that the performability of this programming model will be the performability of the head of the chain.

In an SPMD computation, multiple sub-components operate on different pieces of data. The subcomponents communicate with each other requiring the entire SPMD computation to be scheduled on resources that have similar performability characteristics. The performability of the SPMD computation is: $E_{(SPMD)} = Minimum(E_{systemcomponents})$.

7.2.2 Workflow scheduling

Heuristic techniques are typically used for workflow-level planning and scheduling [205]. Weights are assigned to the nodes of the graph representing the computational needs of the task on particular resources. The edges are assigned values representing the communication (i.e. data transfer

needs) between the adjacent tasks. By applying performability analysis at two levels - the computational resources and the network - we can obtain the application's overall execution time given the failure levels of resources it might encounter. This model can be expanded to model storage and other resources as well. Specifically, using equations (7.1), (7.2) and (7.4) we can obtain the expected accumulated reward rate for an application on the given resources. The projected application running time that accounts for the performance variations during resource failures is given by:

$$T_{projected} = 1/E[Z]_{(computation)} \tag{7.11}$$

where $T_{(projected)}$ is the projected application running time and $E[Z]_{(computation)}$ is the performability of the computational elements.

In systems where application performance is unaffected by reliability levels, systems can be considered to be perfect (i.e., $\rho = 0$ and thus from equation(7.2), the probability of being in the "High" state is one). Thus, the performability for an application would be 1/T and the projected application running time would be T. Note that these are the performance estimates used in current day workflow scheduling (i.e, no failures). Our model enhances existing mechanisms by providing a more detailed estimate of the running time capturing performance variations across reliability levels.

Next, we construct a Markov Reward model for the network availability where the reward rates are the inverse of the data transfer time at different availability and reliability levels. The failure-and-repair rates would represent the availability and reliability of the network links between sites. In this case, the projected data transfer time in presence of various reliability levels would be the

inverse of the network performability.

$$T_{projected-data} = 1/E[Z]_{(communication)} \tag{7.12}$$

where $T_{projected-data}$ is the projected data transfer time based on the performability analysis, and $E[Z]_{(communication)}$ is the network performability.

Once these values are determined, traditional scheduling algorithms [188] can be applied to the workflows and the schedule with minimum makespan can be selected for the workflow execution plan (more details in Chapter 9).

7.2.3 Fault Tolerance Strategies

As discussed earlier, some workflows such as weather forecasting are time sensitive and must meet deadlines for the forecast to be useful. In these cases, users often specify a deadline for workflow completion. When considering the makespan, it is possible to judge whether this deadline can be met by using the projected workflow completion time. Often the composite system reliability can be enhanced by applying additional fault tolerance strategies. Two commonly used fault tolerance strategies are replication and checkpoint-restart; each with different trade-offs. Given unlimited resources, all components could be replicated to increase effective reliability without affecting performance. In practice, users will incur costs on application runs (e.g. service units spent in TeraGrid or cost of resource time on Amazon EC2), necessitating a balance between performance and reliability. On the other hand, checkpoint-restart guarantees a very high level of reliability (probability of successful completion is almost 1) but at the cost of performance degradation due to checkpoint overheads. We use a simple cost-model and the performability analysis presented earlier to determine if a fault tolerance strategy might improve system performability for

an application

Consider the cost of replication to be the lost time on the resources that could have been used by another application. Further assume that an application is replicated on a resource with similar performability. The cost of replication (C_R) can be represented as:

$$C_R = T_{projected} * n \tag{7.13}$$

where n is the number of replicas and $T_{projected}$ is the application running time from the performability analysis. Similarly, the cost of checkpoint-restart (C_{CR}) is represented as

$$C_{CR} = C_{checkpoint} + C_{restart-on-failure} \tag{7.14}$$

$$C_{checkpoint} = C_{per-checkpoint} * T_{projected}/T_{interval} \tag{7.15}$$

where $T_{projected}$ is the application running time from the performability analysis, and $T_{interval}$ represents the optimal checkpoint interval to meet the performability level. Replication will be the preferred fault-tolerance strategy if $C_R < C_{CR}$.

We present a more detailed analysis of the optimal fault tolerance strategy in Section 7.3 for different application running times. Once the fault tolerance strategy is determined performability modeling can be used to recalculate the projected application running time from the model above. Specifically, these values will then be used to estimate task completion times during each iteration of the workflow scheduling algorithm as shown below.

Case 1: Checkpoint-restart. In this case $T_{projected-FT}$ will be the new expected running time ($\rho = 1$) calculated as $T_{projected-FT} = T_{projected} + C_{checkpoint} + C_{restart-on-failure}$.

Case 2: Replication. In replication, if the resources that are used for replication are similar, the $T_{projected-FT}$ is obtained by the performability analysis described (where $\rho = 1 - \pi(1 - \rho_i)$ and $i \in 1, n$) .

7.3 Evaluation

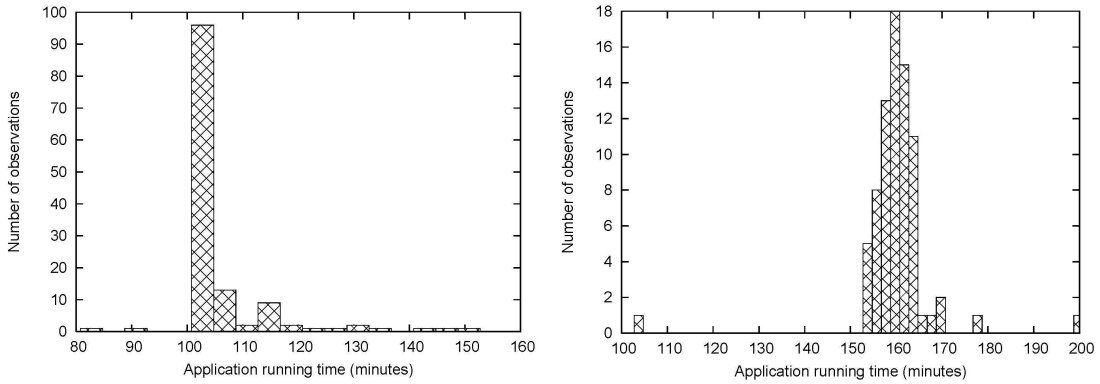
It is critical to understand both the application's characteristics and resource behavior when considering a resource for scheduling or making fault tolerance decisions. We present results collected from the execution of a scientific application on TeraGrid that highlight the variability in application running time in Section 7.3.1. In Section 7.3.2, we present experimental data from a set of meteorological and ocean modeling applications [138] that are subjected to simulated interferences to degrade resource reliability, with consequent changes in observed performance. The experimental data is then provided as input to our performability model and analyzed with respect to varying failure-to-repair ratios. We study the effect of application running time and performance degradation factor(x, n_i) on application performability at different failure-to-repair ratios in Section 7.3.3. We evaluate the various parameters affecting the fault tolerance strategy for applications in Section 7.3.4.

7.3.1 Application Performance Variability

Figure 7.2 shows histograms of application running times of WRF (a weather prediction code, see Table 7.2) on two TeraGrid NCSA clusters observed over two weeks. Mercury is an IBM IA-64 cluster with a mixture of 1.3GHz and 1.5GHz Intel Itanium 2 processors and Tungsten is a collection of Dell PowerEdge 1750 servers with Intel Xeon 3.2GHz processors. The data presented consists of 132 runs on Mercury and 77 runs on Tungsten. On mercury a large number of application runs

Name	Application Description
arps2wrf	Generates initial and lateral boundary conditions for WRF.
wrfstatic	Processes static data sets such as terrain, vegetation, soil texture, etc that serves as input for a meteorological model WRF.
adcirc	Finite element hydrodynamic model for storm surge modeling (run on 64 processors)
wrf	Mesoscale numerical weather prediction system (run on 128 processors)

Table 7.2: Application Descriptions. The table provides a bried description of the application codes from weather and ocean modeling that we use for our experiments.



(a) (b)

Figure 7.2: Application Performance Variation. Figure shows the running time variability observed for WRF over TeraGrid machines (a) Mercury (b) Tungsten.

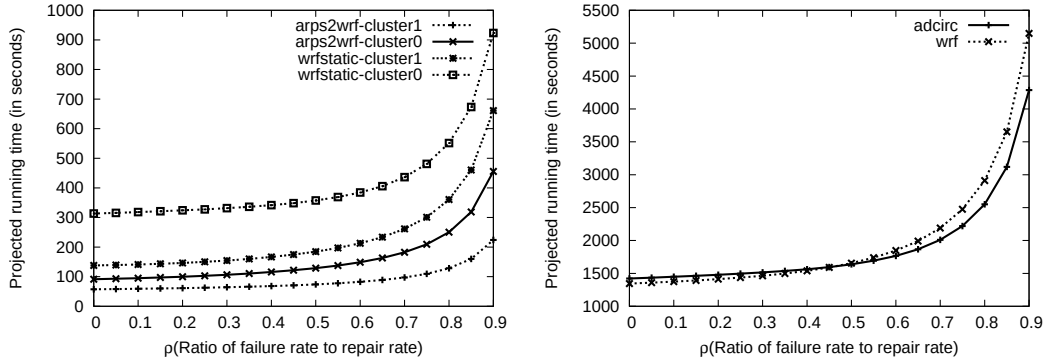


Figure 7.3: Effects of Failure Levels on Applications. The projected application running time for meteorological and ocean modeling applications (a) Short running - arps2wrf, wrfstatic (b) Long running - wrf, adcirc.

take around 103 minutes. However 27% of observations fall outside this range. On Tungsten most run times are distributed from 150 to 170 minutes. This variation of 20 minutes can significantly impact workflow planning strategies. Workflow scheduling needs to account for this variation while mapping applications to resources especially for deadline-sensitive applications.

7.3.2 Effect of Failure Levels on Applications

In this section, we present experimental data on application running times induced with simulated failure levels. Table 2 provides brief application descriptions, which consist of a mix of single-node preprocessing applications and message-passing-interface (MPI) multiprocessor jobs. We subject the applications to simulated availability stress tests that affect memory, cpu and network bandwidth (for the MPI jobs). A matrix multiplication, a program blocking memory and Test TCP (TTCP) benchmarking tool [180] were run individually and then in combination during application execution. Data for single node jobs were collected on a 35 node Linux cluster with Intel Xeon processors running at 3.2 GHz (cluster 0) and on a 70 node Dell PowerEdge cluster, where each node has 2 x 2.66Ghz Intel Woodcrest 5150 (dual core) processors (cluster 1). Data for wrf and

adcirc was collected on the Dell PowerEdge cluster (cluster 1).

The application running times at different failure levels are then substituted in our model (Section 7.1) to study the performability under different failure-to-repair rates. In this experiment, we make a assumption that considers performance levels in the Good and Medium levels to be identical for MPI jobs and Poor and Low states to be identical for the single-node and MPI jobs. This simplifying assumption is appropriate since all systems might not exhibit all failure states and corresponding performance fluctuations. Figure 7.3(a) shows the projected application running time for arps2wrf and wrfstatic on both clusters. Both applications perform better on cluster 1, however the execution time of wrfstatic on cluster1 is affected more than on cluster0 when subjected to failures. Similarly arps2wrf on cluster0 is affected at higher values of failure-to-repair rate. We compare wrf and adcirc (Figure 7.3(b)) which have similar running time (on different number of processors), as the failure-to-repair rates increase we see that wrf is more affected than adcirc. Thus we see that different failures modes and underlying hardware characteristics have an impact on applications.

7.3.3 Factors affecting Performability

In this section, we consider more general cases of application running time to explore the parameter space. Figure 7.4 shows the expected steady state reward rate with varying values of failure-to-repair rates for performance degradation factors x=2 and x=70 for a range of application running times (20 -140 minutes), that are typical of grid applications. At x=2 (Figure 7.4(a)), and for application running time of 20 minutes we see that the performability does not change for smaller values of ρ but decreases significantly for large values of ρ. At x=70 (Figure 7.4(b)), the performability decreases almost linearly for application running time of 20 minutes. As the application running time increases, the expected reward rate stays constant for larger values of ρ. Workflows

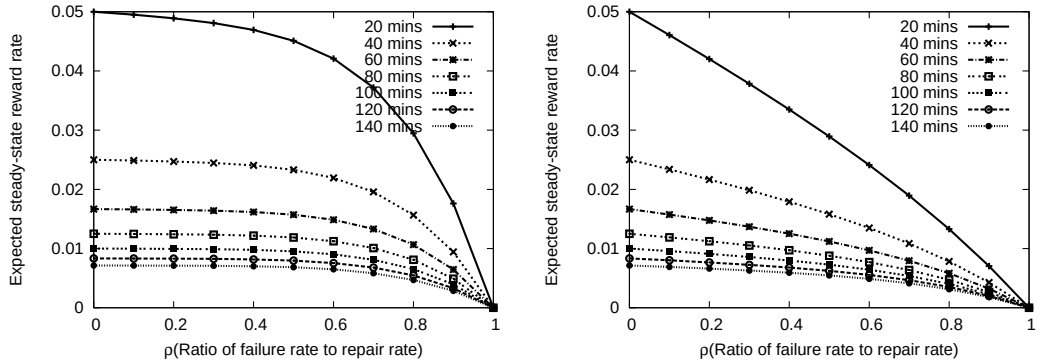


Figure 7.4: Study of Performance with Availability Variations. The expected steady-state reward rate for different application run times with performance degradation factors $n_1 = 1, n_2 = 2, n_3 = 3, n_4 = 4$ (a) $x = 2$ and (b) $x = 70$.

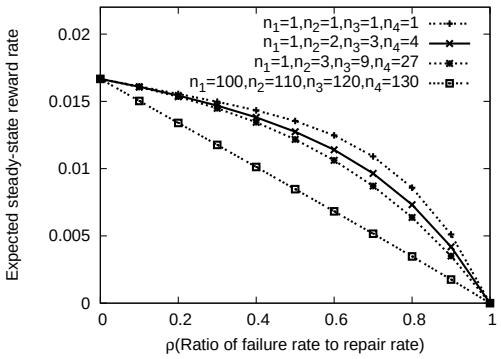


Figure 7.5: Study of Performance Degration Factors on an Application. The expected steady-state reward rate for different n_i values for an application with running time of 60 minutes and $x = 30$.

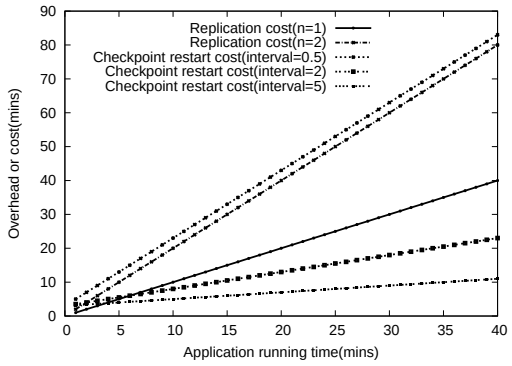


Figure 7.6: Cost analysis of Fault Tolerance strategies. Figure shows a comparison of costs with replication and checkpoint-restart strategy for different application running times where $C_{per-checkpoint} = 1$ and $C_{restart-on-failure}$ is 3.

typically consist of a mix of jobs with different running times. From this analysis we see that longer running components might be less sensitive to ρ allowing more scheduling alternatives than the more sensitive components (low running times) that might need to be run on specific machines. A workflow algorithm in the future might account for these characteristics when optimizing resource choices for workflow tasks. This can help minimize costs in workflow planning. Although generic metrics like MTTF and MTTR can indicate the general suitability of resources, it is critical to consider the application's running time in making resource selection decisions. At small values of n_i the performability decreases linearly. At higher values of n_i the decrease is more rapid as resource reliability decreases (Figure 7.5). For large values of n_i (i.e. performance difference between the High and other states is significant) the performability decreases linearly.

7.3.4 Fault Tolerance Strategies

Fault tolerance strategies depend on application and resource characteristics. To determine a fault tolerance strategy associated costs for each technique must be considered. The most commonly used fault tolerance strategies are replication and checkpoint-restart. From our earlier analysis, the system will replicate if the cost of replication is lower than the cost of checkpoint-restart: $C_R < C_{CR}$.

Figure 7.6 shows the cost of fault tolerance strategies for different replication factors and checkpoint intervals. Replication is more cost effective for smaller application running times. If we checkpoint too frequently, the cost of checkpoint-restart increases, as expected. If the checkpoint interval is very low (i.e, 0.5 minutes) the cost of checkpoint-restart is significantly higher and replicating twice is more cost-effective. Thus, we can apply a fault tolerance strategy and select appropriate values for the replication factor and checkpoint-interval to minimize cost while increasing performability. If the application is very critical and the cost of replication and checkpoint-restart is immaterial both strategies might also be used simultaneously

7.4 Summary

In this chapter, we presented a framework that provides a basis to evaluate the performance of distributed resources in the presence of failures. We explore the trade-offs of performability on resource selection and fault tolerance strategies appropriate for different programming models of scientific applications. The joint treatment of performance and reliability using performability analysis through Markov Models lays the foundation for next generation dynamic workflow scheduling and fault tolerance strategies required in grid and cloud environments. The performability model provides a generic framework that allows plug-and-play of resource behavior to study the

variation in QoS and its effect on performability guarantees to the application. We explore one such approach in using the performability model for workflow scheduling in Chapter 9. As more distributed resource deployments make available underlying resource reliability information, workflow planning components will be able to use that information with application characteristics for appropriate workflow planning and resource management decisions.

8

Resource Layer

Grid and cloud computing systems have evolved to provide different abstractions at the resource layer. We compared and contrasted the different resource models and their interfaces and capabilities in Chapter 3. Each system provides different types of access, cost modeling and Quality of Service(QoS) capabilities. The resource layer is responsible for interacting with these different systems to support mechanisms that can balance the needs of both resource providers and consumers.

Resource provider sites should have autonomy to control how much of each resource type they allocate to each consumer at any given time. Resource consumers need predictable service quality (performance isolation and reliability expectation) even in the presence of competition for shared resources. Service quality is especially crucial for urgent computing applications such as weather prediction and disaster response.

In this chapter, we explore the resource layer abstractions, interfaces and interactions across different resource systems. Specifically,

- We propose a hosting model in which independent, self-contained middleware deployments

run within isolated containers on shared resource provider sites. Sites and hosted environments interact via an underlying resource control plane to manage a dynamic binding of computational resources to containers. Central to the hosting model is GROC (Grid Resource Oversight Coordinator). GROC is an implementation of resource coordinator that manages the dynamic containers (i.e., instances of slots) for the end user. The resource coordinator interacts with the resource mechanisms at the sites to query, procure a dynamic binding of resources. Our implementation is built with Shirako, a leasing framework for cluster sites and the hosted middleware is Globus middleware. However, the design and architecture are more widely applicable to cloud sites.

- We also propose the lowest-common-denominator probabilistic Quality of Service (QoS) abstractions atop grid and cloud services that enables providers to quantify the variation in resource availability in probabilistic measures. Users of both grid and cloud environments cannot expect strong QoS assurances since they experience reliability variations due to hardware and software failures and availability fluctuations from shared user environments. The probabilistic abstractions allows resource providers to realistically quantify the level of service.

The proposed resource layer abstractions in this chapter enables higher level tools to assess resource status and implement higher level policies and techniques to meet user needs (more details in Chapter 9).

The rest of this chapter is organized as follows. We discuss the dimensions to resource control policy in grid and cloud systems. We present the container based hosting model in Section 8.2 and discuss the resource co-ordinator's functions and roles in greater detail in Section 8.3. We discuss our probabilistic QoS model in greater detail in Section 8.4.1. Finally we evaluate the container hosting model and the probabilistic resource acquisition mechanisms in Section 8.5.

8.1 Resource Control Policy

In grid systems, user communities, or virtual organizations (VOs), generate streams of jobs to execute on shared resource sites, e.g., cluster farms. Similarly, cloud clients or SaaS (Software as a Service) providers generate streams of resource requests. Cluster sites and data centers provide computational resources to these virtual organizations. We refer to the entities that generate the jobs as *application managers*. The term denotes a domain-specific entry point to a set of distributed resources; VO users may submit jobs through a portal framework or gateway, a workflow manager, or a simple script interface. Figure 8.1(a) depicts an example of a standard Globus grid with two VOs executing on two sites. A VO's application manager submits each task to a "gatekeeper" at one of the sites, which validates it and passes it to a local batch scheduling service for execution. There are four key aspects to resource control policy in such a system:

- *Resource allocation to VOs.* The sites control their resources and determine how to allocate them to serve the needs of the competing user communities. A site may assign different shares or priorities to contending VOs, and/or may hold resources in reserve for local users.
- *Resource control within VOs.* VOs determine the rights and powers of their users with respect to the resources allocated to the VO.
- *Task routing.* The application managers for each VO determine the routing of tasks to sites for timely and efficient execution.
- *Resource recruitment.* Entities acting on behalf of the VOs negotiate with provider sites for resources to serve the VO's users.

Grid sites such as TeraGrid and Open Science Grid implement their own *resource allocation policies* as job-level policies within the batch schedulers in current practice in grid systems. A scheduler

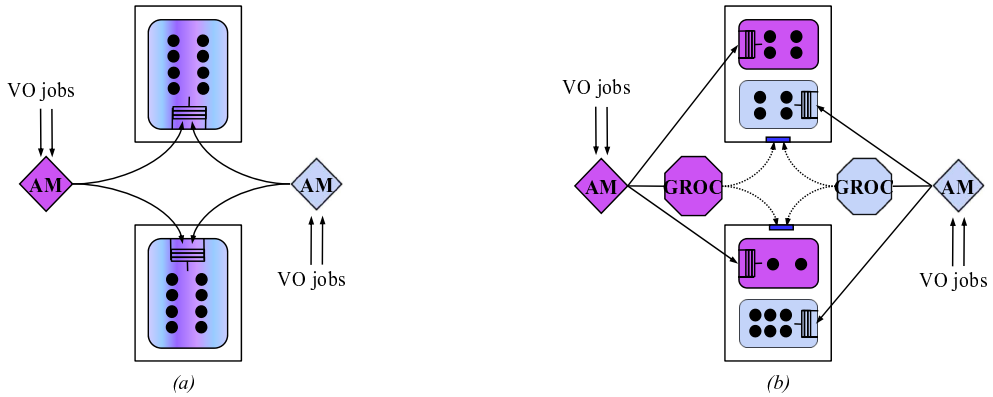


Figure 8.1: Two Architectural Alternatives for Serving Multiple User Communities, or VOs. In (a), the VOs' application manager (AM) submit jobs through a common gatekeeper at each site; job scheduling middleware enforces the policies for resource sharing across VOs. In (b), each VO runs a private grid within isolated workspaces at each site. Isolation is enforced by a foundational resource control plane. Each VO grid runs a coordinator (GROC) that controls its middleware and interacts with the control plane to lease resources for its workspaces.

may give higher priority to jobs from specific user identities or VOs, may export different queues for different job classes, and may support job reservations. *Resource recruitment* is based primarily on reciprocal and social agreements requiring human intervention; a recent example is the notion of *right-of-way tokens* in the SPRUCE [14] gateway extensions for urgent computing. Most cloud deployments have static resource allocation policies. Currently, Amazon EC2 allows users to register on the website and access upto 20 machine instances. Additional instance requests must be pre-approved through out-of-band communication. Many current deployments also rely on ad hoc routing of tasks to grid sites, given the current lack of standard components to coordinate task routing. However for next-generation dynamic application environments it is critical that the software stack has mechanisms to represent and enforce these four dimensions of resource control.

8.2 Container Hosting Model

Figure 8.1(b) depicts the architectural model we propose for hosted grids with container-level resource control. Each site instantiates a logical container for all software associated with its hosting of a given VO. The container encapsulates a complete isolated computing environment or *workspace* [88] for the VO grid's point-of-presence at the site, and should not be confused with the individual JVMs that run Java components at the site. Each VO grid runs a separate batch task service within its workspace. The site implements resource control by binding resources to containers; the containers provide isolation, so each instance of the batch scheduler only has access to the resources bound to its container, and not other resources at the site. Condor-G "gliding-in" [67] provides similar mechanisms to instantiate appropriate Condor services in Globus based batch queue systems. Similar mechanisms are also available in the Virtual Grid Execution System (vgES) to enable slots on grid and cloud systems (discussed in Chapters 3 and 10). However Condor-G and vgES does not address dynamic container resizing that is addressed in the container model we propose in this chapter.

Thus we propose integrating resource control functions at two different levels of abstraction: *jobs* and *containers*. Jobs —individual independent tasks or tasks in a workflow—are the basic unit of work for high-throughput computing, so middleware systems for clusters and grids focus on job management as the basis for resource control. Our premise is that the architecture should also incorporate resource control functions at the level of the logical context or "container" within which the jobs and the middleware services run. Advances in virtualization technologies—including but not limited to virtual machines—create new opportunities to strengthen container abstractions as a basis for resource control and for isolation and customization of hosted computing environments [33, 83, 88, 172, 173].

In essence, in our container model we propose a "Grid" comprising a set of autonomous resource provider sites hosting a collection of independent "grids":

- Each grid serves one or more communities; we speak as if a grid serves a single VO, but our approach does not constrain how a hosted grid shares its resources among its users.
- Each grid runs a private instance of its selected middleware to coordinate sharing of the data and computing resources available to its user community.
- Each grid runs within a logically distributed container that encapsulates its workspaces and is bound to a dynamic "slice" of the Grid resources.

In this chapter, we show how hosted grids can negotiate with the resource control plane to procure resources across grid sites in response to changing demand. We present the design and implementation of a prototype system based on the Shirako [83] toolkit for secure resource leasing from federated resource provider sites. Cluster sites are managed with Cluster-on-Demand [33] and Xen virtual machines [13]; the hosted grid software is based on the Globus Toolkit (GT4) [58]. Within this supporting infrastructure, we explore coordinated mechanisms for programmatic, automatic, service-oriented resource adaptation for grid environments.

8.2.1 Resource Coordinator

While the sites control how they assign their resources to each hosted grid, the grids control the other three policies internally. We propose that each hosted grid include a coordinating manager, which we will call the GROC—a loose acronym for Grid Resource Oversight Coordinator.[1] The

[1]The novelist Robert Heinlein introduced the verb *grok* meaning roughly "to understand completely". The verb "groak" [27] originally was used to refer to *watching people silently while they eat, hoping they will ask you to join them*. Both meanings have significance here. The name GROC emphasizes that each hosted grid has a locus of resource policy that operates with a full understanding of both the resources available to the grid and the grid's demands on its resources. GROC also actively *watches* the resources and can opportunistically procure them.

GROC performs two interrelated functions, which are explained in detail in Section 8.3:

- The GROC is responsible for advising application managers on the routing of tasks to sites. In this service brokering role the GROC can be called a *metascheduler* or *superscheduler*.
- The GROC monitors the load and status of its sites (points of presence), and negotiates with providers to grow or shrink its resource holdings. It may resize the set of batch worker nodes at one or more sites, set up new grid sites on resources leased from new providers, or tear down a site and release its resources.

The GROC thus serves as the interface for a VO application manager to manage and configure its resource pool, and may embody policies specific to its application group. Crucially, our approach requires no changes to the grid middleware itself. Our prototype GROC is a service built atop the Shirako and Globus toolkits and it is the sole point of interaction with the underlying resource control plane.

8.2.2 Resource Control Plane

The GROC uses programmatic service interfaces at the container-level resource control plane to acquire resources, monitor their status, and adapt to the dynamics of resource competition or changing demand. The control plane is based on the SHARP [68] leasing abstractions as implemented in the Shirako toolkit [83]. Each lease represents a contract for a specified quantity of typed resources for some time interval (*term*). Each resource provider runs a local resource manager called Cluster-on-Demand (COD [33]), and exports a service to lease *virtual clusters* from a shared server cluster. Each virtual cluster comprises a dynamic set of nodes and associated resources assigned to some guest (e.g., a VO grid) hosted at the site. COD provides basic services for booting and

imaging, naming and addressing, and binding storage volumes and user accounts on a per-guest basis.

The GROC interacts with the site to configure its virtual clusters and integrate them into the VO's grid (Section 8.3.4). When the lease expires, the grid vacates the resource, making it available to other consumers. The site defines local policies to arbitrate requests for resources from multiple hosted grids. In our prototype the leased virtual clusters have an assurance of performance isolation: the nodes are either physical servers or Xen [13] virtual machines with assigned shares of node resources. In our implementation we use Xen VMs because they boot faster and more reliably than physical servers, but the concept applies equally to physical servers.

8.2.3 Separation of Concerns

While the hosted VOs and their grid middleware retain their control over job management, the GROC managers interact with the resource control plane to drive the assignment of resources to VOs. The assignment emerges from the interaction of GROC policies for requesting resources and the resource provider policies for arbitrating those resource demands. In effect, the architecture treats the grid nodes and their operating systems as managed entities. Provider sites allocate resources to workspace containers without concern for the details of the middleware, applications, or user identities operating within each workspace isolation boundary.

Grid hosting with container-level management is particularly important as the grid evolves toward a stronger separation between resource providers and consumers. TeraGrid and Open Science Grid are examples of the growth of large infrastructure providers in the academic community. A similar trend has been observed in industry from cloud offerings through Amazon, Google, Microsoft, etc. They signal a shift from a traditional emphasis on reciprocal peer-to-peer resource sharing within VOs to a new emergence of resource providers that serve computational resources

to multiple competing user communities or VOs. Containment and container-level management also enable resource providers to serve more diverse needs of their VOs. A resource provider site can host different grid stacks or other operating software environments concurrently.

Our approach assumes that the grid middleware can adapt to a dynamically changing set of worker nodes at the sites. In fact, adaptation is always required in a dynamic world: compute servers may fail or retire, and provider sites deploy new servers in response to bursts of demand or funding. With a container based hosting model, grids and clouds may grow dynamically to use additional resources as they become available. One limitation is that batch services often do not have adequate support to checkpoint or reschedule nodes when worker nodes fail or shutdown. Checkpointing, replication and migration continue to be active research topics, and these capabilities are increasingly crucial for long-running jobs in a dynamic world. We investigated the tradeoffs involved with checkpoint-restart and replication for applications in dynamic resource environments that can be applied to long jobs running in Chapter 7. These principles are applicable to jobs running within a container. Further investigation of these topics during runtime execution is outside the scope of this work.

8.3 GROC

We present the design and implementation of a prototype system that coordinates dynamic resource leasing and task routing, based on the grid hosting architecture outlined above. Our prototype uses the standard Globus Toolkit (GT4) for resource management within each hosted grid: job management, resource discovery, identity management and authorization, and file transfer. Dynamic resource leasing is based on Shirako, a service-oriented toolkit for constructing SHARP resource managers and COD cluster sites [83].

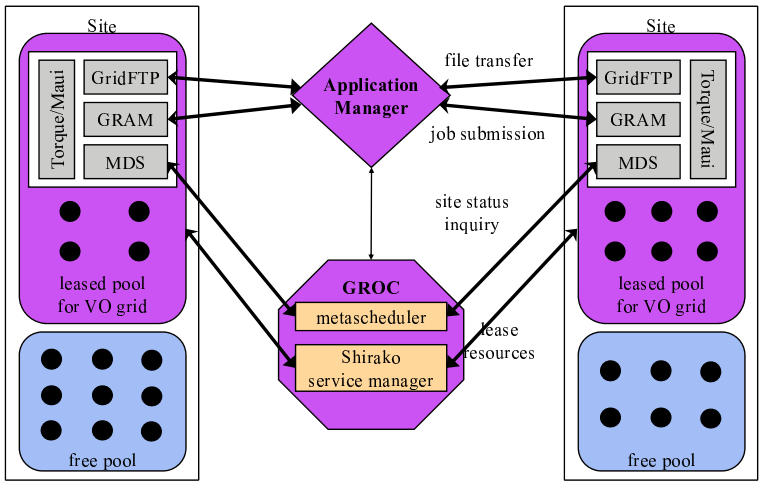


Figure 8.2: GROC Components. Overview of components for a GROC managing a VO grid hosted on virtual clusters leased from multiple cluster sites. The application manager interacts with Globus services, instantiated and managed by the GROC, for job and data management.

Figure 8.2 illustrates the interactions among the most important components within a hosted grid, as implemented or used in the prototype.

- The nucleus of the hosted grid is the GROC, which orchestrates task flow and resource leasing. The GROC is the point of contact between the Globus grid and the Shirako resource control plane.
- The application managers (e.g., portals) control the flow of incoming job requests. They consult the GROC for task routing hints (Section 8.3.2), then submit the tasks to selected sites.
- A Globus Resource Allocation Manager (GRAM) runs on a master node (*head node*) of a virtual cluster at each provider site, acting as a gatekeeper to accept and control tasks submitted for execution at the site.
- The application managers interact with a secure staging service on each head node to stage data as needed for tasks routed to each site, using Reliable File Transfer (RFT) and Grid File

Transfer Protocol (GridFTP).

- When a task is validated and ready for execution, GRAM passes it to Torque, an open-source batch task service incorporating the Maui job scheduler.
- The GROC receives a stream of site status metrics as a feedback signal to drive its resource requests (Section 8.3.1). Each site exposes its status through a Globus Monitoring and Discovery Service (MDS) endpoint.
- The GROC acts as a Shirako *service manager* to lease resources on behalf of the VO; in this way, the GROC controls the population of worker nodes bound to the hosted grid's batch task service pools (Section 8.3.3). The GROC seamlessly integrates new worker nodes into its grid (Section 8.3.4) from each site's free pool.

The following subsections discuss the relevant aspects of these components and their interactions in more detail.

8.3.1 Site Monitoring

In our prototype, the GROC acts as a client of WS-MDS (a web service implementation of MDS in GT4) to obtain resource status at each site, including the number of free nodes and the task queue length for each batch pool. The WS-GRAM publishes Torque scheduler information (number of worker nodes, etc.) through the MDS aggregator framework using the Grid Laboratory Uniform Environment (GLUE) schema. MDS sites may also publish information to upstream MDS aggregators; in this case, the GROC can obtain the status in bulk from the aggregators.

The GROC queries the MDS periodically at a rate defined by the MDS poll interval. The poll interval is a tradeoff between responsiveness and overhead. We use a static poll interval of 600

ms for our experiments. The results of the *site poll* are incorporated immediately into the task routing heuristics. A simple extension would use MDS triggers to reduce the polling, but it is not a significant source of overhead at the scale of our implementation.

8.3.2 Task Routing

A key function of the GROC is to make task routing recommendations to application managers. The GROC factors task routing and other resource management functions out of the application managers: one GROC may provide a common point of coordination for multiple application managers, which may evolve independently. The task routing interface is the only GROC interface used by a grid middleware component; in other respects the GROC is non-intrusive.

To perform its task routing function, the GROC ranks the sites based on the results from its site poll and a pluggable ranking policy. Information available to the policy includes cluster capacity at each site, utilization, and job queue lengths. In addition, the policy module has access to the catalog of resources leased at each site, including attributes of each group of workers (e.g., CPU type, clock rate, CPU count, memory size, interconnect).

The coordinating role of the GROC is particularly important when multiple user communities compete for resources. The GROC maintains leases for the resources held by the VO grid: its task routing choices are guided by its knowledge of the available resources. Since it observes the complete job stream, it can also make informed choices about what resources to request to meet its demand.

The goal of our work is to evaluate the hosting architecture, rather than to identify the best policies. Our prototype policy considers only queue length and job throughput for homogeneous worker nodes. More sophisticated techniques such as batch queue prediction [125] can be used for

job start predictions within the container. Also, we do not consider data staging costs. Job routing in our prototype uses a simple load balancing heuristic. It estimates the aggregate runtime of the tasks enqueued at each site, and the time to process them given the number of workers at each site. It selects the site with the earliest expected start time for the next job.

8.3.3 Resource Leasing

In the absence of support for resource leasing, the GROC could act as a task routing service for a typical grid configuration, e.g., a set of statically provisioned sites with middleware preinstalled and maintained by administrators at each site. In our system, the GROC can also use the resource control to change the set of server resources that it holds. The GROC invokes Shirako's programmatic resource leasing interface to acquire and release worker nodes, monitor their status, and/or instantiate points of presence at new cluster sites when resources are available and demand exists. This control is dynamic and automatic.

The GROC seeks to use its resources efficiently and release underutilized resources by shrinking renewed leases or permitting them to expire. This good-citizen policy is automated, so it is robust to human failure. An operator for the VO could replace the policy, but we presume that the VO has some external incentive (e.g., cost or goodwill) to prevent abuse. Note that our approach is not inherently less robust than a conventional grid, in which a greedy or malicious VO or user could, for example, submit jobs that overload a site's shared storage servers. In fact, the leased container abstraction can provide stronger isolation given suitable virtualization technology, which is advancing rapidly.

The GROC uses pluggable policies to determine its target pool sizes for each site. In Section 8.5.1, we define the policies used in our experiments. The prototype GROC uses a predefined preference order for sites, which might be based on the site's resources or reputation, peering agreements,

and/or other factors such as cost. Similarly, the sites implement a fixed priority to arbitrate resources among competing GROCs.

8.3.4 Configuring Middleware

Typically, grid middleware is configured manually at each site. One goal of our work is to show how to use Shirako/COD support to configure grid points of presence remotely and automatically. The responsibility—and power—to manage and tune the middleware devolves to the VO and its GROC, within the isolation boundaries established by the site. This factoring reduces the site's administrative overhead and risk to host a grid or contribute underutilized resources, and it gets the site operators out of the critical path, leaving the VOs with the flexibility to control their own environments.

Configuration of a COD node follows an automated series of steps under the control of the Shirako leasing core. When a site approves a lease request for new worker nodes, the GROC passes a list of *configuration properties* interpreted by a resource-specific plugin *setup* handler that executes in the site's domain. The *setup* handler instantiates, images, and boots the nodes, and enables key-based SSH access by installing a public key specified by the GROC. It then returns a lease with *unit properties* for each node, including IP addresses, hostnames, and SSH host keys. The GROC then invokes a plugin *join* handler for each node, which contacts the node directly with key-based root access to perform an automated install of the middleware stack and integrate the node into the VO's grid. Similarly, there is a *teardown* handler that reclaims resources (e.g., machines), and a *leave* handler that cleanly detaches resources from the middleware stack. To represent the wide range of actions that may be needed, the COD resource driver event handlers are scripted using *Ant* [6], an open-source OS-independent XML scripting package. We implemented *join* and *leave* handler scripts to configure the middleware components shown in Figure 8.2.

To instantiate a point of presence at a new site, the GROC first obtains separate leases for a master node (with a public IP address) that also serves as a scratch storage server for data staging. It instantiates and configures the Globus components, Torque and Maui on the master, and configures the file server to export the scratch NFS volume to a private subnet block assigned to the virtual cluster. When a new worker node joins, the *join* handler installs Torque and registers the worker with the local master node. The *join* handler for the master configuration is about 260 lines of Ant XML, and the worker join handler is about 190 lines.

Our prototype makes several concessions to reality. It assumes that all worker nodes are reachable from the GROC. It is possible to implement a proxy that handles the worker *join* operations through the public head node for each virtual cluster so that workers may use private IP addresses. This work is outside the scope of this thesis. The *setup* attaches a shared NFS file volume containing the Globus distribution to each virtual cluster node, rather than fetching it from a remote repository. For the sake of simplicity, all the hosted grids use a common certificate authority (CA) that is configured using Globus's SimpleCA, although there is nothing in the architecture or prototype that prevents the hosted grids from each using a private CA. Interaction with the CA is not yet automated; instead, the GROC has preconfigured host certificates for the DNS names that its master nodes will receive for each potential site that it might use. Our implementation also uses a set of common user identities that are preconfigured at the sites. Finally, we prestage all applications and data required by the VO's users when we instantiate the site.

We use the default First Come First Served (FCFS) scheduling policies for Torque/Maui, but the GROC is empowered to set policies at its points of presence as desired. Thus, the application manager is able to rely on the VO's GROC to implement policies and preferences on how its available resources might be used by different members of the community, and to adapt these policies as the resource pool size changes.

8.3.5 Robustness

The GROC is stateless and relies on recovery mechanisms in Shirako, which maintains all lease state in a local LDAP repository. If a GROC fails, it will recover its knowledge of its sites and resource holdings, but it will lose its history of task submissions and the MDS feedback stream from the sites. Once recovered, the GROC maintains its existing leases and monitors grid operation for a configurable interval before adjusting its lease holdings. Reliable job submission and staging are handled using existing Globus mechanisms that do not involve the GROC.

As noted in Section 8.2.3, robust grid services must be capable of restarting jobs when nodes fail or leave the service. In our approach, nodes may depart due to resource competition, as governed by the site policies and the GROC interactions with the dynamic resource control plane. Although the GROC has advance warning of node departures, the Torque batch service in our current prototype is not able to suspend or migrate tasks running on those nodes; thus some tasks may be interrupted. We believe that support for virtual machine *checkpoint/migrate* is a promising path to a general solution. Xen supports live VM migration, but we do not explore its use for robust adaptation.

8.3.6 Security

The important new security requirement of our architecture is that each GROC must have a secure binding to each of its candidate hosting sites. Each SHARP actor has a keypair and digitally signs its control actions. To set up the trust binding, there must be some secure means for each site and GROC to exchange their public keys. Other related systems that delegate policy control to a VO, or a server (such as a GROC) acting on behalf of a VO, also make this assumption. Examples include the VO Membership Service (VOMS) [1] and Community Authorization Service (CAS) [134].

Similarly, Amazon EC2 uses X.509 certificate for authentication to their cloud systems [5].

One solution is to designate a common point of trust to endorse the keys, such as a shared certificate authority (CA). Although each grid selects its own CA to issue the certificates that endorse public keys within the grid, the provider site authorities exist logically outside of the VO grids in our architecture. Thus reliance on a common CA would presume in essence that the public key certificate hierarchy (PKI) extends upwards to include a common CA trusted by all resource provider sites and all hosted grids. An alternative is to rely on pairwise key exchange among the sites and VO operators. In this prototype the public keys for the brokers and GROC s are installed through a manual operator interface.

To instantiate a new site point of presence, the GROC passes the gateway host certificate and private key in an encrypted connection during *join*. Note, however, that the GROC cannot hide the site private keys used by its middleware from the hosting resource provider, since the resource provider knows the private SSH key of each leased node. There are many ways that a malicious resource provider can subvert or spy on its guests. However, our implementation is not inherently less secure than grid and cloud sites today.

8.3.7 Summary

The GROC coordinates container-level functions such as resource leasing and configuration and guides task routing decisions. The GROC enables the higher-level application tools to focus on job and data management and abstracts the resource-level variations through the container. The evolution of computing models such as cloud computing and the variability in these distributed resources requires a concrete representation of resources and their properties that is facilitated by the container representation. In Section 8.4.1 we discuss the resource properties associated with the container model that represent its QoS properties.

8.4 Probabilistic QoS Model

Resource providers often implement policies that balance different customer needs that result in variability of guarantees. For example, in a leasing or cloud system, a broker may oversubscribe resource requests to minimize idle time. Similarly, in batch systems services such as QBETS [125] and VARQ [126] provide probabilistic bounds on when a job or a reservation will start. Thus resource contracts are often not designed to be "strong". In addition, hardware and software services have failure characteristics that the user needs to know about in advance of resource procurement. Thus we propose a probabilistic QoS model in which resource providers can make a promise within a certain guarantee - e.g., a resource provider can say that there is a 95% chance that a user's resource request can be met and a 99% chance the resource will stay up during the allotted time.

Our probabilistic QoS model provides a "least-common denominator model" that abstracts out the differences from the resource models, i.e. grid and clouds. The abstraction captures the common key elements across the systems and allows the higher-level mechanisms (e.g. scheduling) to work with different resource models and underlying policies. This is analogous to the IP hourglass model used in computer networks, where irrespective of the specific protocols in the application layer or transport layer, the only protocol used for passing data packets is the IP.

The core of the probabilistic QoS model is the *slot* abstraction. The *slot* abstraction is the center of the interaction model between the two layers. The concept of decoupled resource selection and scheduling [205] and the slot abstraction [83, 163] has been discussed earlier. A *slot* is a term used to refer to resource units assigned by a resource provider for a specified duration to the consumer which have a set of properties.

Our QoS abstraction allows the higher-level stack to abstract out existing site mechanisms and

policies including underlying scheduling systems and plan for the uncertainty. This does not affect the resource procurement and execution system interaction with underlying systems. Other systems like the virtual grid execution system [90] provide interfaces to abstract the differences in execution system. Thus probabilistic guarantees help resource providers maintain QoS while anticipating unexpected load, utilization and other runtime factors.

We discuss probabilistic resource procurement in Section 8.4.1. We define the properties required on the slot abstraction for our QoS model in Section 8.4.2 . In Section 8.4.3 we discuss the cost models associated with the slots under different resource systems.

8.4.1 Probabilistic Resource Procurement

Explicit resource control is possible in today's batch systems through offline or online advanced reservations that allow users to specify a fixed start time at higher costs. Thus advanced reservations yield resource slots that have guaranteed start and end times and probability of procurement very close to one. We use VARQ (Virtual Advanced Reservations for Queues) [126] based resource slots to determine if effective workflow orchestration is possible without explicit resource control in batch systems. A virtual advanced reservations obtained through VARQ is an instance of the resource slot abstraction with probabilistic bounds on obtaining a slot of certain duration by a given time. In overbooked leasing systems we can calculate an equivalent probability using the number of resource lease requests that can be overbooked.

8.4.2 Resource Properties

We define the following properties on the slot abstraction:

Estimated start and end time. Each resource slot has a projected start and end time during which

the resource will be available. For a batch system, the start and end times are bound by queue wait times. Leases and advanced reservations have fixed start and end-times.

Width. Each slot has a width that denotes the number of computational units (e.g., processors) that will be available for a given resource request.

Projected cost. Each slot has a cost for resource usage over the estimated slot duration. Cost models for each of the resource models are discussed in detail in Section 8.4.3.

Probability of meeting the resource request. We define a probability value that is analogous to the chance that a resource request is successfully fulfilled. Statistical methods have be used to predict stretch factors and probabilities of meeting resource requests using historical traces [125, 126]. The probability of meeting a resource request when using an advanced reservation mechanism or leases is 1. We use job wait time predictions for standard batch queue systems as a means of predicting probability of resource arrival. For virtual advanced reservations, there is a probability associated with the ability to meet the reservation request. In overbooked leasing systems we can calculate an equivalent probability using the number of resource lease requests that can be overbooked.

Probability of resource failures during the estimated duration. Most systems undergo hardware and software failures during their lifetime. The probability of the machine failing during a given duration can typically be calculated using uptimes and downtimes of the nodes and constituent services. Monitoring services on both grid (e.g., NWS [201], INCA [82]) and cloud systems (e.g., CloudStatus [37]) collect some failure information, but these are often not directly accessible to the end user. The properties associated with the slot abstraction enable this information to be easily accessible by the user.

Start early and end extension time flexibilities. In addition to knowing the estimates on start and end time the user is interested in knowing if the resource allocation may start early or if the end

time is extensible. In a traditional batch queue system, a wall clock duration is specified during job submission. In this case the resource cannot overrun the duration but may finish early without a penalty. In the virtual advanced reservation while the end time cannot be extended, the resource can become available earlier. In a leasing system, start times are typically fixed while there may be an option to extend the leases. In the context we consider, late starts and early ends are handled as exceptions since these occur as failures in resource acquisition guarantees or from failures of resources etc. In addition these situations do not incur additional costs for the user.

8.4.3 Resource Cost models

An important decision factor when it comes to selecting resources is the resource cost. HPC systems like the TeraGrid use service units as basic units of TeraGrid time. Services like Amazon EC2 similarly charge per instance-hour. In this section, we discuss the cost models associated with batch and lease systems:

Batch model. In a typical batch model, users pay for only the execution time of the task on the resource. Thus, the cost for an application per processor on a batch system is

$$Cost_{batch} = tb_i * cb_i$$

where tb_i is the time to execute and cb_i is unit cost on resource i.

Capabilities like out-of-band or online advanced reservations come at additional costs which are a function of execution time or a fixed overhead. For e.g., on the SDSC TeraGrid, advanced reservations have a premium that ranges from 1.2 to 2 times the actual cost of the resource. Thus the cost of an offline advanced reservation per processor is given by

$$Cost_{advres} = ts_i * cs_i * premium$$

where ts_i is the time requested and cs_i is the unit cost on resource i. In the virtual reservations mode, the users pay for any additional time that the resource is procured in advance of the actual task execution. Thus the cost of a virtual advanced reservation per processor is

$$Cost_{virtual} = ts_i * cs_i + (idletime)$$

Leases. In a leasing system, a user will pay for a lease that is acquired which includes the running time and wasted time on the resource. The cost of a lease per processor is

$$Cost_{leases} = tl_i * cl_i$$

where tl_i is the time of the lease (in closest hours for EC2) and cl_i is the cost of the resource.

If the same lease is used by multiple jobs or different workloads the wasted cost is amortized over the workflows. For EC2 systems the cost varies from $0.10 to $0.80 based on the size of the instance.

The cost models inherently represent the differences in the overheads in the systems. Our QoS model represents the cost-rate that can be used by different systems to gauge relative costs when considering these systems for workloads.

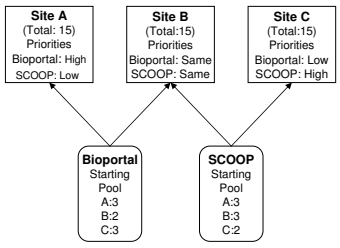


Figure 8.3: GROC Testbed. The testbed has three cluster sites with a maximum capacity of 15 virtual machines each. There are two hosted grids (the Bioportal and SCOOP applications). Each site assigns a priority for local resources to each grid, according to its local policies.

8.5 Evaluation

We have proposed a hosting model and a probabilistic QoS model to facilitate interaction of the higher-level software stack with the underlying resources. In this section, we evaluate implementation that demonstrate these ideas.

We present an experimental evaluation of the container hosting model prototype built atop the Globus and Shirako toolkits to demonstrate its feasibility and understand the trade-offs in different policy and site setup choices in Section 8.5.1 .

Probabilistic QoS models come with some variations that can be bounded within certain limits to be feasible. Virtual advanced reservations are implemented atop batch queue systems as a mechanism to assure resource procurement within certain probabilistic bounds. Virtual advanced reservations by design come with variations in when they start and incur additional costs when they start earlier than required. We present an experimental evaluation of the effect on start-times and costs obtained through virtual reservations atop under-provisioned TeraGrid batch systems in Section 9.6.3.

8.5.1 Container Hosting Model

We conducted an experimental evaluation of the prototype to illustrate how hosted grids configure and adapt their resources to serve streams of arriving jobs. The experiments demonstrate on-demand server instantiation for hosted grids, dynamic adaptation driven by GROC policies, and the interaction of policies at the sites and grids

Application workloads. We consider here two specific grid application services: Bioportal [142], a web-based interface that allows VO users to submit bioinformatics jobs, and SCOOP [138], a system that predicts storm surge and local winds for hurricane events. Bioportal uses a simple policy to route user jobs to a local cluster and the TeraGrid. In its original incarnation it has no mechanism to ensure predictable service quality for its users. We selected four commonly used Bioportal applications (*blast*, *pdbsearch*, *glimmer*, *clustalw*) from the Bioportal tool suite to represent the workload.

The North Carolina SCOOP Storm Modeling system (described in Chapter 2) is an event-based system that triggers a series of Advanced Circulation (ADCIRC) runs on arrival of wind data. Executions are triggered periodically during the hurricane season based on warnings issued by the NOAA National Hurricane Center (NHC). One interesting aspect of SCOOP is its ability to forecast its demand since the hurricane warnings are issued every six hours during storm events. In the original version, a simple resource selection interface schedules the runs when each warning arrives; although SCOOP knows when runs will be issued, it cannot ensure that sufficient resources will be available to complete the models in a timely manner.

Policy. The experiments use GROC policies appropriate for each workload. Bioportal uses an *on-demand* policy that maintains a target upper bound on waiting time. The total number of nodes to

request at each decision point is given by:

$$BioportalRequest(t) = \max\left\{\frac{(WaitingJobs(t) - FreeCPUs(t))}{WaitingFactor * Resources(t)}, 0\right\}$$

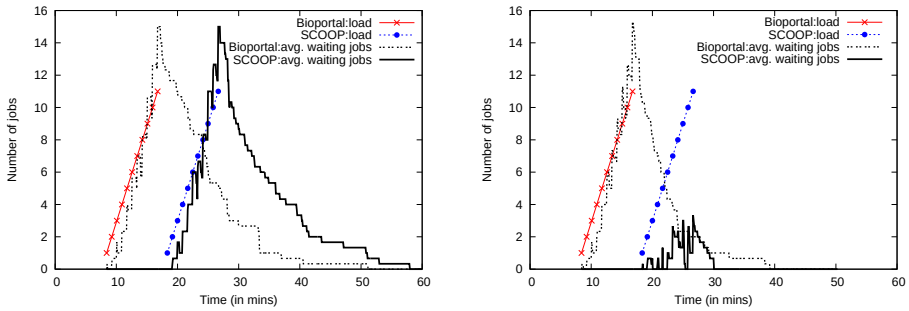
where $WaitingJobs(t)$ are the total number of jobs in the queue at and $FreeCPUs(t)$ are the number of CPUs available and $Resources(t)$ are the total number of resources at time t. Our experiments use $WaitingFactor = 2$.

SCOOP's GROC uses a *look-ahead* policy to reserve resources in advance of expected demand. It considers the current backlog and expected arrivals over a sliding time window. The total number of new nodes to request is given by:

$$SCOOPRequest(t) = \max\left\{\left((WaitingJobs(t) - FreeCPUs(t)) + \sum_{i=t}^{t+\Delta t} ExpectedJobs_i\right), 0\right\}$$

Experimental setup. All experiments run on a testbed of IBM x335 rackmount servers, each with a single 2.8Ghz Intel Xeon processor and 1GB of memory. Some servers run Xen's virtual machine monitor version 3.0.2-2. All experiments run using Sun's Java Virtual Machine (JVM) version 1.5. COD uses OpenLDAP version 2.2.23-8, ISC's DHCP version 3.0.1rc11, and TFTP version 0.40-4.1 to drive network boots.

We partition the cluster into three sites (Figure 8.3). Each site consists of a COD server that configures and monitors allocated machines, a broker server that implements the site's policy for



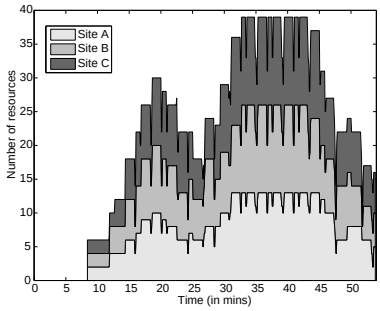
(a) Without advance reservations (b) With advance reservations

Figure 8.4: Effect of reservations. Figure shows the average number of waiting jobs across three sites. In (b), the SCOOP grid reserves servers in advance to satisfy its predicted demand.

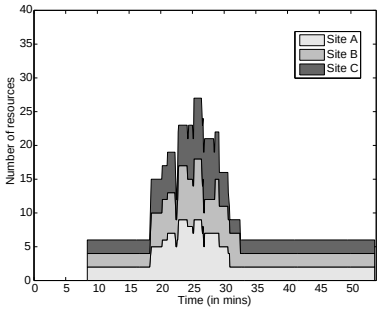
allocating its resources to competing consumers, and five physical machines. The sites divide the resources of each physical machine across three virtual machines, giving a total resource pool of 45 machines for our experiment. Previous work [83] has shown that the leasing and configuration mechanisms scale to much larger clusters. The sites in these experiments use a simple priority-based arbitration policy with priorities as shown in Figure 8.3. All leases have a fixed preconfigured lease term.

Reservations and priority. This experiment illustrates how GROCs procure resources to serve growing load, and illustrates the mechanisms and their behavior. We consider two synthetic load signals that have a linearly increasing number of jobs arriving over a short interval. The duration of the load is 50 minutes and a worker node lease term is 4 minutes.

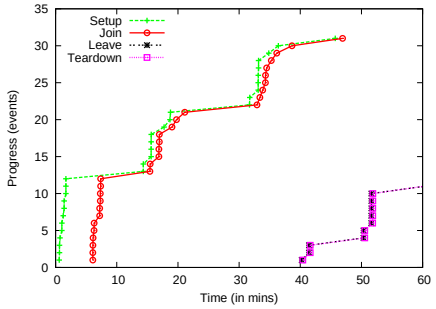
Figure 8.4 shows the average number of waiting jobs across the three sites (a) without and (b) with advance reservations. In both cases, the sites use priorities as shown in Figure 8.3, and Bioportal uses its simple *on-demand* resource request policy. In Figure 8.4 (a), SCOOP's look-ahead horizon is zero, so it effectively uses an on-demand request policy as well. In Figure 8.4 (b), SCOOP reserves resources in advance of its anticipated need, significantly reducing its job delays and queue



(a) Bioportal resource holding at each site



(b) SCOOP resource holding at each site



(c) Progress of server configuration events.

Figure 8.5: Resource Holdings and Priority. Site resources are allocated to competing GROCs according to their configured priorities. (a) shows the decrease in resources available to Bioportal as more machines are reserved to SCOOP, as shown in (b). Bioportal reacquires the machines as SCOOP releases them. (c) shows the progress of resource configuration events on sites and GROCs.

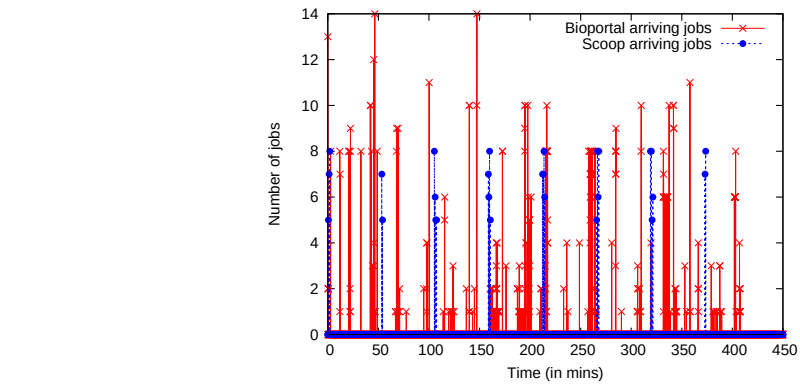
lengths.

Figures 8.5 (a) and (b) show the distribution of resources among the two GROCs, illustrating the impact of site policy. This experiment is slightly different in that the Bioportal load submits jobs at a constant rate after it reaches its peak, producing a backlog in its queues. As more computation is allocated to serve the SCOOP burst, Bioportal's worker pool shrinks. The impact is greatest on Site C where Bioportal has lower priority. As SCOOP's load decreases, Bioportal procures more resources eventually reduces its backlog.

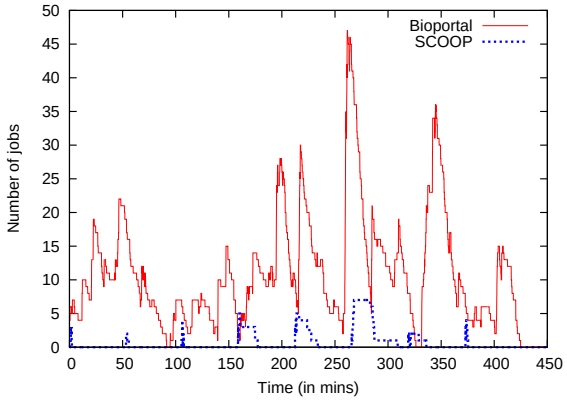
The GROCs adapt to changing demand by adding and removing worker nodes as the experiment progresses, using the mechanisms described in Section 8.3.4. Figure 8.5 (c) shows the completion times of configuration events across all three sites for an experiment similar to Figure 8.5. At the start of the experiment, each GROC leases and configures a master node at each of the three sites. These six nodes boot (*setup*) rapidly, but it takes about 336 seconds for the master *join* handler to copy the Globus distribution from a network server, and untar, build, install, and initialize it. As jobs arrive, the GROC also leases a group of six worker nodes. Once the master nodes are up, the workers *join* rapidly and begin executing jobs; as load continues to build, both GROCs issue more lease requests to grow their capacity. After each worker boots, it takes the GROC's worker *join* handler about 70 seconds to initialize the node with a private copy of Torque, and register it with its Torque master at the site. The GROCs permit some leases to expire as the queues clear; the *leave* (deregister) and *teardown* handlers complete rapidly. In this experiment, the Bioportal takes a while to clear its queued jobs, so the remainder of the *leaves* and *teardowns* occur later in the experiment.

Adaptive provisioning with varying load. This experiment demonstrates adaptive resource provisioning by competing grids under a more realistic load signal. The Bioportal workload consists of a steady flow of jobs, with occasional spikes in job arrivals. The job arrival times were obtained from traces of a production compute cluster at Duke University. We scaled the load signals to a common basis that is appropriate for the size of our resource pools. The SCOOP workload runs a small set of ADCIRC jobs periodically according to a regular schedule. In practice, the resource demand for the runs in each period may vary according to weather conditions or post-processing results. For this experiment we use a synthetic load generator to create load spikes lasting a small time period (approximately 1 minute), at intervals of approximately 50 minutes. The duration of this experiment is 420 minutes and the lease length of each worker node is set to 25 minutes.

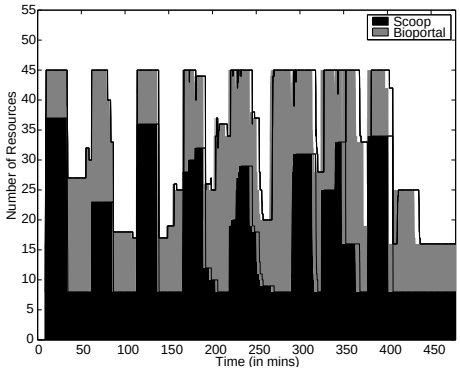
Figure 8.6(a) shows the load signal, (b) the waiting jobs queued at Site A, and (c) the resources



(a) Load signal



(b) Waiting jobs at Site A

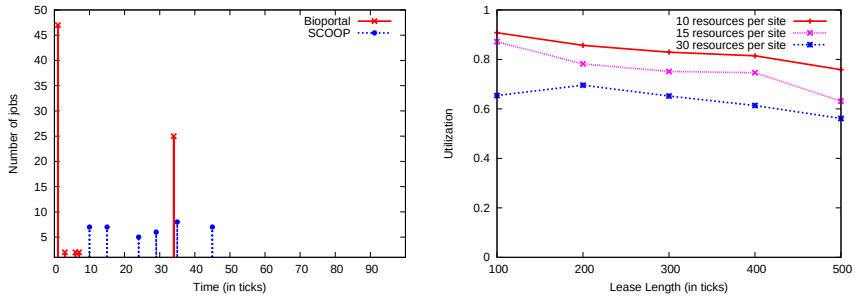


(c) Resource holdings

Figure 8.6: Adaptive Provisioning under Varying Load. The load signal (a) gives job arrivals. (b) shows the waiting jobs queue at Site A, while (c) shows a stacked plot of the resource holdings of each grid across the three sites.

that each GROC holds across the three sites. We see that each GROC is able to procure resources according to its varying load. SCOOP periodically demands resources to complete its runs, temporarily reducing Bioportal's resource holdings. However, Bioportal successfully retrieves resources between SCOOP's periods of activity. For simplicity, we omit the distribution of waiting jobs at Site B and Site C, which are similar to Site A.

Resource efficiency and lease length. The last experiment compares container-level control with job-level control with respect to efficiency and fairness of resource assignments to two competing

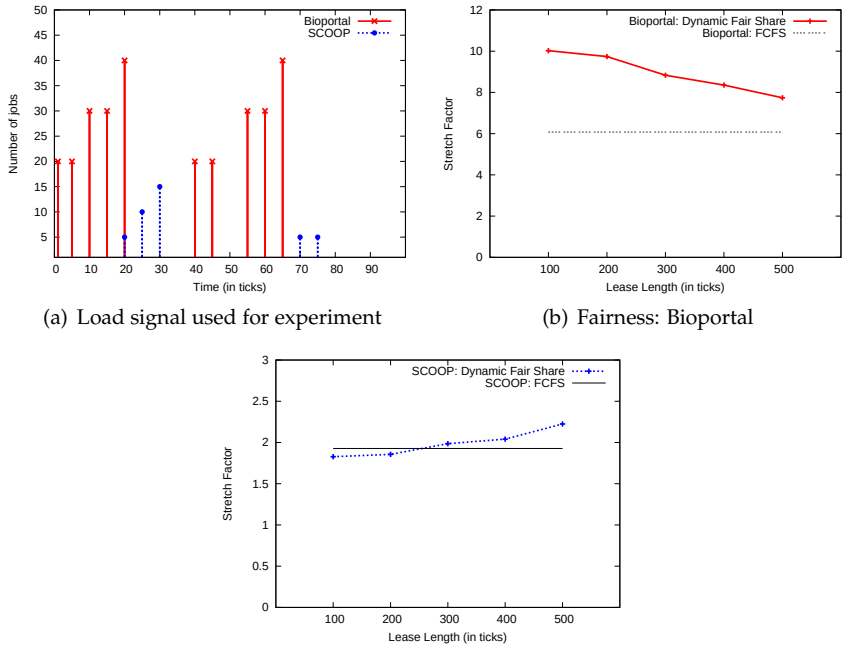


(a) Load signal used for experiment (b) Utilization with varying lease lengths

Figure 8.7: System Efficiency. (a) shows the load signal and (b) the variation of efficiency with lease length across multiple cluster sizes.

VO grids. The power and generality of container-level resource control comes at a cost: it schedules resources at a coarser grain, and may yield schedules that are less efficient and/or less fair. In particular, a container holds any resources assigned to it even if they are idle—in our case, for the duration of its lease. Another container with work to do may be forced to wait for its competitor's leases to expire. Our purpose is to demonstrate and quantify this effect for illustrative scenarios.

In this experiment, the job-level control is a standard First Come First Served (FCFS) shared batch scheduler at each site. The container-level policy is Dynamic Fair Share assignment of nodes to containers: the GROCs request resources on demand and have equal priority at all sites. Node configuration and job execution are emulated for speed and flexibility. We implement a grid emulator as a web service that emulates the Globus GRAM and MDS interfaces (job submission and status query) and also exports an interface to instantiate grid sites and add or remove worker nodes from a site. An external virtual clock drives the emulation. The site emulation incorporates a Maui scheduler with a modified resource manager module to emulate the job execution on worker nodes. Note that the core components (GROC, Shirako/COD, Maui) are identical to a real deployment. One difference is that the emulation preempts and requeues any job running on an expired worker node, although the batch scheduler configured in our prototype (Torque) does not support



(a) Load signal used for experiment

(b) Fairness: Bioportal

(c) Fairness: SCOOP

Figure 8.8: Stretch Factor. We study stretch factor as a measure of fairness, of two competing GROCs - Bioportal and SCOOP with varying lease lengths

preemption.

Figure 8.7 (b) shows the *utilization* of container-level control with different lease lengths using a bursty load signal derived from a real workload trace (Figure 8.7 (a)) across different cluster sizes. We measure utilization as how effectively GROCs use their allocated resources: one minus the percentage of unused computational cycles. As lease length increases, container-level utilization decreases because the system is less agile and it takes longer for resources to switch GROCs. The decline is not necessarily monotonic: if the job and lease lengths are such that jobs complete just before the lease expires, then the Dynamic Fair Sharing container policy will redeploy the servers, maintaining high utilization. However, an advantage of longer leases is that they can reducing "thrashing" of resources among containers; in this emulation we treat the context switch cost as

negligible, although it may be significant in practice due to initialization costs. Also, at smaller cluster sizes, resources become constrained, causing utilization to increase.

To compare job-level and container-level control, we also measure the efficiency of the resource pools. We define *efficiency* as one minus the percentage of usable resources that are wasted. A server is "wasted" when it sits idle while there is a job at the same site which could run on it. By this measure, the efficiency of a site-wide batch scheduler using FCFS is 100%, since it will always run the next job rather than leave a server idle. In contrast, a local batch scheduler running within a container may hold servers idle, even while another task scheduler in a different container has jobs waiting to run. For the given workload, in a resource constrained case (10 resources per site), the average efficiency across sites is 92% and in the overprovisioned case (30 resources per site) the average efficiency is 78%. As with utilization, efficiency is higher on smaller clusters since the GROCs are more constrained and may make better use of their resources. Efficiency is lower on larger clusters—but of course efficiency is less important when resources are overprovisioned.

Fairness is a closely related issue. One measure of fair resource allocation is the relative stretch factor of the jobs executed at a given provider site. Stretch factor is the ratio of completion time to job duration. That is, we might view a site as "fair" if a job incurs equivalent waiting time regardless of which grid submitted the job to the site. (Of course, the benefits of container-level resource control include support for differentiated service and performance isolation, which are "unfair" by this definition.) Both the FCFS job policy and the Dynamic Fair Share container policy strive to be "fair" in that they do not afford preferential treatment. Even so, these simple policies allow one of the GROCs to grab an unfair share of resources if a burst of work arrives while another is idle.

Figure 8.8 shows the average stretch factors for two job streams (Bioportal and SCOOP) running under both job-level and container-level resource control. Bioportal submits an initial burst of short

jobs, which fill the global FCFS queue (for job-level control) or trigger lease requests for a block of servers (for container-level resource control). A subsequent burst of longer SCOOP jobs must wait for servers to become available. These bursts are followed by another pair of bursts of Bioportal and SCOOP jobs as shown in Figure 8.8 (a).

The Bioportal (Figure 8.8 (b)) shows a higher stretch factor than SCOOP (Figure 8.8 (c)) in all cases. In this particular scenario, the SCOOP bursts submit longer jobs to the queue, increasing the waiting time for the subsequent burst of Bioportal jobs. However, resource leasing can allow either workload to hold its resources longer so that some are still available for the next burst. In this case, longer leases improve the stretch factor for Bioportal and increase the stretch factor for SCOOP, improving fairness of the overall system.

In general, efficiency and fairness properties result from the interaction of the policy choices and the workload; it is less significant whether resource control is implemented at the job level or container level. A rich range of policies could be implemented at either level. The advantage of container-level control is that its policies generalize easily to any middleware environment hosted within the containers. On the other hand, the granularity of that control must be coarser to avoid sacrificing efficiency and utilization.

8.5.2 Probabilistic Advanced Reservations

Next, we evaluate our probabilistic advanced reservations atop grid systems. For this experiment, we use resource data from two TeraGrid machines (tagged as *ncsatg* and *abe* in the graphs) located at the National Center for Supercomputing Applications. We obtain resources acquisition probabilities through QBETS and VARQ services. We setup experiments on *ncsatg* and *abe* to obtain probabilistic resource slots using the VARQ service. The experiments request 90 minute, 16 node slots (the approximate time required for a single LEAD workflow) one, two, three and four hours

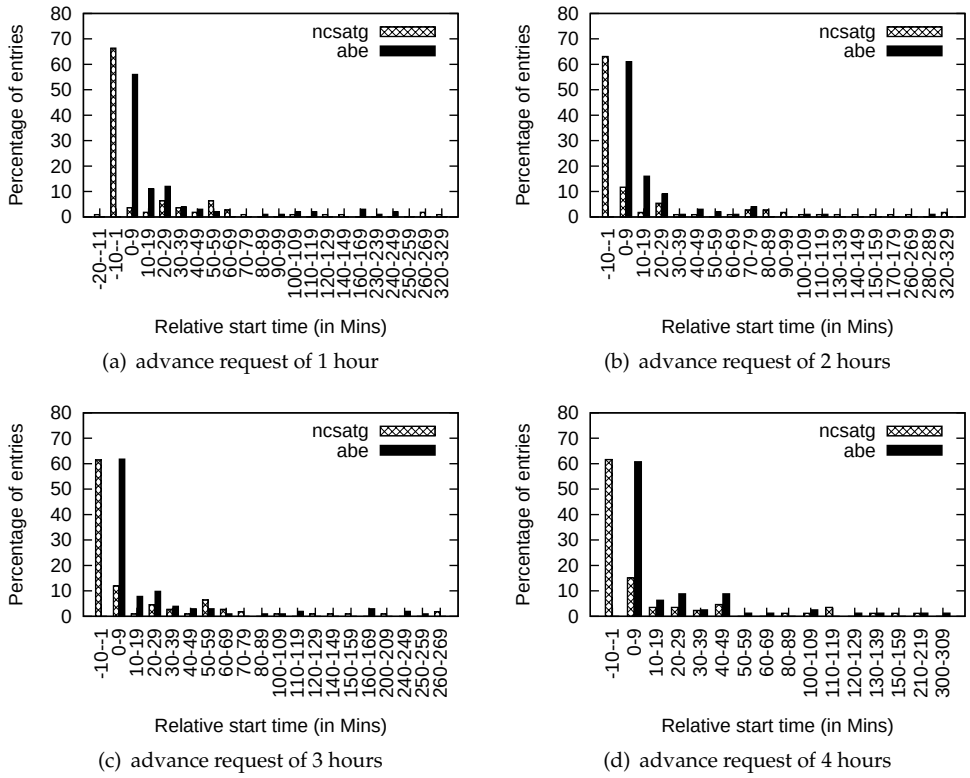


Figure 8.9: Start Times of Probabilistic Advanced Reservations. Probabilistic reservations have variable start times. We show the historgram of difference in actual start times from expected start times on two resources for requests made (a) 1 hour (b) 2 hours (c) 3 hours (d) 4 hours in advance. NOTE: Only intervals with entries have been shown in this graph.

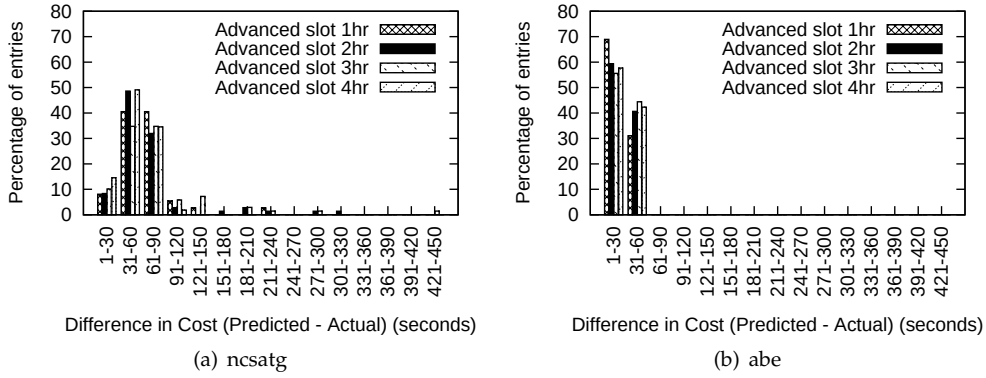


Figure 8.10: Costs of Probabilistic Advance Reservations. Probabilistic reservations incur additional costs if and when they start before expected start time. Here we show the cost variations between the predicted and actual cost over a set of requests on two TeraGrid resources (a) ncsatg (b) abe.

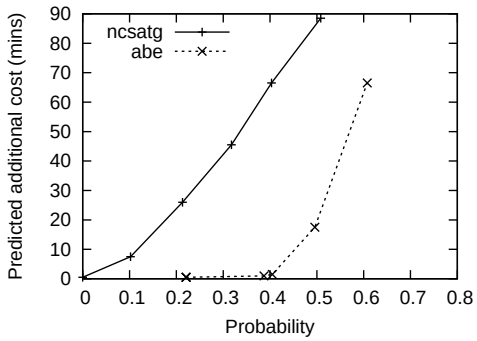


Figure 8.11: Effect on Cost for Different Guarantees. Higher levels of guarantees i.e., higher success probabilities result in greater costs.

in advance, with success probabilities ranging from 0.1 to 0.99.

Start time. When considering probabilistic advanced reservations, the slots can arrive exactly at, before or after the expected start time. Figure 8.9 shows the start time variation for one, two, three four hour advance requests over a period of four weeks. The majority of the experiments start in the [-10,10] minute range around the expected start time.

Cost. If the slot arrives on time or later, there is no extra cost since the job is ready to run. However if the slot arrives earlier, the idle time is the extra cost. In all our experiments the cost incurred is always equal to or lower than that predicted by VARQ. Figure 8.10 shows the distribution of the difference between the predicted and actual costs for all advanced requests on (a) *ncsatg* and (b) *abe*. The largest percentage of runs have a prediction that is higher by 31 to 90 seconds on ncsatg and by 1 to 60 seconds on abe. Finally, Figure 8.11 shows the variation in predicted cost with probability values. The predicted cost increases as the desired probability value increases.

8.6 Summary

The increasing separation between resource providers and consumers makes resource control in today's grid and cloud system both more important and more difficult. This chapter illustrates the dynamic assignment of shared pools of computing resources to hosted environments. It demonstrates the role of the resource coordinator in managing a dynamic binding of resources across different sites driven by workload requirements. Our approach addresses resource control at the container level, independently of the application middleware that runs within the container. The implementation of resource control at the container-level becomes more critical especially in cloud environments.

We also presented a lowest-common-denominator probabilistic QoS model that abstracts the differences in the different systems and lets the application middleware (e.g. workflow tools) concentrate on higher-level mechanisms required to manage user requirements and constraints in variable and competitive resource environments. Our experimental evaluation atop over-provisioned grid systems demonstrates the feasibility of this QoS model.

9

Workflow Orchestration

Distributed resources are increasingly used for time-sensitive workflows such as weather forecasting, storm-surge modeling, etc. These applications typically require a higher-level of QoS that has been hard to guarantee in current day production environments. There is a similar trend in industry where business services are relying on cloud computing to manage its peak workload capacity. These two parallel trends require us to revisit how higher-level tools that coordinate resources and user requirements and their interaction with new resource models.

We investigate these issues in the context of workflow tools that are increasingly used in cyberinfrastructure environments to coordinate data and computational tasks in this chapter. Today's workflow planning techniques can provide a "yes" or "no" answer to the question of whether a workflow will meet its deadline [20, 127]. However this information alone is often insufficient for deadline-driven applications such as weather prediction, where users are willing to run the workflow so long as the odds of completion are "reasonable". Users are often willing to pay extra or trade-off application requirements to ensure timely workflow completion. In addition, even if applications are not deadline sensitive, experiments have uncertainties and users often use resource availability and costs as a criteria to define their experiment parameter space. Current workflow

planning approaches and their interaction with resource models are severely limiting as we move to next generation infrastructure.

We presented a resource layer abstraction that enables uniform access across different computing models and the properties associated with the the resource abstraction in turn help with higher-level decisions (Chapter 8). Thus, we investigate a holistic workflow planning approach that considers workflow characteristics and coordinates (a) resource acquisition, (b) directed acyclic graph (DAG) scheduling, and (c) scheduling enhancements that can improve the chance of workflow completion. We use the term *workflow orchestration* to collectively describe these mechanisms.

The rest of this chapter is organized as follows. We present an overview of workflow orchestration in Section 9.1. We present an approach for DAG analysis that helps understand the resource needs and characteristics of a workflow in Section 9.2. We describe resource acquisition, task mapping and schedule enhancement in greater detail in Sections 9.3, 9.4 and 9.5 Finally, we present our evaluation of our orchestration approaches (Section 9.6) .

9.1 Orchestration: An Overview

Figure 10.1 shows the various aspects of workflow orchestration. We use the term *workflow orchestration* to denote a collection of mechanisms that describe the coordination of workflows and resources to meet end user expectations while accounting for resource characteristics. Resource level activities (e.g. resource querying and acquisition) are managed by interaction with the resource coordinator whereas the workflow planner manages workflow and user level activities(e.g. understanding workflow requirements). The components interact with each other and share logical data structures that hold workflow information and resource information. The shared data structure for workflow information has DAG description as well as user constraints on time, budget,

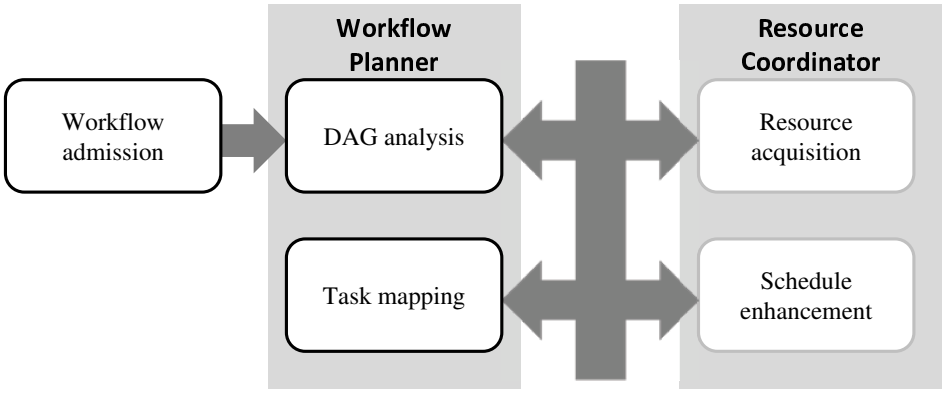


Figure 9.1: Workflow Orchestration Functional Blocks. Workflo orchestration has multiple stages for understanding workflow requirements and constraints, querying resource status and scheduling a workflow. The different functional boxes interact with and share data structures of workflow representation and the schedule.

etc. The resource information is stored in a Gantt Chart structure that has information about the "slots" at different sites and its properties. We discuss the dimensions of workflow orchestration in greater detail:

Workflow Admission. In today's environment, the user submits a workflow to an execution system that uses application performance models and resource monitoring data to make resource mapping decisions. However, before execution starts it is necessary to consider whether or not workflow constraints can be met with available resources and make priority decisions at the application layer. The user and the system have to consider the value of executing a workflow against its associated costs before execution starts. While a workflow might eventually finish, resource time may be wasted if the workflow does not complete by its deadline. In the case of critical applications users are also willing to risk some wastage of resource for increasing the odds of the workflow completing. Thus we need high-level planning and interaction techniques that allow users and service providers to jointly decide if a workflow must be run.

DAG Analysis. As complex workflows with different characteristics and constraints are run in distributed environments, we need automated mechanisms to understand workflow characteristics

that then drive planning strategies appropriately. It is critical to understand the structure and the resource requirements of the workflows.

Resource Acquisition. Resource acquisition is critical for workflow scheduling decisions. Higher-level tools require knowledge of resource quantity, type and availability times to make decisions. In batch systems resource acquisition is closely associated with the execution queue. Jobs are submitted to a queue from which jobs are mapped onto resources. However, as we move to cloud or lease based systems, resource acquisition is a separate phase that precedes planning strategies.

Task Mapping. Job or workflow scheduling strategies have been used for a long time to map tasks onto appropriate resources. Various scheduling strategies have been proposed that account for execution time, data transfer time and monitoring status of the resources while scheduling a DAG. Here we present task mapping approaches that uses (a) probabilities of task completion and, (b) performability as criterion for mapping decisions(Section 9.4). These approaches demonstrate how workflow DAG scheduling strategies can leverage resource properties to plan for not just performance but also reliability.

Schedule enhancement. Both lease and batch systems allow mechanisms to get higher guarantees on QoS through mechanisms such as advanced reservations. These facilities come at higher costs and the system needs to decide if it is worth the cost. We investigate schedule enhancements using advanced reservations in this work.

9.2 Workflow DAG Analysis

Understanding workload characteristics and predicting resource needs has been an area of active research. In today's grid systems people use performance models and historical data to predict resource requirements [112]. On batch systems, users have to specify an expected wall clock time

for their jobs in batch systems. Specifying longer than required durations can result in longer queue wait time, however specifying lower than what is required can result in the job getting killed. As we move to more dynamic resource environments with virtual advanced reservations, leasing and cloud based systems projecting resource requirements during the resource acquisition phase becomes critical and challenging.

Workflow orchestration needs to estimate and specify the set of resources and their properties during resource acquisition phase. In over-subscribed environments resource requests may not be immediately satisfied and it is necessary to iterate the resource request by changing its properties. Thus it becomes critical to do an analysis of the workload to understand its requirements. We propose a heuristic and define a set of properties that help understand the workflow structure and its computational and data requirements. Our methodology provides a strong foundation for understanding workload characteristics and estimating resource requests. We discuss the structural analysis of the workflow in Section 9.2.1 and we outline our approach to understanding the resource requirements of the workflow in Section 9.2.2. Finally, we describe heuristics for determining resource requests based on workflow characteristics in Section 9.2.3.

9.2.1 Structural Analysis

The structural elements of the workflow are considered for higher level understanding of the workflow. The structural characteristics of the workflow are useful if each of the tasks performed approximately the same quantum of work. We define the following properties that capture the characteristics of the workflow.

Maximum Task Width of Workflow. We define the property to capture the maximum number of concurrent tasks at any part of the workflow.

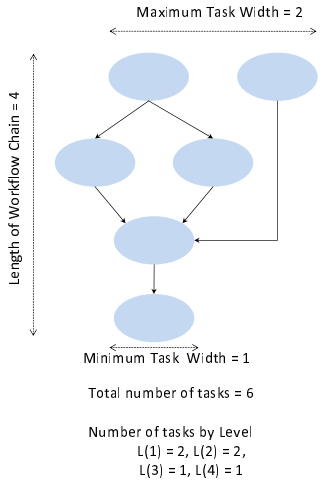


Figure 9.2: Example of Structural Analysis: A simple workflow with its associated structural properties

Minimum Task Width of Workflow. The minimum number of tasks at any level of the workflow gives the minimum width of the workflow.

Length of Workflow Chain. The number of tasks from the start to the bottom of the directed acyclic graphs along its longest path captures the length of the workflow chain.

Number of Tasks at Each Level. A critical element in workflow planning is understanding the possible parallelism achievable during workflow execution thus knowledge of the number of tasks at each level is useful.

Figure 9.2 shows an example workflow and the value of its associated structural properties. The workflow has six tasks and a maximum width of two and minimum width of one. The workflow's length is four. Also based on the structural analysis, there are two tasks in levels one and two, and one task each in levels three and four.

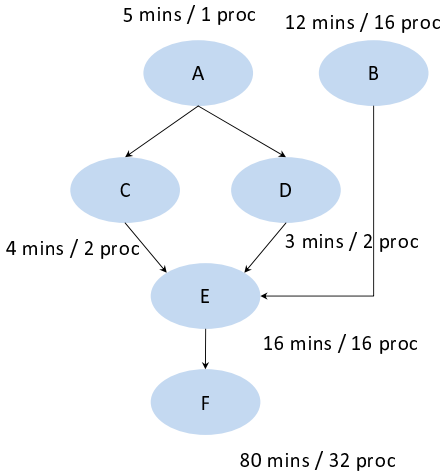


Figure 9.3: Example of Work Unit Analysis. A simple workflow annotated its work quantum characteristics

9.2.2 Work Unit Analysis

We illustrated the resource requirements of scientific workflow examples in Chapter 2. A number of scientific applications run on multiple processors and vary in their execution times. Thus it is important to consider the actual computational and data units required by the workflow in addition to its higher-level structural characteristics. We assume that a DAG description is annotated with rough estimates of the performance information. For each task in the workflow, the time (T) on N processors is specified. For example in Figure 9.3), we see that task A is expected to take five minutes on one processor. In addition the data sizes between two or more tasks in the workflow is specified. Performance models are not always accurate on Grid resources. The following analysis can be plugged in with average numbers to get a rough order estimate. Alternatively worst-case timings can be used which would give a pessimistic schedule.

We define the following properties for our work unit DAG analysis.

Single Processor Computational Units. We calculate the time units the workflow will take if it were to run on a single processor. This is useful in judging the total quantum of work performed

by this workflow.

Ideal Processor Sequential Makespan. It is often useful to understand the sequential aspect of the workflow in a multi-processor environment. Thus we calculate the turn-around time or makespan of the workflow for the case where a resource acquisition strategy is able to get the desired number of processors for each task but at a time only one task can run.

Ideal Processor Makespan. It is the turn-around time of the workflow where there are no resource constraints and resource requests are met immediately. This represents the ideal makespan of the workflow and is usually what is possible in under-subscribed resource environments.

Maximum Processor Width of Workflow. The parallelism of the workflow is an important criteria in resource planning decisions. We capture the maximum number of processors that is required in parallel for a workflow at any given time during its execution.

Minimum Processor Width of Workflow. We also capture the minimum number of processors that are required at any level of the workflow.

Task Data Ratios. For each task in the workflow, we calculate the output to input data ratios, thus enabling us to gauge the data aspects of the task.

Workflow Computational Classification. At a higher level it is often important to understand the computational distribution of the workflow. In our classification we characterize workflows on the distribution of the work load in the top, middle and bottom of the workflow. Our classification includes the following categories

- TOP_COMPUTATION_HEAVY indicates that the top one third of the workflow is computation intensive
- MIDDLE_COMPUTATION_HEAVY indicates that the middle one third of the workflow is computation intensive

- BOTTOM_COMPUTATION_HEAVY indicates that the bottom one third of the workflow is computation intensive
- TOP_MIDDLE_COMPUTATION_HEAVY indicates that the top half of the workflow is computation intensive
- MIDDLE_BOTTOM_COMPUTATION_HEAVY indicates the the bottom half of the workflow is computation intensive
- UNIFORM_COMPUTATION indicates that the workflow is roughly uniform in its computation distribution

The nature of the workflow can determine resource acquisition strategies. For e.g., a BOTTOM_COMPUTATION_HEAVY workflow might benefit from a real-time advanced reservation request.

Workflow Data Classification. We use the ratio of the total output data from the workflow to its input data to classify the workflow based on its data. Our categories for this classification currently are

- DATA_PRODUCER indicates that the outputs of the workflow are larger than the inputs to the workflow.
- DATA_REDUCER indicates that the inputs to the workflow are larger than its outputs.
- DATA_UNIFORM indicates that workflow inputs and output sizes are mostly similar.

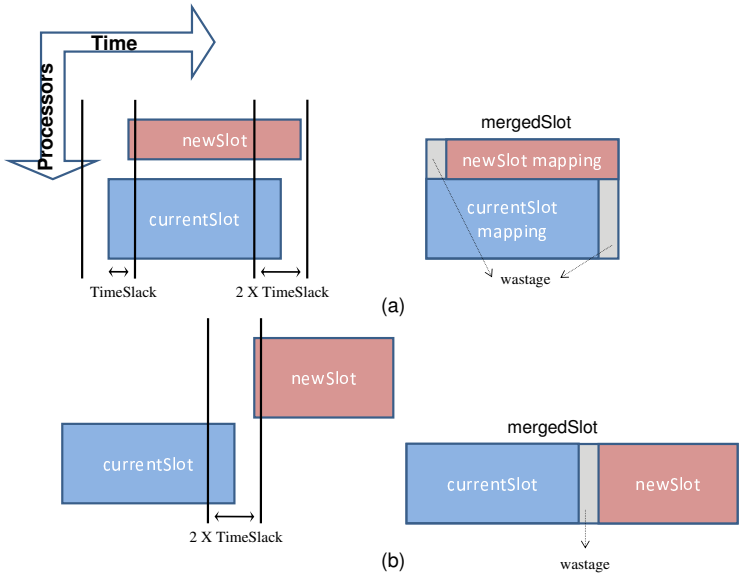


Figure 9.4: Resource Request Merging in Time. Examples that shows how slot merging is applied with timeSlack (a) *newSlot's* start and end times fall within timeSlack units of *currentSlot's* start and end times the slots are merged (b) If the start time of the new slot is *timeSlack* units within the end time of the slot and the processor width is identical, the slots are merged.

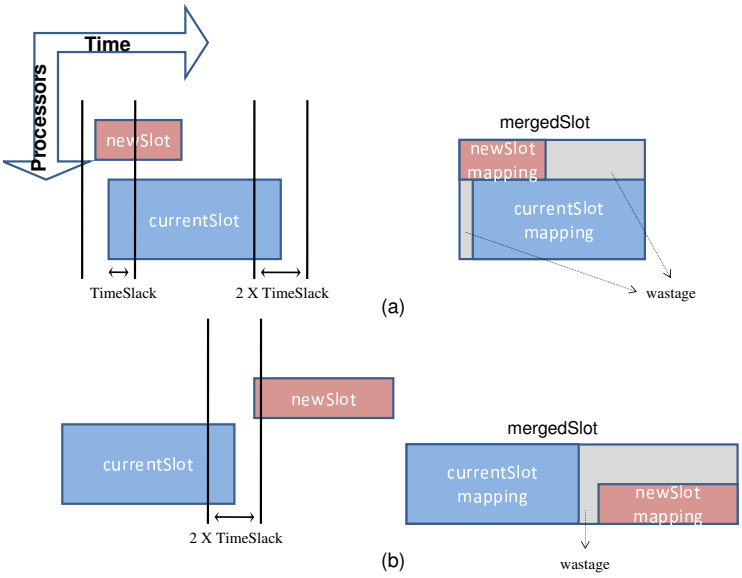


Figure 9.5: Resource Request Merging in Time and Processor Width (i.e.,*processorSlack* = true). Examples that shows how slot merging is applied with *timeSlack* and *processorSlack* (a) *newSlot's* start times falls within timeSlack units of *currentSlot's* start time or *newSlots's* end times fall within *currentSlot's* end times slots are merged (b) If the start time of the new slot is *timeSlack* units within the end time of the slot, the slots are merged.

9.2.3 Resource Requests

Resource requests often come with certain overheads for e.g., wait time in the batch queues or virtual machine startup overhead. Thus it is often beneficial to merge resource requests for multiple consecutive tasks to minimize overheads. This might incur a wastage of a resource when the resource is idle waiting for the next task. However for applications with strict timeliness requirements (e.g. LEAD) this wastage is often inconsequential. Thus we define two properties, on the time and processor width dimension of a slot that control the merging of the slots while trading wastage.

- **Time slack factor.** The *timeSlack* factor captures the variation in time units that can be tolerated from an existing slot's start and end times. For example, using this factor a new slot whose start time is within the start time $\pm$ *timeSlack* units would be considered for a merge. Similarly a new slot that has an end time within the end time $\pm$ *timeSlack* units would be considered for a merge.

- **Processor slack factor.** In addition to the time dimension it is also important to consider the width (i.e. number of processors) of the resource request. A user can specify a true or false value for the processor slack factor to control slot wastage when considering the processor based merging of slots.

The above two parameters help us in defining how resource requests must be formulated for a particular workflow. Figures 9.4 and 9.5 show examples of slot merging without (Case 1 in Algorithm 1) and with processor slack (Case 2 in Algorithm 1). When the *timeSlack* factor is considered for merges, a width-wise merge is performed only when both start and time times of the slots are within $\pm$ *timeSlack* units are merged (Figure 9.4(a)). Similarly, a length-wise merge is performed if

the start time of the *newSlot* is within $\pm$ *timeSlack* units of a *currentSlot* (Figure 9.4(b)) if the processor widths for the two slots are identical.

The *processorSlack* factor allows a more lenient merge. If either the start time of a slot or the end time of a slot is close to the end time within $\pm$ *timeSlack* units of a current slot, a merge is performed (Figure 9.5(a)). Similarly a length-wise merge is performed if the start time of the *newSlot* is within $\pm$ *timeSlack* units of a *currentSlot* (Figure 9.5(b)) irrespective of their processor widths.

Algorithm 1 describes the slot merging methodology for merging an existing slot *currentSlot* with a new slot request *newSlot*. Two slots can be merged either in width, i.e. the slot request's processors is increased to accommodate both requests, or in length where one slots end time is closer to the other's start time. In our methodology we traverse the DAG from top to bottom and comparing each new slot request for a task with already processed slot requests. We obtain the set of slots required by the workflow calculated under the conditions imposed by the slack factors described above. Individual slots from each task in the workflow are merged to come up with an optimal set that are constrained by the time slack and processor slack factors. A time slack factor of zero would be the most aggressive resource request and can result in a separate slot for each task.

Let us now consider the complexity of this heuristic. If a workflow has n tasks, our heuristic in the worst case has to do zero comparisons for the first element, one comparison for the second element, (n-1) comparisons for the n^{th} element. Thus, the worst case complexity of our algorithm is given by:

$$
\begin{aligned}
f(n) &= 0 + 1 + ... + (n-1) \\
&= \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} \\
&= \frac{n^2 - n}{2} = O(n^2)
\end{aligned}
$$

9.3 Resource Acquisition

Newer resource models have changed the way higher-level tools interact with resources. One of the fundamental differences is how and when resources are acquired. In batch queue based systems, resource acquisition is closely tied with the execution system. However with newer resource models the procurement of resources is explicit and occurs before planning strategies can be applied. This presents some new challenges to workflow orchestration including the need to predict resource requirements. Resource acquisition strategies that are possible with workflows are:

Task-based. In a task based strategy, resources are acquired *just-in-time* for each task or each job in the workflow separately. This is similar to the state of the art in workflow grid systems. In today's batch queue based grid systems, each task of the workflow is submitted as a job to the queue. In a task-based strategy the overheads from queue wait time in batch systems and machine startup-overhead in cloud or utility systems are incurred for each task.

Workflow-based. In a workflow based strategy, resources are acquired *prior to scheduling* for an entire workflow. In this case, we need mechanisms to determine appropriate resource requests. Also, gaps in the schedule result in resource wastage that is an additional cost that must be accounted for in workflow planning.

Algorithm 1 Slot merging: Merging *newSlot* with a *currentSlot*

{Case 1: TIME SLACK MERGE: when *processorSlack* is false and only *timeSlack* is specified}
{Case 1(a): WIDTH MERGE if newSlot's start and end times are close to an existing slot, it can be expanded in width}
if $(newSlot.startTime \geq (currentSlot.startTime - timeSlack))$ and
$(newSlot.startTime \leq (currentSlot.startTime + timeSlack))$ and
$(newSlot.endTime \leq (currentSlot.endTime + timeSlack))$ and
$(newSlot.endTime \geq (currentSlot.endTime - timeSlack))$ **then**
  $currentSlot.processorWidth \Leftarrow currentSlot.processorWidth + newSlot.processorWidth$
  Update start time and end time of the currentSlot to accommodate the new slot if required
  $return$
end if
{Case 1(b): LENGTH MERGE: if newSlot's start time is close to the end time and processorWidth of newSlot is less than or equal to currentSlot}
if $(newSlot.startTime \leq (currentSlot.endTime + timeSlack))$ and
$(newSlot.startTime \geq (currentSlot.endTime - timeSlack))$ **then**
  if $newSlot.processorWidth == currentSlot.processorWidth$ **then**
    Update start time and end time of the currentSlot to accommodate the new slot if required
    $return$
  end if
end if
{Case 2: PROCESSOR SLACK MERGE: even a slight overlap in time, consider merging slots}
if processorSlack **then**
  {Case 2(a): LENGTH MERGE: if newSlot's start time is close to the end time}
  if $(newSlot.startTime \leq (currentSlot.endTime + timeSlack))$ and
  $(newSlot.startTime \geq (currentSlot.endTime - timeSlack))$ **then**
    if $newSlot.processorWidth > currentSlot.processorWidth$ **then**
      $currentSlot.processorWidth \Leftarrow newSlot.processorWidth$
      Update start time and end time of the currentSlot to accommodate the new slot if required
      $return$
    else if $newSlot.processorWidth < currentSlot.processorWidth$ **then**
      Update start time and end time of the currentSlot to accommodate the new slot if required
      $return$
    end if
  end if
  {Case 2(b): WIDTH MERGE if newSlot's start or end times are close to an existing slot, it can be expanded in width}
  if $((newSlot.startTime \geq (currentSlot.startTime - timeSlack))$ and
  $(newSlot.startTime \leq (currentSlot.startTime + timeSlack)))$ or
  $((newSlot.endTime \leq (currentSlot.endTime + timeSlack))$ and
  $(newSlot.endTime \geq (currentSlot.endTime - timeSlack)))$ **then**
    $currentSlot.processorWidth \Leftarrow currentSlot.processorWidth + newSlot.processorWidth$
    Update start time and end time of the currentSlot to accommodate the new slot if required
    $return$
  end if
end if

The resource acquisition strategy affects the availability of resources accessible to higher-level workflow planning components. Both task-based and workflow-based resource acquisition have trade-offs. We evaluate the tradeoffs between a task-based and workflow-based resource acquisition in the context of our workflow orchestration implementation (Section 9.6.3). The workflow-based strategy can be expanded to include multiple workflows or the active workload that is visible to the application-level components (more details in Chapter 10).

9.4 Task Mapping

Scheduling distributed workflows on heterogeneous resources is a known NP-complete problem and a number of heuristics have been proposed [127, 188]. These heuristics focus on optimizing the makespan of the workflow using projected application running times and data transfer times. However application running times vary in real-time due to a number of factors including load and availability of machines [141]. Thus we need task mapping strategies that account for performance and reliability. For deadline-sensitive workflows it is necessary to consider the best chance of workflow completion when making resource decisions.

We explored the effect of availability variation on performance in Chapter 7. We develop deadline-driven DAG scheduling approaches focused on probability of a task finishing in Section 9.4.1. Nect, we develop DAG scheduling strategies that use the performability model to account for the performance impact due to availability variation (Section 9.4.2).

9.4.1 Probabilistic DAG Scheduler

We describe a probabilistic workflow scheduling approach that takes into account the probability of resource acquisition as well as the probability of resource failure during the allotted duration

while scheduling each task.

Algorithm 4 describes the probabilistic task mapping approach. The node priority assigning phase traverses the DAG from bottom-up and assigns deadlines for the tasks given a workflow deadline (Algorithm 2). Subsequently the tasks are sorted by deadline for the scheduling phase. Each task T has a duration d and must be scheduled no earlier than *earliestStartTime* and must finish no later than *latestFinishTime*. The only difference in the slot based system is that the algorithm tries to find a space on the slot where the task can be mapped. The difference arises from the resource model characteristics. In a batch queue system, resource requests are bound by the size of the cluster whereas when resource procurement is decoupled from the mapping, the scheduler is bound by the size of the slot already returned by the site. Subsequently all task mappings that meet the task deadline are considered for selection and the best success probability mapping is selected.

For any task in a workflow, the probability that it will succeed depends on the resource on which it is scheduled as well as the probability of its parent tasks finishing. When two tasks are scheduled on independent resource slots their probabilities are independent and the probability of a task is the joint probability of its parent and itself. However in a slot abstraction, if a Task T and its parents are scheduled on the same resource slot then the Task T has the same probability of finishing as its weakest parent. Algorithm 3 shows how the task probability of every task in a workflow is calculated according to this mechanism. Also the probability of a workflow completing is the minimum of the success probability of all tail nodes. The process is repeated for all tasks in the workflow. This heuristic finds a mapping in polynomial time ($O(n^2)$).

9.4.2 Performability based DAG Scheduler

We showed that performability modeling can be used to project the effect on performance from availability variations in Chapter 7. Specifically we showed that we can use the performability

Algorithm 2 Deadline Assignment: Calculating latest completion times for tasks in a DAG

while T in $tailnode$ **do**
  $taskdeadline \Leftarrow DAGdeadline$
  $unassignedtasks \Leftarrow parentsofT$
end while
while T in $unassignedtasks$ **do**
  if children of T have been assigned deadlines **then**
    $taskdeadline \Leftarrow$ minimumAcrossChildren$(deadlineOfChild(T) - durationOfChild(T))$
  end if
  $unassignedtasks \Leftarrow parentsofT$
end while

Algorithm 3 Task Probability: Calculating task success probability

$MinParent \Leftarrow$ Get minimum probability of all parent tasks
if $MinParent$ is on same resource slot S as task **then**
  $taskSuccessProbability \Leftarrow MinParentProbability$
else
  $taskSuccessProbability \Leftarrow MinParentProbability * TaskProbability$
end if

Algorithm 4 DAG Scheduler: Probabilistic DAG Scheduler for batch and slot systems

$Assign$ latest completion times for the tasks (using Deadline Assignment Algorithm (2))
$Sort$ the tasks by latest finish times
for all T in DAG in sorted order **do**
  $earliestStartTime \Leftarrow LatestFinishTime(Parents(T))$
  for each resource slot **do**
    if BATCH **then**
      $latestFinishTime \Leftarrow Maximum(earliestStartTime, (taskDeadline - duration))$
    else
      $latestFinishTime \Leftarrow$ find position where task will fit on slot
    end if
    if task can complete by deadline **then**
      $resourceAcqProb \Leftarrow ProbSlotAcquisition$
      $resourceUpProb \Leftarrow ProbSlotDoesNotFail$
      $taskSuccessProbabilityOnResource \Leftarrow resourceAcqProb * resourceUpProb$
      $taskSuccessProbabilityRelativeToParents \Leftarrow$ calculate task success probability considering placement of parent tasks
    end if
  end for
  $selectedResource \Leftarrow$ Resource where task has
  $Maximum(taskSuccessProbabilityRelativeToParents)$
end for

values to project running time and data transfer times for a task on machines and network with reliability variations.

We modify two commonly used DAG scheduling heuristics - Min-min and Max-min [188] to account for performability.

Min-min: In this heuristic, at each pass the minimum completion times of each task that can be scheduled given its performance model is calculated. The earliest finishing task is then selected to be scheduled next. This is repeated till all components in the workflow are mapped

Max-min: The philosophy behind this heuristic is to schedule the bigger components first. Once the minimum completion time for all possible components is calculated the latest finishing task (i.e. largest component) is then selected to be mapped first.

We modified the implementation of these algorithms to consider the projected times from the performability analysis. The calculate projected running time for the application uses application performance distribution with the resource's failure and repair rates. The algorithms using performability have been shown in Algorithm 5. The approach has the same complexity as the the version of the algorithms that consider just performance i.e. both heuristics find a mapping in polynomial time ($O(n^2)$.)

We compare workflow scheduling simulation results from the performance and performability approaches using failure data collected on production systems at Los Alamos National Laboratory [106, 158] in Section 9.6.2.

Algorithm 5 Performability DAG scheduler: The algorithm shows the modified version of the min-min and max-min heuristic that uses the performability analysis for projected timings. The modified lines from the original algorithm is shown in bold.

$readyTaskList \Leftarrow$ Determine components that are ready to be scheduled based on dependencies
while T in $readyTaskList$ **do**
  for all R in $resourceList$ **do**
    $projectedRunningTime \Leftarrow \frac{1}{Performability_{Computation}(T,R)}$ {from equation 7.11}
    $dataTransferTime \Leftarrow \frac{1}{Performability_{Network}(T,R)}$ {from equation 7.12}
    $EstimatedCompletionTime(T) \Leftarrow MaxParentFinishTime(T) + projectedRunningTime + dataTransferTime$
  end for
  For min-min: Find minimum completion time of task T over all resources R and finalize mapping
  For max-min: Find maximum completion time of task T over all resources R and finalize mapping
  $readyTaskList \Leftarrow$ Determine components that are ready to be scheduled based on dependencies
end while

9.4.3 Hybrid DAG Scheduler

Finally, we developed a hybrid version of the deadline-sensitive DAG scheduler that uses the performability metrics. We also modified the DAG scheduler to consider fault-tolerance strategies (Section 7.2.3) in the mapping phase. The DAG scheduler is similar to the probabilistic DAG scheduler and first assigns deadlines to the tasks in the DAG based on the deadline specified for the workflow. For this step it uses the performability based projected application running time that accounts for impact on running time due to availability variations that can occur during execution. In the next step, for each task in the DAG, it then uses the performability model and the state of the resource from the model (e.g. High, Good) to calculate the projected running time of the application on each of the resources. It finally considers all the resources for the task that can meet the deadline, and the task is mapped onto the resource that will incur the least cost for the application. If fault tolerance is enabled during DAG scheduling, the cost of replication and checkpoint-restart for this application is considered. If the cost of replication is lower, an additional mapping is sought for the task on the resources. If checkpoint-restart is a better strategy, the resources are checked for

availability with the additional time overheads for checkpoint-restart. If resources are available to accommodate the overheads the task mapping is updated appropriately.

9.5 Schedule Enhancement

Both lease and batch systems allow users to get higher QoS guarantees through facilities that come at higher costs, and workflow orchestration must balance costs with the need. In batch queue systems today, users often use online or out-of-band advanced reservations for guaranteed resource access. However these often require considerable lead time and the costs for these are often prohibitive limiting its use only when there is a predictable load anticipated. Cloud systems today work on a model where you are guaranteed immediate access to N resources that are agreed upon out-of-band and thus do not have explicit notion of promise of resources in the future. However as mentioned previously, when cloud systems are over-subscribed, a similar notion can be expected. For workflows, schedule enhancement can be implemented at a task-level or workflow-level or a hybrid approach which we refer to as boundary reservations.

Task-based. In task-based advanced reservations, we query the resource coordinator on a task-by-task basis for the entire workflow to see if the workflow success probability can be enhanced by using an advanced reservation for a task. In our implementation we use the schedule from the probabilistic DAG scheduler sorted by the task success probabilities as a guidance mechanism to enhance the "weak" links in the workflow. While, for time-sensitive applications cost is not a consideration, this mechanism also allows enhancements to the approach that controls acquisition by priority of the tasks and the cost. For each task we query the resource for an advanced reservation and sort the results by the highest probability followed by minimum cost. We pick the best result, if the probability of that task completing is higher than what is obtained through the probabilistic

batch scheduler. With this mechanism, the cost of obtaining the advanced reservation is incurred on each task of the workflow.

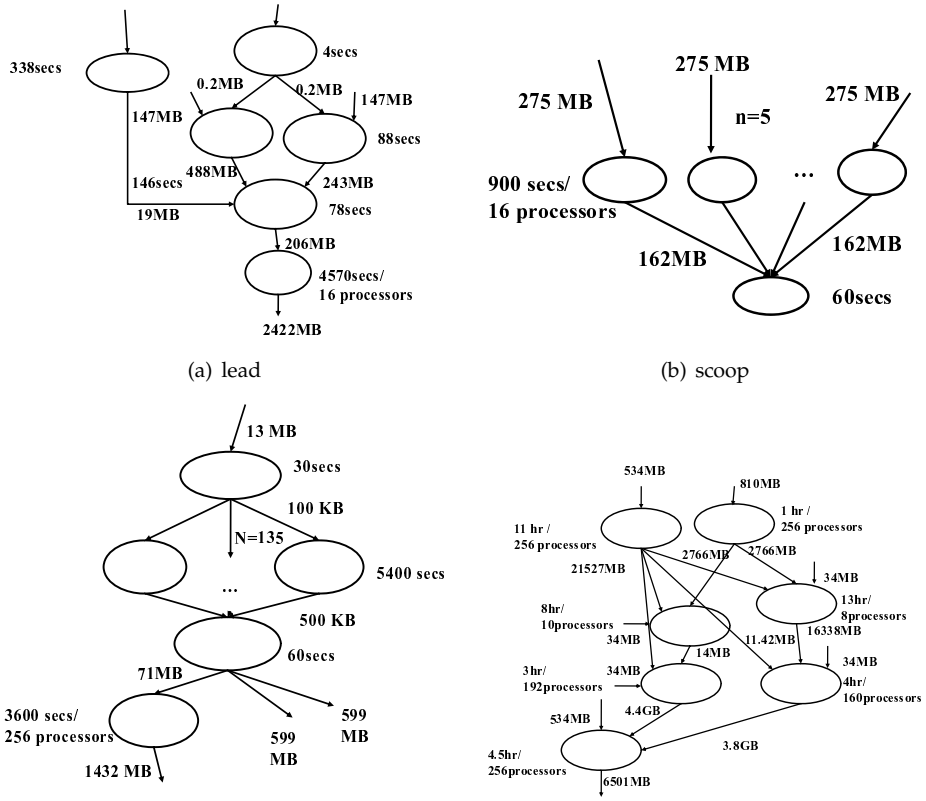
Boundary-based. Workflow components may be scheduled on multiple sites, hence we investigate an intermediate approach of procuring a single advanced reservation on each site for parts of the workflow that are scheduled on it. In this mechanism we consider the earliest task and latest task scheduled on the resource to define the time boundary and the maximum width of any task on the resource as the slot width for the request.

Workflow-based. In workflow-based advanced reservations, the resource is requested for the entire slot during the resource acquisition phase. In this mechanism, the overhead (i.e., wait time on batch systems and machine startup overhead on cloud systems) of slot acquisition is incurred only once for the entire workflow. The cost for this approach is the entire cost of the slot that is used. The wastage in the slot comes from idle time on the slots and earlier slot arrival.

9.6 Evaluation

We have presented a number of workflow orchestration strategies in this chapter. In this section, we evaluate our orchestration strategies in the context of select workflow examples from Chapter 2. The workflow examples used in this evaluation is shown in Figure 9.6. We model the structure of the workflows using Xbaya [161], a graphical composition tool that is used in the Linked Environments for Atmospheric Discovery(LEAD) [54] project.

We evaluate our heuristic for slot request and discuss workflow properties for our example workflows in Section 9.6.1. We evaluate our performability-based and probabilistic DAG schedulers in Sections 9.6.2 and 9.6.3.



(a) lead (b) scoop

(c) motif (d) ncfs

Figure 9.6: Scientific workflow examples.(a) a weather forecasting workflow (b) storm surge modeling workflow (c) domain analysis of biological sequences (d) flood-plain mapping workflow

Workflow	Number of Tasks	Number of Levels	Number of Tasks by level	Maximum Width	Minimum Width
lead	6	4	2,2,1,1	2	1
scoop	6	2	5,1	5	1
ncfs	7	4	2,2,2,1	2	1
motif	138	4	1,135,1,1	135	1

Table 9.1: Structural Workflow Analysis. The table shows the structural properties for some example workflows.

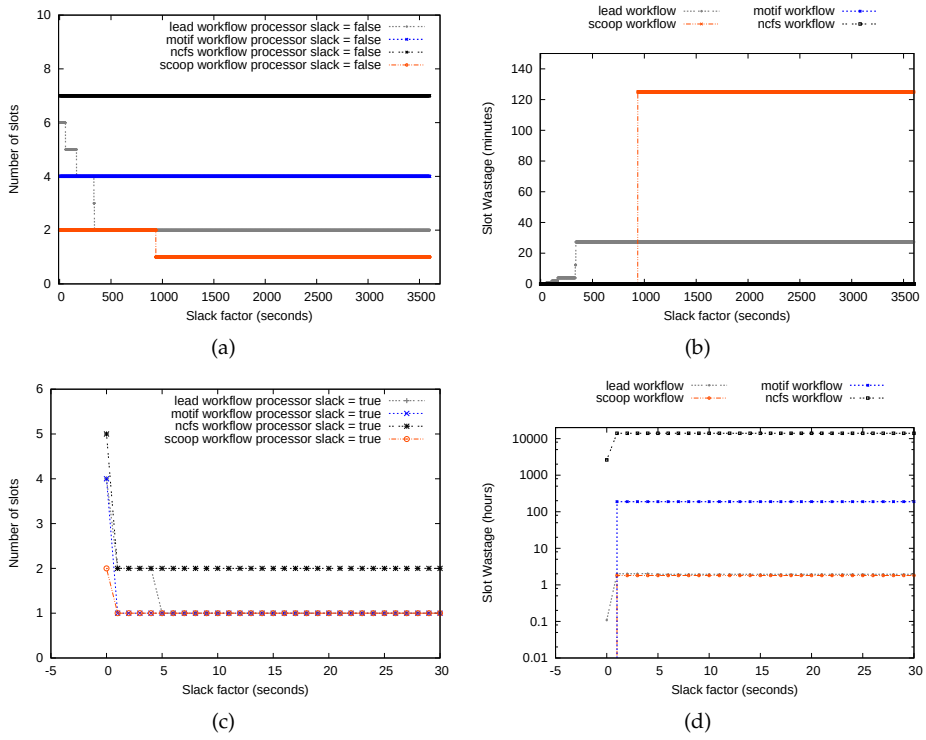


Figure 9.7: Impact of Slack Factor on Slot Requests. The graphs shows the slot analysis for the four sample workflows with varying *timeSlack* and *processorSlack* parameters (a) and (b) show the slot count and corresponding wastage as *timeSlack* varies, (c) and (d) show the slot count and corresponding wastage (in log scale) *timeSlack* varies when *processorSlack*= true.

Workflow	Total computational units	Workflow Data ratio	Computational Classification	Data Classification
lead	20.63 hours	8.0	MIDDLE_BOTTOM	DATA_PRODUCER
scoop	20.82 hours	0.36	TOP_MIDDLE	DATA_REDUCER
ncfs	5968 hours	3.0	TOP	DATA_PRODUCER
motif	459.17 hours	2.13	BOTTOM	DATA_PRODUCER

Table 9.2: Work Unit Workflow Analysis. The table shows the properties that describe that determine the work units to be performed for each of our example workflows.

9.6.1 DAG Analysis

Tables 9.1 and 9.2 show the value of the properties for the example workflows. As mentioned earlier each of the workflows varies significantly in their structure and work units. The **lead** workflow has a small number of tasks and is a data producer and the computation is heavy in the middle to bottom of the workflow. The **scoop** workflow reduces its input data and the top to middle of the workflow has the computation. The **ncfs** workflow is very computationally heavy requiring total of almost 6000 processor hours and the top of the workflow is more computationally heavy. The **motif** workflow has its computation in the bottom of the workflow and is also a data producer.

Figure 9.7 shows the effect on slot count and slot wastage for varying values of the parameters *timeSlack* and *processorSlack*. Figure 9.7(a) shows the slot count as *timeSlack* is varied. The **lead** workflow has a number of small tasks and the slot count decreases quickly with increasing *timeslack*. The **scoop** workflow's slot count decreases at about 900 seconds which is the largest part of the workflow. In Figure 9.7(c) the number of slots decreases rapidly as the *processorSlack*=true since it allows more lenient merging of slots. Figures 9.7(b) and (d) captures ths slot wastage as *timeSlack* is varied. We observe that as the slot count decreases, the wastage from the slot increases since a large part of the slot goes unused. The **ncfs** workflow does not encounter any slot wastage since a slot is requested for each task in the workflow since the tasks are large and merging of slots does not occur. Workflow orchestration strategies need to compare the effects from the overheads associated with individual slots with the wastage on the slots to determine the right slot acquisition parameters.

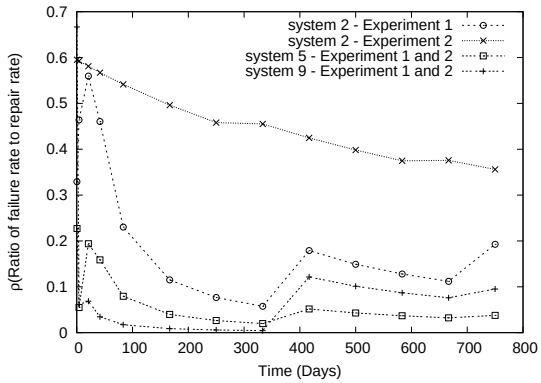


Figure 9.8: Failure Characteristics of Production Systems. Failure to repair rates over time in production use of systems at LANL.

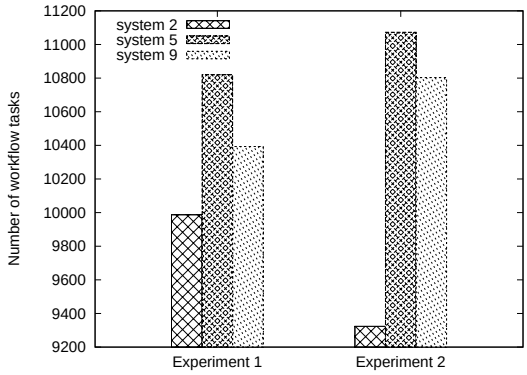


Figure 9.9: Schedule Comparision with Different Availability Levels. Comparison of workflow tasks scheduled on the resources in Experiment 1 with no prior accumulated resource history and Experiment 2 with prior accumulated history for system 2.

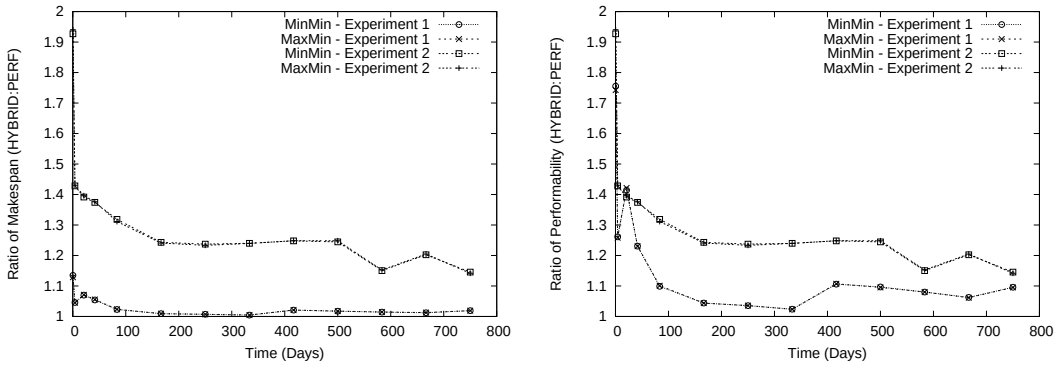


Figure 9.10: Study of Performability based Workflow Schedules On Production Systems. Ratio of a) makespan b) performability over time in production use of systems (averaged over 100 runs per data point.)

9.6.2 Performability Workflow Scheduling Simulation

Next, we present results from a set of experiments in workflow scheduling using a simulation framework. Recently, resource providers such as TeraGrid have been collecting and publishing machine (e.g., Availability Prediction Service [11], INCA Real Time Monitoring Suite [82]). However, the quantity of data is still insufficient for any extensive evaluation. As more data is collected on systems it will be possible to classify the current availability level (e.g. High, Good, etc) of the resource and plug that additional information into the model. We use the operational data made available by Los Alamos National Laboratory (LANL) that includes data collected from about 22 high performance computing systems over a period of 9 years [158]. The reliability data provides significant samples similar to current production HPC systems. The simulation framework evaluates workflow scheduling algorithms using performability and performance as metrics. Numerous studies have proposed various workflow scheduling heuristics for different grid applications [205] ; we choose two scheduling algorithms: min-min and max-min [188] described in Section 9.4.2. With each heuristic we first consider application performance time on each resource in the high state and generate a schedule for the workflow (referred to as PERF hereafter) and then apply the algorithms

shown that considers performability (referred to as HYBRID). Our approach is orthogonal to the choice of the exact algorithm and can be used with other algorithms.

Performance data Our simulation framework uses the **lead** workflow - a mesoscale meteorology workflow that is used for weather prediction. A sample directed acyclic graph is shown in Figure 9.6(a). In the simulation we generate the application run times in different reliability states from a normal distribution [98] where the mean is from real observations on TeraGrid machines. Bandwidth data required to model data transfer times is generated from long-tailed Pareto distributions [86].

Reliability data The LANL reliability data provides details on failures and repair times over the life of a number of systems. We use systems 2 (6152 cpus), 5 (512 cpus) and 9 (512 cpus) in this simulation. We calculate the failure and repair rates at time t for these machines using failure data available till time t. We take this approach since in real systems decisions will be made with data available to date. We select data points from this set of observations for our scheduling simulation and compare the effect on workflows as the systems failure and repair times change over their lifetime. Figure 9.8 shows the failure-to-repair ratio of the data points selected for the simulation for the three systems. In Experiment 1, we use the first 18000 hours of system lifetime for the simulation, i.e. all systems have no prior accumulated resource history. For experiment 2, we consider system 2 to have prior accumulated failure data history i.e., we use system 2's data from 20000 to 38000 hours.

Results. Figure 9.9 shows the total number of workflow tasks scheduled on each resource in each experiment. In Experiment 2, the corresponding failure-to-repair rates for system 2 are significantly higher than the other two resources and we see a drop in the number of components that are scheduled on this resource. Figure 9.10(a) shows the ratio of makespan from the HYBRID approach and the corresponding performance heuristic. At high failure-to-repair rates, the HYBRID

approach produces a longer schedule accounting for the failure possibilities. In Experiment 1, the HYBRID approach produced slightly longer makespans for the workflows than the PERF heuristic. Experiment 2 demonstrated slightly higher increase in the makespan where system 2 had a much higher rate of failure-to-repair ratio compared to the other two systems. This is largely because workflow components do not get scheduled on the best performance machine when reliability of that machine is lower.

In Figure 9.10(b) we compare the performability of the computational parts of the workflows. The performability for the PERF algorithm is calculated using the corresponding failure-to-repair rates of the machines on which each task was scheduled. As expected we see that performability ratio corresponds closely to the resource failure-to-repair rates. As the failure-to-repair rates increase, the performability from the HYBRID approach is significantly higher than the PERF. Thus we see that using performability as a metric can result in a better workflow schedule that accounts for the machine availability in addition to application performance. We see that the difference between min-min and max-min is minimal on the makespan and the performability. These experiments demonstrate the performability and makespan variance over machine lifetimes, emphasizing the importance of performability as a metric in workflow scheduling that improves performability with minimal effect on makespans.

9.6.3 Probabilistic Workflow Orchestration

In this section, we present experimental results that compare and contrast our probabilistic orchestration techniques in grid and cloud environments. In Section 9.6.3, we demonstrated the feasibility of probabilistic slots through trials performed on the TeraGrid. Here we use simulation to compare the effect of orchestration technique parameters when using both TeraGrid and EC2 resources through probabilistic orchestration.

Implementation. We implemented a set of workflow orchestration strategies for resource acquisition, task mapping and schedule enhancement to facilitate comparison. Our implementation consisted of the following planners:

- ***Batch Queue (BQP).*** We implemented a vanilla batch queue scheduler that used batch queue prediction data (BQP) and Availability Prediction Service (AVP) [11] data to select resources during the mapping phase selecting the resource with best probability for each task. The complexity of this algorithm is the complexity of the DAG scheduler which is $O(n^2)$.

- ***Batch Queue and Task-based advanced reservations (Task).*** We use the batch scheduler to map tasks onto resources. We sort the tasks by their success probabilities. We use VARQ to query for each task in the sorted list to see if a task-based advanced reservation enhances the success probability. VARQ queries use a heuristic to vary the parameters to find a possible resource combination that meets the users requirements. The VARQ query is for a fixed duration, width and start time and range of success probabilities. The cost incurred for this mechanism is the total execution time plus additional costs from VARQ for the slots. The complexity of this algorithm is given by the complexity of the DAG scheduler ($O(n^2)$) and the complexity of the VARQ query which is $O(n)$. Thus the complexity of this approach is given by $O(n^2 + n)$ i.e., $O(n^2)$.

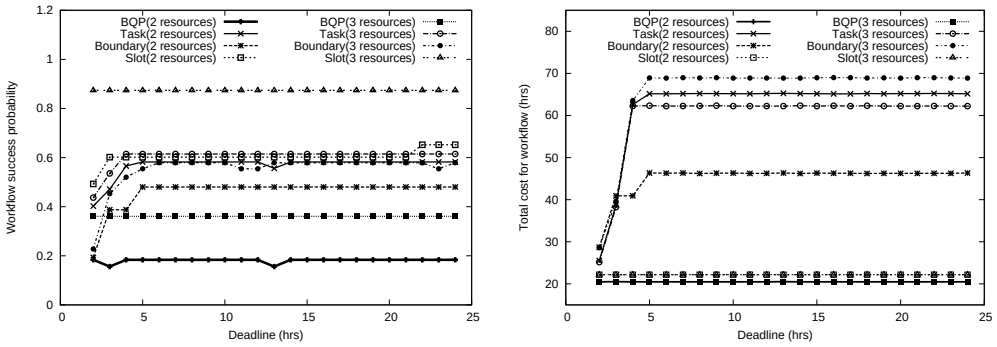
- ***Batch Queue and Boundary-based advanced reservations (Boundary).*** We use the batch scheduler to map tasks onto resources and then use VARQ to procure advanced reservations grouping all the mappings on a single resource into a single slot request. The VARQ query is for a fixed duration, width and start time and range of success probabilities. The cost incurred for this mechanism is the total execution time plus additional costs from VARQ and any slot idle time that comes from gaps in the slots. The complexity of this algorithm is given by $O(n^2 + n)$ i.e. $O(n^2)$.

- ***Batch Queue Advanced Slot and Workflow-based task mapping (Slot).*** In this approach we query the resources to get appropriate "advanced reservation" slots. We then apply the probabilistic DAG scheduler to map tasks onto these slots. For every task we make a VARQ query for a fixed duration and fixed width but for a range of start times and range of success probabilities. The cost incurred for this mechanism is the total execution time plus additional costs from VARQ and any slot idle time that come from gaps in the slots. There might be idle time at the tail end of the slot which is not counted as a cost since in batch systems the resources can be released as soon as jobs are done. The complexity of this algorithm is given by $O(n^2 + n^2)$ i.e. $O(n^2)$.

- ***EC2 Task-based.*** We implement a task-based DAG scheduler for cloud (EC2) systems where resources are procured independently for each task. The bootstrap time for the machines for each task is added to the makespan of the workflow. In addition EC2 rounds up resource usage to the closest hour and thus the wastage on each resource slot also gets added to the cost. The complexity of this algorithm is $O(n^2)$.

- ***EC2 Slot-based.*** We also built a EC2 slot based planner where resources are assumed to be acquired at the start for the entire workflow and subsequently the slot based DAG scheduler is used. Resource usage is rounded to the closest hour and the slot overhead is added to the makespan. The complexity of this algorithm is $O(n^2)$.

Experimental setup. We use the four grid workflow examples (Figure 9.6) that routinely run on TeraGrid and/or other high performance systems. For our batch experiments we use probabilistic resource data from three TeraGrid machines (tagged as *ncsatg, abe* and *uctg*) located at the National Center for Supercomputing Applications and Argonne National Laboratory. We obtain resources acquisition probabilities through QBETS [125] and VARQ [126] and reliability probabilities through



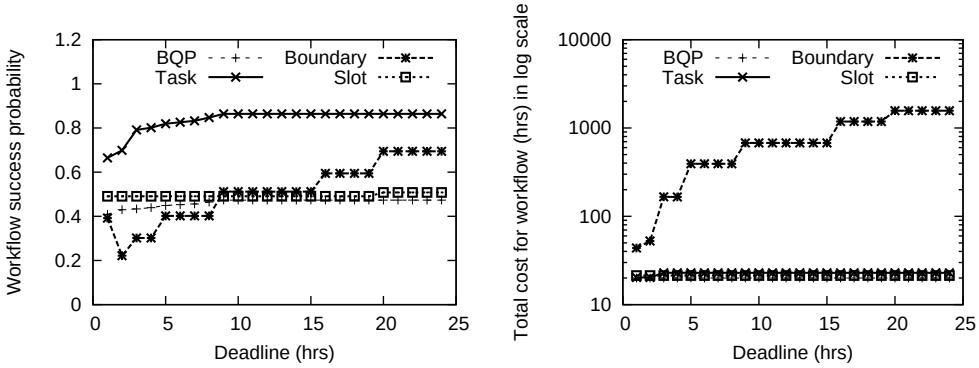
(a) Probability (b) Cost

Figure 9.11: Resource Procurement over Batch Systems for LEAD workflow. Comparison of different resource acquisition techniques for the **lead** workflow. We compare (a) the effective probability and (b) cost as deadline varies upto 24 hours.

AVP [11] for failure probabilities. For EC2 systems, we use the present day cost value of the resources and data transfer.

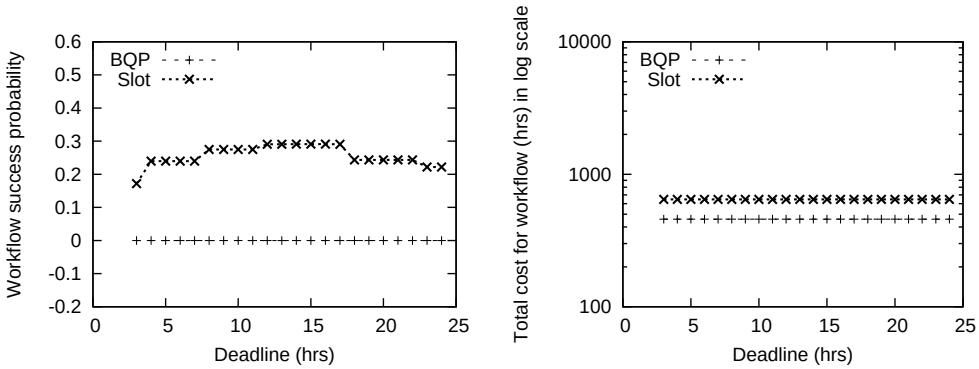
In our first experiment we compare the orchestration techniques discussed above using a workflow planner simulation. We recalculate the probabilities for tasks when schedules are enhanced by one or more mechanisms. We use the cost models (Chapter 8:Section 8.4.3) to calculate the cost for each mechanism as the total number of used CPU hours. In addition, on batch systems, resources can be vacated when a job or all jobs on a slot are done, thus incurring no costs for additional slot time at the end of the schedule. We compare success probability of the workflows, makespans, and associated resource usage costs.

On batch systems, for each workflow type we first use a task-based batch queue scheduler (*BQP*) for the planning. We also use a workflow-level slot based mechanism (*Slot*). For the small workflows, we apply Task-based(*Task*) and Boundary-based (*Boundary*) VARQ requests on the schedule. The probabilistic advanced reservation technique does have a known limitation; if there are multiple concurrent large resource requests made through VARQ, the queries could potentially perturb the predictions by dominating the workload behavior of the system. The perturbations induced



(a) Probability (b) Cost

Figure 9.12: Resource Procurement over Batch Systems for SCOOP workflow. Comparison of different resource acquisition techniques for **scoop** workflows. We compare (a) the effective probability and (b) cost (shown in log scale) as deadline varies upto 24 hours.



(a) Probability (b) Cost

Figure 9.13: Resource Procurement over Batch Systems for Motifi workflow. Comparison of different resource acquisition techniques for **motif** workflow. We compare (a) the effective probability and (b) cost (shown in log scale) as deadline varies upto 24 hours

by such requests are being studied by the VARQ team. Thus for large workflows (**motif**, **ncfs**) we compare only the *BQP* and *Slot* mechanisms since the predictions from VARQ are not guaranteed to be accurate. Each run is repeated multiple times over a period of three weeks. The probability predictions are very stable resulting in identical output.

Batch systems, small workflows. Figure 9.11 shows the probability and cost comparisons for the **lead** workflow for deadlines ranging from two to twenty-four hours on two (*ncsatg* and *abe*) and three resources (additional resource *uctg*). The additional resource has a slightly higher slot acquisition probability. For the **lead** workflow, the *Slot* mechanism assures the highest level of probability among the four techniques. The cost of the slot system is slightly higher than with vanilla *BQP* but considerably lower than both *Task* and *Boundary*-based. We see that there is a slight drop in the success probabilities for a deadline of 13 hours. This variation results from the granularity of the parameter sweep in the heuristic used in VARQ queries. A static advanced reservation on the TeraGrid for a 16 processor, 1.5 hour slot for **lead** workflow would cost anywhere from 24 CPU hours to 48 CPU hours (for premium factors of 1 and 2). The *Slot* based mechanism costs less than that.

Figure 9.12 shows the probability and cost comparisons for the **scoop** workflow for deadlines ranging from one to twenty-four hours. In this case, using *Task*-based slots for the individual tasks yields a higher probability than trying to get one big slot for the five parallel tasks. The *Boundary* slot also yields higher probability values for deadlines that are higher than 15 hours. In terms of cost, however, the boundary slots are significantly more expensive (about 100 to 1000 hrs) compared to less than 25 hours for other mechanism. The static advanced reservation of 80 processors for 17 minutes would cost between 22 and 44 CPU hours for this workflow and the *Slot* mechanism is on the lower end of this range.

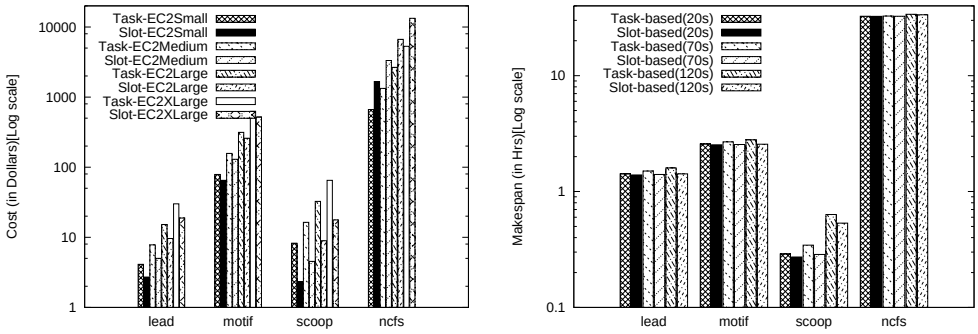
	BQP	Slot
Probability	0.0037	0.0066
Cost (Hours)	5631.5	16640.4

Table 9.3: Resource Procurement for NCFS workflow. The table shows the cost and success probability that can be obtained for an **ncfs** workflow scheduled for a deadline of 36 hours over batch systems.

Batch systems, medium and large workflows. Figure 9.13 shows the probability and cost comparisons for the **motif** workflow for the *BQP* and *Slot*-based mechanisms. The success probability of the workflow from a *Slot*-based system is higher than the *BQP* schedule. However as the deadline increases we see that the probability drops as a result of the reliability prediction for a 256-sized slot dropping. The *Slot* mechanism has a steady cost that is slightly higher than *BQP*. We compare the BQP approach with slot-based approach for the **ncfs** workflow for a 36 hour deadline (Table 9.3). While the success probability from the *Slot* mechanism is slightly higher, the costs are also higher.

Cloud (EC2). Cloud systems today implement *explicit resource control*. However they do have distinct overheads and cost models that affect the nature of workflow orchestration. For this set of experiments we assume EC2 systems have high acquisition (0.9999) and success probabilities (0.9999). We compare and contrast a *Task*-based and *Slot*-based policy. We calculate the EC2 costs for the instance-hours used by the workflow. We consider both computational costs (for different instance sizes) as well as data transfer costs for input and output data transfers to and from the cloud. Figure 9.14 shows the cost comparison and the effect on makespan for the four workflows for different instance sizes and overheads. In Figure 9.14(a) we see that for all instance sizes, the slot-based system incurs lesser cost than a task-based mechanism for the lead, **motif** and **scoop** workflows. However for the **ncfs** workflow, where each task executes for many hours, leaving resources idle in the slot system makes the cost significantly higher than the task-based approach.

Earlier experiments reveal that startup overhead for a small instance image varies from 20 to 30



(a) Cost (b) Makespan

Figure 9.14: Resource Procurement over Cloud Systems. Comparison of (a) cost and (b) makespan from task-based and workflow-based scheduling for workflows on Cloud (EC2) resources. The Y axis is in log scale.

seconds for 1 to 8 virtual machines [129]. We compare the makespans for overheads of 20, 70 and 120 seconds. Figure 9.14(b) shows the effect of startup and shutdown overheads on the makespan. In the task-based strategy the startup and shutdown overheads get added to each task's execution time. Our results show that the slot based system produces better makespans than the task-based systems. As the overheads increase, the difference also increases, as expected.

From our evaluation we see that probabilistic resource decisions help us understand the possibility of meeting a workflow deadline. Slot-based acquisition works well for our small and medium sized (**lead**, **scoop**, **motif**) workflow examples on both batch and cloud systems. For our larger sized workflow example (**ncfs**) the benefits are not substantial due to increased costs.

9.7 Summary

In this chapter, we proposed and evaluated workflow orchestration atop resource models provided by grid and cloud systems. We proposed and evaluated a heuristic for slot requests using DAG analysis methodology. To account for availability variations in distributed resources, we

evaluated a performability based DAG scheduling approach. The experiments demonstrate the effectiveness of using performability as a metric to account for availability variations over machine lifetimes, with minimal effect on makespan.

We design, implement and evaluate task-based and workflow-based deadline-sensitive orchestration algorithms. A workflow-based dynamic resource acquisition and planning strategy works well for all workflows in our example set on both cloud and grid systems but sometimes at a higher cost. Experiments demonstrate that *effective orchestration* is possible even on batch queue systems that have *no explicit resource control* through slots implemented with virtual advanced reservations.

10

Workflow sets

The scientific exploration process often has uncertainties. Application codes have a number of configurable input parameters and often a number of workflows need to be run concurrently. Workflow planning techniques today are focused on scheduling individual DAGs and do not consider the relationship between DAGs and constraints associated with scheduling a set of workflows [20, 112, 127, 205]. However as cyberinfrastructure deployments are used for complex scientific endeavors, support for planning and executing multiple workflows is necessary in the workflow tools. We need a workflow orchestration approach that manages workflow sets to balance cost, performance and reliability while meeting user constraints.

The problem of managing execution of concurrent workflows is especially common in the scientific domains such as weather forecasting, storm surge modeling and other application codes that use Monte-Carlo simulations where computing an exact result is impossible. In these cases it is often necessary to run a large number of model runs with different initial parameters to manage the accuracy of the result. Scientists would like to run an infinitely large set of workflows, but time and resources are limiting factors. It is also necessary to run a minimal number of the workflows to achieve desired accuracy. For such workflow sets, users specify that they minimally require at

least a fraction of the workflows to finish by the deadline. Individual members of the workflow might have different priorities requiring careful consideration of workflow properties while planning. Additionally it is necessary to coordinate individual workflow requirements with constraints on the entire set. For example, we need to determine if available resources should be used for fault tolerance strategies or scheduling additional workflows.

We developed a multi-phase workflow orchestration pipeline to balance performance, reliability and cost considerations for a set of workflows. The workflow orchestration (a) orders workflows within a set, (b) provides mechanisms to schedule minimal workflows required (c) provides provisions to compare the effectiveness of various fault tolerance strategies with scheduling additional workflows. We discuss the workflow orchestration pipeline (Section 10.1). We detail the workflow queue preparation in greater detail in Section 10.2 and discuss the implications on the execution system in Section 10.3. Finally, we consider three different case studies for the study of workflow pipeline with different policies and constraints. Specifically, we discuss implementations of our pipeline policies and present their evaluation as follows:

- We consider the simple case of scheduling a set of workflows on a set of slots and study the effect of various scheduling parameters. (Section 10.4)
- We discuss the integration of the workflow orchestration with the virtual grid execution system and present results from the integrated system and evaluate various parameters (Section 10.5).
- Finally, we present a deadline and budget-sensitive orchestration that uses the performability analysis and trade-offs with different fault tolerance strategies (Section 10.6).

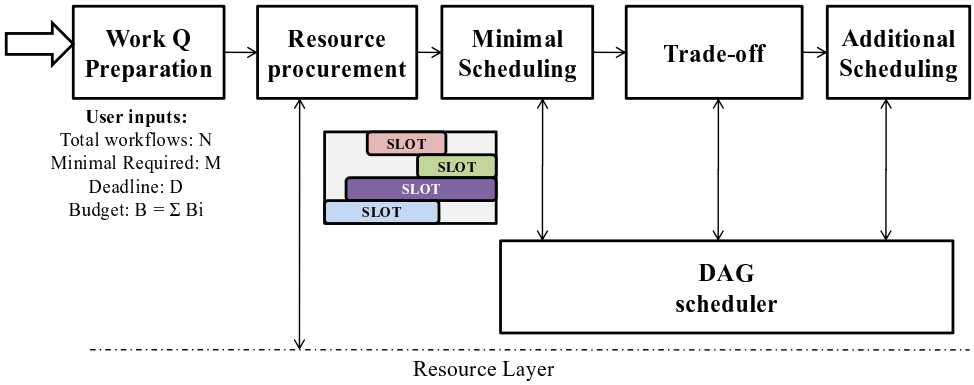


Figure 10.1: Workflow Orchestration Pipeline. It is a multi-phase orchestration strategy for scheduling workflow sets. The user workflows are assigned to priority queues. Next, the workflow constraints guide a resource procurement strategy. The resource procurement step returns a Gantt Chart that consists of a set of slots from the different sites. In the next phase the minimally required workflows are first mapped using a DAG scheduler. Subsequently in the trade-off phase, increasing fault tolerance for a scheduled workflow is compared with scheduling an additional DAG. The more effective schedule to meet workflow constraints is selected. Finally, any additional scheduling to use additional resources with different pricing or scheduling remaining DAGs or increasing fault tolerance is applied.

10.1 Workflow Orchestration Pipeline

Workflow sets require careful coordination of a number of workflow parameters with resource availability. Figure 10.1 shows a workflow orchestration pipeline for deadline-sensitive workflow sets that require a minimal fraction of the workflows to finish by a given deadline. The pipeline receives from the user workflow descriptions for each of the N workflows and a set of constraints that include the minimal number of workflows required (M), budget (B) and a deadline (D) on the entire set of workflows. Additionally each of the workflows in the queue may have additional time, budget or resource constraints. The orchestration pipeline consists of the following stages:

Workflow Queue Preparation. In the first stage of the pipeline, the workflows are ordered in a priority queue. The workflow queue preparation step enables us to classify and order the workflows by their importance and value. The other stages of the pipeline then consider workflows from the priority queue in order for scheduling.

Resource Procurement. In the next stage of the pipeline, resources are procured for the workflow set. Resource procurement strategies can vary based on various factors including budget or time constraints and workflow characteristics.

Minimal Scheduling. The goal of this phase is to schedule the minimal workflows (i.e., M of N) required by the user to complete by the deadline. Subsequently in the pipeline the workflow schedule is improved to either schedule additional workflows and/or increase fault tolerance of existing workflows. If enough resources are not available to schedule at least the minimally required workflows an error is returned to the user. A user might then relax one or more conditions and reinitiate the orchestration process. For scheduling the first M workflows from the priority queue this component repeatedly calls a DAG scheduler (Section 9.4) with all constraints applicable to this workflow derived from its individual as well as set-level constraints.

Trade-off. A challenging problem that is often faced in highly variable distributed systems is the decision on whether available resources should be used for increasing fault tolerance of scheduled workflows or scheduling additional workflows. In this stage of the pipeline we explore this trade-off in the context of workflow sets. In this case a scheduling strategy with fault-tolerance for the M scheduled workflows is compared with a scheduling strategy that includes additional workflows. We use the probability that at least M out of N workflows will complete by the deadline as the criteria for comparing the schedules.

Additional Scheduling. At the end of the pipeline, the orchestration handles special cases such as using more expensive resources to increase the fault tolerance of the resources, using available resource to schedule additional workflows or fault tolerance strategies for workflows or parts of the workflow that were not considered in the trade-off stage.

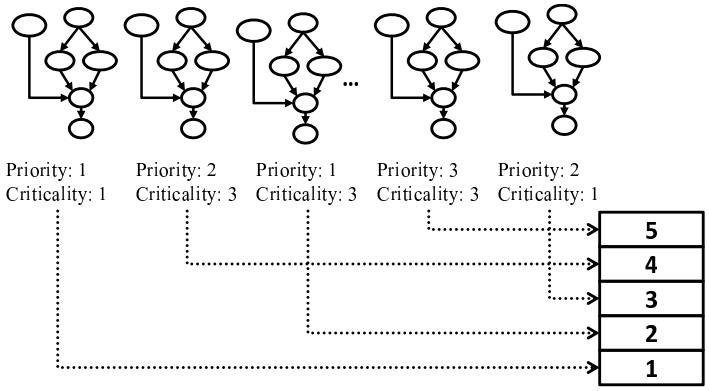


Figure 10.2: Workflow Queue Preparation. Workflows with different priorities and criticalities need to be placed in appropriate sequence for scheduling. Our queue is ordered by priority and then criticality between the elements with same priority.

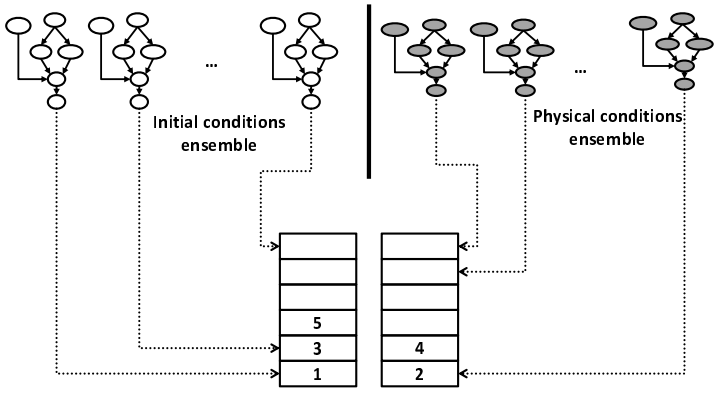


Figure 10.3: Queue of Queues. Often it is necessary to consider two subsets of workflows in conjunction during schedule. Our queue of queues approach enables two sets to scheduled simultaneously.

10.2 Work Queue Preparation

Every workflow in our system has two assigned properties - priority and criticality (Chapter 6). We assume priority and criticality have three values - High, Medium, Low. The priority is assigned by the system and used to implement policy between different users of the system. For example, a scientist would get a higher priority than a student using the system in a workshop. Criticality is the property that enables a scientist to assign relative importance of workflows in a workflow

set. Criticality values are expected to be associated with higher QoS and hence higher costs giving users an incentive to assign accurate criticality values.

A priority queue data structure manages the elements in the queue such that the highest priority element is always at the head of the queue. In our implementation workflows are ordered first by priority and then criticality values. Figure 10.2 shows a simple workflow queue example and ordering of a number of workflows.

Workflow sets in LEAD consist of two type of workflows - "initial condition" ensemble members where only initial conditions were varied and "physics" ensembles where the initial conditions were the same but different physics options to the model runs are used. The scientists require at least a fraction of each of the subsets to complete within a certain deadline to derive accurate results. In this case we require these subsets to be concurrently scheduled such that the required fraction from each subset is scheduled. Figure 10.3 shows the internal representation of the priority queue for such a workflow set that enables the heads of each of the queues to be considered for scheduling simultaneously. Thus, the priority queue is maintained as a queue of queues. Thus in Figure 10.3, there are two virtual queues in our queue representation. Each queue itself is ordered by priority and criticality values.

10.3 Execution Management

Workflow engines execute workflows based on DAG dependencies. However when considering workflow sets the execution plan will include execution dependencies on other workflow's tasks that are scheduled to run on the same resources. Figure 10.4 shows a schedule for two workflows A and B on a single slot. In the generated schedule, B3 and B4 wait for A4 and B2 to finish to start executing. Similarly B5 needs to wait for A6 to complete. Execution management in slots or

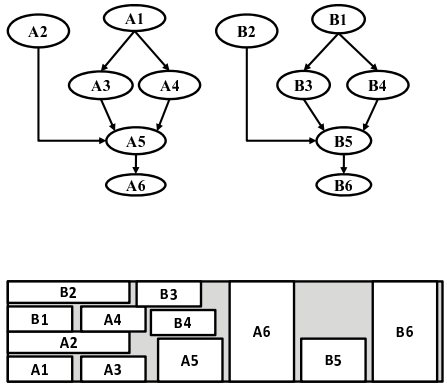


Figure 10.4: Execution dependency. When tasks from different workflows are scheduled on a slot there are additional execution dependencies. B3 and B4 are ready to execute but need to wait for A4 and B2 to finish. Similarly while B5 is ready to execute it must wait for A6.

containers is an active research topic. Batch queue software has been proposed as a mechanism to manage execution [89]. In batch queue systems resources are allocated to the next job in queue for which resources are available. In the above example, the workflow engine would launch B5 when B2, B3 and B4 complete. Similarly A6 would be ready to execute when A5 is complete. During execution, B5 would be ready to run before A6 and hence the slot batch queue software would start executing B5 delaying the execution of A6. This occurs because the slot execution manager (vanilla batch queue software) has no knowledge of the schedule imposed by higher-level tools. Thus we need execution level support for workflow set execution in slots that respects the order of the DAG and other workflow tasks scheduled on the same slot.

We implement a simple slot execution ordering mechanism in the Execution Manager. This ordering mechanism submits jobs to the slot execution batch queue system using the schedule. When a job is received at the execution manager it checks to see if all tasks that are scheduled on the slot before this task have been submitted. If all tasks scheduled before this haven't been submitted the task is saved in a pending queue for later execution. Events in the system such as job submission and job completion triggers a thread that checks the elements in the pending queue

to see if a task is ready to run. This ordering mechanism is sufficient to sequence task execution on slots as per generated schedule. However this ordering mechanism is not completely resistant to failures. If an earlier task fails to arrive, a task will be stuck in the pending queue till it is rectified. For example, if the workflow engine has a transient error which causes A6 to not be launched then B5 will stall as well. Also if an earlier task starts execution and fails on the resource, the current task will start execution not following the scheduling ordering i.e., if A6 fails during execution, B5 will start execution. Then a rescheduled A6 will need to wait for B5 to finish or use other mechanisms such as checkpointing or force B5 to vacate resources. Thus, in our implementation, the ordering mechanism depends on external mechanism such as the monitoring system to diagnose errors and rectify it.

10.4 Scheduling Workflow Sets Without Fault Tolerance

We consider a simple case of scheduling a workflow set with the constraint of M out of N workflows must complete by a given deadline with no fault tolerance. We detail the problem description in Section 10.4.1 and discuss the pipeline policies in Section 10.4.2. We present evaluation results in Section 10.4.3.

10.4.1 Problem Description

We consider a workflow set $W = \{W_1, W_2, ..., W_n\}$ where workflow W_i is a description of a DAG that specifies the ordering of task execution. In addition for each task $Task_k$ its execution on resource R_j is given by $[n, T_1]$ where n is the number of processors required for the task and T_1 denotes execution time of the application. Workflow W_i has higher priority than workflow W_j where $j > i$. The workflow set is specified to have the following constraint: workflows M where

$M \leq N$ must complete by a deadline D.

10.4.2 Pipeline Policies

We consider a simple pipeline that consists of workflow queue preparation, resource procurement, minimal scheduling of M DAGs and additional scheduling to handle any additional DAGs possible. We use the probabilistic DAG scheduler from Section 9.4.1.

We implement a simple set of policies using the slot based mechanism to procure resources for a *workflow set* and use the slot based DAG scheduler to meet the constraint of scheduling at least M out of N workflows by a given deadline.

In our implementation, the resource acquisition policy asks for slots for the duration between expected start time and the deadline for the workflow set. We make a resource query that is designed to ask for a resource width that minimally can satisfy the constraint M and possibly more. The minimum width is calculated as:

$$minWidth = (M * durationOf(workflow))/(durationOfSlot) * widthOf(workflow)$$

The maximum width is calculated as:

$$maxWidth = (N * durationOf(workflow))/(durationOfSlot) * widthOf(workflow)$$

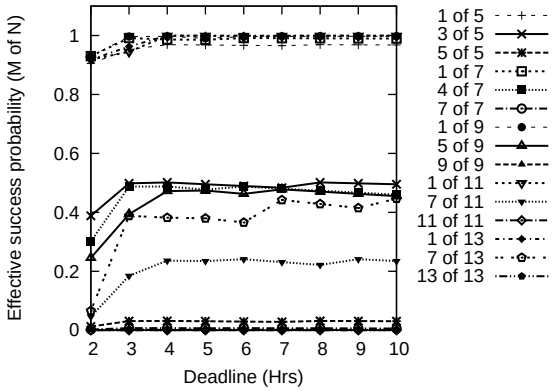
where $durationOf(workflow)$ and $widthOf(workflow)$ are the makespan and slot width for a single DAG and $durationOfSlot$ is the time between the start of the workflow set and the deadline. The results from the resource query are sorted by highest success probability and maximum processor width and the best result is picked for the schedule. The maximum number of possible DAGs are scheduled on this slot and we calculate the effective success probability of M-out-of-N workflows completing [137].

10.4.3 Evaluation

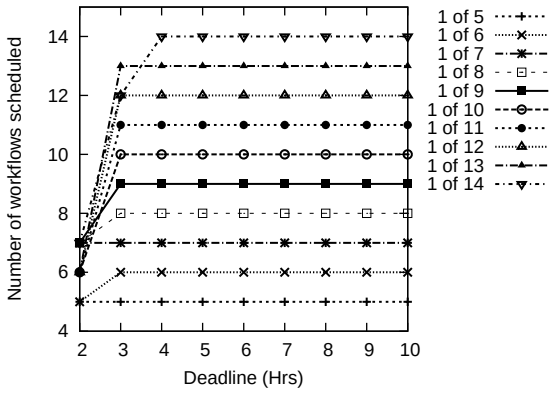
We perform a set of experiments with the **lead** workflow set to meet the constraint that minimally M out of N workflows must complete by deadline D. We assume workflows are scheduled for a start time that is 12 hours which is a reasonable time frame for advanced reservation requests. We explore the variation of the following parameters- effective success probability, deadline, M and N. Figure 10.5(a) shows the variation in the effective success probability of getting M out of N workflows with deadline and different M/N pairs. For short deadlines, limited resource time is available and we see slightly lower success probabilities. As expected, the success probability achievable increase as the deadlines are further out and remains fairly steady thereafter. For a given workflow set with N workflows, as M (the required number of workflows) increases we see that the effective success probability decreases. Figure 10.5(b) shows the number of workflows that were scheduled for a given M/N combination as the deadline varies. For short deadlines, the number of workflows scheduled is often less than N (the total), however at larger deadlines, all N workflows are scheduled. Finally, Figure 10.5(c) shows the variation in the effective success probability with varying M at a deadline of 7 hours. We see that there is a rapid decrease in probability as M increases for a given N since as more workflows are required to complete the guarantee that the system can make is lower that all the required ones will complete. Thus by changing the deadline and the value of M the user can determine various schedules that meet the user's needs.

10.4.4 Summary

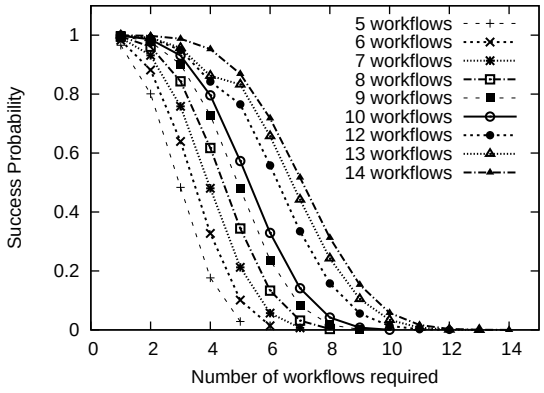
This case study demonstrates the resource procurement policies and scheduling techniques to schedule the maximum number of workflows given the amount of resources available. The orchestration is simple since it does not account for fault tolerance strategies or any other trade-offs.



(a) probability with D



(b) no. of workflows with D



(c) probability with M

Figure 10.5: Study of Deadline and Accuracy Scheduling of Workflow Set. We apply a slot based workflow orchestration to a workflow set to meet the constraint of at least M out of N workflows must finish within the deadline D. We study the variation of (a) probability with deadline for different M/N, (b) number of workflows that get scheduled with deadline D (c) variation of effective probability with variation in M for different N values and deadline of 7 hours

10.5 Scheduling over Grid and Cloud Resources with Fault Tolerance

Scientific workflows often have access to disparate set of resources that are allocated through different mechanisms. For example - LEAD uses a TeraGrid allocation for workflow execution. Optionally, LEAD can use Amazon EC2 resources that are priced differently from the TeraGrid resources. While these additional resources may not be used regularly, their use might be justified for situations where higher QoS is needed. In this case study, we study the impact of scheduling **lead** workflow sets on a mix of TeraGrid, local grid and cloud sites and EC2 resources.

This pipeline implementation uses the Virtual Grid Execution System (vgES) (described in Chapter 3) that provides an execution abstraction over grid and cloud systems. The system enables users to query, procure and execute on resources shielded from specific resource mechanisms. In addition, we use the VGrADS task-based fault tolerance implementation. The task-based fault tolerance implementation enables us to address the issue of if an individual task's success probability can be enhanced by replication.

10.5.1 System Design

Figure 10.6 shows a comparison of current day cyberinfrastructure deployments with the LEAD-VGrADS integrated system. Figure 10.6(a) shows workflow execution control flow in HPC scientific environments today. Users use science gateways or portal interfaces to compose, launch and monitor workflows. The portal interacts with a workflow engine that manages the task dependencies and execution. For each task in the workflow, the workflow engine invokes the corresponding application service that has knowledge about the application's configuration and data. The application service interacts with distributed sites using specific interfaces. In this architecture, resource

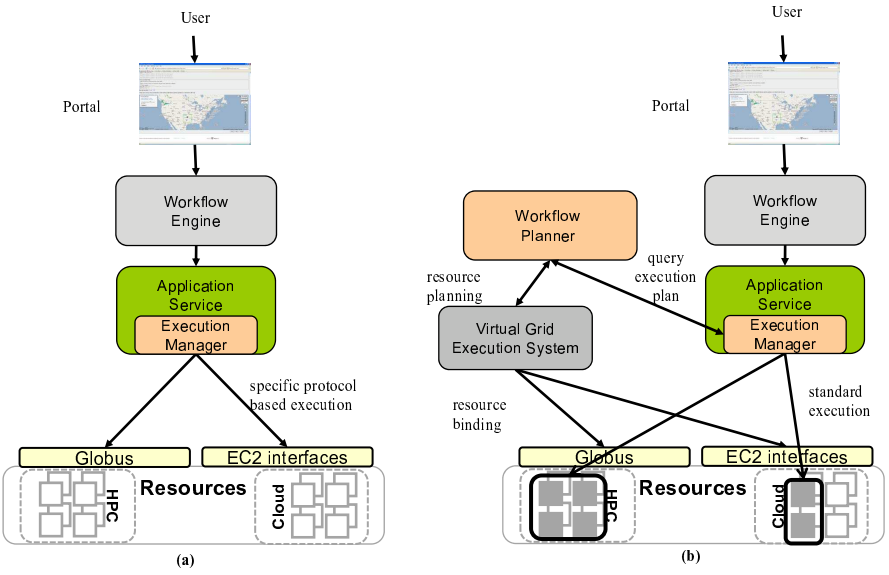


Figure 10.6: Comparison of the LEAD-VGrADS collaboration system with cyberinfrastructure production deployments.

decisions are ad-hoc and distributed across different components. This makes workflow planning and real-time adaptation extremely difficult and challenging. The performance and reliability variations associated with distributed environments, application requirements such as deadlines, cost factors associated with recent cloud computing requires better coordination of different resources types to meet user requirements.

Figure 10.6(b) shows the integrated system. The user interaction with the system remains identical to current day systems. However in addition to the normal execution flow, the workflow planner and the Virtual Grid Execution System (vgES) handle resource planning for the execution. This integrated system is an implementation of the WORDS architecture. The vgES system shields much of the differences of different execution systems and provides a single interface for querying and procuring resources across grid and cloud sites. In the vgES system, resources are represented as a hierarchical tree structure. For the purposes of this integration effort, the vgES system was expanded to return a Gantt chart representation of the slots.

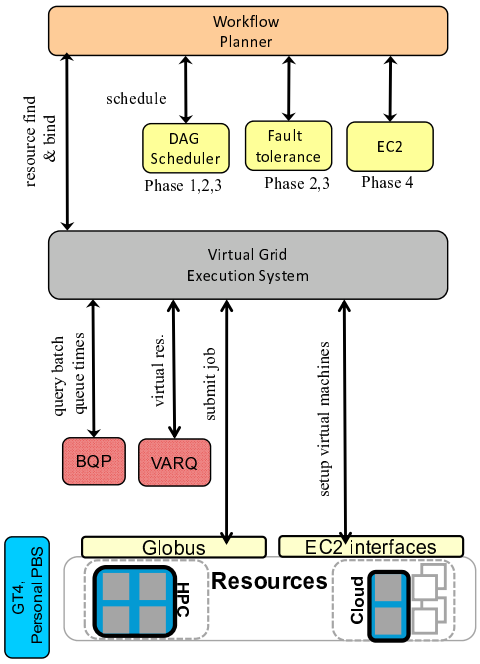


Figure 10.7: Interaction of Workflow Planner with VGrADS components. The workflow planner iteratively queries for resources and once sufficient resources are obtained initiates the resource binding process. The resource binding by vgES consists of a series of steps that include procuring the resources and setting up the resource to be ready for application execution. In parallel, the workflow planner determines the workflow execution plan on available resources.

In the integrated system, the workflow planner initiates a query for resources. The vgES system then interacts with diverse resource interfaces including grid and cloud systems to procure resources for a workflow or a set of workflows. The vgES system also sets up the resources assigned by the sites with standard interfaces that allows the execution manager in the application service to interact with each of the sites for job execution and file transfer agnostic to what specific sites are running. In our implementation each site hosts a Globus-PBS interface letting the LEAD infrastructure operate with the sites as it does with traditional batch queue supercomputing centers.

Figure 10.7 shows the interaction of various system components. The first step initiated by the workflow planner is resource procurement. The planner triggers the activities of vgES system and

returns back a set of slots that represent the resources assigned for this request. The vgES system interacts with grid and cloud systems with specific interfaces. It interacts with the virtual advanced reservation system to procure slots probabilistically on the batch queue sites. It also uses any advanced reservations that has been be procured at the sites. In addition, it interacts with web service interfaces with Amazon EC2 [5] and local cloud sites running Eucalyptus [129]. The workflow planner receives a set of slots from vgES and determines using simple policies if it is sufficient for its requirements. If resources received are insufficient or do not meet the original requirements the workflow planner iteratively relaxes constraints on the resource request and requeries the system. Once the workflow planner is satisfied with the requests, it requests vgES to "bind" or start the resource setup process.

The workflow planner then implements a four stage planning process for the LEAD workflow sets with the deadline and accuracy constraint. For the implementation we used an emulation framework (Appendix A) that mimics the LEAD cyberinfrastructure components. We describe the problem description for the workflow planning and the pipeline policies in Sections 10.5.2 and 10.5.3.

10.5.2 Problem Description

We consider a workflow set $W = \{W_1, W_2, ..., W_n\}$ where workflow W_i is a description of a DAG that specifies the ordering of task execution. The resources are available on sites $R_{total} = R_1 ... R_w$ where R_w is a higher priced resource and must be used sparingly.

For each task $Task_k$ its execution on resource R_j is given by $[n, T]$ where n is the number of processors required for the task and T denotes execution time of the application. Workflow W_i has higher priority than workflows W_j where $j > i$. The workflow set is specified to have the following constraint: workflows M where $M \leq N$ must complete by a deadline D.

10.5.3 Pipeline Policies

We detail our pipeline policies in greater detail in this section:

Resource Procurement. Each resource site R_j is queried for a certain number of processors from now till the deadline. The workflow orchestration has two goals: to meet the specified deadline and schedule the maximum number of workflows in the given time such as to increase the probability of at least the minimum required workflows complete. Thus we pick an aggressive resource request policy querying all sites for the maximum duration. When vgES returns a set of slots, we use the width and total computational units required as a coarse grained criteria to determine if sufficient resources are available. If the planner determines insufficient resources are available, it relaxes the minimum success probabilities required of the slots and queries the system again.

Minimal Scheduling. The goal of this stage of the pipeline is to schedule the first M out of N workflows in W over the set of resources other than R_w (Amazon EC2 in our implementation). The probabilistic DAG scheduler (Section 9.4.1) is used for scheduling each DAG. If M DAGs can't be scheduled, the planner exits with an error.

Trade-off. In the trade-off stage we compare scheduling additional workflows with increasing the fault-tolerance of one or more tasks of the scheduled M workflows. We compare the success probability of M out of N workflows completing as the criteria for picking the schedule at this stage. Probabilities of tasks completing are computed using the failure probability of the resources and the probabilities of its parent tasks. We maintain a queue of tasks from the scheduled M workflows that are sorted by their probability of completion. We compare:

- a schedule where a task T_i in T_m is replicated one or more times on available resources where T_i has the least success probability $\in T_M$ (The replication query is made to the VGrADS fault tolerance component to see if the success probability of an individual task can be increased.),

and

- a schedule from scheduling additional workflow Wi using the probabilistic DAG scheduler (Section 9.4.1).

The schedule that yields the higher success probability is selected and this step is repeated till no additional workflows can be scheduled or all tasks in the original M workflows have been checked to see if fault tolerance can be applied.

Additional Scheduling. If any workflows in W have not been scheduled in the earlier step, an attempt is made to schedule those. If any tasks in T have not been checked for fault tolerance in the earlier step, an attempt is made to replicate those tasks to increase its success probability. Finally, each of the tasks in the workflow is checked to see if using Amazon EC2 has an effect on the success probability. Costs on Amazon EC2 are bounded by the number of processors (16 in our setup) and the deadline.

10.5.4 Evaluation

In this section, we study the performance and behavior of our integrated system. Specifically we study the a) event timeline, b) comparison of resource acquisition times from batch and cloud resources, c) parameters of the workflow orchestration schedule.

Experiment Setup. Our setup consists of a mix of batch and cloud resources as shown in Table 10.1. The testbed consists of a virtual machine (bottlenose) where our entire software stack is hosted. The software consists of an Apache ODE workflow engine, the workflow planner service, the vgES code base and associated databases. The distributed infrastructure consists of local batch queue systems run at RENCI/Univ. of North Carolina - Chapel Hill (*kittyhawk*), Univ. of California- Santa Barbara(*mayhem*), NCSA TeraGrid (*mercury*), Eucalyptus based cloud resources at Univ of Houston

Machine	Software configuration	Processors Used	Mode
Mercury/NCSA	Globus, NWS sensor, PBS	32	Advanced reservation
KittyHawk/RENCI	Globus, NWS sensor, PBS	32	Advanced reservation
Mayhem/UCSB	Globus, NWS sensor, PBS	8	Batch queue
EC2	Web service interface	16	Cloud
UHEuca	Eucalyptus, NWS	16	Cloud
UCSBEuca	Eucalyptus, NWS	6	Cloud
UTKEuca	Eucalyptus, NWS	6	Cloud
tg-nws.cs.ucsb.edu	NWS memory and name server	1	NWS
bottlenose/UCSB	MySQL database for vgES and Apache ODE, vgES, WorkflowPlanner service, Application Service emulation, Apache ODE workflow engine	1	Launch of software
RENCI Euca	Eucalyptus, MySQL database for vgES and Apache ODE, vgES, WorkflowPlanner service, Application Service emulation, Apache ODE workflow engine	1	Launch of software

Table 10.1: Demonstration Testbed for LEAD-VGrADS. Setup configuration of grid and cloud resources in our testbed.

(*uheuca*), Univ of Tennesse - Knoxville(*utkeuca*)), Univ of California - Santa Barbara(*ucsbeuca*) and Amazon EC2(*ec2*). For these set of experiments advanced reservations were procured on *mercury* and *kittyhawk* clusters. In addition, each of the cloud sites was set up with a Linux image with the application binaries and data sets. In these experiments, we submit eight LEAD workflows which are ready to run in five minutes from start of the experiment. The data set we use for the LEAD workflow set is a small regional weather forecast and takes about 1.5 hrs to complete. The first few steps of the workflow take a few minutes on single processors and the weather forecasting model (WRF) takes over an hour and fifteen minutes on 16 processors. The constraint on the workflow set is that at least one workflow must complete by a deadline of 2 hours.

Event timeline Figures 10.8 and 10.9 show the snapshot of an experiment timeline. Figure 10.8 shows the timeline of the planning phase of the orchestration. Each line on the graph represents the duration of a particular event where the ends of the line signify the start and end time of the event. In the first step, the planner queries the VGES which takes about 49 seconds. Once a set of slots are determined to be sufficient, the binding process starts on all the sites in parallel. While the binding is in progress, the planner queries bandwidth across all pairs of sites and launches the phased workflow planning. The four phases takes about 7 seconds and complete well before the resource procurement is complete. The resource procurement duration varies by site, but resources are ready within 20 minutes. Once the resources are ready, the workflows begin execution (Figure 10.9). In this snapshot the workflow1 failed and hence finished fairly early. All other workflows complete by the deadline of 2 hours as expected. These figures demonstrate the effectiveness of our approach in scheduling workflow sets across distributed resource sites.

These experiments are repeatable and while there is some runtime variation, the figures are representative of the successful runs. It must be noted that the resources in our testbed have high success probabilities (e.g., through advanced reservations) and hence have a fairly high probability

of meeting the deadline. The effectiveness of the approach for meeting deadlines depends on the probability of the resource set that is available to the user. Additional optimizations in the timings are possible (e.g.,decreased levels of logging) when applied to production environments.

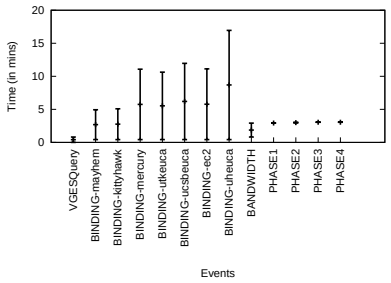


Figure 10.8: Planning Timeline. The graph shows the timeline of the planning phase in the system. The orchestration system queries the virtual grid execution system. Once prerequisite amount of resources are obtained, the vgES system is directed to start the binding process. Simultaneously, bandwidth between the sites is queried and the multi-phase pipeline process is launched. The end of the bind process signifies that the resources are ready to execute jobs.

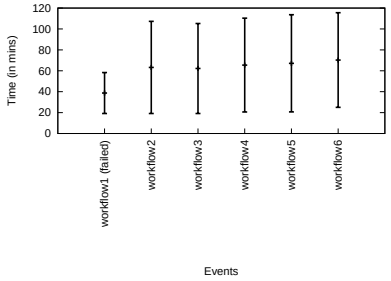


Figure 10.9: Execution Timeline. The graph shows the timeline of execution of the workflows in the system. In this run, workflow1 failed and hence completed earlier. All other workflows completed by its deadline.

We compare the start and end times of workflow execution with that predicted by the schedule (Figure 10.10). We see that the start and end times in the scheduler are different on the order of 13 to 22 minutes. The slots returned by VGES do not consider the overheads associated with resource procurement and setup. From vgES's perspective the slots are assigned to the user at the time the resources were requested. The application-level planning in today's systems has no estimates on

the overheads. A simple workaround might be to assume a maximum overhead to the start time of the slots. But long term, there is a need for better prediction and estimation mechanisms for resource setup. Additionally, tools are required to support dynamic and staggered execution to maximize slot utilization.

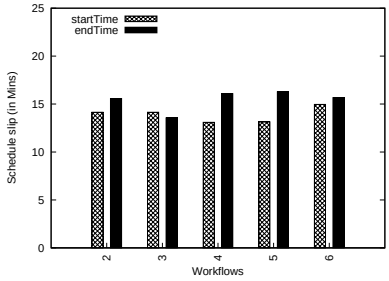


Figure 10.10: Comparison of Proposed and Actual Schedule. The graph shows a comparison of workflow start and end times with the generated schedule. The difference in start time is due to lack of tools for predicting resource binding and setup.

Resource Binding Figure 10.11 shows the average time required for binding each of the sites over nine to thirteen runs. The variation in the number of data points comes from failures at sites in some runs. The error bars show the minimum and the maximum values seen in the experiment set. The batch systems (*kittyhawk, mayhem and mercury*) take lesser time to setup than the cloud sites. The cloud sites based on Eucalyptus (*uheuca, ucsbeuca, utkeuca*) has more overhead than setting up the batch systems since the virtual machines need to be booted with the image. The cloud sites - uheuca and ec2 take longer since they boot 16 nodes. There is some variation in the bind time at Amazon EC2, kittyhawk and mercury that is the result of runtime characteristics such as the load on the machine. In this graph, we see that the overheads from cloud computing are slightly higher than from batch systems. However the overhead on all sites is less than 25 minutes, which is not significant for long running workflows. The batch queue wait times are minimized in these set of experiments since we have advanced reservations on the sites.

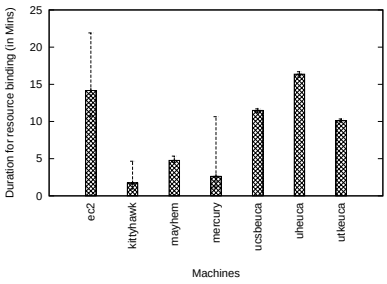


Figure 10.11: Resource Binding. The graphs shows the time at each site for resource binding or procurement. The average values are shown as bars and the high and low values are shown with error bars.

Scheduling Parameters In this set of experiments we query the virtual grid execution system for scheduling different sized workflow sets and evaluate the schedule that is generated by the orchestration component.

Figure 10.12 shows the number of workflows scheduled at different deadlines and sizes of the workflow set where at least five workflows (i.e., $M = 5$) must complete. When the workflow set has five workflows (i.e., $N = 5$), there are sufficient resources to schedule five workflows at any deadline. However when there are more workflows at a deadline of two hours only five workflows are scheduled. As the deadline is extended, resources are available for longer durations enabling scheduling of additional workflows such as for $N = 13$ all workflows can be scheduled for a deadline of five and six hours. Thus as the deadline increases, more workflows can be scheduled since more resource time is available, as expected.

Figure 10.13 shows the number of workflows that are scheduled by the planner with varying deadlines and for different values of M, i.e., number of workflows that must minimally complete out of ten workflows in a workflow set. A value of zero for the number of workflows scheduled indicates the planning failed to find a schedule that meets the accuracy constraint and hence zero workflows were scheduled to run. At a deadline of two hours, when five workflows were required,

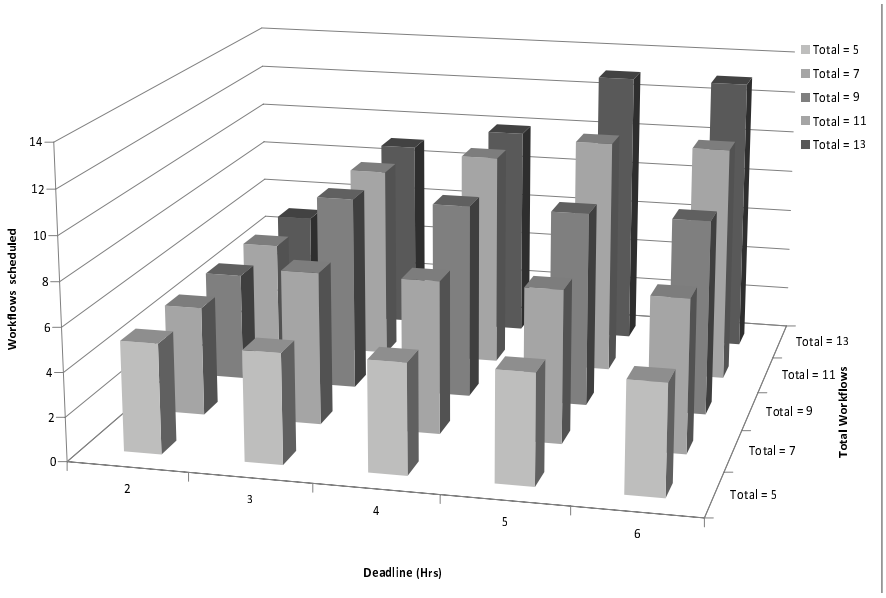


Figure 10.12: Schedule for Different Workflow Set Size. Number of workflows scheduled with deadline and different number of workflows in the set. This graph shows the case for when five workflows, i.e., $M = 5$

the planner was able to meet the minimum criteria. When $M > 5$ the workflow planner is unable to find a schedule that meets minimum criteria at a deadline of two hours. Similarly, at a deadline of three hours and when ten workflows are required, the planner is unable to find a schedule. Thus, if a user requires that more workflows must complete within a deadline, an acceptable schedule may not be found.

Figures 10.14, 10.15 and 10.16 shows the effect on the schedule with varying values of M for a set of workflows with nine members. Figure 10.14 shows the effective success probability of the schedule generated at different deadlines. The final effective success probability of the schedule meeting the deadline and accuracy constraints is close to 1 at low values of M. However as the number of workflows that are required to complete increases the success probability drops. Figure 10.15 shows the corresponding workflows that are scheduled at different values of M and

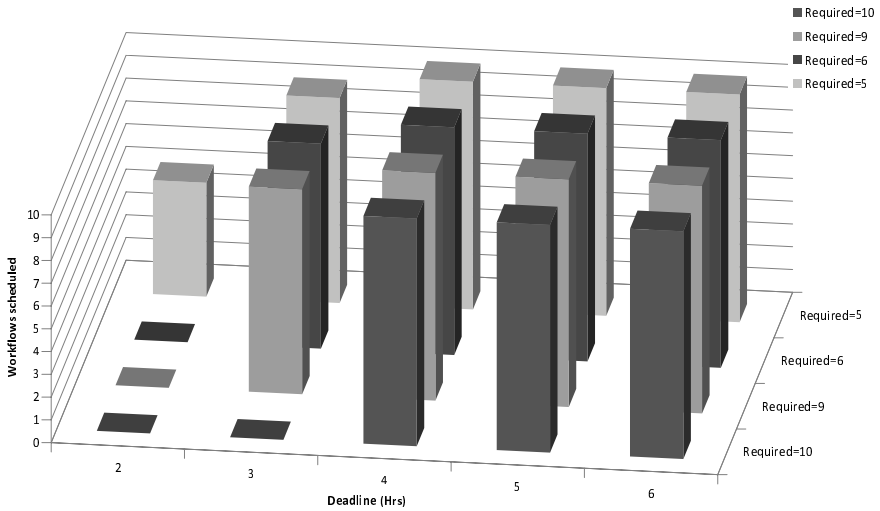


Figure 10.13: Schedule for Different User Requirements. Number of workflows scheduled with deadline and different quantity of workflows required. The graph shows the case for a total of ten workflows in the set.

deadline. We see that in the case of deadline of two hours only five workflows are scheduled. In all other cases, all nine workflows are scheduled. Figure 10.16 shows the replicas scheduled with varying M and at different deadlines. We see that as the deadline is extended for the same number of workflows, the number of replicas increases. Similarly as the number of workflows required increases. For the replication, each task is expected to meet a minimum success probability and as the number of workflows required increases, the effective success probability decreases requiring more replications.

Finally, in Figure 10.17, we compare the effects of each phase of our pipeline scheduling. Our resource set has high success probability values associated with them. We compare the effective success probability of the schedule at different stages for a workflow set with five workflows to be scheduled for a deadline of two hours. We see that each of the phases increase the effective success probability. Also in the case where a large number of workflows are required, the effective

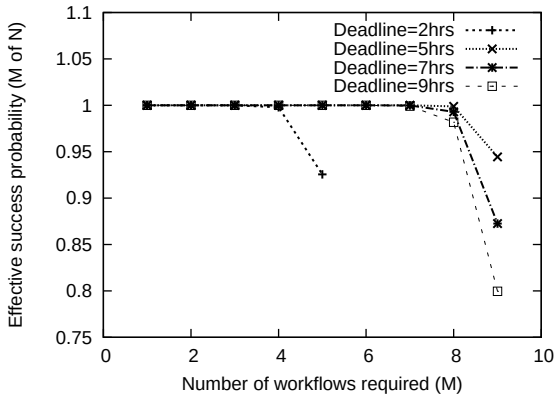


Figure 10.14: Schedule Success Probability with Different User Requirements. Effect on success probability of the workflow schedule with varying number of workflows required for a workflow set schedule with nine workflows.

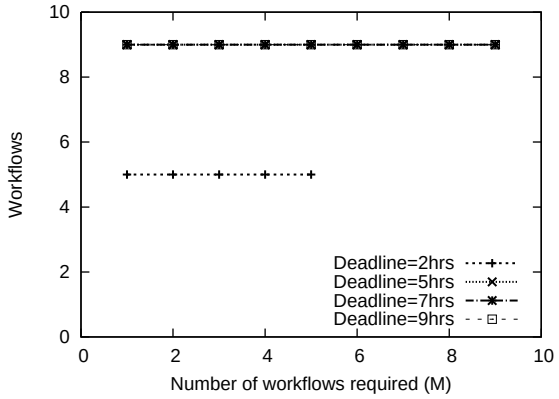


Figure 10.15: Schedule for Varying M. Effect on number of workflows scheduled with varying number of workflows required for a workflow set schedule with nine workflows.

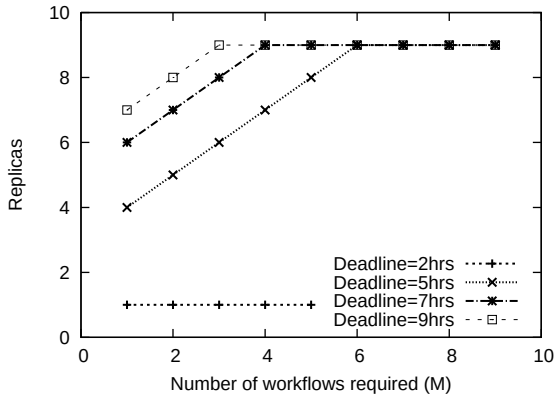


Figure 10.16: Effect on Fault Tolerance Strategy with Varying User Requirements. Number of replicas in the schedule with varying number of workflows required for a workflow set schedule with nine workflows

success probability is lower in the initial phases and is substantially improved by using Amazon EC2 resources in the schedule.

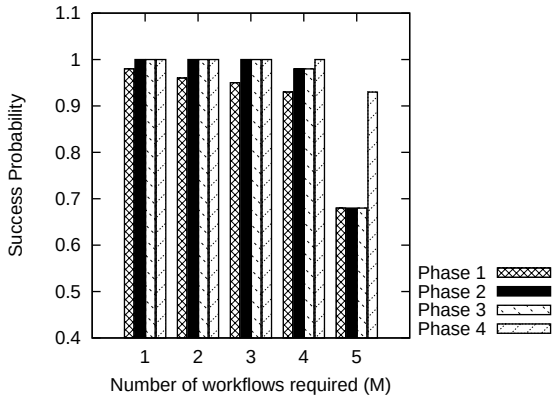


Figure 10.17: Effect on Success Probability from the Pipeline Scheduling. In this graph we see the success probability at the end of each of the phases for a workflow set with five workflows and a deadline of two hours

10.5.5 Summary

This case study enables us to study (a) the trade-offs between replicating certain tasks in the workflow and scheduling additional workflows, (b) the implications of using Amazon EC2 as a overflow resource for scientific workflows. However this case study is simplistic because often users will have budgets in terms of service units of dollars that they would like to spend for an experiment. In addition, a post-scheduling task-based fault tolerance strategy has certain problems. For example, a task could have been replicated if it was considered as part of the DAG scheduling strategy. It is possible that after scheduling M workflows the task cannot be replicated since that space is occupied by task from another workflow. In the next use case we consider an orchestration pipeline that overcomes these issues.

10.6 Deadline and Budget Sensitive Workflow Orchestration

We present a multi-phase workflow planning approach that is used to schedule a set of workflows with time and cost considerations using the performability approach discussed in Chapter 7. We state the formal problem description in Section 10.6.1. We present details on the implementation of the multi-phase planning strategy in Section 10.6.2 and present evaluation results in Section 10.6.3.

10.6.1 Problem Description

We consider a workflow set $W = \{W_1, W_2, ..., W_n\}$ where workflow W_i is a description of a DAG that specifies the ordering of task execution. The workflows have access to resources $R_total = \{R_1, ..., R_w\}$. In addition for each task $Task_k$ its execution on resource R_j is given by $[n, T_1, T_2..., T_5]$

where n is the number of processors required for the task and $T_1, T_2..., T_5$ denotes execution time of the application in High, Good, Medium, Low, Poor reliability states.

Workflow W_i in the set has higher priority than workflows W_j where $j > i$. The workflow set is specified to have the following constraint: workflows M where $M \leq N$ must complete by a deadline D within a total budget of B.

The budget B is specified across all the set of resource sites $R = R_1...R_w$ such that the budget for Resource R_j is B_j and $B = B_1 + B_2 + ... + B_w$. Each resource site R_j has failure-rate λ_j, repair-rate μ_j and has cost-rates $c_0...c_4$ in the degraded states.

10.6.2 Pipeline Policies

We apply a multi-phase planning approach to schedule workflows and fault-tolerance to increase the probability of a workflow-set meeting its constraints. The multi-phase approach trades performance and fault tolerance to meet the M out of N workflow accuracy and the deadline constraints on the given resources. We outline here the policies at different stages of the planning strategy:

Resource procurement. The first stage of the workflow planning is to procure resources for the workflows. Each resource site R_j is queried for a certain number of processors for the time duration from now till the deadline for a given budget B_j. Our workflow orchestration has two goals: to meet the specified deadline and schedule the maximum number of workflows in the given time such as to increase the probability of at least the minimum required workflows complete. Thus we pick an aggressive resource request policy querying all sites for the maximum duration. We use the budget at each site, B_j, and the maximum width of resources required at the site w and try to procure the best resource we can for the given duration between now and the deadline. We

also assume that a maximum duration of the resource is required i.e., resources are required for the entire duration from now till the deadline. Thus,

$$TotalBudget(B_j) = (costrate) * w * D \tag{10.1}$$

Thus cost-rate that we can afford at a given site is given by $\frac{B_j}{w*D}$.

The workflow planner uses the monitoring status of each of the sites and procure the resources where the cost-rate $\leq \frac{B_j}{w*D}$. Thus we acquire a set of resources where each resource $_j$ is described by $[D, w, state_j]$ where D gives the duration and w is the number of resources allocated and $state_j$ describes the current reliability state of the machine. Other appropriate policies for resource acquisition can be applied to trade-off system state with cost.

Minimal scheduling. In the next stage, an attempt is made to schedule the first M out of N workflows. For each of first M out of N workflows in W the deadline-sensitive Hybrid DAG scheduler (Section 9.4.3) is invoked. At this stage no fault-tolerance policies are applied. If M DAGs can't be scheduled, the planner exits with an error.

Trade-off. Next, the goal is to determine if the resources should be used to schedule additional workflows or use the resources for increasing the fault-tolerance for each of the M workflows. We compare the success probability of M out of N workflows completing as the criteria for picking the scheduling at this stage. Probabilities of tasks completing are computed using the failure probability of the resources in the complete fail state (Chapter 7:equation 7.1) and the probabilities of its parent tasks. The following two strategies are compared:

- a schedule where a workflow $Wi \in W_M$ has fault-tolerance enabled where Wi has the least success probability $\in W_M$ using the DAG scheduler from Chapter 9:Section 9.4.3,

- a schedule from mapping an additional workflow Wi without fault tolerance,

The schedule that yields the higher success probability is selected and this step is repeated till no additional workflows can be scheduled or all of the original M workflows have been checked to see if fault tolerance can be applied.

Additional Scheduling. If any workflow in W has not been scheduled in the earlier step, an attempt is made to schedule them. Finally, the fault tolerance strategy is applied to one or more workflows that are scheduled without fault tolerance to increase the probability of meeting the constraints.

10.6.3 Evaluation

In this section, we evaluate the parameters of our multi-phase deadline-driven workflow planning strategy over the lifetime of the LANL system. We study the impact of the following factors on the scheduling (a) failure-to-repair ratios (b) deadlines (c) user budgets. The results shown are for scheduling minimally three out of ten workflows.

Figure 10.18 (b) shows the number of workflows scheduled by the multi-phase workflow planning approach. When the deadline is two hours, only the minimal number of workflows get scheduled. At a deadline of three hours, in the beginning and end of the system lifetime where failure rates are high, only five workflows are scheduled but otherwise six workflows are scheduled. We observe that the number of workflows scheduled increases with the deadline and is sometimes affected by system properties. For example, when deadline is six hours we see that the number of workflows scheduled drops at system lifetime of about 175 hours and 500 hours, but we see a corresponding increase in number of replicas and checkpoints (Figures 10.18(b) and (c)).

Figures 10.19(a),(b) and (c) show the number of workflows scheduled, number of tasks that are

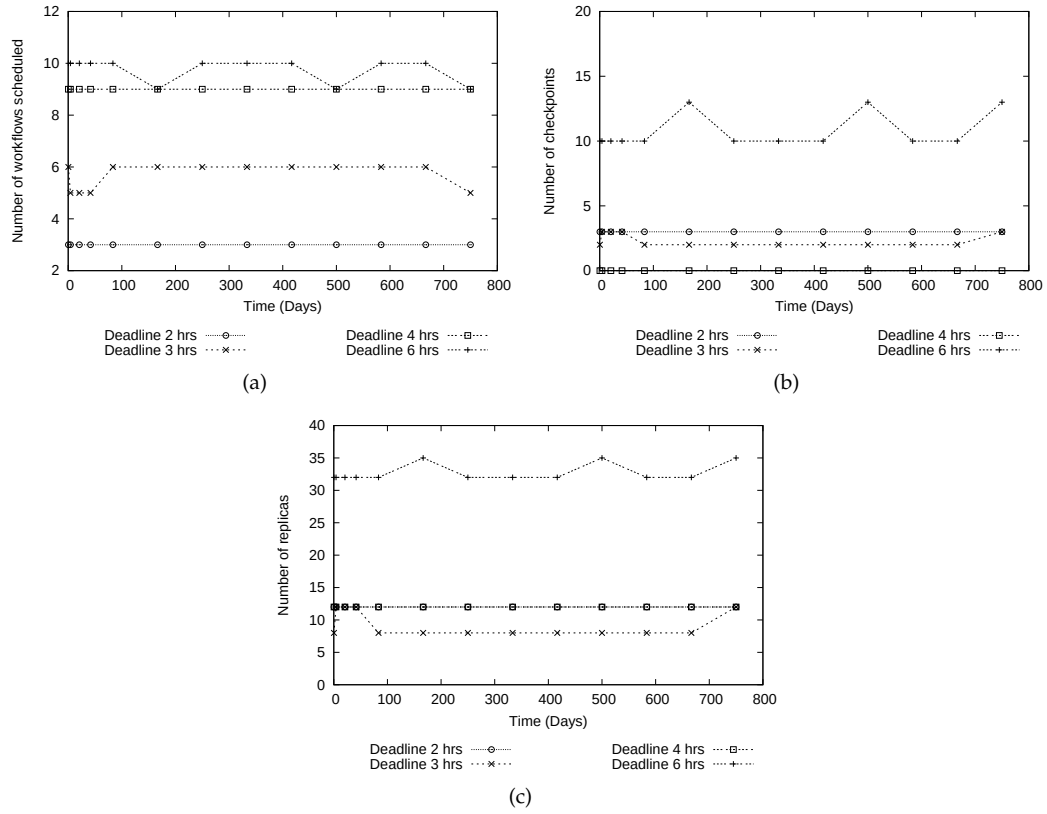


Figure 10.18: Performability Workflow Set Scheduling Over System Lifetime. Number of (a) workflows (b) checkpoints (c) replicas scheduled over the production use of systems lifetime at LANL.

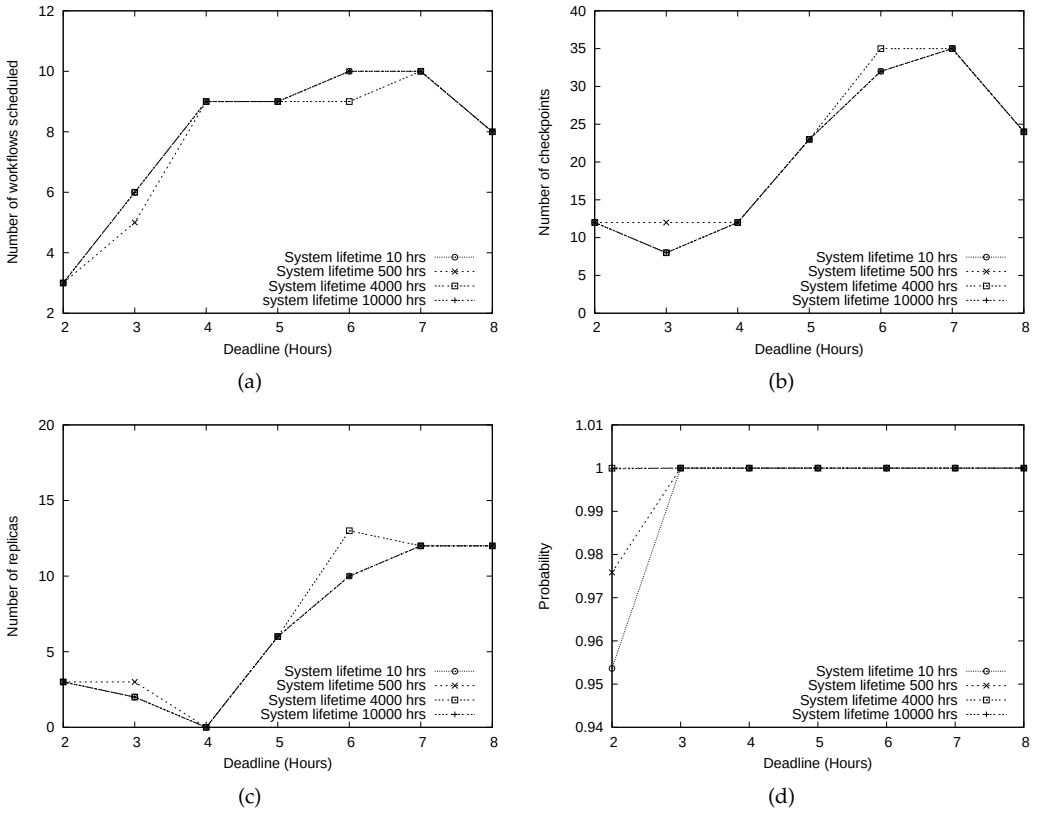


Figure 10.19: Performability Workflow Set Scheduling with Varying Deadline. Number of a) workflows b) tasks checkpointed c) tasks replicated (d) effective success probability with variation in deadline.

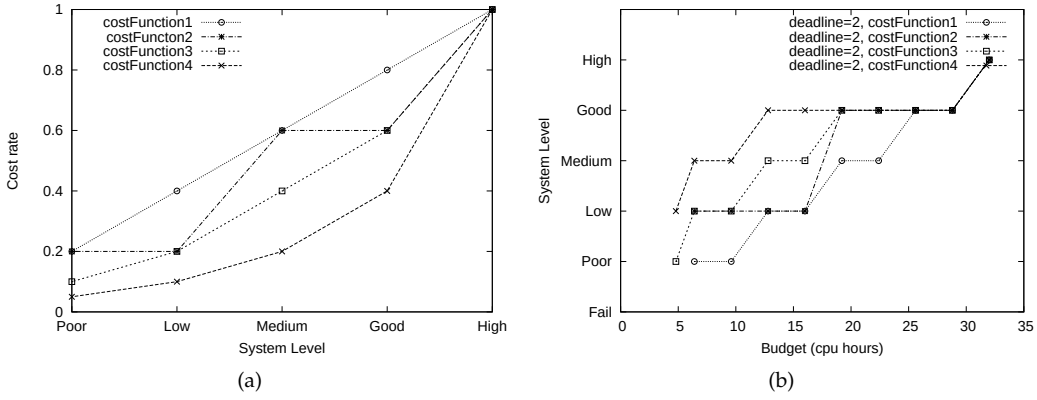


Figure 10.20: Effect of Budget on Resource Availability Level. Resource state variation with budget (a) shows four cost rate functions we consider for the resource state (b) shows the variation of resource stability level with budget.

checkpointed and replicated as deadline is varied. We see that as the deadline interval increases the system is able to schedule more workflows and guarantee higher level of fault-tolerance by checkpointing and replicating tasks since resources are available for a longer duration. The behavior is mostly similar across different points of system lifetime. Figure 10.19(d) shows the variation of the effective success probability achieved with our planning approach. At shorter deadlines, we see that the probabilities are lower especially in a system's early life (e.g., system lifetime of 10 hrs and 500 hrs) when failure-rates are high.

Finally, we study the resource state variation with allocated budget. Figure 10.20(a) shows the four costFunctions we consider for this analysis. For example, costFunction1 has a linear increase in cost with the different resource states. For each of the cost functions, we vary a sites's budget at a deadline of two hours and study the state of the resource that can be afforded. As expected, as the budget is increased the resource state improves (Figure 10.20(b)). The linear cost function (costFunction1) shows the slowest improvement in resource state with budget since the cost rates are not significantly different between each state.

Our evaluation demonstrates the importance of using performability as a basis of workflow scheduling and planning to achieve a certain level of QoS in degradable systems. The experiments demonstrates the effect of varying various parameters to achieve user constraints such as deadline and requiring minimally M out of N workflows under budget considerations.

10.6.4 Summary

Our workflow planning strategy accounts for performance, reliability and associated costs. This multi-phase planning strategy (a) guarantees at least minimum number of required workflows are scheduled, (b) applies fault tolerance strategies and/or (c) schedules additional workflows such that it meets a deadline and a budget constraint. The resource acquisition has a complexity of $O(n)$

and the DAG scheduler has a worst case complexity of $O(n^2)$. Thus, the workflow orchestration approach has worst case complexity of $O(n^2)$. Next, we evaluate the parameters in the workflow set planning strategy.

10.7 Summary

In this chapter, we demonstrated various orchestration strategies for workflow sets to balance cost, performance and reliability of the scheduled workflows. A coordinated effort across the resource and workflow layer, enables an orchestration approach that can increase the success probability of meeting user constraints even in the presence of resource variability. The different parameters (e.g., deadline, cost) determine the exact schedule. The multi-phase workflow orchestration approach provides a strong foundation for next-generation workflow planning and scheduling in grid and cloud environments.

11

Conclusions and Future Work

Recent advances in computing, i.e., emergence of virtualization technologies, web-services and multi-core processors, have accelerated advances in grid computing and spearheaded cloud computing business models. These changes in turn necessitate the need to closely examine the software stack that runs atop these systems and services provided by distributed data centers to provide predictable Quality of Service to scientific and business applications.

We developed WORDS that abstracts differences between specific resource models and provides a clear separation of concerns between application and resource layers in providing QoS to end user. In the context of this architecture, we investigate the capabilities required in grid and cloud resource control mechanisms to provide predictable performability guarantees. We also explore the application-level algorithms for deadline-sensitive workflow orchestration that can balance cost, performance, reliability. Specifically, we validate our hypothesis and answer the following research questions (Chapter 5):

1. Is a common abstraction possible that captures the different properties of grid and cloud systems and yet enables higher level systems to be shielded from specific system implementations?

A: Yes. We present a common resource abstraction across different resource control mechanisms. We demonstrate how effective workflow orchestration can be built on top of that abstraction while being shielded from specific system properties. The abstraction is based on the lowest common denominator of the system properties, yet is also flexible enough to accommodate advanced system properties such as resizable dynamic resource containers.

2. What information is required in next-generation data-center interfaces to improve support for dynamic adaptive workflows?

   A: Next-generation data center/resource interfaces need to reflect information regarding QoS parameters including performance, failures and cost that users or tools acting on behalf of users can use to trade-off various system parameters.

3. Can users be allowed to express dynamic user and resource constraints?

   A: Yes. Application interfaces need to balance simplicity of use with giving users more control that facilitate intelligent and autonomic scheduling decisions.

4. Is it possible to provide predictable quality of service atop systems that do not provide explicit resource control?

   A: Yes. We show that a probabilistic QoS model allows us to provide predictable quality of service atop systems such as batch systems that do not provide explicit resource control.

5. How can workflows account for variability in performance, and reliability that are inherent to distributed large-scale systems?

   A: By using performability analysis based on Markov Reward Models, we show how workflows can account for variability in performance and reliability and represent associated costs allowing workflows to do appropriate trade-offs.

6. How can workflow sets be scheduled to meet multiple constraints such as deadline and accuracy? How can higher-level tools determine appropriate fault tolerance strategies with cost and other constraints?

   A: We show that by using an orchestration pipeline and a probabilistic QoS model we can quantify the trade-offs between different constraints enabling a richer schedule that accounts for the system properties and user constraints.

Next we outline the specific contributions of this work and future directions that are motivated from the results in this work.

11.1 Resource Layer

We explore the interfaces and mechanisms required at the resource layer in next-generation data centers supporting grid and cloud systems.

Resource Abstraction. WORDS lays the foundation for next generation dynamic adaptive distributed environments. The resource abstraction in WORDS enables resource providers to provide specific bounds on QoS and facilitates higher-level applications to interact with and compare QoS capabilities of different resource systems without needing to know specific policies or implementation details. This abstraction provides the necessary framework for dynamic resource contracts that are required as cloud computing business models become mainstream.

Probabilistic QoS model. The cloud computing paradigm provides a clear separation between resource, service providers and consumers. This makes predictable resource control with appropriate QoS abstractions critical in today's distributed environments. We explore and demonstrate probabilistic bounds on resource procurement and failure characteristics as a feasible approach

to providing QoS service to end-user applications. The probabilistic QoS model accounts for uncertainty in procurement and quantifies availability characteristics in highly variable distributed systems. As cloud computing systems develop, it is important to explore QoS contracts between different level providers (e.g., IaaS - Infrastructure as a service providers, SaaS - Software as a service providers) in cloud environments that capture the resource behavior, business models and consumer requirements. To meet this goal, it will be necessary to explore the set of services that need to be hosted inside and outside cloud systems to facilitate end-user resource interactions for workload and data management, monitoring and adaptation for QoS and cost.

Performance, reliability and cost-based contracts With large scale deployments of resources at cloud computing centers, service providers need to handle degraded services at the compute, storage and network levels to manage availability variations in hardware and software. We propose and evaluate a Markov Reward Model for performability analysis i.e., measuring the effect of availability variations on performance and cost [141]. More recently energy efficiency or "green computing" has become central to design of data centers. Long term, we need policies and cost-models that enable higher-level applications to trade-off system parameters such as energy usage. The proposed performability model can be further expanded to build dynamic resource contracts between resource providers and consumers enabling criteria such as energy usage and failure characteristics to become part of the mainstream business model.

Container-provisioning. Cloud computing models have spearheaded the use of customized virtual environments. In this thesis, we illustrate the dynamic assignment of shared pools of computing resources to hosted grid environments. Our implementation shows how to extend grid management services to use a dynamic leasing service to acquire computational resources and integrate them into a grid environment in response to changing demand. Each site controls a dynamic assignment of its local cluster resources to the locally hosted grid points of presence. Our approach

addresses resource control at the container level, independently of the software that runs within the container. Further investigation is required for policy choices required for resource provisioning at the data center.

11.2 Application Layer

We develop the interaction of high-level application tools (e.g., workflow tools) with next-generation data centers and interfaces required at the user-level for easy access to cloud computing systems.

Constraint Specifications. We propose a user-level constraint space that allows users to specify higher-level constraints such as budget, priorities, etc. on their work units to guide orchestrations decisions. It is important to further explore constraint specifications at the user-level that allow end users to specify performance, availability, cost, security and other QoS expectations. These need to be mapped to appropriate policies at the resource level.

Programming models. Different programming models and tools have evolved in the grid and cloud computing space. Higher-level workflow tools have various internal representations for dependencies between different tasks. The Web Services Business Process Execution Language (WS-BPEL) [202] is one such executable language for specifying interactions between web service workflow components. On the other hand tools such as Apache Hadoop [7] and Dryad [84] have evolved to handle execution in parallel clustered systems. Scientific applications are often composed through Message Passing Interfaces, Master-Worker, Divide and Conquer or Single Program Multiple Data programming paradigms. There is a disconnect between these programming models at different levels requiring specific tools for specific applications. For example, workflow tools today see only task dependencies, whereas resource planning in Hadoop develop execution

plans based on resource availability and other concurrent workloads. The execution framework in WORDS includes a prototype execution system that managed runtime execution dependencies in addition to the task dependency managed by an Apache ODE [8] workflow engine. It is necessary to further explore the interaction between application and cloud-level execution tools to manage execution based on user constraints and resource availability characteristics.

Dynamic Scheduling. We explore multi-phase workflow orchestration approaches that balance performance, reliability and cost trade-offs for deadline-sensitive workflows that have accuracy and timeliness constraints. The workloads that were considered were scientific workflows from diverse domains including bioinformatics and biomedicine, weather and ocean modeling. Dynamic scheduling techniques need further exploration for scientific workloads that require timely coordination of streaming data from instruments, computational resources and access to large-scale storage systems and periodic human intervention to resolve uncertainties during runtime that were identified as requirements in our workflow survey. Dynamic scheduling is also critical for mainstream business workloads such as mobile applications that process large data sets on distributed cloud resources.

Adaptation. Uncertainty is a key element of distributed system. Preplanning techniques need to be accompanied by real-time adaptation. An adaptation infrastructure needs to be supported with monitoring infrastructure to detect performance and reliability fluctuations and policies at different levels. For example,the adaptation infrastructure can receive millions of adaptation events sometimes requiring conflicting actions requiring an arbitration policy. The WORDS architecture provides the infrastructure to further explore adaptation requirements in next-generation grid and cloud systems.

A

Workflow Emulator

In the last few years, workflows and workflow tools have become an integral part of cyberinfrastructure. These workflow systems often interact with other middleware including planners, schedulers, web services, provenance systems and resource-level services. The complex requirements of this software stack drives the research in computer science to investigate and apply innovative techniques and mechanisms to manage these environments. The complexity and cost of these systems often makes it hard to experiment and test new computer science mechanisms during actual workflow execution. Often there is also a need to replay a workflow execution to inspect and generate the data events associated with workflow execution. We used a simple service based workflow emulation model that serves as a benchmark platform to experiment with mechanisms and policies in a controlled environment.

Figure A.1 shows the workflow emulation architecture. It consists of an existing workflow engines that works coordinates an emulation service to recreate the workflow execution flow. The workflow engine in our particular implementation is Apache ODE [8], a BPEL based workflow engine that invokes the emulation service in place of the application service that is invoked during real execution for each step in the workflow or directed acyclic graph(DAG). The emulation service follows a state based execution flow that captures different stages of task execution including data transfer, computation, post-processing, etc.

The specific information such as task execution time and data transfers are retrieved from a local

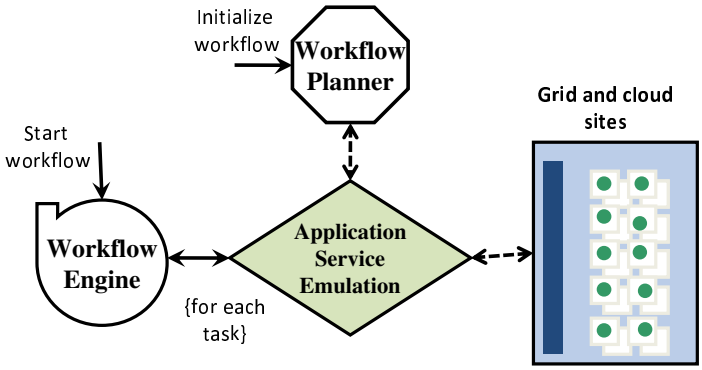


Figure A.1: Workflow Emulation Architecture. The figure shows the interaction of the different components in the emulation environment.

database for each workflow during execution. The emulation service may also interact with external systems like a grid emulator that emulates application running on different resource provider sites [139]. For purposes of this research, the service emulation interacts with a workflow planner service.

The workflow emulation architecture is a minimalistic framework that can be configured with external event handlers to register the execution activities that is of interest for a particular case study. In addition to this research it is being used to generate a provenance database for workflows [51].

A.1 Application Service Handler Interface

The application service emulation execution flow is captured with the state diagram shown in Figure A.2. The service is configurable to allow event handlers to be associated with each state for a particular workflow or task execution. Each state may be associated with multiple event handlers (e.g. for provenance generation and scheduling) that are specified before execution. The states are described below.

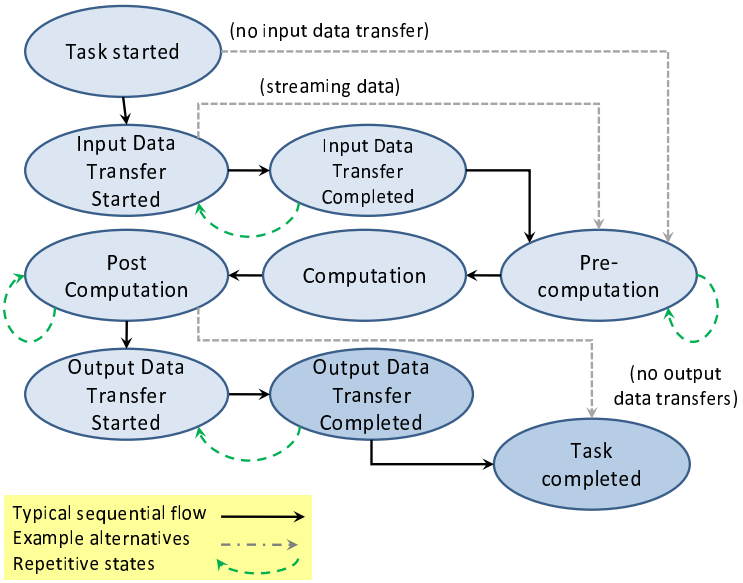


Figure A.2: Application Service Emulation Execution Flow. The figure shows the different states supported by the emulation execution flow for each task in the workflow. Some states may be skipped or repeated for different scenarios.

Task Started. This marks the start of an activity or task. The handlers for this state may invoke a workflow planning component to get resource information and then retrieve the task information (e.g. execution time, data sizes) based on the resource information. In addition handlers may publish events to signify the task was invoked.

Input Data Transfer Started and Input Data Transfer Completed. These states are used to capture the data transfers that may be required for the particular task at hand. These steps will be repeated for each data product that may be required by the computation. These steps can also be skipped for special circumstances - e.g. if there are no input data transfers that are required or the input data transfer completed state may not be relevant in the case of streaming input data.

Pre-computation. In this state typically pre-computation steps are invoked. For example, input data products may be registered with a meta-data catalog or specific resource based information for the computation may be retrieved

Computation. In this state the emulation of the computation stage will be emulated. This could be a NOP (no operation) for the application execution time or an external grid emulator can be invoked.

Post computation. This represents the post-computation activities that can include data product registration.

Output Data Transfer Started and Output Data Transfer Completed. These states are used to capture the data transfers that are required for the output data generated by the emulated task. These steps will be repeated for each data product that is required by the computation. These steps can also be skipped if data transfers are not required for this task.

Task Complete. This is the final state in a normal execution flow and the result of the task is sent back to the workflow engine which then uses that to invoke the next task in the DAG.

A.2 Orchestration Handler

The application service handler interface is customizable. The orchestration handler in the emulator handles the following to demonstrate and evaluate components of this research:

- For every task in the workflow it queries the workflow planner to determine the resources the task must run on. There cab be replicas for the tasks and the resource mapping for each of them is returned.
- It interacts with the expanded execution manager interface and triggers job submission with a description of the job. This is repeated for each replica.
- It queries the execution manager for job status and wait till the job succeeds or fails.

This emulation architecture is used in concert with other system components as a demonstration of the orchestration mechanisms over the Virtual Grid Execution System (vgES) (described in Chapter 10).

B

LEAD Portal Resource Usage Analysis

The deployed LEAD system consists of a portal [1] that allows users to interact with meteorological workflows that run across a distributed testbed. The portal provides access to pre-composed workflows that can be configured with specific data. The workflow types supported consist of the weather forecasting and data mining workflows. The portal also provides other tools for visualization, to create and configure application services, and manage data products. The LEAD meta-data sub-system collects and manages workflow data of the user's experiments. We present here a higher-level analysis of the data to highlight the types of workflows supported in the system and the performance and reliability characteristics experienced by these workflows.

The data presented here is from user experiments run between October 2007 and March 2009. Our system allows users to delete their experiments and the meta-data for the experiments is purged from the system. Thus, more experiments were run than what is detailed here. The experiments also consist of developer run workflows that were used to test the system components. Other studies by other project members capture the problems and solutions that were put in place for handling failures associated with job submission [114] and file transfer [171] that were encountered and rectified in the infrastructure.

[1] https://portal.leadproject.org

B.1 System Details

The LEAD portal supports three primary type of workflows - data mining, and NAM-initialized and ADAS-initialized weather forecasting workflows (described in Chapter 2: Section 2.2). Another workflow type that has been deployed in the portal and primarily used for testing is a simple Echo workflow.

The different execution components of the LEAD system publish notifications to a notification bus. The LEAD meta-data system subscribes to the notifications and manages the data in backend databases.

The LEAD infrastructure consists of a diverse set of geographically distributed resources. The LEAD production services including the portal, application services are hosted on the tyr cluster. The TeraGrid resources provide back-end computational infrastructure for job execution. Table B.1 shows the configuration of these machines. The TeraGrid systems are all batch queue resources and applications incur wait time on these resources. During workshops and significant events, administrators of the LEAD system procure out-of-band advanced reservations on these systems to reduce wait times.

B.2 Overview

We parse the execution metadata to determine the type of the workflow. The parser detects three primary types of workflows - NAM, ADAS and MINING workflows. The NAM and ADAS workflows differ in just one task and sometimes hard to classify them when there are missing notification messages or the workflow has failed. In addition some workflows have missing notifications and can't be classified and are tabulated as UNCLASSIFIED.

Machine	Usage	Specifications
tyr	LEAD services	16 nodes, Dual AMD 2.0GHZ Opteron, 16GB memory per node, Redhat Enterprise Linux
anl	computation (Argonne TeraGrid)	62 nodes, Dual Intel Itanium 2 processors, 4GB memory per node, Redhat Enterprise Linux
bigred	computation (IU TeraGrid)	768 nodes, Dual-core 2.5GHz PowerPC 970MP, 8GB memory per node, Suse Enterprise Linux
mercury	computation (NCSA TeraGrid)	887 IBM nodes: 256 nodes with dual 1.3 GHz Intel Itanium 2 processors (half with 4 GB of memory per node, and the rest with 12 GB of memory per node), 631 nodes with dual 1.5 GHz Intel Itanium 2 processors (4 GB of memory per node), SuSE Linux
tungsten	computation (NCSA TeraGrid - decommissioned)	1280 nodes with dual Dell PowerEdge 1750 server, 3GB memory per node, Redhat Linux (decommissioned in 2008)

Table B.1: LEAD Production Testbed. Machine specifications of systems used by LEAD.

Type	Total	Success	Recovered	Failed
NAM	1483	386	141	956
ADAS	1237	301	148	787
MINING	337	267	8	46
ECHO	68	8	0	60
NAM or ADAS	473	4	0	469
UNCLASSIFIED	574	4	0	570
Total	4172	971	297	2904

Table B.2: LEAD Production Workflow Completion States. The table shows the classification of workflows that are in different termination states - success, failed or recovered.

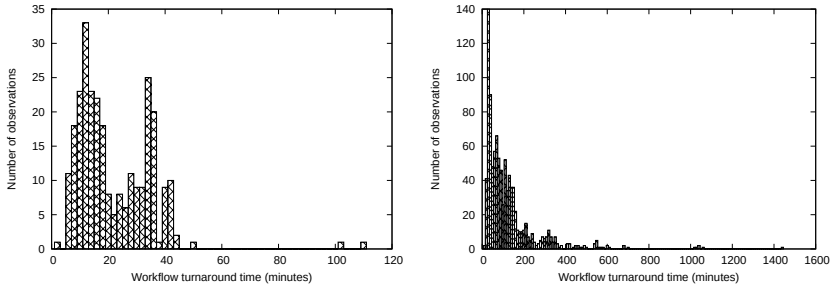
A workflow can be in one of three final states in our system. The workflow can complete successfully without encountering failures or can fail during execution. In addition the LEAD system has built in fault tolerance strategies such as task replication [87] and retries that were deployed in the system during the time frame of this data collection. We classify a workflow that has duplicate tasks as RECOVERED. A duplicate task will occur if the task failed or data transfer failed or the system timed out due to large queue wait times and was resubmitted to a different resource.

Table B.2 shows the classification of the workflows by type and the number of workflows in each of the final states. The majority of the workflows (76%) were identified as NAM or ADAS workflows, of which 11% of the workflows could not be distinguished.

In the weather forecasting workflows only 35%(NAM) and 36% (ADAS) workflows completed successfully with or without recovery. In contrast 81% of the MINING workflows completed successfully out of which only 2% had to be recovered. The higher rates of success for the MINING workflow are probably due their shorter running duration and hence less likely to encounter failures.

About 14% of the workflows could not be classified into correct workflow types. The successful workflows in this class had missing notification messages that prevented them from being classified automatically. The largest number of errors (85%) in the workflows are immediately after workflow initialization. The remaining errors are from computational task or data transfer failures.

Figure B.1 shows the turnaround time distribution for the data mining and meteorological workflows. The total turnaround time of workflows can vary due to a number of reasons including user inputs, data transfer times, execution time and batch queue wait times. The data mining workflow takes on the order of few minutes to a couple of hours though the largest number of workflows finish within 40 minutes. The weather forecasting workflows vary from a few minutes to a number of hours.



(a) dataMining (b) meteorology

Figure B.1: LEAD Workflows Performance Variation.

B.3 Data Mining Workflow

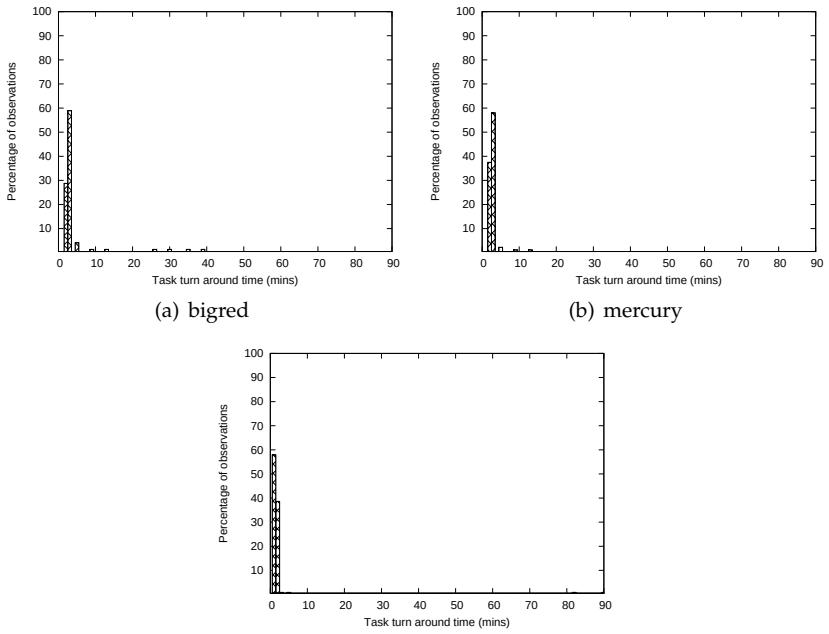
The data mining workflow (Chapter 2, Section 2.2) has three primary tasks in the workflow for a) storm detection, b) removing attributes and c) spatial clustering. We study the distribution of the execution time of these tasks.

The storm detection algorithm algorithm takes in the order of two to three minutes for the majority of the runs on all three machines(Figure B.2). However we see some variation on all machines and the maximum variation is seen on tungsten.

The remove attibutes and spatial clustering tasks, in most cases, takes between one and three minutes on all machines (Figures B.3 and B.4). The remove attributes is fastest on tungsten where 60% of the entries complete within one minute. The perturbation in execution time is minimal for both these tasks.

B.4 Weather Forecasting Workflow

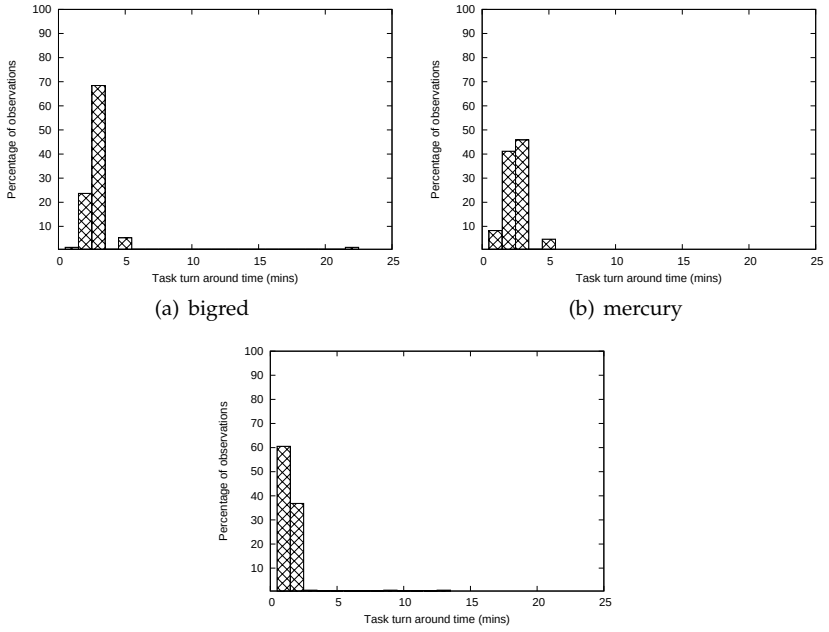
The weather forecasting workflows consists of six tasks. These are a) Terrain Preprocessor b) WRF Static c) Nam Initial interpolators d) NAM Lateral or ADAS interpolators e) ARPS2WRF f)



(a) bigred (b) mercury

(c) tungsten

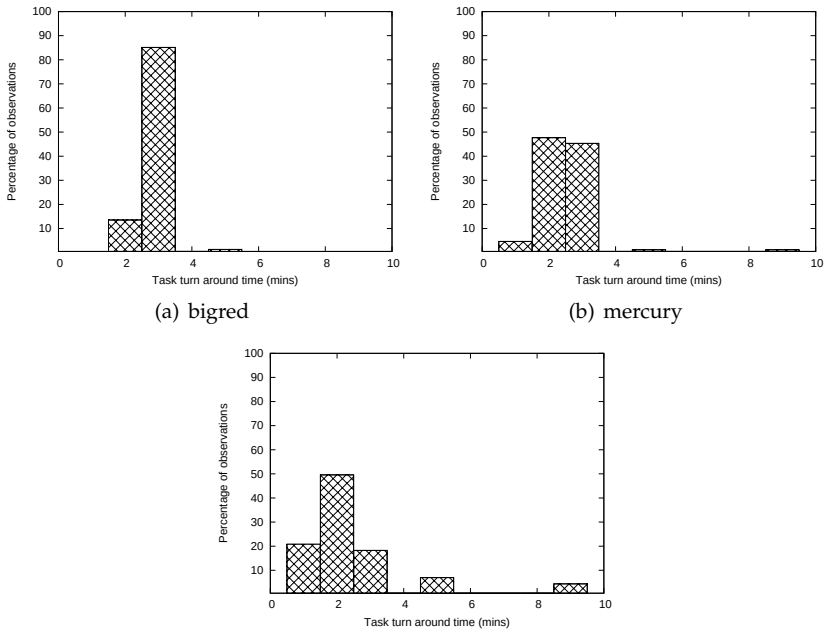
Figure B.2: Storm Detection. The distribution of execution times for the storm detection algorithm code on bigred, mercury and tungsten.



(a) bigred (b) mercury

(c) tungsten

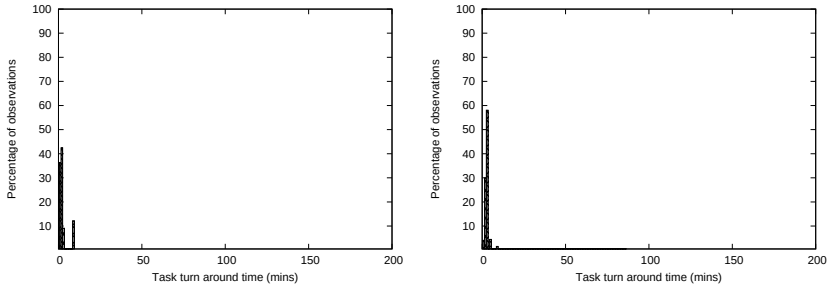
Figure B.3: Remove attributes. The distribution of execution times for the remove attributes code on bigred, mercury and tungsten.



(a) bigred (b) mercury

(c) tungsten

Figure B.4: Spatial Clustering. Execution times on bigred, mercury and tungsten.



(a) anl (b) bigred

(c) mercury (d) tungsten

Figure B.5: Terrain Preprocessor. Execution times (a) anl (b) bigred (c) mercury (d) tungsten.

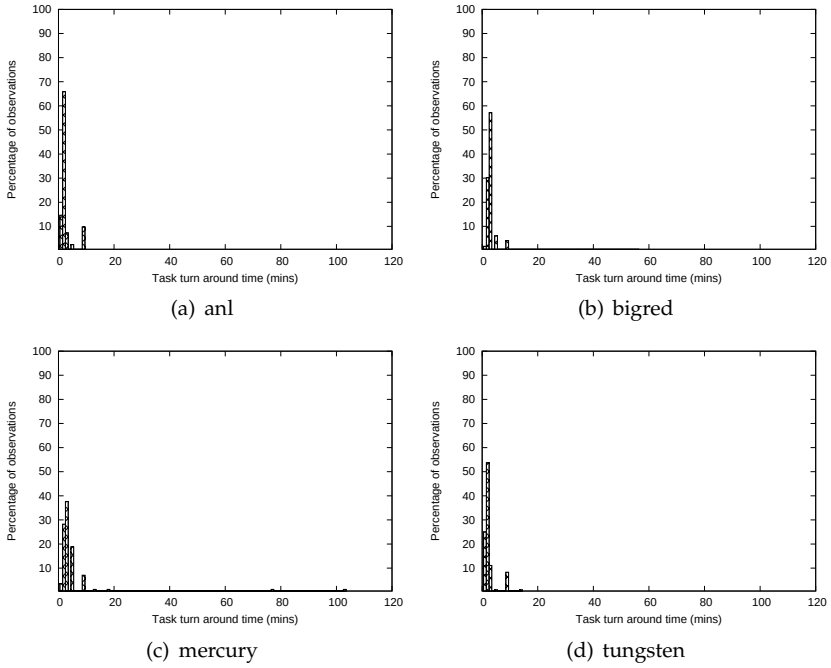
WRF. In this section, we discuss the execution times observed for these tasks.

Terrain Preprocessor. The Terrain Preprocessor task takes order of a few seconds to few minutes on all the machines. The perturbation is infrequently outside the five minute range(Figurefig:terrain) and primarily see on mercury.

WRF Static. WRF Static typically takes order of two to three minutes for execution. The application experiences some perturbation on mercury but rarely outside the 20 minute range (Figure B.6).

Interpolators. There are three types of interpolator tasks in the LEAD workflows. NAM Initial (Figure B.7) has a typical execution time of two to three minutes and has minimal variation outside the 20 minute mark. NAM Lateral and ADAS exhibit similar behavior (Figures B.8 and B.9).

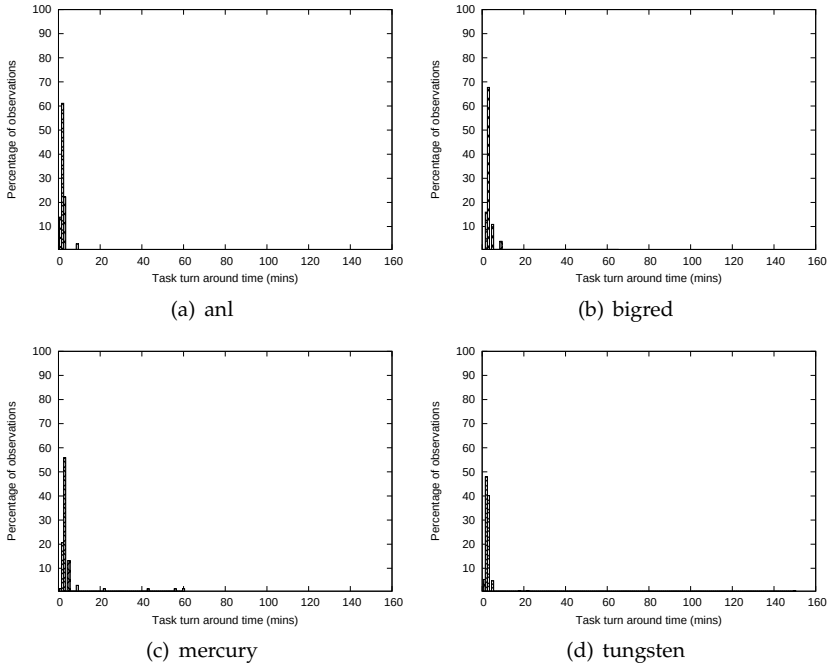
ARPS2WRF. This application takes between two to fives minutes typically to execute on all machines. The application experiences more fluctuations on mercury and tungsten where less than



(a) anl (b) bigred

(c) mercury (d) tungsten

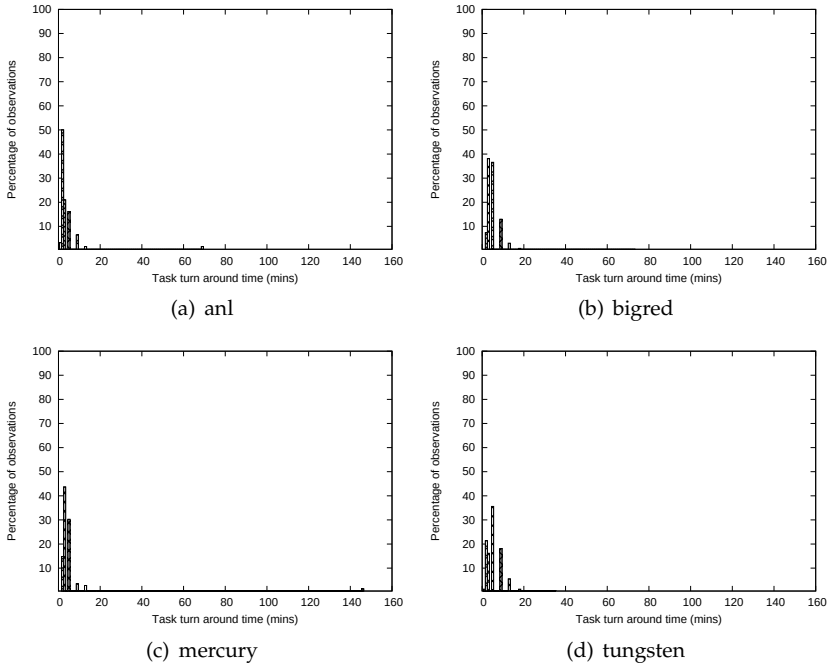
Figure B.6: Wrfstatic.Execution times (a) anl (b) bigred (c) mercury (d) tungsten.



(a) anl (b) bigred

(c) mercury (d) tungsten

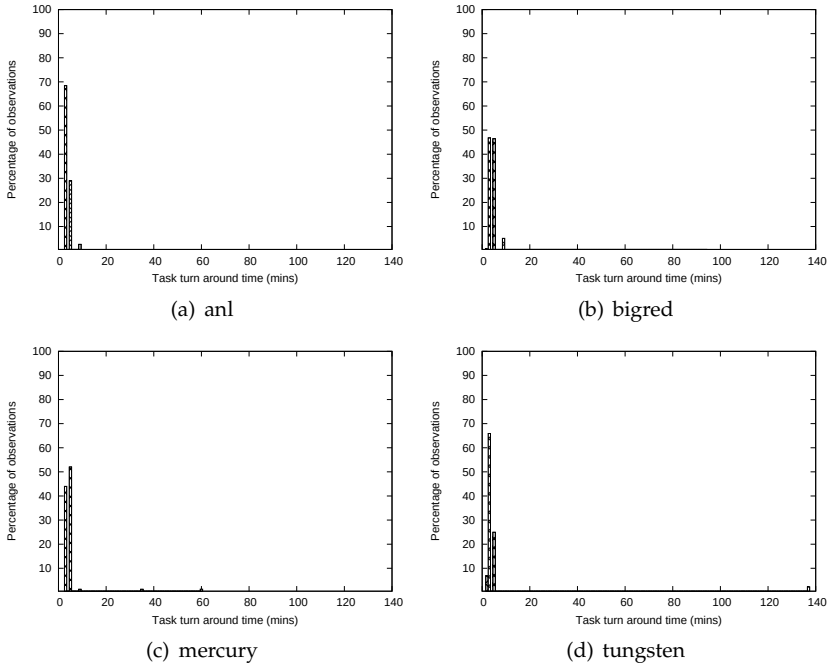
Figure B.7: Nam Initial Execution times (a) anl (b) bigred (c) mercury (d) tungsten.



(a) anl (b) bigred

(c) mercury (d) tungsten

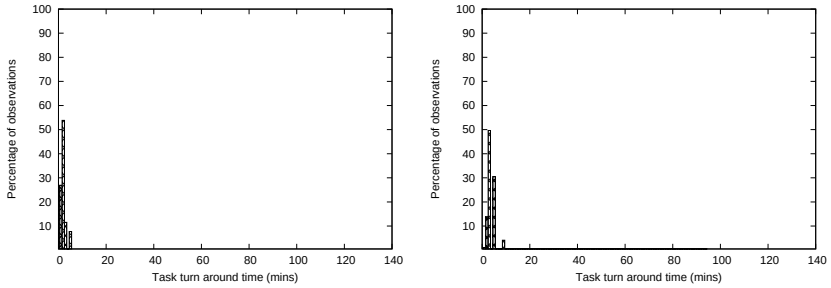
Figure B.8: Nam Lateral Execution times (a) anl (b) bigred (c) mercury (d) tungsten.



(a) anl (b) bigred

(c) mercury (d) tungsten

Figure B.9: ADAS Execution times (a) anl (b) bigred (c) mercury (d) tungsten.



(a) anl (b) bigred

(c) mercury (d) tungsten

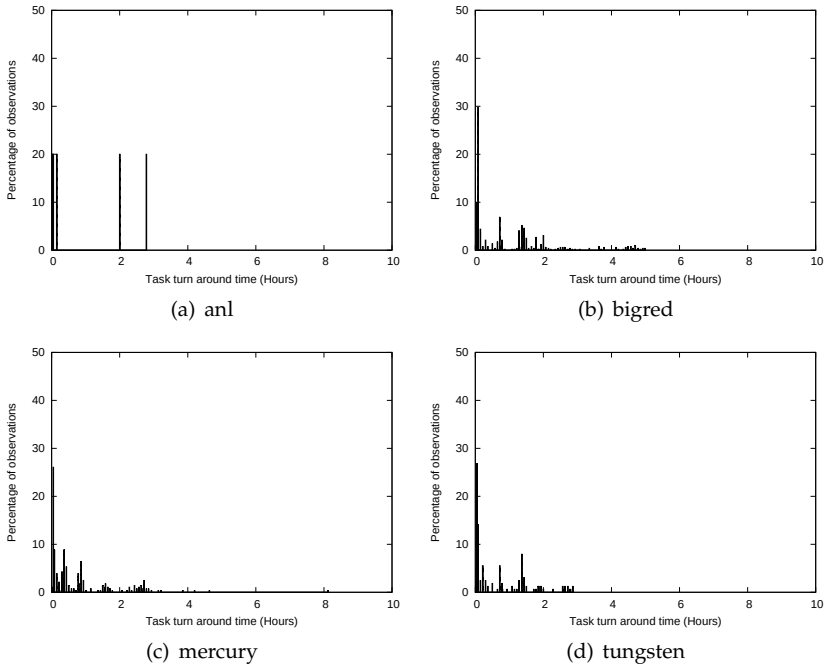
Figure B.10: ARPS2WRF Execution times (a) anl (b) bigred (c) mercury (d) tungsten.

50% of the entries fall within the same minute range.

WRF. WRF which is a MPI application takes order of few minutes to a few hours based on the input data size. Here we see that there is a huge variation in the execution time and less than 30% of the entries fall in the same range.

B.5 Summary

The data from the LEAD system shows that workflows experience failures and performance fluctuation during execution. While the perturbation in execution time for individual applications is in the order of a few minutes and considered minimal, these can severely impact the workflow turnaround time.



(a) anl (b) bigred

(c) mercury (d) tungsten

Figure B.11: WRF Execution times (a) anl (b) bigred (c) mercury (d) tungsten.

Bibliography

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS, An Authorization System for Virtual Organizations. In *Proceedings of the First European Across Grids Conference*, February 2003.

[2] G. Alonso, C. Hagen, D. Agrawal, A. E. Abbadi, and C. Mohan. Enhancing the Fault Tolerance of Workflow Management Systems. *IEEE Concurrency*, 8(3):74–81, 2000.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows, 2004.

[4] S. F. Altschul, W. Gish, E. M. W. Miller, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 214(1-8), 1990.

[5] Amazon Web Services `http://aws.amazon.com/`.

[6] Apache Ant. `http://ant.apache.org/`.

[7] Apache hadoop. `http://hadoop.apache.org/core`.

[8] Apache ODE. `http://ode.apache.org/`.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing.

Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[10] D. Atkins. A Report from the U.S. National Science Foundation Blue Ribbon Panel on Cyberinfrastructure. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 16, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Availability Prediction Service. `http://nws.cs.ucsb.edu/ewiki/nws.php?id=Availability+Prediction+Servic%e`.

[12] Avian Flu Grid Website. `http://avianflugrid.pragma-grid.net/`.

[13] P. Barham, B. Dragovic, K. Faser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[14] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh. SPRUCE: A System for Supporting Urgent High-Performance Computing. In *IFIP WoCo9 Conference Proceedings*, 2006.

[15] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan. New Grid Scheduling and Rescheduling Methods in the GrADS Project. *International Journal of Parallel Programming (IJPP)*, Volume 33(2-3):pp. 209–229, 2005.

[16] F. Berman, G. C. Fox, and anthony J.G. Hey. *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, April 2003.

[17] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive

Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(4):369–382, April 2003.

[18] G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. M. Costa, H. A. Duran-Limon, T. Fitzpatrick, L. Johnston, R. S. Moreira, N. Parlavantzas, and K. B. Saikoski. The Design and Implementation of Open ORB 2. *IEEE Distributed Systems Online*, 2(6), 2001.

[19] B. Blanton, H. Lander, R. A. Luettich, M. Reed, K. Gamiel, and K. Galluppi. Computational Aspects of Storm Surge Simulation. 2008.

[20] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *CCGRID*, pages 759–767, 2005.

[21] J. Brevik, D. Nurmi, and R. Wolski. Predicting Bounds on Queueing Delay for Batch-scheduled Parallel Machines. In *Principles and Practice of Parallel Programming (PPoPP 2006)*. ACM, 2006.

[22] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). Technical Report UCB/EECS-2007-7, EECS Department, University of California, Berkeley, January 11 2007.

[23] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous Concurrent Modeling and Design in Java (Volume 2: Ptolemy II Software Architecture). Technical Report UCB/EECS-2007-8, EECS Department, University of California, Berkeley, January 11 2007.

[24] J. Buisson, F. Andri, and J.-L. Pazat. Dynamic Adaptation for Grid computing. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005 (European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers)*, volume 3470 of *LNCS*, pages 538–547, Amsterdam, June 2005. Springer-Verlag.

[25] R. Buyya, D. Abramson, and J. Giddy. Economy Driven Resource Management Architecture for Computational Power Grids, 2000.

[26] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. *In Proceedings of 4th International Conference on High Performance Computing in ASis-Pacific Region, IEEE Computer Press*, cs.DC/0009021, 2000.

[27] J. H. Byrne. *Mrs. Byrne's Dictionary of Unusual, Obscure, and Preposterous Words: Gathered from Numerous and Diverse Authoritative Sources*. Citadel, 1976.

[28] CaGrid Taverna Workflows. `http://www.cagrid.org/wiki/CaGrid:How-To:Create\_CaGrid\_Workflow\_Usi%ng\_Taverna`.

[29] Cancer Biomedical Informatics Grid (caBIG). `http://www.cagrid.org/`.

[30] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut. Quality of Service for Workflows and Web Service Processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281–308, April 2004.

[31] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. In *Conference on Advanced Information Systems Engineering*, pages 13–31, 2000.

[32] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in Legion. *Future Generation Computer Systems*, 15(5–6):583–594, 1999.

[33] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic Virtual Clusters in a Grid Site Manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on*

High Performance Distributed Computing (HPDC'03), page 90, Washington, DC, USA, 2003. IEEE Computer Society.

[34] A. Chien, H. Casanova, Y.-S. Kee, and R. Huang. The Virtual Grid Description Language: vgDL. UCSD Technical Report CS2005-0817, University of California San Diego, 2005.

[35] A. Chong, A. Sourin, and K. Levinski. Grid-based Computer Animation Rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 39–47, New York, NY, USA, 2006. ACM.

[36] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming Scientific and Distributed Workflow with Triana Services. *Concurrency and Computation: Practice and Experience (Special Issue: Workflow in Grid Systems)*, 18(10):1021–1037, 2006.

[37] Cloudstatus – `http://www.cloudstatus.com/`.

[38] Condor DAGMan. `http://www.cs.wisc.edu/condor/dagman/`.

[39] T. W. Crockett. An Introduction to Parallel Rendering. *Parallel Computing*, 23(7):819–843, 1997.

[40] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based Resource Management, 2005.

[41] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems, 2002.

[42] K. Czajkowski, I. T. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. In *HPDC*, 1999.

[43] C. da Lu and D. A. Reed. Assessing Fault Sensitivity in MPI Applications. *Proceedings of Supercomputing*, 2004.

[44] A. Darling, L. Carey, and W. chun Feng. The Design, Implementation, and Evaluation of mpiBLAST. *4th International Conference on Linux Clusters: The HPC Revolution 2003*, June 2003.

[45] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. pages 137–150.

[46] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow Management in GriPhyN. Grid Resource Management, J. Nabrzyski, J. Schopf, and J. Weglarz editors, Kluwer, 2003.

[47] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1:25–39, 2003.

[48] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing, Vol. 1, No. 1,, 2003.

[49] E. Deelman and Y. Gil. Report from the NSF Workshop on the Challenges of Scientific Workflows. *Workflow Workshop*, 2006.

[50] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The Cost of Doing Science on the Cloud: The Montage Example. In *Proceedings of SC'08*, Austin, TX, 2008. IEEE.

[51] Digital Data Provenance. `http://www.dataandsearch.org/provenance/`.

[52] A. Downey. Predicting Queue Times on Space-Sharing Parallel Computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[53] A. Downey. Using Queue Time Predictions for Processor Allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.

[54] K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda. Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Engg.*, 7(6):12–29, 2005.

[55] R. Duan, R. Prodan, and T. Fahringer. Run-time Optimization for Grid Workflow Applications. In *7th IEEE/ACM International Conference on Grid Computing (Grid'06), IEEE Computer Society Press*, 2006.

[56] L. M. eSolva. Parallel Programming Models and Paradigms. In *High Performance Cluster Computing: Programming and Applications*, 1999.

[57] N. K. et al. Pan-STARRS Collaboration. *American Astronomical Society Meeting*, (206), 2005.

[58] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *Network and Parallel Computing, IFIP International Conference*, pages 2–13, 2005.

[59] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. End-to-end Quality of Service for High-end Applications. *Computer Communications*, 27(14):1375–1388, 2004.

[60] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[61] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[62] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann, November 2003.

[63] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[64] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[65] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation, 2000.

[66] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov. 2008.

[67] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)*, 00:0055, 2001.

[68] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 133–148, New York, NY, USA, 2003. ACM Press.

[69] Geni. `http://www.geni.net/`.

[70] GEON Website. `http://www.geongrid.org`.

[71] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial Intelligence and Grids: Workflow Planning and Beyond. IEEE Intelligent Systems, special issue on e-science, Jan/Feb, 2004.

[72] Google AppEngine. `http://code.google.com/appengine/`.

[73] GridEngine Website. `http://gridengine.sunsource.net/`.

[74] A. S. Grimshaw, W. A. Wulf, and C. T. L. Team. The Legion Vision of a Worldwide Virtual Computer. *Commun. ACM*, 40(1):39–45, 1997.

[75] L. E. Grit. *Extensible Resource Management for Networked Virtual Computing*. PhD thesis, Durham, NC, USA, 2007.

[76] GUR SDSC Co-scheduling system. `http://www.sdsc.edu/us/tools/coschedule.html`.

[77] B. R. Haverkort, R. Marie, G. Rubino, and K. Trivedi. *Performability Modelling*. Wiley, Chichester, England, 2001.

[78] R. Huang, H. Casanova, and A. A. Chien. Using Virtual Grids to Simplify Application Scheduling . In *IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006.

[79] S. Hwang and C. Kesselman. A Flexible Framework for Fault Tolerance in the Grid. *Journal of Grid Computing*, 1:251–272, 2003.

[80] S. Hwang and C. Kesselman. Gridworkflow: A Flexible Failure Handling Framework for the Grid. *HPDC*, 00:126, 2003.

[81] A. Iamnitchi and I. T. Foster. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. In *International Conference on Parallel Processing*, pages 4–14, 2000.

[82] INCA Real Time Monitoring Suite. `http://inca.sdsc.edu/`.

[83] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *USENIX Annual Technical Conference*, pages 199–212, 2006.

[84] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, 2007.

[85] J.L.Vazquez-Poletti, E. Huedo, R. Montero, and I.M.Llorente. Coordinated Harnessing of the IRISGrid and EGEE Testbeds with GridWay. *Journal of Parallel and Distributed Computing*, 66(5):763–771, May 2006.

[86] J.Schopf and F. Berman. Performance Prediction in Production Environments. Proceedings of IPPS/SPDP, 1998.

[87] G. Kandaswamy, A. Mandal, and D. A. Reed. Fault Tolerance and Recovery of Scientific Workflows on Computational Grids. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 777–782, Washington, DC, USA, 2008. IEEE Computer Society.

[88] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving Quality of Service and Quality of Life on the Grid. *Scientific Programming*, 13(4):265–276, 2005.

[89] Y.-S. Kee, C. Kessleman, D. Nurmi, and R. Wolski. Enabling Personal Clusters on Demand for Batch Resources Using Commodity Software. In *International Heterogeneity Computing Workshop (HCW08) in conjunction with IEEE IPDPS08*, April 2008.

[90] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient Resource Description and High Quality Selection for Virtual Grids. In *Proc. of 5th IEEE Symp. on Cluster Comp. and the Grid*, 2005.

[91] Y.-S. Kee, K. Yocum, A. A. Chien, H. Casanova, and H. Casanova. Improving Grid Resource Allocation via Integrated Selection and Binding. In *International Conference on High Performance Computing, Network, Storage*, 2006.

[92] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt,

D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a Framework for Preparing and Executing Adaptive Grid Programs. In *Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL*, April 2002.

[93] O. Khalili, J. He, C. Olschanowsky, A. Snavely, and H. Casanova. Measuring the performance and reliability of production computational grids. In *The 7th IEEE/ACM International Conference on Grid Computing*, 2006.

[94] J. Klingemann, J. Wasch, and K. Aberer. Deriving Service Models in Cross-Organizational Workflows. In *RIDE*, pages 100–107, 1999.

[95] P. Koksal, I. Cingil, and A. Dogac. A Component-Based Workflow System with Dynamic Modifications. In *Next Generation Information Technologies and Systems*, pages 238–255, 1999.

[96] F. Kon, F. Costa, G. Blair, and R. H. Campbell. The Case for Reflective Middleware. *Commun. ACM*, 45(6):33–38, 2002.

[97] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, C. Magalhaes, and R. H. Campbell. Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 121–143, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[98] W. Kramer and C. Ryan. Performance Variability of Highly Parallel Architectures. International Conference on Computational Science, 2003.

[99] H. M. Lander, R. J. Fowler, L. Ramakrishnan, and S. R. Thorpe. Stateful Grid Resource Selection for Related Asynchronous Tasks. Technical Report TR-08-02, RENCI, North Carolina, April 2008.

[100] Large Hydron Collider. `http://lhc.web.cern.ch/`.

[101] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[102] X. Li, B. Plale, N. Vijayakumar, R. Ramachandran, S. Graves, and H. Conover. Real-time Storm Detection and Weather Forecast Activation through Data Mining and Events Processing. *Earth Science Informatics*, May 2008.

[103] M. Litzkow and M. Livny. Experience with the Condor Distributed Batch System. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, October 1990.

[104] C. Liu, M. E. Orlowska, and H. Li. Automating Handover in Dynamic Workflow Environments. In *CAiSE '98: Proceedings of the 10th International Conference on Advanced Information Systems Engineering*, pages 159–171, London, UK, 1998. Springer-Verlag.

[105] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *In Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[106] Los Almos Reliability Data. `http://institutes.lanl.gov/data/fdata/`.

[107] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System, 2005.

[108] R. Luettich, J. J. Westerink, and N. W.Scheffner. ADCIRC: An Advanced Three-dimensional Circulation Model for Shelves, Coasts and Estuaries; Report 1: Theory and Methodology of ADCIRC- 2DDI and ADCIRC-3DL. *Technical Report DRP-92-6, Coastal Engineering Research Center, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS*, 1992.

[109] V. Lynch, J. Cobb, E. Farhi, S. Miller, and M. Taylor. Virtual Experiments on the Neutron Science TeraGrid Gateway. *TeraGrid*, 2008.

[110] R. Maia, R. Cerqueira, and F. Kon. A Middleware for Experimentation on Dynamic Adaptation. In *ACM/IFIP/USENIX 3rd International Workshop on Adaptive and Reflective Middleware*, Grenoble, France, November 2005.

[111] G. Malewicz. Parallel Scheduling of Complex DAGs Under Uncertainty. In *Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms(SPAA)*, pages 66–75, 2005.

[112] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling Strategies for Mapping Application Workflows onto the Grid. In *High Performance Distributed Computing (HPDC 2005).*, pages 125–134. IEEE Computer Society Press, 2005.

[113] P. J. Mangan and S. W. Sadiq. A Constraint Specification Aproach to Building Flexible Workflows. *Journal of Research and Practice in Information Technology*, 35(1):21–39, 2003.

[114] S. Marru, S. Perera, M. Feller, and S. Martin. Reliable and Scalable Job Submission: LEAD Science Gateways Testing and Experiences with WS GRAM on TeraGrid Resources . *TeraGrid Conference*, June 2008.

[115] Maui Scheduler Website. `http://www.clusterresources.com/products/maui/`.

[116] J. F. Meyer. On Evaluating the Performability of Degradable Computing Systems. *IEEE Trans. Computers*, 29(8):720–731, 1980.

[117] J. Michalakes, J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. The Weather Reseach and Forecast Model: Software Architecture and Performance. *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, October 2004.

[118] Microsoft Azure. `http://www.microsoft.com/azure/`.

[119] J. W. E. Nabrzyski, J.M. Schopf. *Grid Resource Management*. Kluwer Publishing, 2003.

[120] NEES Cyberinfrastructure Center Website. `http://it.nees.org`.

[121] W. M. v. d. A. Nick Russell Arthur H.M.ter Hofstede and N. Mulyar. Workflow Management Coalition Terminology Glossary, 1999.

[122] Nirvanix. `http:///www.nirvanix.com`.

[123] North Carolina Floodplain Mapping Program. http://www.ncfloodmaps.com/.

[124] D. Nurmi, J. Brevik, and R. Wolski. Minimizing the Network Overhead of Checkpointing in Cycle-harvesting Cluster Environments. In *Proceedings of Cluster 2005*, 2005.

[125] D. Nurmi, J. Brevik, and R. Wolski. QBETS: Queue Bounds Estimation from Time Series. In *Proceedings of 13th Workshop on Job Scheduling Strategies for Parallel Processing (with ICS07)*, June 2007.

[126] D. Nurmi, J. Brevik, and R. Wolski. VARQ: Virtual Advance Reservations for Queues. *Proceedings 17th IEEE Symp. on High Performance Distributed Computing (HDPC)*, 2008.

[127] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction. In *Proceedings of SC'06*, Tampa, FL, 2006. IEEE.

[128] D. Nurmi, R. Wolski, and J. Brevik. Model-Based Checkpoint Scheduling for Volatile Resource Environments. Technical Report 2004-25, University of California Santa Barbara, Department of Computer Science, Santa Barbara, CA, 93106, 2004.

[129] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus:A Technical Report on an Elastic Utility Computing Archietcture Linking Your

Programs to Useful Systems. Technical Report 2008-10, University of California, Santa Barbara, California, August 2008.

[130] OASIS Web Services Resource Framework Technical Committee. *Web Services Resource Framework (WSRF) v1.2 Specification*, April 2006.

[131] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1067–1100, 2006.

[132] Open Science Grid Website. `http://www.opensciencegrid.org/`.

[133] PBSPro Website. `http://www.altair.com/software/pbspro.htm`.

[134] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. In *Proceedings of the Third IEEE International Workshop on Policies for Distributed Systems and Networks*, June 2002.

[135] B. Plale, D. Gannon, J. Brotzge, K. K. Droegemeier, J. Kurose, D. McLaughlin, R. wilhelmson, S. Graves, M. Ramamurthy, R. D. Clark, S. Yalda, D. A. Reed, E. Joseph, and V. Chandrashekar. CASA and LEAD: Adaptive Cyberinfrastructure for Real-time Multiscale Weather Forecasting. *IEEE Computer*, (39):66–74, 2006.

[136] Planetlab. `http://www.planet-lab.org/`.

[137] G. Radke and J. Evanoff. A Fast Recursive Algorithm to Compute the Probability of M-out-of-N events. In *Reliability and Maintainability Symposium, 1994. Proceedings., Annual*, 1994.

[138] L. Ramakrishnan, B. O. Blanton, H. M. Lander, R. A. Luettich, Jr, D. A. Reed, and S. R. Thorpe.

Real-time Storm Surge Ensemble Modeling in a Grid Environment. In *Second International Workshop on Grid Computing Environments (GCE), Held in conjunction ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis*, November 2006.

[139] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase. Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Computing, Networking, Storage and Analysis*, November 2006.

[140] L. Ramakrishnan and D. A. Reed. Monitoring and orchestrating a mesoscale meteorological cyberinfrastructure. In *21st International Conference on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, 85th AMS Annual Meeting*, January 2007.

[141] L. Ramakrishnan and D. A. Reed. Performability Modeling for Scheduling and Fault Tolerance Strategies for Grid Workflows. In *ACM/IEEE International Symposium on High Performance Distributed Computing*, 2008.

[142] L. Ramakrishnan, M. S. Reed, J. L. Tilson, and D. A. Reed. Grid Portals for Bioinformatics. In *Second International Workshop on Grid Computing Environments (GCE), Held in conjunction with ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis*, November 2006.

[143] L. Ramakrishnan, Y. Simmhan, and B. Plale. Realization of Dynamically Adaptive Weather Analysis and Forecasting in LEAD. In *In Dynamic Data Driven Applications Systems Workshop (DDDAS) in conjunction with ICCS (Invited)*, 2007.

[144] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[145] R. Raman, M. Livny, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching, 2003.

[146] D. A. Reed, C. da Lu, and C. L. Mendes. Reliability Challenges in Large Systems. *Future Generation Computer Systems*, 22(3):293–302, 2006.

[147] D. A. Reed and C. L. Mendes. Intelligent Monitoring for Adaptation in Grid Applications. *Proceedings of the IEEE*, Volume 93(2):426– 435, 2005.

[148] M. Reichert and P. Dadam. ADEPT flex -Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[149] RENCI Computational Resources. `http://www.renci.org/resources/computing.php`.

[150] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive Control of Distributed Applications. In *HPDC*, pages 172–179, 1998.

[151] L. D. Rose and D. A. Reed. SvPablo: A Multi-Language Architecture-Independent Performance Analysis System. In *International Conference on Parallel Processing*, pages 311–318, 1999.

[152] N. Russell, A. H. ter Hofstede, D. Edmond, and W. M. van der Aalst. Workflow Exception Patterns. *In E. Dubois and K. Pohl, editors, Proceedings of the 18th Conference on Advanced Information Systems Engineering (CAiSE'06)*, 4001:216–232, 2006.

[153] N. Russell, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. *In O. Pastor and J. Falcao e Cunha, editors, Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, 3520:216–232, 2005.

[154] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, 1996.

[155] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos. Scheduling Workflows with Budget Constraints. In S. Gorlatch and M. Danelutto, editors, *Integrated Research in GRID Computing*, CoreGRID, pages 189–202. Springer-Verlag, 2007.

[156] Salesforce. `http://www.salesforce.com/`.

[157] V. Sander, W. Adamson, I. Foster, and R. Alain. End-to-End Provision of Policy Information for Network QoS, 2001.

[158] B. Schroeder and G. Gibson. A Large-scale Study of Failures in High-performance Computing Systems. In *Proceedings of the International Conference on Dependable Systems*, June 2006.

[159] SCOOP Website. `http://scoop.sura.org`.

[160] SDSC User Settable Reservations. `http://portal.sdsc.edu/sdsc?cid=reservations`.

[161] S. Shirasuna. *A Dynamic Scientific Workflow System for Web Services Architecture*. PhD thesis, Indiana University, Department of Computer Science, September 2007.

[162] M. Silberstein, D. Geiger, A. Schuster, and M. Livny. Scheduling Mixed Workloads in Multigrids: The Grid Execution Hierarchy. In *Proceedings of the 15th IEEE Symposium on High Performance Distributed Computing (HPDC-15)*, Paris, France, June 2006.

[163] G. Singh, C. Kesselman, and E. Deelman. Application-level Resource Provisioning on the Grid. In *Proceedings of 2nd IEEE Intl Conference on e-Science and Grid Computing*, Amsterdam, 2006. IEEE.

[164] A. Slominski. Adapting BPEL to Scientific Workflows. In I. J. Taylor, E. Deelman, D. Gannon, and M. S. Shields, editors, *Workflows for e-Sciences: Scientific Workflows for Grids*, pages 210–228. Springer, To appear 2007.

[165] A. Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.

[166] W. Smith, I. Foster, and V. Taylor. Scheduling with Advanced Reservations. In *Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 127–132.

[167] W. Smith, I. Foster, and V. Taylor. Scheduling with Advanced Reservations. In *International Parallel and Distributed Processing Symposium*, pages 127–132, 2000.

[168] W. Smith, V. Taylor, and I. Foster. Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer Verlag, 1999.

[169] Q. Snell, M. Clement, D. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-Scheduling. In *6th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153, 2000.

[170] Sonian. `http://www.sonian.net/`.

[171] Y. Sun, S. Marru, and B. Plale. Experience with Bursty Workflow-driven Workloads in LEAD Science Gateway. *TeraGrid Conference*, June 2008.

[172] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Proceedings of the Third Virtual Machine Research and Technology Symposium (VM)*, May 2004.

[173] N. Taesombut and A. Chien. Distributed Virtual Computers (DVC): Simplifying the Development of High Performance Grid Applications. In *Proceedings of the Workshop on Grids and Advanced Networks*, April 2004.

[174] I. Taylor, M. Shields, and I. Wang. Resource Management for the Triana Peer-to-Peer Services. In J. Nabrzyski, J. M. Schopf, and J. Węglarz, editors, *Grid Resource Management*, pages 451–462. Kluwer Academic Publishers, 2004.

[175] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, December 2006.

[176] TeraGrid Service Units. `http://kb.iu.edu/data/ascf.html`.

[177] TeraGrid Data Allocation. `http://datacentral.sdsc.edu/how\_to\_apply.html`.

[178] TeraGrid Service Units. `http://www.teragrid.org/cgi-bin/kb.cgi?docid=ascf`.

[179] TeraGrid Website. `http://www.teragrid.org`.

[180] Test TCP. `http://www.pcausa.com/Utilities/pcattcp.htm`.

[181] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[182] N. Thomas. Challenges and Opportunities in Grid Performability. Technical Report Series CS-TR-842, University of NewCastle upon Tyne.

[183] N. Thomas. Report on First Workshop on Grid Performability Modelling and Measurement. `http://www.nesc.ac.uk/talks/379/report.pdf`.

[184] J. Tilson, A. Blatecky, G. Rendon, G. Mao-Feng, and E. Jakobsson. Genome-Wide Domain Analysis using Grid-enabled Flows. 2007.

[185] J. Tilson, G. Rendon, G. Mao-Feng, and E. Jakobsson. MotifNetwork: A Grid-enabled Workflow for High-throughput Domain Analysis of Biological Sequences:Implications for Study of Phylogeny, Protein Interactions, and Intraspecies Variation. 2007.

[186] J. L. Tilson, M. S. Reed, and R. J. Fowler. Workflow for Performance Evaluation and Tuning. *IEEE Cluster*, 2008.

[187] Torque Website. `http://www.clusterresources.com/pages/products/torque-resource-manager.%php`.

[188] N. B. Tracy D. Braun, Howard Jay Siegel. A Comparision of Eleven Static Heuristics for Maping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. J. of Parallel and Distributed Computing 61, 810-837., 2001.

[189] UDDI Spec Technical Committee. *UDDI Specification Version 3.0.1.*, October 2003.

[190] Unidata Local Data Manager (LDM). `http://www.unidata.ucar.edu/software/ldm/`.

[191] S. Vadhiyar and J. Dongarra. A Metascheduler for the Grid. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[192] W. M. P. van der Aalst, Ter, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.

[193] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(6):685–699, 2006.

[194] Vertica. `http://www.vertica.com/cloud`.

[195] J. S. Vetter and D. A. Reed. Real-Time Performance Monitoring, Adaptive Control, and Interactive Steering of Computational Grids. *IJHPCA*, 14(4):357–366, Winter 2000.

[196] I. Wang, I. Taylor, T. Goodale, A. Harrison, and M. Shields. gridMonSteer: Generic Architecture for Monitoring and Steering Legacy Applications in Grid Environments. In S. J. Cox, editor, *Proceedings of the UK e-Science All Hands Meeting 2006*. EPSRC, CD Rom Proceedings, 2006.

[197] J. B. Weissman. Fault Tolerant Computing on the Grid: What are My Options? In *HPDC*, 1999.

[198] J. B. Weissman, S. Kim, and D. England. A Framework for Dynamic Service Adaptation in the Grid: Next Generation Software Program Progress Report. In *19th International Parallel and Distributed Processing Symposium*, 2005.

[199] D. N. Wfmc-Tc. Workflow Management Coalition Terminology Glossary, 1999.

[200] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing Market-Based Resource Allocation Strategies for the Computational Grid. *The International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.

[201] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[202] WS-BPEL Technical Committee. *Web Services Business Process Execution Language Version 2.0, Public review draft*, November 2006.

[203] J. Yu and R. Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Rec.*, 34(3):44–49, September 2005.

[204] J. Yu and R. Buyya. Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms. *Scientific Programming*, 14(3-4):217–230, 2006.

[205] Y. Zhang, A. Mandal, H.Casanova, A. Chien, Y. Kee, K. Kennedy, and C. Koelbel. Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling. CCGrid, May 2006.

[206] Y. Zhou, T. Kelly, J. L. Wiener, and E. Anderson. An Extended Evaluation of Two-Phase Scheduling Methods for Animation Rendering. In *Job Scheduling Strategies for Parallel Processing, 11th International Workshop*, pages 123–145, 2005.

Curriculum Vitae

Lavanya Ramakrishnan

Education

M.S. Computer Science, Indiana University, 2002

B.E. Computer Engineering, VJTI, University of Mumbai, India, 2000

Professional Experience

Senior Research Programmer (2004-2006)
Renaissance Computing Institute, Chapel Hill, NC

Served as techincal lead on a number of national inter-disciplinary collaborations. Responsibilities included providing coordination and leadership for the research and development of open-source distributed solutions (e.g., Grid) to serve the needs of application scientists. Key areas of research included portal solutions, resource management, performance and reliability monitoring, and workflow orchestration. Other responsibilities included project management, assisting in the development of technical strategies for the institute and hiring of personnel, proposal development, mentoring graduate students, and representing the team at key national meetings.

Research Engineer (2002-2004)
MCNC, Research Triangle Park, NC

Developed middleware and security architectures for grid based systems including the North Carolina BioGrid. Integrated various middleware solutions for access to grid resources. Worked on workflow audit representation for an insider detection system.

Givens Associate (Summer 2002)
Argonne National Laboratory,Argonne, IL

Developed a graphical interface for composing and launching applications in Common Component Architecture (CCA) frameworks. Developed prototype XML schema for standardization of component metadata to facilitate portability between frameworks.

Academic Experience

Research Assistant (2007-Present)
Indiana University, Bloomington, IN

Developing an architecture and associated techniques for workflow adaptation required in the context of meteorological workflows for the Linked Environments for Atmospheric Discovery (LEAD) project in distributed environments (e.g., Grid, cloud). Specifically this architecture enables proactive planning and adaptation across the multi-layered web service architecture to balance the performance and reliability needs of the application.

Research Assistant (2001-2002)
Indiana University, Bloomington,IN

Worked on developing an authorization framework for a component based distributed environment, XCAT. Component specification was based on the Common Component Architecture (CCA) Forum. Worked on designing the security architecture for the Application Factory, that helps solve the problem of building reliable, scalable grid applications, by separating the process of application deployment from execution.

Teaching Assistant (2000-2001)
Indiana University, Bloomington,IN

Teaching, grading and designing lab activities for "Introduction to Computing" course in the Department of Computer Science.

Honors, Awards and Grants

1. Co-Principal Investigator (PI: Jeffrey S. Chase, Duke University), A Grid Service for Dynamic Virtual Clusters, National Science Foundation Middleware Initiative [2003-2007].
2. Indiana University School of Informatics Diversity Committee Scholarship for Grace Hopper [2007].
3. Anita Borg Institute's Grace Hopper and Tapia Bridge Day Scholarship [2007].
4. Google Anita Borg Scholarship Finalist [2007].
5. Best employee award, MCNC, NC [2004].
6. Givens Associate Fellowship, Argonne National Laboratory, IL [2002].
7. J.N. Tata Scholarship for higher studies abroad [2000].
8. Maharashtra State Fellowship for undergraduate studies [1996-1999].

Activities

1. Technical/Program Committee: Grid Computing Environments Workshop (GCE) [2007,2008].
2. Volunteer:
   - Anita Borg Institute for Women and Technology's Women of Vision Gala Awards Dinner [2007, 2008].
   - Girls for a Change Summit [2007].
3. Reviewer:
   - TeraGrid Conference [2007,2008]
   - International Conference on Distributed Computing Systems (ICDCS) [2007]
   - IEEE Internet Computing (Special Issue: Virtual Organizations) [2007]
   - ACM Symposium for Applied Computing [2005]
4. Participated in Global Grid Forum [2002-2004].
5. Technical Evaluation Committee, MCNC- Research and Development Institute [2003].
6. Student Volunteer at ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis [2001].

Selected Publications

1. L. Ramakrishnan, D. Nurmi, A. Mandal, C. Koelbel, D. Gannon, T. M. Huang, Y.S. Kee, G. Obertelli, K. Thyagaraja, R. Wolski, A. Yarkhan, D. Zagorodnov, VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance, To appear *Proceedings of the ACM/IEEE SC2009 Conference on High Performance Computing, Networking, Storage and Analysis*, 2009.
2. L. Ramakrishnan and D. A. Reed, Predictable quality of service atop degradable distributed systems, in *Journal of Cluster Computing*, 2009.
3. R. J. Fowler, T. Gamblin, G. Kandaswamy, A. Mandal, A. K. Porterfield, L. Ramakrishnan, and D. A. Reed, *High Performance Computing and Grids in Action*, ch. Challenges of Scale: When All Computing Becomes Grid Computing. IOS Press, 2008.
4. L. Ramakrishnan and D. A. Reed, PerformabilityModeling for Scheduling and Fault Tolerance Strategies for Scientific workflows, in *ACM/IEEE International Symposium on High Performance Distributed Computing*, 2008.

5. L. Ramakrishnan, Y. Simmhan, and B. Plale, Realization of Dynamically Adaptive Weather Analysis and Forecasting in LEAD, in *Dynamic Data Driven Applications Systems Workshop (DDDAS) in conjunction with ICCS (Invited)*, 2007.

6. L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase, Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control, in *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Computing, Networking, Storage and Analysis*, (Tampa, Florida), November 2006.

7. L. Ramakrishnan, B. O. Blanton, H. M. Lander, R. A. Luettich, Jr, D. A. Reed, and S. R. Thorpe, Real-time Storm Surge Ensemble Modeling in a Grid Environment, in *Second International Workshop on Grid Computing Environments (GCE), Held in conjunction ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis*, November 2006.

8. L. Ramakrishnan, M. S. Reed, J. L. Tilson, and D. A. Reed, Grid Portals for Bioinformatics, in *Second International Workshop on Grid Computing Environments (GCE), Held in conjunction with ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis*, November 2006.

9. K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda, Service- Oriented Environments for Dynamically Interacting with Mesoscale Weather, *Computing in Science and Engg.*, vol. 7, no. 6, pp. 12-29, 2005.

10. L. Ramakrishnan, Securing Next-Generation Grids, *IT Professional.*, vol. 6, no. 2, pp. 34-29 2004.

11. D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski, *Grid Computing: Making the Global Infrastructure a Reality*, ch. 9, Grid Web Services and Application Factories. Wiley, 2003.

12. T. J. Smith and L. Ramakrishnan, Joint Policy Management and Auditing in Virtual Organizations., in *GRID*, pp. 117-124, 2003.

13. L. Ramakrishnan, H. N. Rehn, J. Alameda, R. Ananthakrishnan, M. Govindaraju, A. Slominski, K. Connelly, V. Welch, D. Gannon, R. Bramley, and S. Hampton, An Authorization Framework for a Grid Based Common Component Architecture, in *Proceedings of the 3rd International Workshop on Grid Computing*, Baltimore,Maryland, pp. 169-180, Springer Press, 2002.

14. D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan,

F. Bertrand, K. Chiu, M. Farrellee,M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz, Programming the Grid: Distributed Software components, P2P and GridWeb Services for Scientific Applications, *Journal of Cluster Computing*, vol. 5, pp. 325-336, July 2002.