2011

Integrating multiple clusters for compute-intensive applications

Zhifeng Yun
Louisiana State University and Agricultural and Mechanical College

INTEGRATING MULTIPLE CLUSTERS FOR COMPUTE-INTENSIVE APPLICATIONS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Zhifeng Yun
B.S., Huazhong University of Science and Technology, China, 2004
M.Sc., University of Southampton, U.K. 2006
August 2011

Acknowledgments

This work would not have been possible without the help and support of my current and former colleagues at the Center for Computation & Technology and the Department of Electrical & Computer Engineering at Louisiana State University.

I am very grateful to my advisory committee: Dr. Jagannathan Ramanujam, Dr. Daniel S. Katz, and Dr. Gabrielle Allen, for all the support and guidance throughout my studies and preparation of this dissertation. I would like to thank Dr. Jianhua Chen, Dr. R. Vaidyanathan, and Dr. John DiTusa for being a part of my committee.

Special thanks to Dr. Lei Zhou who inspired much of this work by proposing challenging problems and contributing to the ideas, implementations, and publications making up this dissertation, and to Dr. Sun Joseph Chang for the kindly support and constant encouragement throughout this journey.

I would like to thank Dr. Shantenu Jha for his advice and suggestion of the ideas and publications, Xin Li during the collaboration of the EnKF simulation and Maoyuan Xie for the support of the implementation of cluster abstraction. I also would like to thank Matthew Woitaszek, Michael Oberg and Ben Mayer for their warmly help during my stay in NCAR and great support for my research work on system administration, networking and virtual machine setups.

Finally, I would like to thank my wife Ai Geng and my parents for their love, support, and understanding during this journey. I would also like to thank all my friends here who made my stay at LSU an enjoyable and a memorable one.

This work was in part sponsored by the NSF CyberTools (NSF award #EPS-0701491) projects, the NCAR's NSF research grant for the SIParCS program, the U.S. Department of Energy (DOE) under Award Number DE-FG02-04ER46136 and the Board of Regents, State of Louisiana, under Contract No. DOE/LEQSF(2004-07).

Portions of this research were conducted with high performance computational resources provided by the Louisiana Optical Network Initiative (http://www.loni.org/).

Table of Contents

List of Tables

List of Figures

Abstract

Multicluster grids provide one promising solution to satisfying the growing computational demands of compute-intensive applications. However, it is challenging to seamlessly integrate all participating clusters in different domains into a single virtual computational platform. In order to fully utilize the capabilities of multicluster grids, computer scientists need to deal with the issue of joining together participating autonomic systems practically and efficiently to execute grid-enabled applications.

Driven by several compute-intensive applications, this theses develops a multicluster grid management toolkit called **Pelecanus** to bridge the gap between user's needs and the system's heterogeneity. Application scientists will be able to conduct very large-scale execution across multiclusters with transparent QoS assurance. A novel model called DA-TC (Dynamic Assignment with Task Containers) is developed and is integrated into **Pelecanus**. This model uses the concept of a task container that allows one to decouple resource allocation from resource binding. It employs static load balancing for task container distribution and dynamic load balancing for task assignment. The slowest resources become useful rather than be bottlenecks in this manner. A cluster abstraction is implemented, which not only provides various cluster information for the DA-TC execution model, but also can be used as a standalone toolkit to monitor and evaluate the clusters' functionality and performance.

The performance of the proposed DA-TC model is evaluated both theoretically and experimentally. Results demonstrate the importance of reducing queuing time in decreasing the total turnaround time for an application. Experiments were conducted to understand the performance of various aspects of the DA-TC model. Experiments showed that our model could significantly reduce turnaround time and increase resource utilization for our targeted application scenarios. Four applications are implemented as case studies to determine the applicability of the DA-TC model.

In each case the turnaround time is greatly reduced, which demonstrates that the DA-TC model is efficient for assisting application scientists in conducting their research.

In addition, virtual resources were integrated into the DA-TC model for application execution. Experiments show that the execution model proposed in this thesis can work seamlessly with multiple hybrid grid/cloud resources to achieve reduced turnaround time.

Chapter 1
Introduction

Cluster-based computing has been dominantly adopted by compute-intensive applications, and myriad investments in cluster systems have been made by institutions and governments around the world. However, individual clusters still have rather limited capacity and cannot meet the growing demands of many large-scale scientific and engineering applications. Although teraflop or even petaflop systems are available nowadays, these large parallel systems are very expensive, and in general not dedicated, but shared by a large number of applications designed by many different users. Also, it has been shown that parallel applications that have been written for homogeneous single cluster systems do not run efficiently on multicluster systems [vNMB+00]. Thus, scientists have been pursuing approaches to sharing the workload of individual applications across multiple existing networked clusters so as to achieve a substantial increase in computational capability and furthermore to improve resource utilization without any additional investment.

There is now considerable literature on supporting research into multicluster systems [Aum02, BBE03, BAN00, CFM03, HJS+04, HJSN04]. Based on differences in their network connection, multicluster environments can be classified into two categories: super-cluster systems and multicluster grids [AD03]. The former refers to a system in which the participating clusters are homogeneous, centrally administered and connected by a dedicated high-speed network. Such an architecture provides better predictability and reliability, but it has a dedicated and fixed setting, which seriously limits its scalability. In addition, a super-cluster system can be prohibitively expensive. The latter (i.e., multicluster grid) is a grid environment in which the computational resources are multiple clusters linked by the Internet. These clusters are typically self-administered, and may be heterogeneous and globally distributed. Since each participating resource is shared by multiple users, multiclusters can reuse the existing cluster infrastructure to achieve a substantial capacity

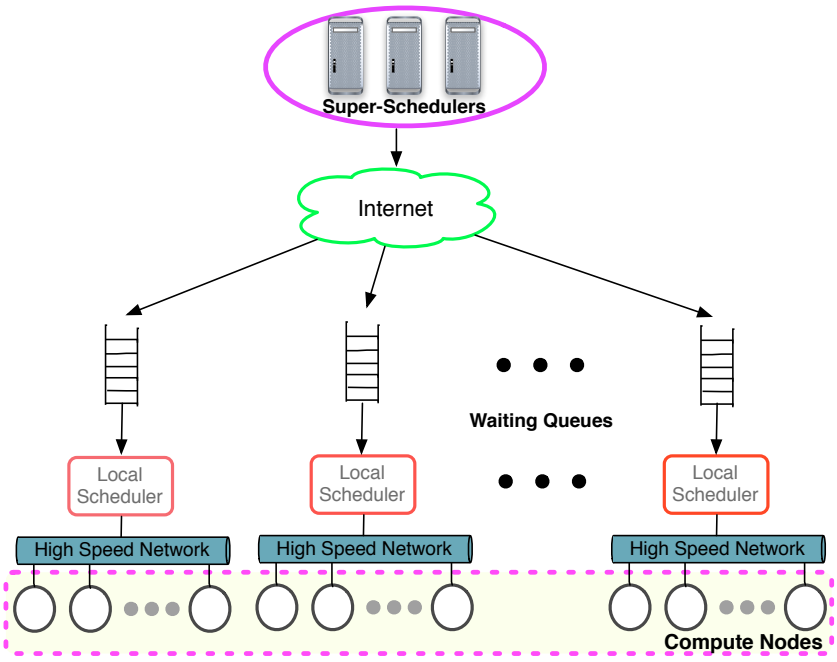


FIGURE 1.1: The multicluster structure.

increase. In addition, multicluster grids have an open architecture and can be easily extended to meet new demands as they arise. However, as a particular case of grid computing, multicluster grids face the same challenges as any other grid environment, including security, performance prediction, reliability and metascheduling.

The structure of a multicluster environment is shown in Figure 1.1. There is a super-scheduler over multiple clusters that assigns application tasks onto participating clusters for execution. The assigned tasks are viewed as normal jobs by the local scheduling systems on the participating clusters. The local scheduling system arranges actual job executions under its own scheduling policies. However, it is challenging to efficiently manage application execution across a grid, due to the nature of participating clusters and network connection [FKT01]. In general, the participating clusters are geographically distributed, heterogeneous and independently administered. The network connection provided by the Internet has security vulnerabilities and generally unpredictable performance. Because the completion of a submitted application depends on the completion of all the application tasks assigned to different clusters, the cluster that completes its tasks latest is the

bottleneck in the application's execution. Application execution also has to take into account the risk of any system failure in the participating clusters.

Several compute-intensive applications motivate us to conduct research on efficient system management for multiclusters. We have worked closely with research groups in various disciplines, such as renewable natural resources, petroleum engineering, mechanical engineering, and computational chemistry. All the applications from these groups are compute-intensive, and these research groups have already invested in clusters as computational platforms. While super-cluster systems are not always available and/or affordable, these groups eagerly expect to construct multiclusters to provide sufficient computational capabilities to advance their research and leverage their existing cluster investments. However, under current circumstances, in order to leverage multiple clusters' computational capability for a large-scale compute-intensive application, a scientist has to perform several tasks: (i) organize the execution scenario for each application, (ii) check the cluster systems' information through diverse information services provided by each cluster, (iii) pass through different security mechanisms, and (iv) then submit tasks on different cluster systems. All these steps are executed manually without QoS (quality-of-service) assurance. In addition, it is challenging to efficiently manage the application components and change the number of executables once they are submitted to grid resources. This situation suffers from a lack of dynamic load balancing, resulting in poor execution performance scalability. It is also very difficult to handle applications that have malleable or recursive characteristics, where problem scale cannot be known until application execution runtime. Inverse modeling is a typical example that requires dynamic handling; it has been extensively adopted by various application domains such as coastal modeling, weather prediction and subsurface modeling. The process of inverse modeling consists of multiple iterations. The number of iterations and each iteration's simulations can only be determined by dynamically assimilated data *during* the inverse modeling.

In order to improve application execution with QoS, and furthermore enhance resource utilization across a multicluster environment, in dissertation, we have developed a dynamic load assign-

ment execution model (DA-TC) and a corresponding service called the Grid Execution Management Service (GEMS) to support such application scenarios. Using this novel application execution model, we are able to improve resource interoperability and enhance application execution and monitoring in multicluster environments. Experiments show that this can significantly reduce turnaround time and increase resource utilization for certain applications such as collaborative mechanical design [YXZ+08], reservoir uncertainty analysis [LAC+08], subsurface inverse modeling [LLW+07], sawing optimization [YCL+08], the Daymet application acceleration [Yun10], and water nucleation simulation [YKX+07]. Based on this model, we have developed a toolkit named **Pelecanus** to efficiently manage the resources and to provide high-level services to application scientists.

Recently, the use of cloud computing technology for application execution has become increasingly popular. By using the virtualization and on-demand virtual machines in the cloud environment, users can start executing their jobs immediately without any waiting time. Besides this, it also provides users the ability to configure the virtual machines beforehand with the necessary software and hardware support for the application execution, so that the virtual machines can be used for job execution without any further software installation and configuration after booting. In this dissertation, we present approaches to integrate cloud resources into our DA-TC model to take advantages of this new technology for application execution.

We aim to provide high-level services and tools that will enable application scientists to concentrate on their primary research and significantly improve the utilization of existing cluster systems in terms of reliability, predictability, confidentiality, and usability. Users will benefit from our research on application execution, resource applicability and utilization, and system management. An application scientist will be able to conduct very large-scale execution across multiclusters with transparent QoS assurance. Small resources that are part of the multicluster environments will be able to make contribution to the whole large application execution, which may otherwise

be abandoned due to the long turnaround time. Our research will facilitate system management through providing high-level system configuration, monitoring and steering services.

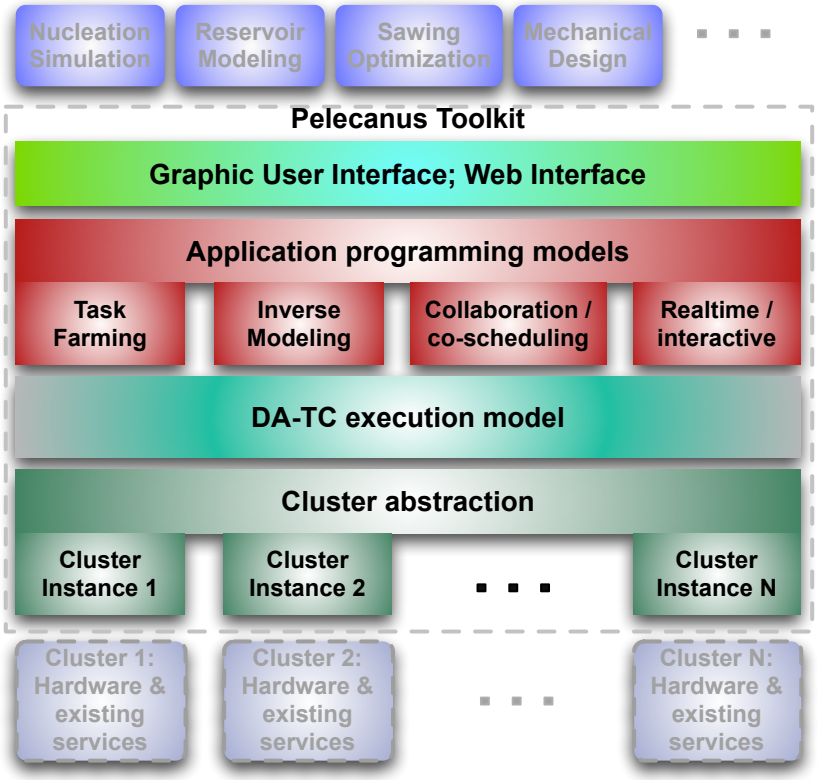


FIGURE 1.2: Schema of research project.

Figure 1.2 gives the schema of our research. Our toolkit serves as the middleware between the user and physical clusters. The DA-TC model is the core component of this toolkit. It integrates multiple cluster for application execution and provides monitoring information and control of the jobs. Several application execution scenarios are supported, including task farming, inverse modeling, co-scheduling and time-critical support. High-level interfaces, such as GUI and web portal, are provided in this toolkit, which gives the user the ability to directly control the workflow and each individual job executing in the DA-TC model. At a lower level, cluster abstraction is used to unify the description of and access to resources in this multi-cluster environment. It hides the heterogeneity of the physical clusters and provides a uniform interface for our DA-TC model.

The rest of this dissertation is organized as follows. We present background and related work in Chapter 2. In Chapter 3, we give the details of our **Pelecanus** toolkit, including the components of the DA-TC execution model, the cluster abstraction that can be used to monitor the cluster functionality and report cluster information for various workload allocation policies in DA-TC model, and the targeted execution scenarios. Chapter 4 shows how the proposed model can improve multicluster grid performance via theoretical analysis and experiments. In Chapter 5, we describe the large-scale applications we have dealt with and the approaches we deployed our execution model to these applications to achieve reduced turnaround time. Chapter 6 shows how the cloud computing technology can be benefit to our application execution by integrating hybrid grid/cloud resources through our model. The conclusions and future directions are discussed in Chapter 7.

Chapter 2
Background and Related Work

In this chapter, we discuss several existing grid computing technologies for improving application execution across a grid.

The Network Weather Service (NWS) [Wol03] provides methods for predicting the performance of computational grid resources using computationally inexpensive statistical techniques. The predictions generated in this manner are intended to support adaptive application scheduling at the super-scheduler level (shown in Figure 1.1). However, once application tasks are dispatched to local scheduler systems on participating resources, the local scheduler systems take control of task execution. As mentioned in Chapter 1, the bottleneck for application execution is the completion of the tasks assigned to the *slowest* computing resource. One might consider technologies to migrate queued tasks among the participating clusters in multicluster grids, but the migrated tasks have to be located at the end of local scheduling queues on the targeted clusters, waiting for resource allocation.

Pegasus [DSS+05] is a framework for mapping complex scientific workflows onto distributed systems. Condor [TTL02] software has to be installed on a user's local machine and Globus [FK99] software has to be installed on the head nodes of the various computing sites. Condor-G is used to schedule the submitted jobs in the workflow to the remote resources. The Globus installation at the remote sites takes care of receiving the job specification and submitting the jobs to the local resource management system, such as PBS [PBS10], Condor, and LoadLeveler [Loa]. There are three major issues that come up in Pegasus. The first is the overloading of the head node on a participating resource with too many Globus jobs, although Pegasus adopts clustering technologies and limits the number of jobs on a single site to decrease head node load average. The third is that

each job submitted to a remote site can spend a lot of time on waiting in the local scheduling queue for resource allocation. The third is that Condor and Globus installation are required.

A solution to the second issue identified above in order to make full use of available resources, is Condor Glidein technology [FTL+01], which provides a mechanism to temporarily add remote resources to the local Condor pool. However, it has limitations and imposes the overheads of the start of a complete set of Condor daemons (i.e., Condor must be running) on the set of resources, and the Condor configuration has to be changed to provide access permission to the remote resource. The Glidein technology heavily depends on Condor and Globus installation, and it may still suffer from the head node overloading problem. In addition, it does not solve the execution bottleneck caused by the *slowest* participating resource.

LSF MultiCluster™ [Xu01] is a software that enables load sharing among disparate LSF clusters. It supports transparent access to remote cluster configuration and load information, distribution of jobs among clusters, importing and exporting of batch queues between clusters, configurable user account mapping at the individual user level, and automatic file transfer. However, all the operations performed are limited to clusters with an LSF installation.

SAGA [SAG10] is a standardization effort for grid application programming. It attempts to bridge the gap between the existing grid middleware and application needs. SAGA provides a mechanism for efficient execution of applications with many loosely-coupled sub-tasks based on the *big-job* abstraction. This abstraction supports the clustering of sub-tasks into a larger big-job and effective dispatching of the sub-tasks. Large chunks of resources will be allocated to a single job to reduce the communication and synchronization cost. However, this suffers from the problem of deciding the size of a big-job (how big is big enough) and how many sub-tasks there should be per big-job; in addition, the problem caused by the *slowest* cluster is not addressed by SAGA.

While most researchers have put much effort into high level solutions for grid computing environments, our research focuses on load-balancing and enhancing efficiency after application tasks have been dispatched to the local resource management system. Since for each task, significant

amount of time is wasted in waiting for resource scheduling in the queue of local batch systems, reducing the queuing time is an important approach to reducing the application turnaround time. Many resources have a long queue waiting time due to the high utilization level, leading to performance degradation for large scale applications, since each task will experience long delays in the queue. Sometimes the queue waiting time overhead is even longer than the application's runtime [ShSV+08].

Normally, a batch queue system is a combination of a parallelism-aware resource management system and a policy based job scheduling engine. First-come-first-serve (FCFS) is a simple initial scheduling policy used by many site administrators to configure their batch systems. Other parameters, such as specific job and/or user priority, can be taken into account to tune the scheduling policy. Many sites also use FairShare [Fai10] and some sort of backfilling [JSC01, WF01] to prevent starvation.

Predicting the time individual jobs will spend waiting in the queue is a difficult problem [BNW06, Dow97b, Dow97a, STF99], not only because the resource demand is unpredictably driven by users, but also because the scheduling policies are usually not completely disclosed publicly (since users may not be entirely satisfied with their respective priorities). Moreover, in order to ensure that the utilization of such expensive resources is maximized, the total user allocations typically exceed the feasible occupancy. The effects of this constantly changing interaction between users and the in-place batch-scheduling policies often result in queuing delays that fluctuate through several orders of magnitude [NWB08a].

While advance reservations [Mac07b, YKA05] can be used to reduce or eliminate waiting time, they are not available at all sites. Besides, they usually involve system administrator assistance and require notice beforehand. Furthermore, Snell et al. [SCJG00] showed that advance reservations can decrease system utilization and have the potential to introduce deadlocks. Another method is to use virtual reservation technology [KKNW08, NWB08b]. However, this relies on the predictions (e.g., those made by QBETS [NBW07]) to implement a reservation. Although this approach does

not require modification to local scheduling policies or scheduler submission protocols, it can not provide accurate prediction. QBETS only predicts bounds by estimating percentiles of the empirically observed delay distribution, and does not take into account possible bursty job submissions at run time. While Zhang [ZKC09] proposed a way to overlay the execution time of previously submitted jobs and the currently queued jobs in order to reduce overhead, it requires complicated algorithms to make decisions on the submission time, since if the wait time for the newly submitted job is less than its predecessor's run time, this new job must pad its requested time to honor the dependencies. Balancing these effects requires heuristic scheduling.

As many sites do not give users the ability to make advance reservations, resource leases provided by "pilot job" approaches [FTL+01, WGLT06] are likely to be a good alternative [FK08]. The objective of pilot-based submission is to obtain a time-constrained lease of a number of physical cluster nodes. The lease is defined in terms of the expected duration of the slot and the requested resources. With pilot-based infrastructures, users submit their jobs to a centralized queue or job repository. A pilot job will download a real job from a repository once it starts executing, hence these jobs are executed asynchronously by running pilot jobs. Examples of pilot-based workload management systems include DIRAC [TGSR04], glideinWMS [Sfi07], and PanDA [KAN+09]. In general this approach allows fast execution. However, it still faces the problems of deciding the number of pilot jobs and the duration of resource leases beforehand. Usually the number of submitted pilot jobs is very large, where only some of them will run the actual jobs, and the others will abort [Pil10], which will cause resource waste.

We have worked closely with application scientists in various disciplines. Their applications are compute-intensive, and usually require thousands of dependent or independent jobs to complete in a limited timeframe. Submitting large numbers of such jobs to a local resource management system will lead to low job throughput because of the high queuing and dispatching overheads. Besides, each cluster usually has a maximum limit of the number of submitted jobs for each user. Clustering multiple jobs into a single larger job can reduce the number of total jobs submitted by each user;

however, it is not suitable for some applications where the problem scale cannot be determined before submission. It is also hard to maintain good load balancing due to the heterogeneous job execution time of the underlying resources. The users will lose the ability to monitor and control individual jobs in this way, which is an important requirement, especially in the case of job failure. These groups want to construct an easy-to-use user-space submission environment that does not rely on any software installation to execute the jobs. Dynamic scheduling is important in this case; otherwise users have to wait for the slowest cluster in order to collect all the results. Fault tolerance is another issue that needs to be addressed by users, since they have to bear responsibility for the overall system failures of any participating resources.

The biggest concern for the users is the long queuing time, since jobs have to wait in queues for resources to be available. For multi-iteration applications, each job has to wait multiple times. Advanced reservations are a way to reduce the queuing time, but this is not always available. Virtual reservation can only predict the successful rate about the jobs to be executed before a deadline, but can not guarantee the application is finished if there is timeframe requirement. The pilot job based approach can be used to reduce the queuing time, but there are several disadvantages with this approach. First, the pilot job will pull the real job from a central authority when it is ready to execute, which provides no way for the central authority to provide monitoring information since it does not track which task is in which pilot job. Second, each failure of a pilot job when it is performing a downloaded task will cause a missed result. Third, it is difficult to match the job execution time to the lease time of each pilot job, which will cause resource wasting if the rest time of the pilot job is not long enough to execute the next job.

To address all of these problems, we have developed a new user-space execution model called DA-TC (Dynamic Assignment with Task Containers) for application execution in multicluster grids, motivated by application scientists' requirements. The DA-TC model is designed to be executed in user space, which means that there is no specific system configuration and software installation required on any participating cluster. It significantly decreases execution turnaround

time, since dynamically assigned tasks can immediately be executed once a task container captures resource allocation. It also increases the reliability for application execution in multicluster grids, since any system failure from a single participating cluster can not affect runtime based dynamic load balancing strategy.

Chapter 3
Pelecanus: A Multicluster Management Toolkit

We have developed the **Pelecanus** toolkit whose goals are to (a) improve resource utilization, (b) speed up application execution, and (c) facilitate resource and application monitoring in a multicluster grid environment. **Pelecanus** can be used as a standalone tool with a Graphical User Interface (GUI), or can be accessed via a web interface, i.e., a grid portal. **Pelecanus** adopts virtualization technologies for efficient resource and application management. The dynamic assignment with task container (DA-TC) execution model is used to schedule tasks across participating clusters. **Pelecanus** is designed to execute in user space, and thus does not require any special system configuration and/or software installation on participating clusters.

3.1 Overview

There are four major objectives for the design of **Pelecanus**:

1. It should significantly decrease application turnaround time. An application is not complete until the completion of all its tasks, which may be assigned to different clusters. Thus, the turnaround time of an application execution is determined by the last completed task, no matter how quickly the other tasks are executed. **Pelecanus** employs a dynamic load balancing technique so that all tasks can be completed at, or almost at, the same time. This can significantly reduce execution turnaround time.

2. It should increase the reliability of application execution in multicluster grids. In particular, **Pelecanus** should tolerate failures of individual clusters.

3. It should provide flexible and user-friendly interfaces for application scientists to perform their tasks. These application scientists are typically not computer-savvy, and making **Pelecanus** easy-to-use is the key to its adoption in practice.

4. This toolkit should be executed in user space so that no special system configuration and/or software installation is required on participating clusters.

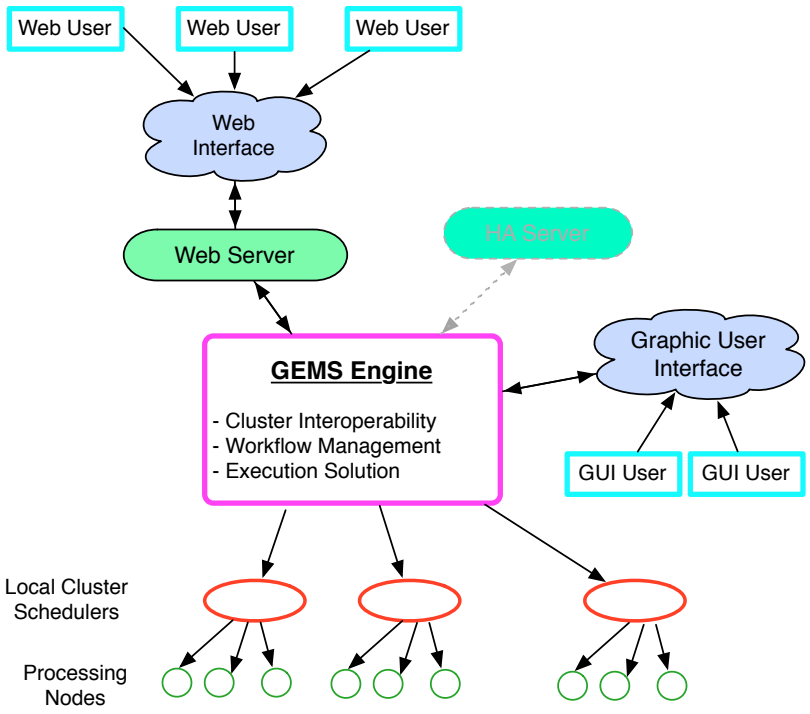


FIGURE 3.1: **Pelecanus** architecture.

The architecture of **Pelecanus** is depicted in Figure 3.1. **Pelecanus** sits on top of local cluster schedulers, and is responsible for resource management, workflow control, execution management and user interaction, at the inter-cluster level. The Grid Execution Management Service (GEMS) engine that we have developed provides the core services, including cluster interoperability, workflow management and pattern-based execution implementation. There are two interfaces: a standalone Graphical User Interface (GUI) and a Web Interface (i.e., grid portal). A high-availability server [LSLS05] can also be used for reliability improvement.

The following are the major features provided by **Pelecanus**:

1. **Resource abstraction and monitoring**. Automated tests are integrated into **Pelecanus** to evaluate clusters' functionality and useful information is abstracted for the user to review. Users can select suitable clusters to execute their applications based on this information.

2. **Application specification**. **Pelecanus** allows the user to specify the executable, resource requirements, dataset location(s) and the execution pattern of an application. An application workflow template is provided for each execution pattern.

3. **Application monitoring and steering**. Once an application is submitted, **Pelecanus** allows the user to monitor the execution progress of the whole application and/or any particular component on a cluster. In addition, **Pelecanus** allows the user to adapt the workflow based on the runtime status of resources and execution progress.

4. **Data management**. Support for data manipulation is provided in **Pelecanus** to allow application deployment on remote clusters and massive input dataset operations.

5. **Security service**. A single entry point is provided for a user to access clusters and grid services, based on existing technologies for credential and access management such as MyProxy and Secure Shell (*ssh*).

3.2 Execution Management

Pelecanus enhances application execution management by adopting the DA-TC model, which is the key component to provide these functions. This DA-TC execution model is based on **D**ynamic **A**ssignment with **T**ask **C**ontainer. It is designed to improve application execution in a multicluster grid environment.

3.2.1 Architecture

In the DA-TC model, an application is assumed to consist of a large number of tasks, among which the parallelizable tasks are executed on the participating clusters of a grid. In the rest of this document, "task" refers to parallelizable tasks of an application. Each task can consist of

one or more sequential or MPI jobs. We assume that task dependency is dealt with by workflow management; the DA-TC model handles the execution of parallelizable tasks generated by that workflow manager. It is also assumed that there is no inter-task communication[1].

The DA-TC model introduces the task container (TC) concept. This idea draws inspiration from the previous "placeholder job" implementation [PLG02, Pin03], but is intended to provide a host environment for each application during the whole execution cycle. It can be viewed as an extension of the pilot-based infrastructure. A task container waits for resources in a participating cluster, and after being allocated resources, it provides a lightweight hosting environment for task execution. A TC is viewed as a normal job by a local resource scheduling system. It is submitted into a queue and waits for resource allocation. The local scheduler allocates resources to a TC under its own scheduling policies. The resources assigned to a TC are released after the TC's execution ends. From a task execution perspective, a TC is a host environment. It provides a standardized method of managing the lifecycle of task execution on any participating cluster. Each task has associated task metadata. A TC retrieves task execution requirements from the task metadata and takes actions to perform a task, including stage-in, invocation, task termination, task execution monitoring, stage-out, etc. A TC is a lightweight environment. It can be easily deployed, and can launch any existing "legacy" task executable on participating clusters.

The DA-TC execution model adopts a dynamic task assignment strategy and employs an application execution agent (AEA). The AEA maintains the tasks waiting for execution and is in charge of the assignment of tasks to an individual TC. A task container reports any status changes of itself and the running task(s) to the AEA. Based on the runtime status of task containers, the AEA takes actions to assign tasks to different task containers. Different task scheduling algorithms are adopted by the AEA. Each task assigned on a TC (or say, a participating cluster) does not need to wait for resource allocation in the local scheduling system since the TC already holds the required resources. Therefore, the tasks on a TC can be executed immediately.

[1]Task clustering technologies can partially remove the inter-task communication.

Our DA-TC model is different from a pilot-based infrastructure. In pilot job mechanisms, users submit their jobs to a centralized queue. The pilot jobs are broadcast to all the available computing elements [Nil08]. A pilot job is not committed to any particular task, and may not be even bound to a particular user; this allows many users to exploit a single pilot infrastructure within one virtual organization [Pil]. Once the pilot jobs get started, they "pull" jobs from the queue to perform the actual work. If there are no jobs waiting for the pilot, the job exits immediately [Pil10]. But in our DA-TC model, the TC submission is based on the scheduling results. Each use of this model is initiated by one user, and all the service units used by each TC are counted on this user. The centralized AEA can control the jobs performed by each TC by active assigning the jobs. The TCs will not exit until received the termination signal from the AEA, so that this model can be used to execute multi-stage applications.

During the whole application execution, the AEA is responsible for several functions including (a) deploying and submitting task containers to participating clusters, (b) monitoring container status, (c) dynamically orchestrating workflow and assigning application tasks, and (d) steering application, task and container executions. An application execution typically employs multiple TCs. The number of TCs for an application execution depends on user configuration. A submitted TC waits for resource allocation in the queue. Once a TC obtains resources, it is used to execute the tasks dynamically assigned by the AEA. TCs take responsibility for managing the task execution lifecycle and holding resources until application execution is accomplished[2].

Figure 3.2 shows the interaction between the AEA and the TCs. To carry out an application execution, the first thing for the AEA to do is to submit TCs to participating clusters. The submitted TCs are placed as normal jobs in the scheduling queues of participating clusters, waiting for resource allocation by local resource management systems. One participating cluster may host multiple task containers, according to different load balancing strategies adopted by the AEA. Af-

[2]The termination of application execution can be decided at run time by the AEA, which provides a way for users to interact with application execution at run time in batch systems.

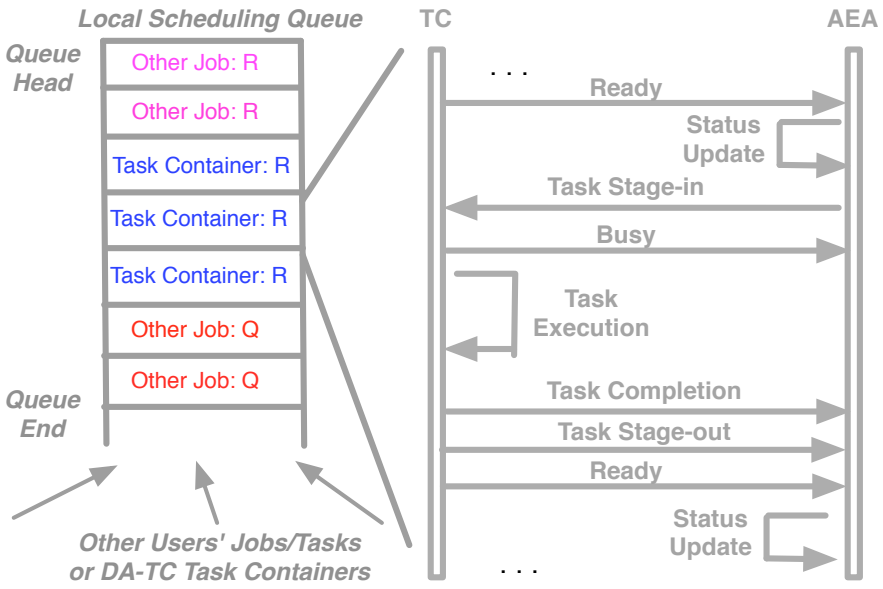


FIGURE 3.2: The interaction diagram between AEA and TC. "R" denotes running and "Q" queuing. "Other" delegates the jobs submitted by other users.

ter a TC obtains the required computing resources from a local scheduling system, it communicates with the AEA for task assignment. First, the TC sends a message to the AEA to say that it is ready to run a task. Second, the AEA updates the TC status table and then one or more tasks are selected, based on application workflow management strategies. Third, task stage-in, execution and stage-out are performed, and the status tables associated with tasks and TCs on the AEA are updated. After a task is completed successfully, the AEA and the TC are ready for the execution of the next task.

The DA-TC model adopts a dynamic load balancing strategy. TCs with allocated computing resources can be assigned application tasks, while other TCs still in the local queues continue waiting for resources. *Fast* clusters will be assigned more tasks for execution. Each task assigned to a participating cluster can be executed immediately. Figure 3.3 shows a runtime scenario of the DA-TC model. Ten TCs have been scattered on three clusters, according to the scheduling algorithms of the AEA. The three TCs on Cluster 1 are running, accepting tasks from the AEA; the three TCs on Cluster 2 are still waiting; and two of the four TCs on Cluster 3 are performing tasks while the other two are waiting in the queue. Application execution is making progress, no

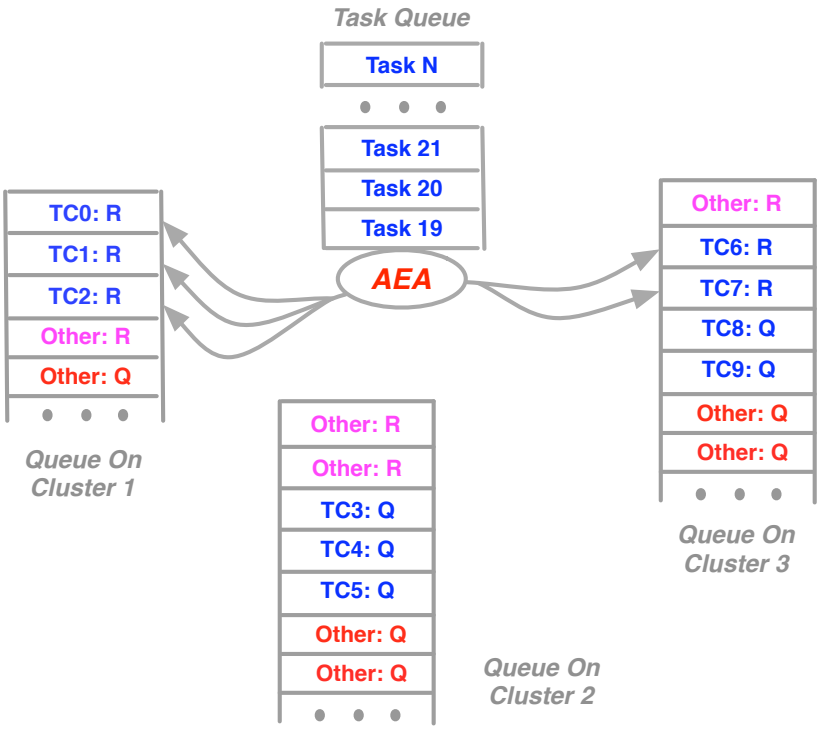


FIGURE 3.3: The DA-TC runtime scenario. "R" denotes running and "Q" queuing. "Other" delegates the jobs submitted by other users.

matter how slow Cluster 2 is. It also does not matter if system failure happens on any participating cluster during application execution, since the ongoing tasks will be resubmitted by AEA after a certain amount time if it does not receive the results from the TCs. Subsequent tasks will not be submitted to this unavailable system because the AEA can not get the status of the TCs in such system. Therefore, this mechanism can increase execution reliability for each application.

3.2.2 Grid Execution Management Service

Based on the DA-TC model, we have designed and implemented an execution service called Grid Execution Management Service (GEMS). This service communicates with the local resource management systems and the grid middleware on the participating clusters, if it exists, to carry out the application execution process. No specific software installation on participating clusters is needed to run this service. The only requirement for local resource management administration

is to provide an access mechanism for a user, e.g., *Globus* or *ssh*, and GEMS will transparently run grid-enabled applications on available clusters.

Next, we explain the GEMS implementation issues related to heterogeneity, application execution agent, task container and interaction.

3.2.2.1 Heterogeneity

There are three kinds of heterogeneity involved in the development of GEMS: access mechanisms, local resource management systems (LRMS), and cluster settings. The access mechanism for a cluster specifies how to inter-operate with the resources and the services a cluster provides. For example, some clusters can be reached by *ssh*, while some use *Globus* access. GEMS interacts with clusters for TC deployment, data transfer, local work directory identification, etc. In the current setting of GEMS, we are using *ssh* and *scp*, which are available for most clusters for user login and file transfer.

Different clusters have different LRMS installations, including PBS, Condor, LSF, SGE, and so on. GEMS has to communicate with different LRMSs for monitoring queue status, generating job scripts of TCs, and submitting/withdrawing TCs assigned to participating clusters. An ensemble of LRMS-related operations have been abstracted. In the current implementation, GEMS supports PBS management systems. It would be easy to be extended to support other LRMSs, e.g., Loadleveler and LSF. Moreover, DRMAA [DRM10] provides a high-level Open Grid Forum [OGF06] API specification for the submission and control of jobs submitted to the distributed resource management systems within a grid architecture. It will make the extension easier if we use DRMAA to interface with different resource management systems.

Cluster settings refer to the different operating systems, network connections, memory size, etc., in the participating clusters that can be be used to meet the resource requirements of a task. There are two issues when dealing with different cluster settings. The first is related to resource requirements of tasks. The current version of GEMS uses the information from the cluster abstraction for TC assignment. The second issue is related to binary executables. It is required to provide dif-

ferent binary executable versions of containers and tasks for each operating system appearing in participating clusters. GEMS selects the executables to match different clusters automatically.

3.2.2.2 Application Execution Agent (AEA)

The AEA plays the metascheduler role in GEMS. There are three phases for the AEA to perform for an application: execution preparation, dynamic task assignment, and execution termination. The actions in execution preparation include:

1. obtaining static and dynamic information for all participating clusters;
2. deciding how many TCs will be assigned to each participating cluster;
3. deploying TC executables on each cluster;
4. initializing TC (as well as tasks) status tables;
5. submitting all the TCs to the participating clusters;
6. waiting for TC status to be ready.

The second phase is dynamic task assignment by communicating with TCs, i.e., dynamic load balancing. This phase conducts the essential execution of an application. Once the AEA understands that a TC is ready, it selects one or more appropriate tasks according to the workflow management and task metadata, updates the status of the TC and task(s) to "*stage-in*", and then sends the task metadata to the TC. The TC takes care of actual execution of the task(s), such as stage-in, invocation, and stage-out. Task metadata includes task dependency, executable location, data location, resource requirements, etc. During this phase, the AEA maintains the status of all TCs and tasks.

After all the tasks have been completed, the AEA takes actions to terminate the running TCs, and, if applicable, withdraw queued TCs. It is possible that even if all the tasks of an application are finished, there could be some TCs still waiting for resource allocation in the queues of the *slow*

clusters[3]. This requires the AEA to interact with local scheduler systems on participating clusters to delete the queued TCs.

GEMS implements static load balancing for task container distribution and dynamic load balancing for task assignment. The latter means that the AEA assigns tasks according to the runtime status of application execution, which has been discussed in [LYA+08]. Here, we provide further explanation of the the task container distribution strategy. The first step is to obtain the static resource description and runtime status of the participating clusters. The static resource description of a single cluster includes operating system, local resource management system, work directory, node number, CPU number, CPU speed, memory size, and so forth. As for the runtime status, the current implementation of GEMS just considers the number of running and queued jobs. Further runtime status information, such as performance prediction and history records, can be included in the future. After this, GEMS uses the available information to decide how many task containers will be distributed on a particular cluster, based on computation capability of each cluster and load balancing strategies. Currently, four different allocation strategies are compared and used to determine the task container dispatch, which will be described in Section 3.4. We say that TC distribution is static because once submitted, a TC cannot be migrated.

3.2.2.3 Task Containers (TCs)

From the viewpoint of the local resource management system, a task container is a job submitted into a job queue. Thus, a job script is needed to describe the information about a task container, e.g., resource requirements and username. When running an application, GEMS automatically generates TC job scripts against various LRMSs with TC executable location, username, application name, resource requirements, etc.

In order to provide a host environment for task execution, the implementation of a TC adopts the logic shown in Algorithm 1. This pseudocode is straightforward. A TC takes care of task execution and status update. It exits and releases resources only when the application termination

[3]Some clusters may be overloaded by other users and may take a long time to allocate resources for its assigned TCs.

Algorithm 1 The pseudocode of TC logic.

```
while TRUE do
  if application termination signal then
    exit 0;
  else
    update the status;
    get next task metadata from AEA;
    retrieve data information from metadata;
    access data;
    execute task;
    update the status;
  end if
end while
```

signal from the AEA is detected. The task lifecycle management provided by the current GEMS version includes execution environment setup, stage-in, invocation, stage-out, and termination.

While a TC manages the lifecycle of a task, the LRMS and the AEA jointly handle lifecycle management of a TC. A local resource management system is in charge of queuing, resource allocation, and execution of a task container. The AEA takes responsibility for submitting a TC, sending a termination signal to destroy a TC, and launching commands to withdraw a queued task container. To implement TC withdrawal, GEMS retrieves the job id of a queued TC from an LRMS using the application name and user id, and then issues a job deletion operation, e.g., *qdel* for PBS.

3.2.2.4 Interaction Between AEA and TCs

Efficient and robust interactions between the AEA and TCs provide GEMS with a better ability to perform application execution, in comparison with pilot-based infrastructure. For example, the AEA can conduct dynamic task assignment after it captures the runtime status of the TCs. In contrast, in a pilot-based mechanism, the pilot jobs contact a passive central authority when they are ready to execute, and the central authority has no ability to actively assign tasks to each pilot job. Furthermore, our model provides easy tracking of application execution progress for status monitoring, since the AEA has the status of each task on each TC. But in a pilot-based mechanism, there is no way for the central authority to provide monitoring information since it does not track which tasks are in which machine. In addition, our model provides better fault tolerance because

the AEA can resubmit a task when it does not get its status from the TC after a certain amount of time. But for the pilot-based case, each failure of a pilot job when it is performing a downloaded task will cause a missed result.

GEMS develops a communication protocol and transfer mechanism for interaction between the AEA and TCs. Simplicity and effectiveness are the fundamental principles used to design the communication protocol and the transfer mechanism, due to the unreliable network connections in a grid environment. Before describing the GEMS solutions to these interactions, we introduce two tables maintained by the AEA that play important roles during application execution progress. First is the TC table. Each container has an entry in this table. Recorded information includes which cluster the container is assigned to, what status it has, which task is running in the container, etc. The second is the task table. Each entry in this table refers to a task. Task metadata, e.g., dataset location, and task status are listed in an entry. By checking this table, GEMS can find out how many tasks have been completed, how many tasks are running, and how many tasks are still waiting.

Interaction between the AEA and TCs occurs in two directions: TCs send status updates to the AEA, and the AEA dispatches tasks to TCs. All TCs are required to update their status to the AEA. Each TC has one of three status values: Queued (Q), Ready (R), or Busy (B). The AEA takes different actions based on checking the status of a TC. For example, the AEA will assign a task to a TC if its status is "Ready." In order to dispatch a task to a TC, the AEA retrieves task description from the task table and then sends task metadata to a TC. The TC uses the metadata to take actions for task execution. All these interaction activities are based on file transfer.

GEMS uses two methods to improve transfer performance and reduce overhead time. The first method is that only the "Ready" status needs to be transferred from a TC to the AEA; other status updates are performed by the AEA itself. This is used to minimize information exchange. For example, when starting to deploy a task to a TC, the AEA changes the status from *Ready* to *Busy* without communicating with the TC. The second method is that a TC combines task stage-out

(or task completion status) and status update (e.g., ready for next task) into a single data transfer process, which reduces the network connection overhead.

3.2.3 QoS Analysis

The dynamic task assignment strategy and the task container methodology in DA-TC essentially improve the application execution across multicluster environment in terms of turnaround time, reliability, and execution monitoring.

Application turnaround time can be significantly reduced by this execution model. Turnaround time is the time interval between the submission time and the completion time, i.e., task waiting time in remote queues plus execution time (*wall time*) in multiclusters. Execution time depends on different solver equations and optimization algorithms. While application researchers work on high performance algorithms to reduce execution time, we emphasize effective utilization of computing resources to shorten waiting time. A TC is used to request and hold resources for task execution, which provides immediate execution for tasks that are dynamically assigned by the AEA. Using this dynamic load balancing method, the *fast* clusters will be assigned more tasks. The execution bottleneck caused by the *slow* clusters is eliminated. Any participating cluster can make a contribution to speed up application execution. The overall waiting time of tasks is greatly shortened and resource utilization is enhanced.

The reliability of application execution can be improved since we can recover from a system failure during execution and the overall results for the user are not affected. A task will not be assigned to a participating cluster if a valid TC status can not be provided to the AEA due to network disconnection, system maintenance, or system failure. Task completion status is monitored by the AEA at runtime. If a task execution error happens, the AEA can intelligently make decisions, e.g., resubmission on the same or different TC, to try to fix the problem. Take the diagram in Figure 3.3 as an example. If Cluster 1 and Cluster 2 are not available for some reason, the AEA will not receive “Ready” status of TCs from these two clusters and no task will be assigned to them. Meanwhile, if the AEA can not detect the status of running tasks, certain actions will be

taken, such as resubmitting these tasks on Cluster 3. Application execution fails only if all the participating clusters are not available.

The monitoring and steering of application progress is very easy in this model since AEA keeps updating the status of both TCs and tasks. Compared to the normal case where user needs to login to the remote cluster, type in the correct commands, and only get the information for the fraction of tasks on that cluster, user can easily retrieve the status of each task and each TC by sending requests to the AEA in this model. The user also can change the workflow manually if needed, such as add new task(s), modify datasets, suspend execution, etc.

3.3 Cluster Abstraction

Although the DA-TC execution model provides a mechanism to seamlessly integrate multiple clusters for application execution, there are some challenges that make the implementation difficult. These challenges include how to discover resource information, how to allocate appropriate resources, and how to integrate them since they are heterogeneous in architecture and software, etc. Therefore, it is useful to provide a cluster abstraction in a multicluster environment, which greatly improves resource utilization with easy-to-use access. The cluster abstraction allows a user to view any underlying cluster as a generic computing unit for use, no matter how diverse underlying clusters in a multicluster environment are. On top of the cluster abstraction, integrated services can be easily generated.

This section presents a cluster abstraction to unify describing and accessing resources in a multicluster grid. After investigating various cluster resource management systems, a reference description of our cluster abstraction is presented to cover cluster computing characteristics, hiding the individual cluster details. Following this concept, a deployment service is discussed. This deployment service is one of the essential components in our DA-TC model. Our DA-TC execution model is built on top of the cluster abstraction, and uses the information provided by the cluster abstraction to make scheduling decisions for the task containers. Furthermore, this service can be used

as a separate automated system to monitor and evaluate the clusters' functionality and network performance.

3.3.1 Introduction of the Cluster Abstraction

Abstraction and virtualization have been important tools in computer system design for a long time. Due to the incredible complexity of computer systems, these technologies have been widely adopted by users to better use these systems by virtualizing separate levels of the hierarchies with well-defined interfaces. The simplifying abstractions hide lower-level implementation details from higher-level applications, therefore reducing the complexity of the development process. One good example is disk storage, where operating systems abstract hard disk physical addressing details, i.e., sectors and tracks, so that to application software, the hard disk appears as a set of files of variable sizes. Higher-level applications can operate on those files without any knowledge about the actual physical organization of the hard disk.

The cluster abstraction in our case describes clusters in a uniform virtual cluster format and provides uniform interfaces for accessing those resources. Figure 3.4 illustrates the concept of cluster abstraction. Through uniform high level interfaces, users and upper-level applications can operate on the virtual clusters in the virtual cluster pool (VCP). The complexity and heterogeneity of the underlying physical clusters are hidden from the upper-level applications and/or users.

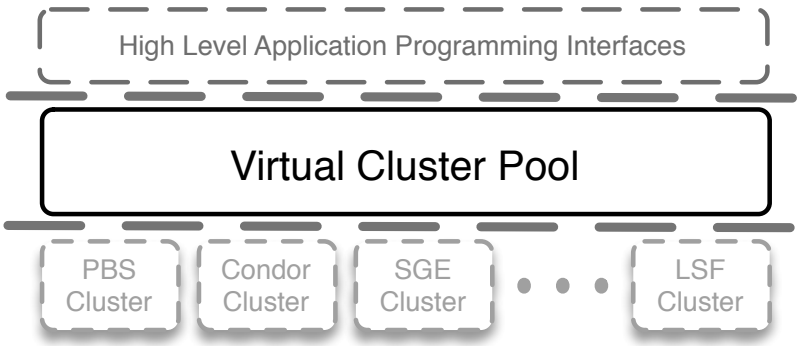


FIGURE 3.4: The basic concept of the cluster abstraction. The virtual cluster pool here hides the details for the heterogeneous multicluster environment from high level services and users.

3.3.2 Cluster Description

To efficiently abstract information, we need to understand the system and users' need. Despite the similarity at the architecture level, most configurations of clusters are different in some way, such as in their local resource management systems. Moreover, their runtime status can be different, e.g., workload, network bandwidth, and availability of compute nodes. Therefore, we need to know what information is of interest, what technologies should be used to abstract the information, and what interfaces should be provided.

3.3.2.1 Local Resource Management Systems

Each cluster employs a resource management system, called a Local Resource Management System (LRMS). LRMSs place jobs in one or more queues with certain resource requirements, and make decisions on the places and times for job execution. Some of them offer a checkpointing service, which helps the user to resume job execution from a certain point after a job or machine failure occurs during the execution process. Widely adopted resource management systems include PBS, SGE, Condor, etc.

PBS, the Portable Batch System [PBS10], is a batch job and computer system resource management software package. It accepts batch jobs, shell scripts and control attributes as input. Jobs are preserved, protected, and then executed, after which the output, the exit status, and possible error messages are delivered to the submitter. PBS includes four components: commands, the job server, the job executor, and the job scheduler. PBS commands provide a platform for users and administrators to submit, monitor, modify, and delete jobs. The PBS job server is essential to PBS, because it provides basic batch services such as receiving, creating and modifying a batch job, protecting the job against potential system crashes, and executing the job. The PBS job executor is a daemon that receives jobs from the job server and actually places these jobs into execution. This executor is also responsible for returning the job's output in response to the output return request from the job server. The PBS job scheduler is a daemon that can be created by each individual local resource. PBS job schedulers communicate between each other and also with the job server and

the job executors for job availability and system resource status. The scheduler basically contains information about the usage policy for each site and controls which job to be run, and when and where to run the job. All of these four components working together as a whole system ensure the stability and reliability of PBS.

The Univa Grid Engine (previously Sun Grid Engine or Oracle Grid Engine) [UGE] allows users to submit computational tasks to an Grid Engine controlled system with transparent workload distribution. Univa Grid Engine accepts batch jobs, interactive jobs, and parallel jobs. A checkpointing mechanism is used to ensure correct recovery from a job or system failure. Firstly, jobs are submitted from the *submit host* to the queue hosted by each *execution host*, where job attributes are reviewed and such decisions as whether the job may be migrated are made. The queues and jobs are controlled by the *master host*, which runs the *Master daemon* and the *Scheduler daemon*. The *execution hosts* with the *Execution daemon* are responsible for the execution of jobs and the update of job status and workload to *Master daemon*. The *Communication daemon* is used for all communication among the components. Throughout the lifetime of the jobs, users can delete jobs and check job status, and the administrators can manage the queues and jobs.

Condor [TTL02] is a specialized job and resource management system (RMS) for compute-intensive jobs. The system also utilizes a checkpointing mechanism to guarantee execution of jobs. Since Condor is a batch system, it also provides a queuing mechanism, scheduling policy, priority scheme, and resource classifications. Jobs are submitted to a job queue, which is referred to as a "Condor pool," and they are executed on distributed computational resources. Results and logs of the job execution are returned to the local submit machine. However, unlike traditional batch systems that can only operate on dedicated machines, Condor can also schedule jobs on non-dedicated machines.

The cluster abstraction uses the information provided by these heterogeneous local batch systems and other resource management services to generate virtual cluster images.

3.3.2.2 Static and Dynamic Information

As defined previously, a multicluster environment consists of a group of clusters that have their own local resource management tools. The configuration of the clusters varies in several ways, such as number of computational nodes, number of CPUs on each node, the cluster operating system and file system, available memory and network connections. Because of unpredictable events, the status of clusters may change in several aspects, e.g., the number of available nodes. Since all clusters need local resource management systems to manage their computational tasks, jobs must be scheduled according to their LRMS's scheduling policy. In addition to static configuration information, two important dynamic information parameters are introduced to measure the performance of a cluster: the number of running jobs and number of queued jobs. Some popular LRMSs already provide interface to gather these information.

PBS provides interfaces to access the static configuration as well as the dynamic status. Using these interfaces, information such as the number of nodes, operating system, number of CPUs per node, size of memory, and scheduling policy, which are referred as static configurations, can be easily gathered. Moreover, other information such as the status of each node, the assigned memory, the assigned CPUs, the number of jobs in the queue, and the number of running jobs, is available. This is the dynamic status information of the cluster.

The information that can be queried from Condor includes operating system, hardware architecture, memory, state of the node (Claimed or Unclaimed), node activity (Idle, Owner, or Busy), average load, activity time of the node, number of jobs running, number of jobs queued, command for the jobs. These information can be grouped into two categories, the first three as static information, and the rest of them as dynamic information.

Univa Grid Engine defines values for each node that include system architecture, number of processors, total memory, total swap space, total virtual memory, used memory, used swap, number of executing jobs, number of queued and waiting jobs, etc.

For the convenience of higher-level applications, the type of LRMS used in each cluster and the home directory of the user are considered as static information. When adding a new cluster into the pool, since the system has no knowledge of this resource, it has to query the LRMS unless the user specifies it. After the initial query, access and operation to the cluster will be much easier. Moreover, due to unexpected node crashes, the number of available nodes, thus available CPUs, can change, and therefore should be classified as dynamic information. Since all computations will take some memory for processing, the size of available memory can be surely identified as dynamic information.

From the discussion above, it is easy to conclude that these LRMS share the same static information and provide similar dynamic information. The static information for the cluster should contain number of CPUs, operating system, total size of memory, number of nodes, the user's home directory, and the DNS name for the cluster. The dynamic information should include available nodes, available CPUs, available memory, number of running jobs, and number of queued jobs.

Abstracting this dynamic resource information and heterogeneity is challenging in a multicluster environment, and is hindered by the lack of a consistent overview of available computational resources. We use an XML schema for a uniform resource description for each cluster. The information for all available clusters are stored in one XML document, whose form is defined by the schema. All information entries are defined as simple elements in the schema file. The "*static_info*" is a complex element which has all the elements representing static information. Another complex element called "*dynamic_info*" contains all the elements representing dynamic information. These two complex elements are belonging to the "*cluster*" element which is the root element.

The XML schema for the cluster abstraction is provided in Figure 3.5. One sample XML description of a cluster available in Center for Computation & Technology at Louisiana State University is shown in Figure 3.6.

```
<!-- definition of simple elements -->
<xs:element name="DNS_name" type="xs:string"/>
<xs:element name="total_nodes" type="xs:positiveInteger"/>
<xs:element name="total_cpus" type="xs:positiveInteger"/>
<xs:element name="total_mem" type="xs:positiveInteger"/>
<xs:element name="available_nodes" type="xs:positiveInteger"/>
<xs:element name="available_cpus" type="xs:positiveInteger"/>
<xs:element name="available_mem" type="xs:positiveInteger"/>
<xs:element name="LRMS" type="xs:string"/>
<xs:element name="architecture" type="xs:string"/>
<xs:element name="running_jobs" type="xs:positiveInteger"/>
<xs:element name="queuing_jobs" type="xs:positiveInteger"/>
<xs:element name="home_dir" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="static_info">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="DNS_name"/>
      <xs:element ref="total_nodes"/>
      <xs:element ref="total_cpus"/>
      <xs:element ref="total_mem"/>
      <xs:element ref="architecture"/>
      <xs:element ref="home_dir"/>
      <xs:element ref="LRMS"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="dynamic_info">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="available_nodes"/>
      <xs:element ref="available_cpus"/>
      <xs:element ref="available_mem"/>
      <xs:element ref="running_jobs"/>
      <xs:element ref="queuing_jobs"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="cluster">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="static_info"/>
      <xs:element ref="dynamic_info"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

FIGURE 3.5: The pre-defined XML schema file.

```
<cluster>
  <static_info>
    <DNS_name>eric.loni.org</DNS_name>
    <total_nodes>128</total_nodes>
    <total_cpus>512</total_cpus>
    <total_mem>2127389696kb</total_mem>
    <architecture>linux</architecture>
    <LRMS>PBS</LRMS>
    <home_dir>/home/zyun</home_dir>
  </static_info>
  <dynamic_info>
    <available_nodes>127</available_nodes>
    <available_cpus>508</available_cpus>
    <available_mem>2009359872kb</available_mem>
    <running_jobs>11</running_jobs>
    <queuing_jobs>39</queuing_jobs>
  </dynamic_info>
</cluster>
```

FIGURE 3.6: A sample resource entry in a resource list.

3.3.2.3 User Environment

Most of the time, users need specific software to compile and execute their programs. Therefore, it is necessary to provide information about the software environment of each cluster so that users can choose the one that best meets their requirements. This information can be gathered from tests run from a standard user account, in order to reflect regular user experiences. It is important not to gather this information based on tests run from a system administrator's account, which may have special privileges and use custom shell initialization files. We provide several basic tests to aggregate and display data from each cluster. These user-level tests are performed periodically to monitor the grid behavior over time. Dedicated monitoring software, such as INCA [INC10], is deployed and integrated into our platform to guide the user in choosing the right clusters.

3.3.2.4 Network Information

Since the DA-TC model involves multiple clusters connected by the Internet, network performance is another important aspect of execution. A fast connection between two sites can save the user a lot of time in data movement, which is especially important for data-intensive applications. We deployed Speedpage [Gri] to perform automatic tests in order to monitor and evaluate the clusters' file transfer performance.

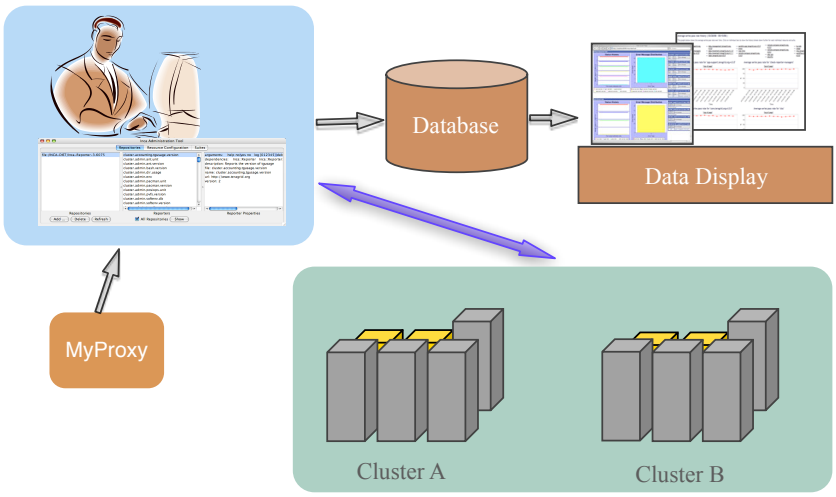


FIGURE 3.7: Speedpage Components.

Figure 3.7 shows the components of Speedpage. It needs the *Globus* toolkit, a MySQL database, and a web server with PHP support in order to perform its test. Before testing, a valid certificate is needed. This is required for MDS host resolution, automated scratch list generation, initial file staging, and for *globus-url-copy*. Speedpage is able to perform the *Globus* third party transfer between two GridFTP servers. The file size and the transfer time are recorded and the network speed is calculated. Speedpage is executed from each resource to every other resource (all-to-all) to measure site-to-site performance. Table 3.1 shows the source and destination URLs, initiated from one cluster (Louie) in LONI to all the other LONI clusters. The source site, destination site, each test's time stamp, and speed information are stored in the database. The web server retrieves the information from the database and displays it on the Speedpage website. It also allows users to specify the time duration or particular site in order to narrow down the information they are interested in.

In order to differentiate the reason for unsuccessful test cases, different error messages are used to determine whether a detected test failure stems from a faulty test, a unmatched file size, or a failed resource. Therefore, users can choose the best suitable clusters for job execution and reduce the chance of system and network failure during their application's execution by examining the data

TABLE 3.1: Source and destination URL list

SOURCE	DESTINATION
gsiftp://louie1.loni.org:2811/dev/zero	gsiftp://louie1.loni.org:2811/dev/null
gsiftp://louie1.loni.org:2811/dev/zero	gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE.Louie.dev.zero
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://louie1.loni.org:2811/dev/null
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://eric1.loni.org:2811/scratch/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://ducky.loni.org:2811/scratch/local/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://zeke.loni.org:2811/scratch/local/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://neptune.loni.org:2811/scratch/local/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://qb1.loni.org:2811/scratch/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://oliver1.loni.org:2811/scratch/zyun/SPEEDPAGE.Louie
gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE	gsiftp://louie1.loni.org:2811/scratch/zyun/SPEEDPAGE.Louie

on the Speedpage website. Moreover, this information can be used by the system administrators to identify, analyze, and troubleshoot user-level grid failures, thereby leading to a more stable and dependable grid infrastructure.

3.3.3 Uniform Resource Access

Since the cluster abstraction hides complex details about the actual cluster hardware and software configurations and status from a user, the multicluster environment looks like a virtual cluster pool to the user. Despite the heterogeneity of different clusters, users can query information from the environment or configure their own execution environment (Section 3.3.3.2) via simple uniform interfaces. Because different interfaces are provided by different LRMSs, the conventional approach for utilizing the multicluster environment requires expertise with all those LRMSs, which increases the difficulty. The high level interfaces of the cluster abstraction concept should cover these diversities, and be versatile for all kinds of clusters, such that these interfaces should enable users (clients) to uniformly access an individual resource or a specific execution environment. These interfaces can be categorized into two groups according to their specific functions, **information related** and **operation related**.

3.3.3.1 Information Interfaces

Since upper-level applications or users may need information on certain resources or certain jobs, the information related interfaces are divided into two categories, at the resource level and at the task level.

- **Resource Level Information Interface**

The resource level information interfaces should be flexible enough to satisfy any query, from detailed information for one specific cluster to list of resources with demanding configurations. For instance, an execution management system wants to make a decision on task allocation within a set of resources for which a specific user has authorization. In this case, the execution management system needs information about these clusters. The information interface should be able to return a set of detailed information for each resource on the list. During this process, all that the execution management system knows is simply a list of clusters with DNS names only.

- **Task Level Information Interface**

The task level information interface is responsible for any information requests about tasks on the virtual cluster pool. Requests include status of some specific jobs, or a set of jobs with same configuration, jobs belonging to a specific user, or jobs on a specific location or in some execution environment with certain configurations, etc. One good example is that a user wants to know the status of all his jobs that have been submitted to the multicluster environment. In this case, without knowledge about the command for the LRMSs on each cluster and without operating on each one individually, the status information should be returned to the user from the interface by simply specifying the execution environment and the username.

3.3.3.2 Operation Interfaces

Here, Operation Interfaces refers to a set of APIs responsible for specific operations on the virtual cluster pool, ranging from submitting tasks to managing customized execution environments. Before we discuss the details, the term "*execution environment*" needs to be clarified. The execution environment for a multicluster grid is a subset of clusters that satisfy certain hardware and soft-

ware requirements from the user. It is defined in an environment file, which can be used for future operation on this environment. This environment file simplifies the process of specifying execution environment with multiple complex constrains.

Due to different levels of access permission, the operation interface contains two levels of interfaces: the administrative operation interface and the user operation interface.

• **Administrative Operation Interface**

The administrative operation interfaces can provide cluster level or queue level administration. Operations include adding a new cluster to the virtual cluster pool, deleting a virtual cluster from the pool, changing the scheduling policy on a specific cluster, managing user priorities, and other regular queue operations. Administrative regular queue operations include deleting a user's jobs or jobs idling for an excessive long time, holding and releasing any job, reserving resources, etc. The administrative operation interface is effective only to those clusters that offer authorization as administrator to the user using this interface. This means that, by default, all end users are considered as normal users and can use the user operation interface. Only when the user has administrative permission to a cluster can he perform administrative operations on that specific cluster.

• **User Operation Interface**

Since this interface is only for a normal user, all regular job operations such as submitting, deleting, holding, and releasing jobs can be only done to those jobs that are owned by that user. According to the concept of cluster abstraction, the virtual cluster pool should be in user space, which means all users can manage their own virtual cluster pool by adding or deleting clusters. Users can customize their own execution environment by specifying requirements. After the specification, the service will select virtual clusters that satisfy the requirements and form them into a unique execution environment with a user-specified name. This name can be used to retrieve configurations for the defined environment in other interfaces.

To sum up, the administrative operation interface and the user operation interface share a subset of interfaces, with the only difference being the level of access. These interfaces are the regular job operations.

3.3.3.3 Execution Environment

As mentioned above, all the information about the execution environment is stored in an environment file. The environment is represented as an arbitrary collection of entries that provide access information as well as other content. The characteristic information for the environment defines the common features for these resource entries, ranging from list of DNS names to arbitrary conditions on number of nodes. These definitions equip the deployment service with the ability to intelligently filter an existing virtual cluster pool for qualified resources.

The deployment service should be able to make appropriate adjustments to the execution environment. That is, after initialization of the environment, any event corresponding to a change of any configuration or status triggers reevaluation of the current environments. This process filters the updated virtual cluster pool for qualified resources, and updates the information for the environment, which results in potential changes for the list of resource entries. Certain possible changes include removing such resource entries that fail in the process of reevaluation, or replacing such entries with qualified new entries. Reevaluation uses the characteristic information for standards, therefore it makes no change to the property of the environment. Modifications to the characteristics of the environment can be done by using high level operation interfaces and triggers the reevaluation process. In conclusion, all execution environment modifications comprise three phases: obtaining updated characteristic information required for reevaluation; filtering the virtual cluster pool with this updated information; and updating the resource list with qualified data entries.

3.3.4 Service Deployment

The deployment service has a set of components that implement the concept of the cluster abstraction, ensures that the service is in user-space and is cluster independent. The architecture is shown in Figure 3.8.

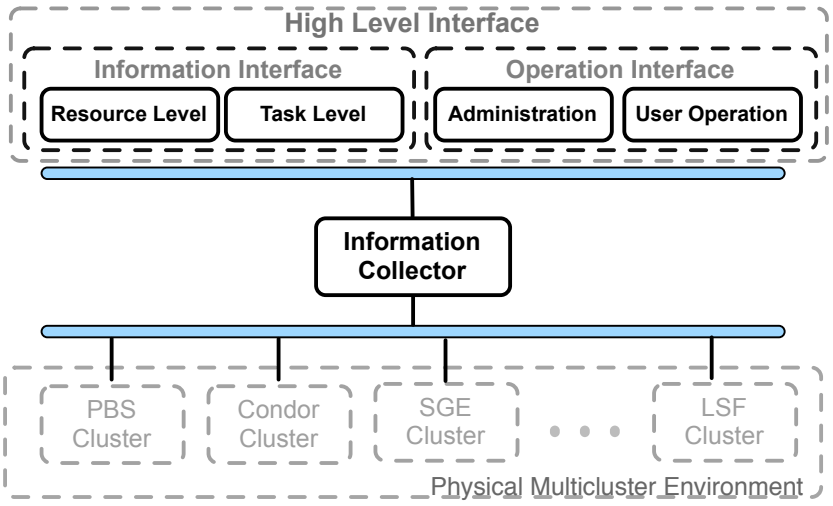


FIGURE 3.8: The service components responsible for the implementation of the cluster abstraction.

3.3.4.1 Service Components

From Figure 3.8, we can identify all the components responsible for implementation and their relationships.

- **Information Collector**

Since the primary objective for the cluster abstraction is describing diverse participating clusters and providing uniform access to them, the information collector is able to communicate with all resources in the pool via their LRMSs.

The information collector broadcasts requests for required information to all available clusters in the pool, in other words, to all authorized resources. These resources are identified by the client, using a list of DNS names. Since the static configurations are considered as constant for a small time period, after the initial request, the information daemon only updates this information after a large time interval, which is specified by administrator. Since the dynamic status is changing every minute or even second, the information collector updates the dynamic information after fairly small time interval period. After the information collection, the collector passes all information to the virtualization service, which creates virtual clusters in the virtual cluster pool.

• **High Level Interfaces**

The high level interfaces provide user the uniform access and operation on heterogeneous multi-cluster environment. By using the information provided from a virtual cluster pool, these interfaces ensure efficient and easy operations on the virtual clusters, i.e., task operations and resource operations. They also equip users with the capability for developing customized program logic, which makes execution management, workflow management and so on much easier and more convenient.

3.3.4.2 Security

At deployment, information exchanges between clusters and client machines can be expected, which are subject to authorization and authentication before data transfer. Since the deployment service uses *ssh* or *Globus* for communication, we use the security mechanism of these two toolkits for authentication. Once a client logs in, the service should allow this authorized client to operate without further authorization requests.

3.4 Task Container Scheduling

An application execution typically employs multiple task containers, depending on user configuration. By tracking the status of clusters, we keep an updated list of available clusters in the pool. Through the cluster abstraction, we are able to capture resource information and use load balancing scheduling strategies to dispatch the task containers.

Table 3.2 shows typical static and dynamic information for the resources gathered by the information collector. It describes a computing resource by the architecture, CPU number, CPU speed, memory size and local resource management system. The architecture factor can be employed to decide which type of algorithms and simulators should be staged. From the table, we can see that the Eric is a Linux cluster with PBS as its LRMS. It has 500 CPUs available and the CPU speed is 2.33GHz. 72 jobs are running on Eric and 129 jobs are queued at this moment. Zeke is a AIX machine with 104 CPUs available. Its resource management system is LoadLeveler. It has 4 jobs running and no jobs queued. Additionally, there is a work directory (Remote home_dir) for each

facility. This is the home directory of the local account of a grid user. The Stage In/Out module uses the work directory to update the executable, carry out the execution, and download the results.

Several different task container allocation strategies have been implemented, including randomly assigned, weighted workload allocation, shortest expected delay, and mean response time.

Before job execution, the users will define the total number of task containers, J, and choose the number of n suitable resources to execute the application. Then, these task containers are distributed by applying the scheduling scheme and, as a result, a fraction P_i of the TCs are allocated to cluster C_i.

When we submit the task containers randomly, they will be spread out evenly onto all available clusters. So in this case, P_i is computed as

$$P_i = \frac{1}{n}. \tag{3.1}$$

The weighted workload allocation strategy (WWA) [HJSN04] takes into account the heterogeneity of the clusters' performance, and dispatches a number of task containers to a resource according to its computational capability. The computational capability of a resource C_i can be measured as

$$c_i = CPU_{Number} \times CPU_{Speed}, \tag{3.2}$$

and P_i in this case is computed as

$$P_i = \frac{c_i}{\sum_{k=1}^{n} c_k}, \tag{3.3}$$

where $c_k =$ the computational capability of a participating resource C_k.

In some cases, one or more of the selected clusters may be extremely slow, which results in a very small value for the fraction: $\frac{c_i}{\sum_{k=1}^{n} c_k}$ ($i \in [1 \cdots n]$). In this case, we will not assign any TCs to these clusters, since once tasks get put on these extremely slow clusters, the whole job will not finish for a long time.

In the shortest expected delay strategy (SED) [TC00], the TCs are dispatched according to the queue length and the job processing speed of each participating cluster. Assuming the queue length

TABLE 3.2: Static and dynamic information of resources

CLUSTER NAME	LOCAL RMS	AVAILABLE NODES	AVAILABLE CPUS	ARCHITECTURE	CPU SPEED	TOTAL MEMORY	FREE MEMORY	RUNNING JOBS	QUEUING JOBS	REMOTE HOME_DIR
queenbee.loni.org	pbs	663	5304	linux	2327.506MHz	16430580 kB	13521960 kB	93	80	/home/zyun
eric.loni.org	pbs	125	500	linux	2327.504MHz	8169236 kB	186096 kB	72	129	/home/zyun
louie.loni.org	pbs	127	508	linux	2327.530MHz	8169236 kB	3357556 kB	32	1	/home/zyun
oliver.loni.org	pbs	128	512	linux	2327.530MHz	8169236 kB	426212 kB	27	0	/home/zyun
poseidon.loni.org	pbs	127	508	linux	2327.527MHz	8169236 kB	6667876 kB	7	6	/home/zyun
painter.loni.org	pbs	126	504	linux	2327.528MHz	8169236 kB	361440 kB	13	7	/home/zyun
bluedawg.loni.org	loadleveler	3	24	AIX	1902 MHz	15424 MB	15424 MB	8	0	/mnt/home/zyun
zeke.loni.org	loadleveler	13	104	AIX	1902 MHz	15424 MB	15424 MB	4	0	/mnt/home/zyun
neptune.loni.org	loadleveler	11	88	AIX	1902 MHz	15424 MB	15424 MB	3	0	/mnt/home/zyun
ducky.loni.org	loadleveler	4	32	AIX	1902 MHz	15424 MB	15424 MB	9	0	/mnt/home/zyun
lacumba.loni.org	loadleveler	14	112	AIX	1902 MHz	15424 MB	15424 MB	0	0	/mnt/home/zyun
philip.hpc.lsu.edu	pbs	37	584	linux	2926.073MHz	49433728 kB	36822780 kB	55	33	/home/zyun
tezpur.hpc.lsu.edu	pbs	355	1420	linux	2992.539MHz	8158324 kB	6433744 kB	81	55	/home/zyun

of a resource C_i is Q_i, then the normalized load will be $\frac{Q_i+1}{c_i}$. Since the TCs will be submitted based on the least normalized load, P_i is measured by

$$P_i \sim \left(\frac{Q_i+1}{c_i}\right)^{-1}. \tag{3.4}$$

Notice that in this case the TCs are not proportionately divided among the clusters according to their speeds. In fact, P_i is proportional to the square of CPU_{Speed}. Clusters with slow speeds will receive much less work than the fast machines. A previous study [LM99] showed that it is beneficial to allocate a disproportionately higher fraction of the workload to the more powerful computers at low and moderate loads, while at high load, it is better to keep the machines more balanced.

In the mean response time strategy (MRT), TCs will be submitted based on a fraction that aims to minimize the mean response time of the application submitted to all participating clusters. The response time of a job is its waiting time in the queue plus its execution time. Therefore, the average response time of the jobs in cluster C_i can be computed as

$$T_i = W_i + \frac{1}{\mu_i}, \tag{3.5}$$

where W_i is the mean waiting time of the jobs in cluster C_i and μ_i is the average job service rate of a node in C_i. The mean response time of the application over these n participating clusters can be computed as

$$T = \sum_{k=1}^{n} P_i T_i. \tag{3.6}$$

We need to find a workload allocation P_i ($i \in \{1 \cdots n\}$) that minimizes T in order to achieve the best mean response time. By applying the Lagrange multiplier theorem [Ber96] to this equation, this problem can be reduced to solving

$$\begin{cases} \sum_{i=1}^{n} P_i = 1 \\ \frac{\partial(\sum_{i=1}^{n} P_i T_i)}{\partial P_k} - \upsilon \frac{\partial(\sum_{i=1}^{n} P_i - 1)}{\partial P_k} = 0. \end{cases} \tag{3.7}$$

Due to the complexity of T_i (Eq. 4.7), it is impossible to find a general symbolic solution for P_i ($i \in [1 \cdots n]$). However, He et al. [HJS+06] have presented an approaching algorithm to calculate

P_i. The basic idea is to use the binary search technique to search for υ and P_i in their respective search spaces to see if the calculated values fit the constrains. For each arbitrary υ in its search space, a set of specific values of P_i can be obtained. Then, the algorithm adds P_i ($i \in \{1 \cdots n\}$) to obtain P_{sum}. If the sum if greater than 1, it means the current υ is too high and a lower value should be used to calculate the set of P_i. On the other hand, if the sum is less than 1, a higher value should be used for the next iteration of the computation of P_i. This process is repeated until the set of values satisfy Eq. 3.7.

Others scheduling algorithms, such as Load-Dependent Static policy (LDS) [BZ92] and Maximum Throughput policy (TP) [CK79], have also been analyzed in terms of their ability to satisfy performance requirements in heterogeneous multiprocessor systems. These algorithms can also be extended to our multi-cluster system. However, due to the complex expression of the load and throughput in multi-cluster system, we will not be able to obtain a numerical solution but we can use the binary search algorithm instead. This forms part of our future work.

3.5 Usage Scenario

One important design goal of **Pelecanus** is to provide application scientists with an easy-to-use interface to hide the complexity of back-end services, such as security checking and execution management. The interface also needs to be flexible, to allow application scientists to fully exploit the resources that are available in a multicluster grid environment.

A typical usage scenario of **Pelecanus** consists of the following steps:

1. By accessing the standalone GUI or the web interface, an application researcher logs into **Pelecanus** and obtains the necessary authorization for using the available resources.

2. The researchers chooses the AEA machine and the participating execution clusters by selecting from the matched results from the cluster abstraction. The AEA machine can also be one of the execution clusters.

3. The researchers specify the number of total task containers to be used for the execution. By choosing the individual scheduling policy, the number of task containers for each participating cluster will be calculated.

4. The researcher chooses which built-in execution pattern the application will adopt. Based on the execution pattern chosen, **Pelecanus** provides an application specification interface that allows the researcher to enter information such as the executable location, the data source, and the workflow description.

5. The GEMS engine, the core of **Pelecanus**, reads the application specification and the set of tasks for execution and then submits these tasks to participating clusters based on the run-time status of the task containers.

6. The researcher monitors the execution progress, and may steer the application workflow as needed, which is enabled by the dynamic load balance strategies adopted by the GEMS engine.

7. The researcher receives the output and an execution report after the application execution is completed.

3.6 Targeted Execution Scenarios

The **Pelecanus** toolkit can be used to manage application executions across many domains, including task farming, inverse modeling, resource co-allocation, and time-critical support. The first two types of execution scenarios have been extensively used by various scientific and engineering applications, such as optimization, Monte Carlo simulations, model validations, etc.

3.6.1 Task Farming

An important category of grid applications is task farming, in which large numbers of somewhat independent tasks are dispatched on remote computing resources. Task farming is simple, yet powerful enough to formulate distributed execution of many application areas such as: radiation equip-

ment calibration analysis, searching for extra-terrestrial intelligence, protein folding, molecular modeling for drug design, etc. However, how to provide efficient scheduling strategies for load sharing the tasks across a grid is still an open research issue.

A parameter sweep is a classic example of task farming. A parameter-sweep application model is a combination of task and data parallel models, and applications formulated to use this model contain a large number of independent jobs operating on different data sets. A range of scenarios and parameters to be explored are applied to program input values to generate different data sets. The programming and execution model of such applications resembles the Single Program Multiple Data model. The execution model essentially involves processing N independent jobs (each with the same task specification, but a different dataset) on M distributed computers, where N is typically much larger than M.

3.6.2 Inverse Modeling

Inverse modeling is a kind of multi-step application execution scenario, where at the end of each step, all the data is collected, and a check is made to see if the data satisfies with a pre-defined criterion. Each step depends on the result of the previous steps. This has been extensively adopted in science and engineering applications. In petroleum engineering, for example, model inversion is important to determine values of model parameters and to make accurate predictions. It is used to calibrate subsurface properties (e.g., porosity, permeability, and hydraulic conductivity) in a subsurface simulation model. In this way, the computed values of observables, such as rates, pressures (or head), and saturations, at different observation locations, are in reasonable agreement with actual measurements of those quantities. Nowadays, the increase in sensor deployment in oil and gas wells for monitoring pressure, temperature, resistivity, and/or flow rate (i.e., smart/intelligent wells) has added impetus to continuous model updating. Instead of simultaneously using all recorded data to generate an appropriate reservoir flow model, it has become important to capture reservoir flow information by incorporating real time data. Automatic and real time adjustment procedure is needed for efficient model inversion.

Since in grid-enabled inverse modeling, users need all the components of previous iteration of inverse modeling in order to initiate the next iteration, computing synchronization is necessary among steps. Without effective computing synchronization, application execution reliability and time-critical updating would be seriously compromised by the *slowest* computing resource(s), due to the nature of grid resources, i.e., geographical distribution, heterogeneity, and self-administration. **Pelecanus** can be used to leverage the effect of slower clusters and achieve synchronization among computing steps.

3.6.3 Co-allocation

Co-allocation is a term that refers to the coordinated allocation of multiple resources that belong to different administrative domains in order to solve a complex problem. Co-allocation is a hot research topic in the distributed computing community. The main challenge of co-allocation is to precisely forecast when a particular task will be executed on a participating resource due to the scheduling policies of the resource. To achieve co-allocation, users need to seek local administration support, for example, resource reservation [Mac07a].

Pelecanus provides a co-allocation strategy suitable for any resource without special local administration involvement. To conduct co-allocation, it needs to submit task containers in advance to remote sites and track the status of these containers. Once all the containers are ready, **Pelecanus** launches collaboration tasks to these containers for execution.

3.6.4 Time-critical support

Many compute-intensive applications require real-time response, e.g., hurricane simulation [BAS+05]. In order to guide the decision making process, meteorologists predict the possible directions of a hurricane according to real-time information. High performance computing is required to conduct such time-critical simulations. However, although traditional grid computing provides massive and cost-efficient computation power, it has weak support for event-driven applications. Usually, dedicated facilities have to be used for this kind of applications.

It becomes possible under **Pelecanus** to run some time-critical applications in a multicluster grid environment. What we can do is just submit TCs in advance, making sure these TCs running before hurricane simulation is launched. Although some resources will be wasted before simulation, this can still reduce the costs of terminating all running and queued jobs. In this way, we build a virtual cluster with computation power partially from each participating cluster, whereas those clusters can still make progress for the jobs submitted by each user.

3.7 Chapter Summary

In this chapter we described a toolkit that we have developed called **Pelecanus** in order to efficiently manage application execution in a multicluster grid environment. The novelty of **Pelecanus** lies in the fact that it adopts the task container and dynamic task assignment techniques to improve resource interoperability, execution management, and application monitoring. The cluster abstraction is implemented to hide the heterogeneity of the underlying systems and provide uniform interfaces for the toolkit to operate with these resources. **Pelecanus** is particularly suited for applications such as simple task farming, inverse modeling, co-allocation and time-critical support.

Chapter 4
Performance Evaluation

Different components of our toolkit, including the DA-TC execution model, cluster abstraction, and task container scheduling are presented in Chapter 3. In this chapter, we will study the performance of our toolkit by comparing with the traditional method that does static load balancing. Theoretical analysis and experiments are carried out and show that our model can provide better scalability and reduced application turnaround time because of dynamic load balancing and reduced queuing time in each application iteration.

4.1 Mathematical Model

The multicluster architecture consists of n local clusters, denoted as C_i, where $i \in [1 \ldots n]$. Each cluster C_i has m_i computing nodes[1] with relative speed s_i. Given two nodes with relative speed s_1 and s_2, if a job takes t units of time on first node, that same job will take $t \cdot \frac{s_1}{s_2}$ on the second node. We assume an average job size of l, which is the completion time of the job when it is executed on an idle machine with relative speed 1. We assume that each cluster is locally homogeneous, which means that the CPUs in each local cluster have the same speed. We further assume that our system is greedy[2].

The multicluster grid we consider has a super-scheduler to which each local cluster is connected. Each local scheduler uses a single waiting queue to accommodate the jobs received from super-scheduler and sends these jobs based on a First-Come-First-Served (FCFS) basis to free processing nodes to execute. The execution is non-preemptive. Each cluster C_i can be modeled using an M/M/m_i queuing model [Kle75] by Kendall's notation, which means arrivals are a Poisson process; service time is exponentially distributed; there are m_i servers; the length of queue in which

[1] We only consider the single CPU per node. It can be easily extended to multicore system if job is scheduled at the core level.
[2] A system is *greedy* if it never leaves any resource idle unless there is no job waiting for that resource.

arriving jobs wait before being served is infinite; and the population of jobs available to join the system is infinite. Suppose in our case, the average job arrival rate to cluster C_i is λ_i (that is, the delay between two successive arrivals follows an exponential distribution with mean λ_i^{-1}); and the average job service rate of a node in cluster C_i is μ_i (the average execution time has an exponential distribution with mean μ_i^{-1}).

4.2 Queue Size and Waiting Time

Let N_i be the number of jobs in C_i (including running and queued jobs). We know from [Nel95] that the probability that N_i equals k is

$$P[N_i = k] = \begin{cases} p_0 \frac{(\rho_i m_i)^k}{k!} & k \leq m_i \\ p_0 \frac{\rho_i^k m_i^{m_i}}{m_i!} & k \geq m_i, \end{cases} \tag{4.1}$$

where p_0 is the limiting probability that the system contains 0 members and is given by

$$p_0 = \left[\sum_{k=0}^{m_i-1} \frac{(\rho_i m_i)^k}{k!} + (\frac{(\rho_i m_i)^{m_i}}{m_i!})(\frac{1}{1-\rho_i}) \right]^{-1}, \tag{4.2}$$

and ρ_i is the utilization of the systems:

$$\rho_i = \frac{\lambda_i}{m_i \mu_i} = \frac{\lambda_i}{m_i s_i \mu}, \tag{4.3}$$

where μ is the baseline job service rate.

The probability that the newly arriving jobs have to join the queue is equal to the probability that all of the m_i nodes are busy. Therefore, if there are k jobs in the queue, then there should be k (in the queue) + m_i (in processing) jobs in C_i. And the probability that there are no jobs in the queue is $P[N_i \leq m_i]$. Hence, the probability of Q_i jobs in the queue is

$$\begin{aligned} P[Q_i = k] &= \begin{cases} \sum_{n=0}^{m_i} P[N_i = n] & k = 0 \\ P[N_i = k + m_i] & k > 0 \end{cases} \\ &= \begin{cases} \sum_{n=0}^{m_i} p_0 \frac{(\rho_i m_i)^n}{n!} & k = 0 \\ p_0 \frac{\rho_i^{k+m_i} \cdot m_i^{m_i}}{m_i!} & k > 0. \end{cases} \end{aligned} \tag{4.4}$$

Using Eq. 4.4, we can find the average queue size of C_i to be

$$\begin{aligned} E[Q_i] &= \sum_{k=1}^{\infty} k \cdot p_0 \frac{\rho_i^{k+m_i} \cdot m_i^{m_i}}{m_i!} \\ &= p_0 \frac{(m_i\rho_i)^{m_i}}{m_i!} \sum_{k=1}^{\infty} k\rho_i^k \\ &= \frac{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{\rho_i}{(1-\rho_i)^2}}{\left[\sum_{k=0}^{m_i-1}\frac{(\rho_i m_i)^k}{k!} + (\frac{(\rho_i m_i)^{m_i}}{m_i!})(\frac{1}{1-\rho_i})\right]}. \end{aligned} \tag{4.5}$$

Then by Little's Law, we can get the expected waiting time in the queue as

$$\begin{aligned} E[W_i] &= \frac{E[Q_i]}{\lambda_i} \\ &= \frac{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{\rho_i}{\lambda_i(1-\rho_i)^2}}{\left[\sum_{k=0}^{m_i-1}\frac{(\rho_i m_i)^k}{k!} + (\frac{(\rho_i m_i)^{m_i}}{m_i!})(\frac{1}{1-\rho_i})\right]}. \end{aligned} \tag{4.6}$$

The job turnaround time is its waiting time in the queue plus its execution time. Hence, the average job turnaround time is

$$E[T_i] = E[W_i] + \frac{1}{\mu_i}. \tag{4.7}$$

From Eq. 4.6 and Eq. 4.7, we can find the ratio of waiting time to the total turnaround time as

$$\begin{aligned} R_i = \frac{E[W_i]}{E[T_i]} &= \frac{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{\rho_i\mu_i}{\lambda_i(1-\rho_i)^2}}{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{\rho_i\mu_i}{\lambda_i(1-\rho_i)^2} + \left[\sum_{k=0}^{m_i-1}\frac{(\rho_i m_i)^k}{k!} + (\frac{(\rho_i m_i)^{m_i}}{m_i!})(\frac{1}{1-\rho_i})\right]} \\ &= \frac{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{1}{m_i(1-\rho_i)^2}}{\frac{(m_i\rho_i)^{m_i}}{m_i!}\frac{1}{m_i(1-\rho_i)^2} + \left[\sum_{k=0}^{m_i-1}\frac{(\rho_i m_i)^k}{k!} + (\frac{(\rho_i m_i)^{m_i}}{m_i!})(\frac{1}{1-\rho_i})\right]}. \end{aligned} \tag{4.8}$$

The above analysis only considers the system in the non-saturated case ($\rho_i < 1$). However, most HPC resources are typically over-committed [NWB08a]. Because it is difficult to predict resource demand in an environment where demand is driven by research, and to also ensure that the utilization of expensive resources is maximized, total user allocations typically exceed feasible occupancy. Therefore, it is worthwhile to consider the system when $\rho_i > 1$.

Let $A_i(t)$ be the random variable of the number of jobs submitted to C_i between 0 and t, and let $D_i(t)$ be the random variable of the number of jobs completing on C_i between 0 and t. Since the

system is saturated, we can always get $Q_i(t) = N_i(t) - m_i$. From [BGJ06], we find

$$\begin{aligned} E[Q_i(t)] &= E[N_i(t)] - \varepsilon(m_i) \\ &= E[A_i(t)] - E[D_i(t)] - \varepsilon(m_i) \\ &\sim_t \lambda_i t - m_i \mu_i t \\ &= \lambda_i t (1 - \frac{1}{\rho_i}), \end{aligned} \tag{4.9}$$

where $f_1(t) \sim_t f_2(t)$ means that $\lim_{t\to\infty} \frac{f_1(t)}{f_2(t)} = 1$.

Therefore, the expected queue waiting time is

$$E[W_i(t)] = \frac{E[Q_i(t)]}{\lambda_i} = t(1 - \frac{1}{\rho_i}). \tag{4.10}$$

Given the execution time as $\frac{1}{\mu_i}$, we can calculate the ratio of waiting time to the total execution time as

$$R_i = \frac{E[W_i]}{E[T_i]} = \frac{(\rho_i - 1)\mu_i t}{(\rho_i - 1)\mu_i t + \rho_i}. \tag{4.11}$$

4.3 System Performance

We will compare the performance of two execution methods. The first one is the traditional method where tasks are directly assigned to the participating clusters, and the number of tasks on each cluster is based on the scheduling policy in Section 3.4. This strategy has been adopted by ResGrid [LHK+06]. The other execution method is based on the DA-TC execution model.

In order to fairly compare the performance of these methods, we use simple task clustering technology [DSS+05] to make sure that the number of jobs[3] assigned to a cluster in the traditional submission method is equal to the number of task containers in the DA-TC model.

4.3.1 Traditional Method

• For Task Farming

Suppose the total number of tasks is N, each with an average task size l, and the total number of task containers (jobs) is J. According to Section 3.4, we know that the number of TCs (jobs)

[3]Multiple tasks are merged into one single job.

submitted to cluster i is $J_i = J \cdot P_i$, where P_i is the fraction of total TCs allocated to cluster C_i based on scheduling.

Therefore, the workload submitted to cluster i is $L_i = \frac{N}{J} \cdot l \cdot J \cdot P_i = N \cdot l \cdot P_i$.

In the following, we assume the tasks we submit are sequential jobs; each TC (job) will take one CPU to execute. For MPI jobs, each TC will occupy multiple CPUs. However, the basic idea is the same. Hence, in the sequential case, if the number of TCs (jobs) submitted to cluster i is less than the number of CPUs (m_i), the expected execution time will be

$$X_i = \frac{L_i}{J \cdot P_i \cdot s_i \mu} = \frac{N \cdot l}{J \cdot s_i \mu}. \tag{4.12}$$

However, if the number of TCs (jobs) submitted to cluster i is larger than the number of CPUs (m_i), then the expected execution time will be

$$X_i = \left\lceil \frac{J \cdot P_i}{m_i} \right\rceil \cdot \frac{N \cdot l}{J \cdot s_i \mu}. \tag{4.13}$$

Since the total execution time in multicluster environments is the maximum execution time of any participating cluster, we can find the turnaround time in this case as

$$T = \max_{i=1}^{n} (E[W_i] + X_i), \tag{4.14}$$

where

$$X_i = \begin{cases} \frac{N \cdot l}{J \cdot s_i \mu} & J \cdot P_i \leq m_i \\ \left\lceil \frac{J \cdot P_i}{m_i} \right\rceil \cdot \frac{N \cdot l}{J \cdot s_i \mu} & J \cdot P_i > m_i. \end{cases} \tag{4.15}$$

• For Inverse Modeling

In inverse modeling, all the data will be collected at the end of each step, and a check is done to see if the data satisfies the stopping condition. If not, further iterations will be used to generate new data. Therefore, the data and settings of the next step depend on the results of its previous step(s). Algorithm 2 provides a simple workflow of inverse modeling. The first step is that large-scale task farming is performed. Once all the tasks are completed, the results are analyzed to see if the stopping condition is satisfied. If not, task farming settings and datasets are modified and the next task farming iteration is undertaken. These steps repeat until the results satisfy the requirements.

Algorithm 2 Logic of inverse modeling

```
while stopping condition unsatisfied do
  task farming;
  if all tasks are completed then
    collect results;
    if results satisfy stopping condition then
      exit 0;
    else
      reset new task sets;
      generate new datasets;
    end if
  end if
end while
```

Suppose there are K ($k \in [1 \cdots K]$) iterations. Since at each step, we need to collect all the results from the previous step, we need to wait for all the results to complete in each iteration, and do the analysis, and then submit the new execution tasks to different local clusters. In this traditional method, each new task submitted has to wait in the queue again to wait for execution. Therefore, we know from the above derivations that in this traditional method, each iteration will take T_k time to execute:

$$T_k = \max_{i=1}^{n}(E[W_i] + X_i). \tag{4.16}$$

Thus, the total turnaround time in this inverse modeling scenario is

$$T_{total} = \sum_{k=1}^{K} T_k = \sum_{k=1}^{K} (\max_{i=1}^{n}(E[W_i] + X_i)), \tag{4.17}$$

where K is the number of iterations, and n is the number of clusters.

4.3.2 DA-TC Method

In the DA-TC execution model, the TCs will hold the resource until all the tasks have finished execution. The AEA will dynamically assign tasks to TCs when the TC status is "Ready". The number of TCs submitted to different clusters is based on Section 3.4.

• For Task Farming

Suppose we order the local clusters based on their estimated queuing time. As showed in Figure 4.1, we have $E[W_1] \leq E[W_2] \leq \cdots \leq E[W_n]$. Since the TCs in clusters C_i $i < n$ will experience

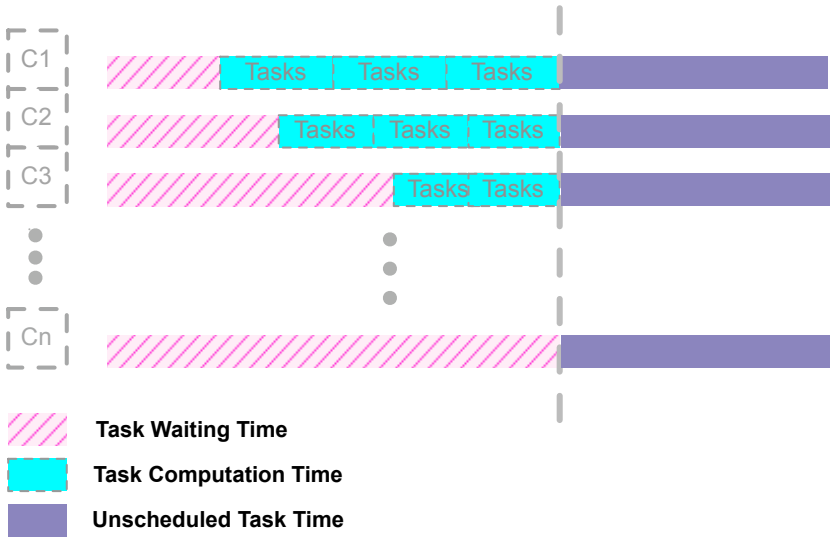


FIGURE 4.1: Waiting and execution time for each local cluster in the DA-TC model.

less waiting time than the TCs in cluster C_n, when the TCs in cluster C_n are allocated resources, the TCs in cluster C_i $(i < n)$ have already executed $\widetilde{N}_i$ tasks,

$$\begin{aligned} \widetilde{N_1} &= \frac{E[W_n]-E[W_1]}{\frac{l}{s_1\mu}} \cdot F_1 \\ \widetilde{N_2} &= \frac{E[W_n]-E[W_2]}{\frac{l}{s_2\mu}} \cdot F_2 \\ &\vdots \\ \widetilde{N_{n-1}} &= \frac{E[W_n]-E[W_{n-1}]}{\frac{l}{s_{n-1}\mu}} \cdot F_{n-1}, \end{aligned} \tag{4.18}$$

where $F_i = J \cdot P_i$ if $J \cdot P_i \leq m_i$, or m_i if $J \cdot P_i > m_i$.

Therefore, when the TCs in cluster C_n are ready to execute tasks, there are $\widetilde{N} = N - \sum_{i=1}^{n-1} \widetilde{N}_i$ tasks left.

In some specific cases, the TCs in the cluster with the longest waiting time may not be assigned any task since all tasks have already been executed before those TCs are allocated resources. This again reflects the advantages of DA-TC model, where slower or long waiting time clusters may receive less or even zero tasks to execute, while faster or short waiting time clusters receive more jobs to execute.

In the DA-TC model, each TC in participating clusters will finish the execution at almost the same time due to the dynamic assignment nature of AEA. Clusters with more computational capability will execute more tasks, while clusters with less computational capability will execute fewer tasks. Thus, we can take the turnaround time of cluster C_n to approximate the turnaround time of whole execution. Since the remaining tasks will be assigned based on the run-time status of TCs in different clusters, the number of tasks assigned to cluster C_n is based on the fraction of its capability to the total system computational capability. Therefore, there will be $\widetilde{\widetilde{N_n}}$ tasks assigned to cluster C_n,

$$\widetilde{\widetilde{N_n}} = \frac{F_n s_n \mu}{\sum_{i=1}^{n} F_i s_i \mu} \cdot \widetilde{N} = P_n \cdot \widetilde{N}. \tag{4.19}$$

Thus, we can calculate the execution time of these tasks as

$$X_n = \frac{\widetilde{\widetilde{N_n}} \cdot l}{F_n s_n \mu}, \tag{4.20}$$

and the total execution time is

$$T = T_n = E[W_n] + X_n. \tag{4.21}$$

• For Inverse Modeling

Since in the DA-TC model, the new tasks in next iteration do not need to want in the queue for resources to execute, the waiting time can be reduced in all remaining iterations. All tasks only need to be in the queue once.

For the first iteration, the execution time should be the same as Eq. 4.21. For next each iteration, the execution time from the workload allocated to cluster C_n should be

$$T_n{}' = \frac{P_n \cdot N \cdot l}{F_n s_n \mu}. \tag{4.22}$$

Thus, the total turnaround time in DA-TC model for this inverse modeling scenario is

$$T_{total} = T_n + \sum_{k=2}^{K} T_n{}' = E[W_n] + X_n + \sum_{k=2}^{K} T_n{}'. \tag{4.23}$$

4.3.3 Performance Comparison

For task farming, by comparing the Eq. 4.14 and Eq. 4.21, we can find that the main speedup comes from the load balancing. Instead of waiting for the slowest cluster to finish the job, the DA-TC's dynamic assignment will be able to ensure the balancing so that all the clusters will be able to finish their assigned tasks at almost the same time. The scheduling policy works as a coarse-grained method to clustering several small tasks to a job to execute on each cluster in traditional method; the DA-TC model serves as a fine-grained method, and the number of small tasks executed in each task container is based on the run-time availability of TC and tasks. It is especially useful when the number of tasks is uncertain at the beginning of the execution.

For inverse modeling, we can gain even more from the DA-TC model. By comparing the Eq. 4.17 and Eq. 4.23, it is easy to find that the DA-TC model can not only provide fine synchronization at each iteration, but also eliminate the waiting time for the following stages. User will benefit from this model for the flexibility of changing the parameters and number of tasks during the execution, and the reduction of the total turnaround time. During this long time execution, the fluctuation of each participating resource will only cause the AEA to automatically rebalance the number of jobs submitted to the cluster at each stage, but will not exert a great influence to the total performance.

4.4 System Evaluation

Without loss of generality, our multicluster grid testbed consists of four Linux clusters with local scheduling system PBS, connected by the Internet[4]. Each cluster can be accessed by Globus GRAM or ssh.

Application scenarios involved in the experiments are task farming and inverse modeling. In task farming, all the tasks are assigned to a multicluster grid and the results are sent back without further processing. In inverse modeling, multiple task farming steps are required and each step needs the results of the previous step(s). These two scenarios are the fundamental methodologies

[4]The experiment can easily be extended to heterogeneous participating clusters with different local scheduling systems and access methods.

in the implementation of many large-scale applications. The experimental load characteristics are described in Section 4.4.4 and Section 4.4.5, respectively.

A multicluster grid is a highly dynamic environment, in which each participating cluster has its own user community with diverse job arrival patterns and workload characteristics. To minimize the effects of uncertainty, we executed the same application multiple times during different time of each day and calculated the mean value of the turnaround time for each configuration. The experiments here focus on compute-intensive applications. The steps for one submission are as follows:

1) Generate application tasks;

2.1) Submit application to the multicluster grid to execute by the traditional and DA-TC methods;

2.2) Repeat Step 2.1 with different configurations;

3) Repeat Step 2 multiple times at different time of each day;

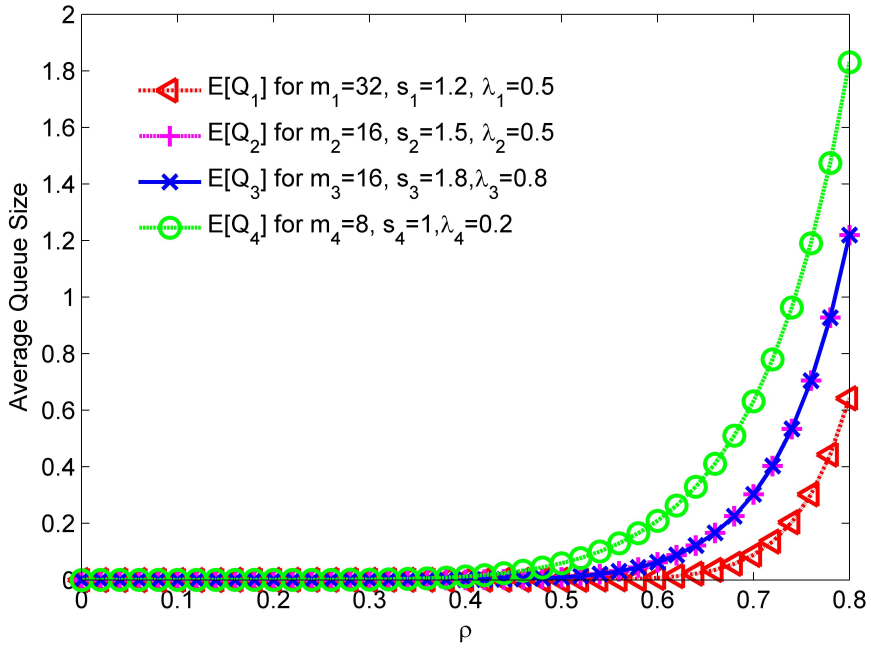
4) Calculate the mean value of the turnaround time with traditional method;

5) Calculate the mean value of the turnaround time with DA-TC.

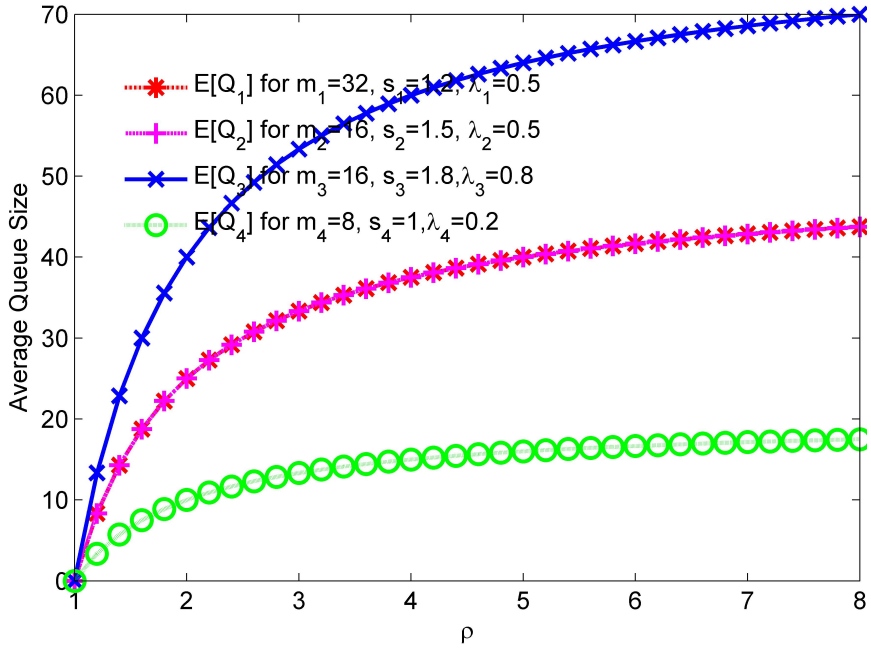
4.4.1 Batch Queue

In order to illustrate the importance of reducing queue time in the total turnaround time, we plot the queue size and ratio of queue time to the application turnaround time in Figure 4.2 and 4.3. The four local clusters have 32, 16, 16, 8 CPUs, and the relative speed is 1.2, 1.5, 1.8, and 1.0, respectively. From Eq. 4.5 we know when $\rho < 1$, the queue size is only related to m_i and ρ, therefore cluster C_2 and C_3 have the same queue sizes with respect to ρ, though they have different relative speed. $E[Q_i] < E[Q_j]$ if $m_i > m_j$. However, when $\rho > 1$, the queue size is related to λ_i and ρ, and this is reasonable since in this case, all the CPUs are busy and all the jobs arrival need to wait in the queue. Therefore, queue size will increase faster when the jobs arrival more frequently.

From Eq. 4.8 we can see that when $\rho < 1$, the ratio of queue time to the application turnaround time is a function of m_i and ρ. Figure 4.3(a) shows that $R_i < R_j$ when $m_i > m_j$. But when $\rho > 1$,

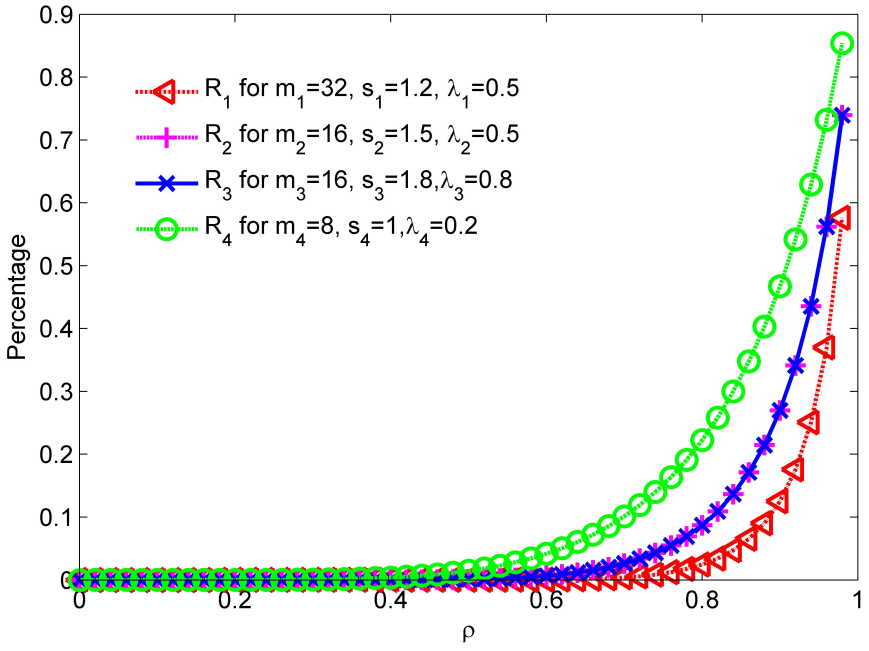


(a) $\rho < 1$

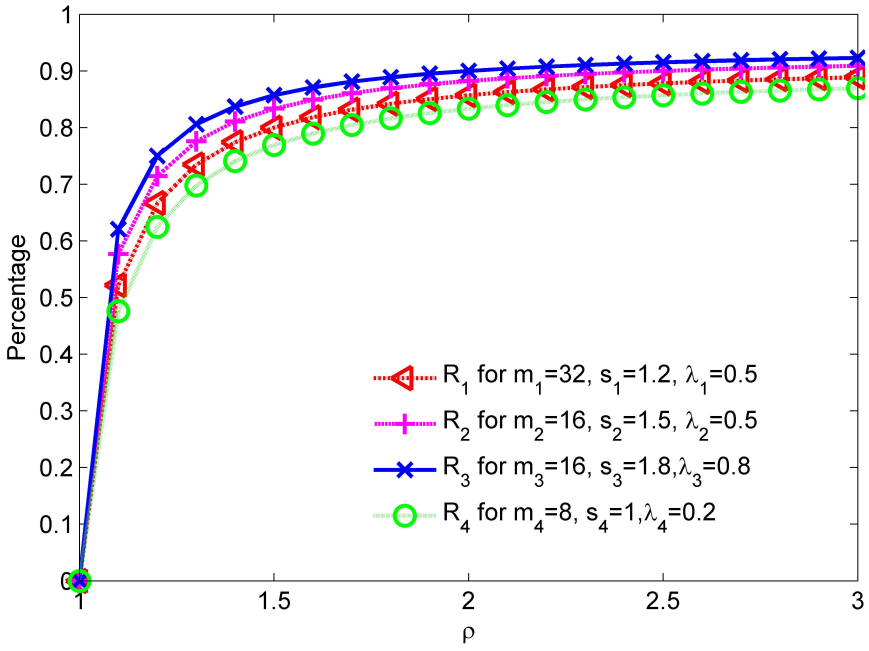


(b) $\rho > 1$

FIGURE 4.2: Queue waiting job size.



(a) $\rho < 1$



(b) $\rho > 1$

FIGURE 4.3: Ratio of waiting time to total turnaround time.

the ratio is a function of s_i and ρ. From Figure 4.3(b) we can see $R_i < R_j$ when $s_i < s_j$. This is because the same task needs less execution time on cluster C_j when $s_i < s_j$, and the ratio, $\frac{WaitingTime}{WaitingTime+ExecutionTime}$, becomes larger.

From Figure 4.3 we can see that under high utilization, each job will experience very long waiting time, and the ratio of waiting time to the turnaround time is very high. Raicu et al. [RZD+07] also conclude that queue time takes the largest portion of the task turnaround time by conducting experiment on the Falkon task execution framework. Normally the execution time is less than 30% of the total turnaround time. Therefore, reducing the waiting time can be a huge acceleration for the whole job execution.

4.4.2 Effect of Number of Participating Clusters

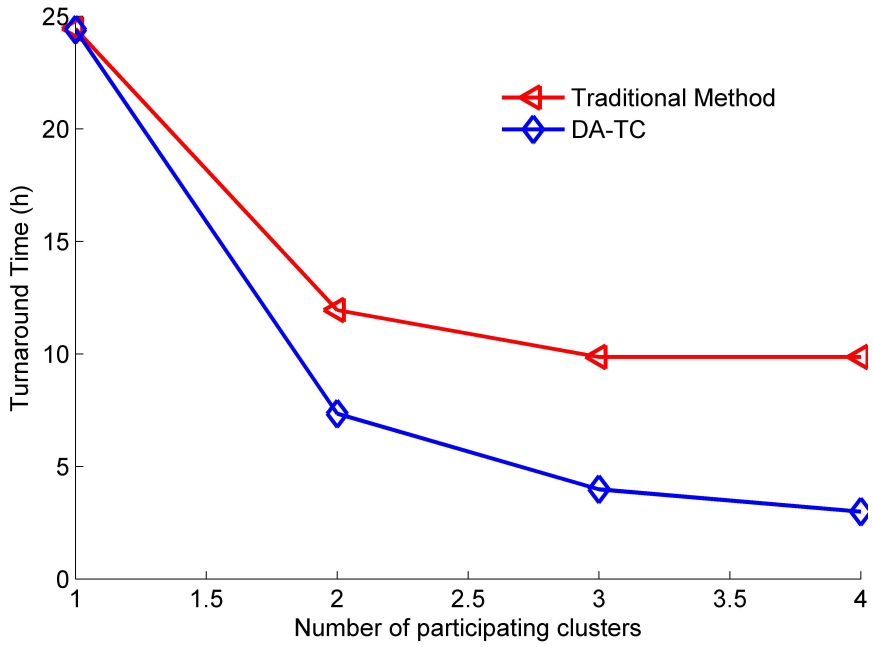


FIGURE 4.4: Effect of different number of participating clusters.

Figure 4.4 illustrates the effect of different number of participating clusters in both the traditional method and the DA-TC model. Table 4.1 lists the parameters of the participating clusters. The total number of tasks is 500 and the number of TCs (jobs) is 60. Cluster 1 is used to get the result for only one participating cluster case, and Cluster 1 and 2 are used to generate the result for two

TABLE 4.1: Parameters of participating clusters

CLUSTER NO.	NO. OF CPUS	RELATIVE SPEED	JOB ARRIVAL RATE
1	8	1	0.2
2	16	1.5	0.8
3	16	1.8	0.5
4	32	1.2	0.5

participating clusters case. For the three participating clusters case, Cluster 1, 2, and 3 are used, and for the four participating clusters case, Cluster 1, 2, 3, and 4 are used.

From this figure we can see that using the traditional method, we cannot always achieve better performance when more clusters are added in. This is because one or more of the participating clusters become the bottleneck during the execution. The total turnaround time can not be further reduced. But in the DA-TC model, the slower clusters become a benefit instead of the bottleneck. Each cluster can make contribution to the total execution. Slower clusters get few tasks to execute, while faster clusters get more tasks to execute. By this model, we can avoid the situation where that faster clusters have already finished all tasks, but users have to wait for slower clusters' results, which happens in traditional method. Therefore, in the DA-TC model, the turnaround time can be reduced when more clusters are used.

4.4.3 Effect of Scheduling

Figure 4.5 shows the performance of submitting the TCs (jobs) based on different scheduling policies introduced in Section 3.4 in both traditional and DA-TC methods. In this figure, the top solid lines show the performance of the traditional method, and the lower dashed lines show the results of the DA-TC method. These four scheduling algorithms are Randomly Assigned, Weighted Workload Allocation (WWA), Shortest Expected Delay (SED), and the Mean Response Time (MRT).

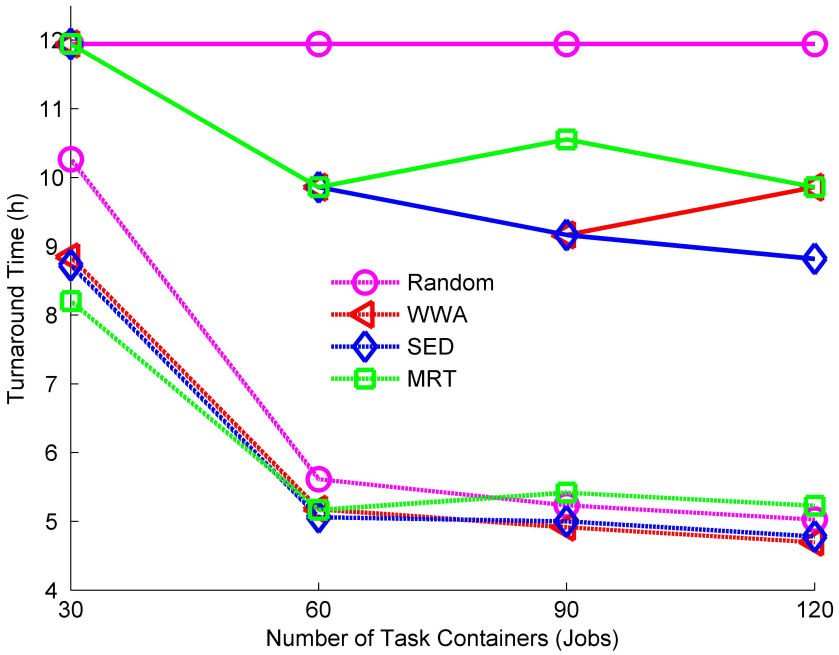


FIGURE 4.5: Comparison of different scheduling policies in traditional and DA-TC methods.

Based on these algorithms and the parameters of the participating clusters listed in Table 4.1, we can get the fraction P_i of total TCs (jobs) submitted to each cluster, as shown in Table 4.2.

From the figure we can conclude the following.

1. In all four cases, the turnaround time of the DA-TC model is much less than the time in the traditional method. This is because the DA-TC model is able to achieve load balancing, making all participating clusters finish the execution at almost the same time. But in the traditional method, no matter how carefully one chooses the scheduling algorithm, the completion time of the jobs in each cluster will be different.
2. The turnaround time will decrease when increasing the number of TCs in DA-TC model. However, in the traditional method, increasing the number of big jobs will make the turnaround time even longer. This is because the large number of big jobs may saturate the machine, even though the executing time of each big job is reduced.
3. The difference of the results among these four policies in DA-TC model is much less than them in traditional method. This is mainly because the DA-TC model implements the static

TABLE 4.2: The fraction P_i of total TCs (jobs) submitted to each cluster

CLUSTER NO.	NO. OF CPUS	RELATIVE SPEED	JOB ARRIVAL RATE	**RANDOM**	**WWA**	**SED**	**MRT**
1	8	1	0.2	0.25	0.08	0.12	0.05
2	16	1.5	0.8	0.25	0.24	0.29	0.33
3	16	1.8	0.5	0.25	0.29	0.40	0.20
4	32	1.2	0.5	0.25	0.39	0.19	0.42

submission of the TCs but dynamic submission of each task. Because of the dynamic assignment of each task, the random submission, which is a horizontal line in the traditional method, can still have similar performance as other policies in the DA-TC method.

4.4.4 Effect on Task Farming

We investigate the effect on task farming from two aspects: 1) an identical application submitted with different number of task containers; 2) the number of task containers fixed with different size of applications. Due to the almost identical performance achieved in the DA-TC model, we use the weighted workload allocation algorithm as the scheduling policy for the following experiments.

• **An Identical Application Submitted with Different Number of Task Containers**

The application consists of 500 tasks. The execution time range of each task is from 1 minute to 30 minutes. The execution time of a single task is generated randomly. If this application was submitted on a single CPU, the total CPU time would be $(15 \times 500) \div 60 = 125 hours$.

We submit the application with different numbers of task containers (jobs) (30, 60, 90, 120) and obtain the results shown in Figure 4.6. We also plot the results predicted by theoretical analysis from Eq. 4.14 and Eq. 4.21 to compare. The following three conclusions can be drawn:

1. Application turnaround time for the DA-TC model is significantly improved over turnaround time in the traditional method.

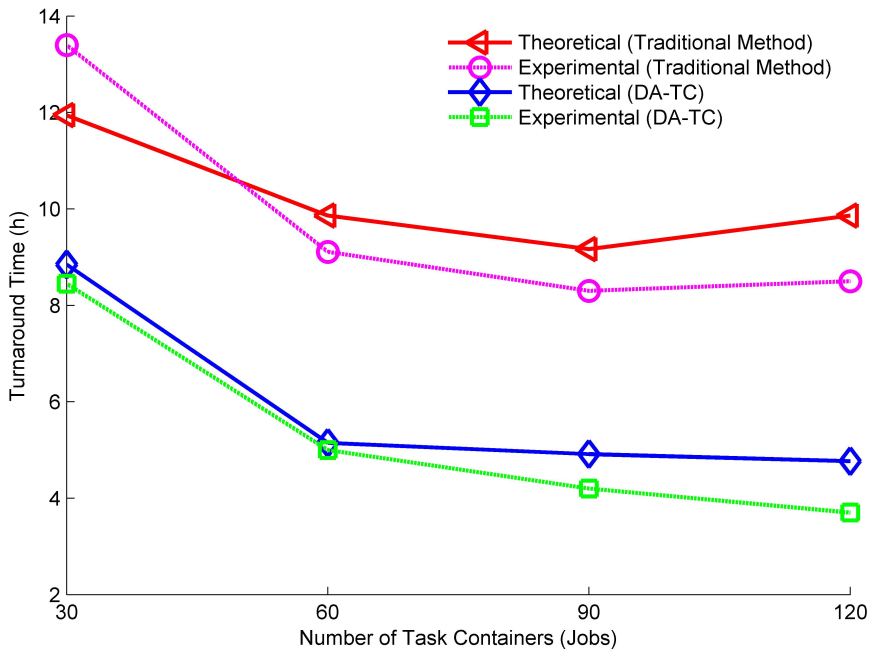


FIGURE 4.6: Task farming comparison between DA-TC and traditional method: an identical application submitted with different number of task containers.

2. DA-TC shows the nature of scalability, which means that turnaround time can be decreased by increasing the number of task containers. A multicluster grid with more participating clusters can provide better performance for application execution.

3. The turnaround time in the traditional method is even longer with 120 task containers (jobs) than with 90 task containers (jobs). The possible reason is that the performance of one participating cluster becomes the bottleneck. The traditional method can not always achieve better performance when more number of clusters are participated.

• Fixed Number of Task Containers (Jobs) for Different Sizes of Applications

Four applications with fixed number of task containers (jobs) are submitted. The fixed number of task containers (jobs) is 60. The four applications have 200, 400, 600 and 800 tasks, respectively. The execution time range of each task is from 1 minute to 30 minutes. The execution time of a task is generated randomly. If these applications were submitted on a single CPU, the total CPU times would be 50, 100, 150 and 200 hours, respectively.

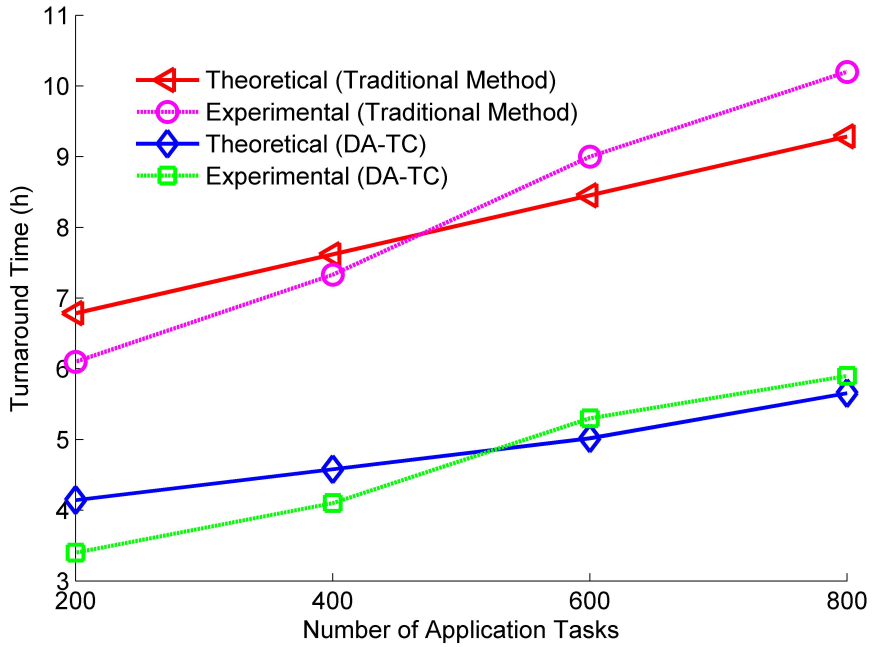


FIGURE 4.7: Task farming comparison between DA-TC and traditional method: the number of task containers (jobs) fixed with different size of applications.

Figure 4.7 shows the the results of the application executions. It is obvious that the turnaround time is greatly reduced if applications with different size are submitted via DA-TC, instead by the traditional method.

4.4.5 Effect on Inverse Modeling

In order to simplify our experiments without loss of generality, we use a fixed number of iterations of task farming to simulate automatic inverse modeling execution. We performed four inverse modeling processes with 3, 6, 9 and 12 iterations, respectively. Each iteration has 100 tasks. The execution time range of each task is from 1 minute to 30 minutes. The execution time of a task is generated randomly. If the tasks of one iteration were submitted on a single CPU, the total CPU times would be 25 hours. The number of task containers (jobs) is 20.

Figure 4.8 demonstrates the experimental results comparing the turnaround time under the two different submission strategies: DA-TC and the traditional method. We observe that when the number of iterations increases, the turnaround time increases much faster under the traditional method than under DA-TC. The major reason is that task containers in DA-TC never release the resources

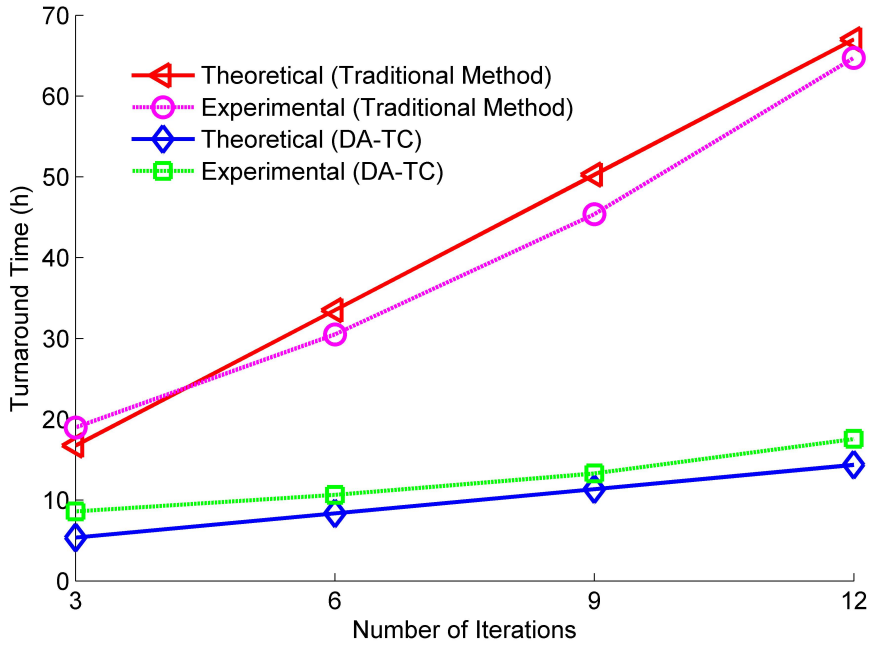


FIGURE 4.8: Experimental results to compare turnaround time of automatic inverse modeling under two different submission strategies: DA-TC and traditional method.

until the whole automatic inverse modeling process is completed, but for the traditional method, the tasks in each iteration are submitted to the end of local scheduling queues. In the DA-TC model, all runs in a container have only one queue wait, and dynamic load balancing also speeds up the execution.

4.5 Chapter Summary

In this chapter we presented the performance evaluation of our DA-TC method in multicluster grids. We first show that reducing the queuing time can be a huge enhancement for the total application turnaround time. Then, we compared the performance of two different execution methods: with and without DA-TC, both theoretically and experimentally. Experiments show that our DA-TC model can easily eliminate the bottleneck caused by the slow clusters during the execution and make them contribute to the total application execution, therefore providing better scalability than the method without DA-TC. Application turnaround time can be significantly reduced since there is no need to queue each individual task, and also because of the dynamic load balancing

technique. This model is very useful when the number of tasks is uncertain at the beginning of the execution. It also provides the user the ability to change the workflow during the execution.

Chapter 5
Application Case Studies

Our toolkit has been successfully deployed on multiple applications across various disciplines, such as renewable natural resources, petroleum engineering, mechanical engineering, and computational chemistry, etc. In this chapter, we will introduce these compute-intensive applications and describe the challenges and how our toolkit can benefit these applications by simply utilizing our toolkit to execute the applications across the multicluster environment.

5.1 Collaborative Mechanical Design

Computer-aided mechanical design is widely employed in the modern world, and it is one of the most active research areas in the academic and industrial sector. It deals with design of almost everything, from home devices to spacecraft. The major concerns are designing the product to meet functional requirements and ensuring that the product is sufficiently robust and that it will meet various constraints, i.e., weight and volume, material availability and production functionality. In general, mechanical design is an optimization process for achieving the desired product functions under multiple constraints.

A typical computer-aided mechanical design has three major phases, as shown in Figure 5.1: (1) *Preliminary Design (e.g., Geometry Design)*: Engineers create a set of preliminary models that have the required functions and meet the given constraints, based on the previous experience, professional standards, and technical regulations. (2) *Structure Verification*: After creating the preliminary models, FEA simulations [ANS] are conducted to check if the preliminary models are robust enough to endure given working conditions. (3) *Performance Evaluation*: CFD simulations [Flu] are executed if flow is involved when final product is in use. This phase can occur prior to the Structure Verification phase in some circumstances.

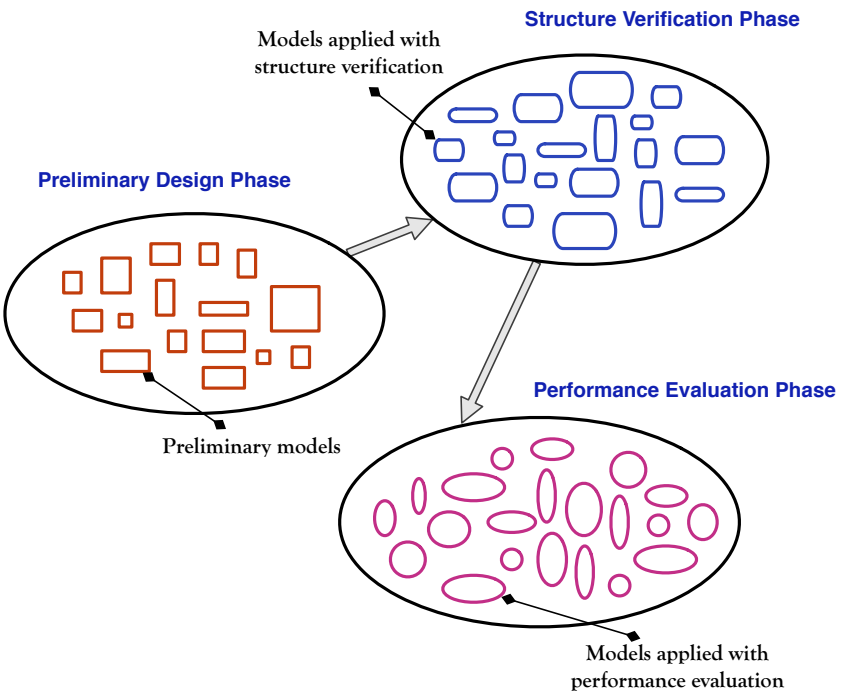


FIGURE 5.1: Mechanical design strategy.

Traditionally, only a very small portion of preliminary design models are presented to the second and third design phases. The selection of a design to next stages is based on whose major features are fairly close to the final requirement [Tho04]. The major advantage of this design strategy is apparent: it cuts the computation time of the two design phases that follow. But it risks the possibility of eliminating good or even the best models at the very beginning, without further evaluation. The cost of such risks can sometimes be unaffordable.

5.1.1 Challenges

If sufficient computational resources are available, it is unnecessary to eliminate any preliminary models in the first design phase, as this could lead to the loss of valuable results. All possible promising preliminary models could be sent to the structure verification and performance evaluation phases, and all possible combinations of working conditions could be evaluated. This new design idea ensures that the best designs would not be discarded from the design process due to lack of computation power. However, the FEA and/or CFD simulation of a typical single model

with a one-foot length in each of the three dimensions takes several hours on an ordinary desktop. Suppose 100 preliminary models are available, and 3 groups of parameters in the structure verification phase and 3 working conditions in the performance evaluation phase need to be evaluated for each model, then there are $100 \times 3 \times 3$ cases to be studied. Even assuming that each procedure of structure verification and performance evaluation only takes one hour, 900 hours would be needed to finish the job. A grid-enabled simulation environment is one method than can be used to reduce the execution time.

5.1.2 System Realization

We have implemented a DA-TC based problem solving environment to meet the challenges explained above. This tool benefits mechanical engineers by: 1) equipping mechanical engineering simulations with supercomputing capabilities and reducing simulation time; 2) making it possible to use a grid environment to evaluate all possibilities, making sure that the best designs will not be discarded during the design procedure.

The DA-TC enabled system workflow is shown in Figure 5.2. Before initiating the simulations, a set of input files that define major geometry parameters, physical properties, and other coefficients needed in design procedures should be first specified. These data, in input files, are referred to as configuration parameters for the program. Then, numerical parallel simulations are invoked across grid computing resources, generating results for further analysis. The whole procedure includes:

1. **Data Preparation.** A local directory, containing a work directory, is created on the AEA machine. The input geometry file and other input files required by the preliminary modeling, FEA, and CFD phases are saved in this directory.

2. **Scheduling.** The user specifies which clusters will be used to run the simulations. The cluster abstraction provides the information needed to make the scheduling decision. The scheduling service uses this information to decide on the number of task containers submitted to each participating cluster.

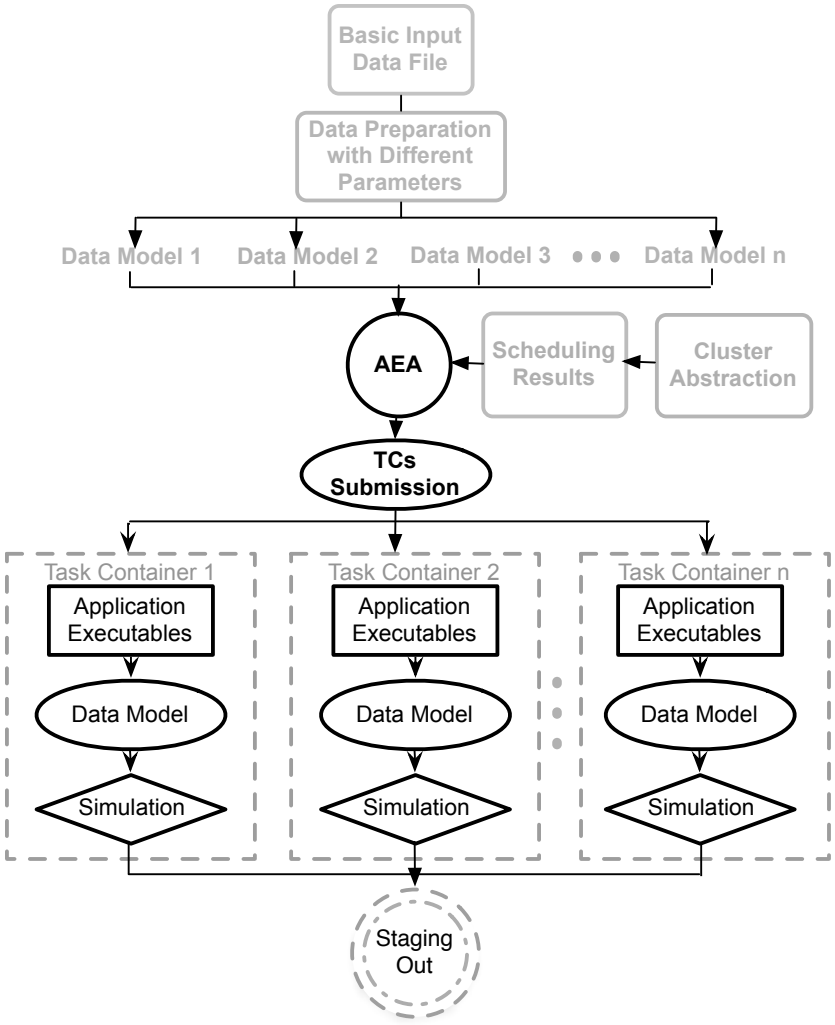


FIGURE 5.2: Execution management.

3. **Task container submission.** The AEA submits the task containers based on the scheduling results.

4. **Application executables and data stage-in.** Once a task container is active (has gone through the queue) on the resource, it will communicate with the AEA to update its status. The AEA will assign the next task to this container. The container can stage-in the executables and the assigned task from the position defined by the AEA. The AEA can dynamically assign tasks to different containers based on the run-time status of these task containers.

5. **Job Execution.** All computations contain preliminary design, structure verification and performance evaluation. The preliminary design phase takes its input from the remote data storage. The output of the preliminary design is the input for structure verification. The performance evaluation phase takes its input directly from the input geometry file.

6. **Retrieving Results.** The computational nodes write the results back to the data repositories.

7. **Stage-Out.** The results are transferred back to the AEA machine for further analysis and visualization.

5.1.3 Case Study

One scenario in mechanical design is to design a wall to protect peoples' lives and properties downstream of a dam in case the dam is broken, as shown in Figure 5.3. Engineers first need to select the cross-sectional shape of the wall, the dimensions of the wall, the distance from the dam to the wall, and the material to be used. There are multiple options for each variable, which can be combined together to form hundreds of possible models. For instance, the cross-section shape can be a rectangle, triangle, trapezoid or other forms; the wall material can be concrete or metal; the wall dimensions and distance from the wall to the dam can be set to many different values, as well as multiple water levels in the dam.

Next, performance evaluation is performed on each preliminary model of the protective wall. For each preliminary model, CFD results provide information about how soon water will impinge on the wall once the dam is broken, what pressure the water flow will cause on the wall, what the flow velocity will be when water reaches the wall, and what the water flow pattern will be at the downstream section of the wall.

Based on preliminary models of the protective wall and CFD results (i.e., pressure distributions on the wall surface), FEA is then conducted on each preliminary model to examine stress distributions in the wall. Different wall materials are checked in this phase. Structural flaws may be found, and the lifetime of the designed walls can be predicted. Qualified models will be selected,

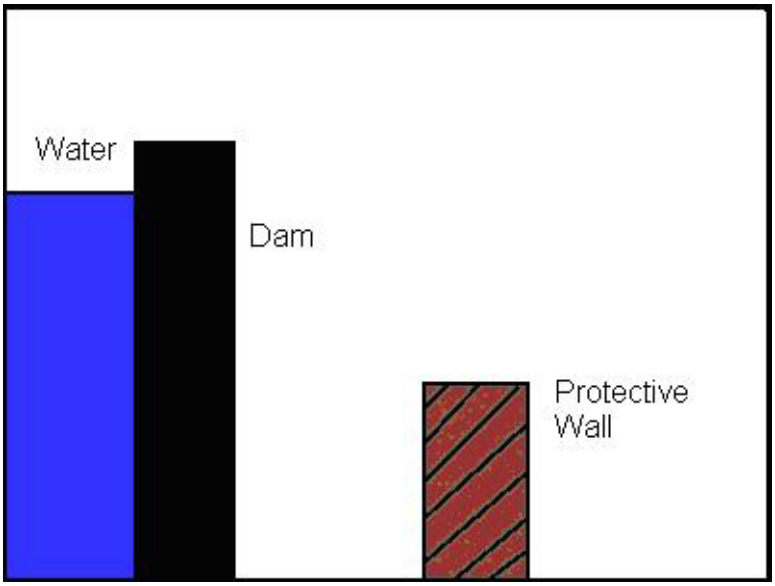


FIGURE 5.3: Design protective wall for preventing flood damage.

from which the best model can be found. Otherwise, engineers have to return to preliminary design phase again to make improvements.

This case study has been implemented and was used to validate the architecture addressed above. It was implemented by using three free software packages, e.g., Gmesh [GR06], Z88 [Rie06], and OpenFOAM [Ope] for the Preliminary Design, Structure Verification, and Performance Evaluation phases, respectively. All these software toolkits are open-source and were pre-installed on the computational nodes.

Data Preparation. In the beginning, all the parameters have to be specified in the *parameter* files, including input geometry file name, dimension of the object, case name, etc. In this case study, parameters such as height and width of the wall, shape of wall cross-section, algorithms to be used in OpenFOAM and Z88, dam water level, and materials to be used to create dam, are required in these parameter files.

The selected geometries are shown in Table 5.1, where H and W represent height and width, respectively. In this case study, there are $10 \times 5 \times 3 \times 2 + 10 \times 5 \times 3 \times 2 + 20 \times 5 \times 3 \times 2 = 1200$ geometry models in total. So the user needs to define 1200 files in the AEA machine, where each

file contains the parameters for one *H* and *W* combination, one distance value, one water level value, and one material value.

TABLE 5.1: Pre-defined geometries

CROSS SECTION	WALL DIMENSION	DISTANCE	WATER LEVEL	MATERIALS
Rectangle	Ten *H* and *W* combinations	Five values	Low, Normal, High	Steel, Concrete
Triangle	Ten *H* and *W* combinations	Five values	Low, Normal, High	Steel, Concrete
Trapezoid	Twenty *H*, upper and bottom *W* combinations	Five values	Low, Normal, High	Steel, Concrete

Task Container Submission. The user can also define the total number of task containers. The AEA submits the TCs based on the scheduling decision. In this case, we submitted ten task containers to Eric, Louie, Oliver, and Queen Bee (systems in our LONI [LON09] Linux cluster environment).

Execution. After some TCs acquire resources, the AEA will assign the task to these TCs. Normally we name user-created files by number, so in this case, we name them from *1* to *1200*. The AEA assigns the tasks in ascending order. For example, if task container 4 is running parameter file 345, and at this time task container 1 is ready to run, the AEA will assign file 346 to task container 1 to execute.

As shown in Figure 5.4, there are several major steps in this simulation: mesh file generation; mesh file conversion; geometry file conversion; structure verification, and performance evaluation. *Preliminary Design.* The user-defined input file is sent to the Mesh module. The module generates the *.geo* file using the parameters specified in *parameter* file. This *.geo* file defines the shapes of the

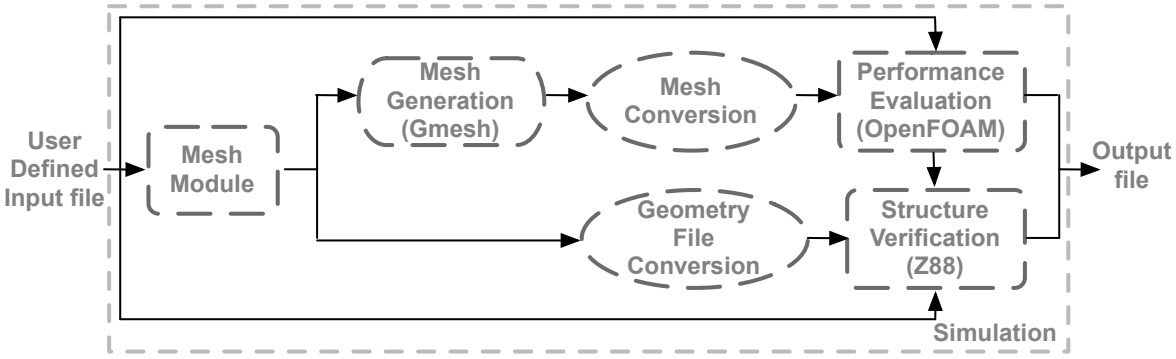
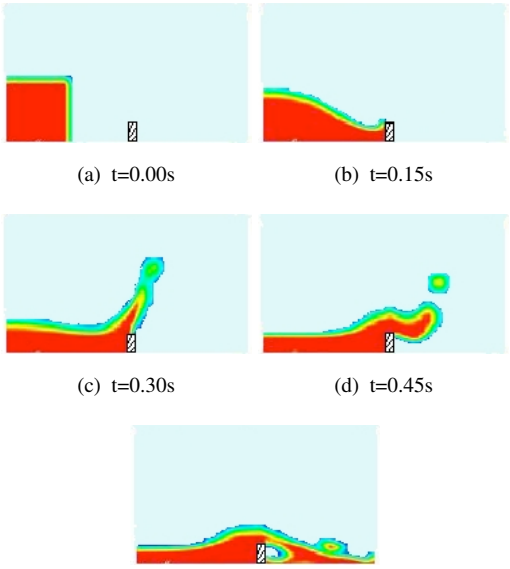


FIGURE 5.4: Simulation workflow.

wall cross-section, the dimensions of the wall, the variable distance between the dam and the wall, and the instructions for how to mesh the geometry. Then, Gmesh reads in the *.geo* file to generate the mesh for the 2D and 3D models. With the *.geo* input file, Gmesh works as a CAD software tool in this case.

Performance Evaluation. The mesh files from Gmesh are converted into OpenFOAM mesh format. Then an OpenFOAM solver (InterFoam in this case) reads flow properties, and initial and boundary conditions from the input files. The input file provides the initial conditions, with three values of water level: low, normal, and high, specified. The results of CFD simulations include water-air distributions in the space studied, flood velocity fields, and pressure distributions on the wall surfaces. Figure 5.5 shows how the water flow impinges on one of the wall designs when the dam is broken at a normal water level, and the flow patterns prior to and after the water reaches the wall.

Structure Verification. The model geometry from the *.geo* file is imported, and meshed into Z88 format. The input file containing material properties, boundary and/or initial conditions, is also needed for Z88. The pressure field data from OpenFOAM is sent to the Z88 module and merged with the Z88 input files to provide boundary conditions. Z88 solvers then read the mesh files and input files, and calculate displacements, stresses, and nodal forces of the wall. Two types of wall material are tested. One is steel and the other is concrete. By evaluating the FEA results, the qualified walls are identified for different water levels, and the wall lifetime can also be predicted,

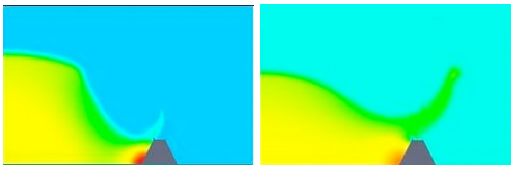


(a) t=0.00s (b) t=0.15s

(c) t=0.30s (d) t=0.45s

(e) t=0.55s

FIGURE 5.5: CFD simulations of the flow impinging on a protective wall with rectangular cross-section.



(a) t=0.015s (b) t=0.020s

FIGURE 5.6: The pressure distributions around the protective wall, where the red color indicates higher local pressure.

based on structural stress distributions. Figure 5.6 shows the pressure distributions around the protective wall.

This specific experiment introduced 1200 parameter combinations. By using our DA-TC execution model, we were able to finish the entire simulation within 100 hours. This is not just because each node in our LONI clusters is more powerful than a desktop system, but also because of the collaborative effort of multiple task containers across the participating clusters and the reduction of queuing time of each job during the execution.

5.2 Sawing Optimization

Trees are products of nature. No two trees are the same. Logs produced from trees are therefore different from each other. Externally, they are different in size, shape and straightness. Internally, defects such as knots, decays, worm holes, are distributed in various parts of the log. Sawyers have the unenviable job of, in a few seconds, taking a quick look at the external shape of the log, guessing what is inside, and proceeding to cut the log into lumber. Invariably, mistakes are made in guessing what defects are inside the log, and the sawing sequence used to cut the log fails to extract the maximum value from the log. What is needed is a sawing optimization system that can see both the external shape and internal defects of a log, determine the best way to cut the log into lumber, and then proceed to cut the log accordingly to capture the maximum value from each log. Fortunately, in the 1980s, computed tomography (CT) scanning technology was developed, which presents the opportunity to acquire cross-sectional images of logs without destroying the log itself [FB87, TWJ+84]. By using advanced CT log scanning technology, developing software packages devoted to simulated sawing of virtual logs reconstructed from CT images became possible [BFT02, SLA96]. TOPSAW, a **T**raining and **Op**timization system for **SAW**ing Logs, was developed to find the optimal cutting pattern for each log based on CT images.

• Description of TOPSAW

The TOPSAW program uses CT scanned images to reconstruct a virtual log. As the schematic for TOPSAW shows in Figure 5.7, the TOPSAW sawing optimization system will first scan a log with an X-ray CT scanner to acquire its cross-sectional images. These log scanning images are combined to reconstruct the 3-D virtual log on the computer. From there, TOPSAW generates full length cut-faces at various rotational angles at various depths. The sawing optimization software then identifies the internal defects in the log and boxes the defects on the cut-face, and then grades the lumber to be cut according to the National Hardwood Lumber Association (NHLA) grading rules. The software also winnows thousands of sawing options to quickly find the optimal sawing sequence to maximize the value realized from each log. The best sawing sequence generating the

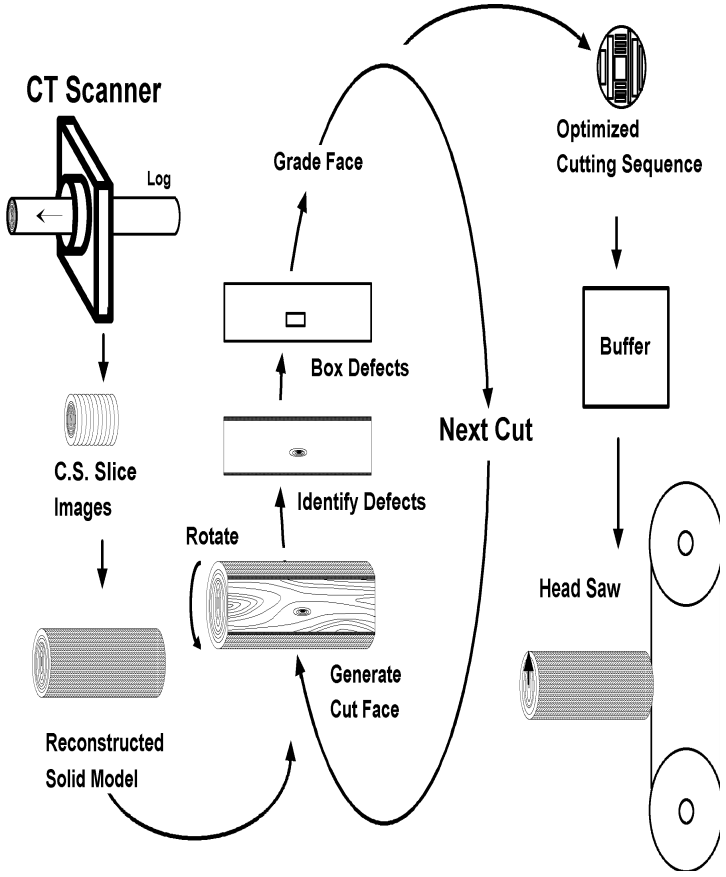


FIGURE 5.7: The schematic of the TOPSAW sawing optimization system.

highest total lumber value will be stored in the memory and a barcode corresponding to the optimal cutting pattern is attached to the scanned log and sent to the actual saw for cutting. The total value of the boards estimated by the software matched 97% of the value produced at the sawmill. This laid the foundation for TOPSAW to be used as a practical tool to analyze different sawing patterns and to determine the value of lumber thus produced [GC98].

• Log Scanning and Defect Detection

Due to the weight limitation of medical scanners, in the past nearly all log scanning has been carried out in short sections. At LSU, we recently had the opportunity to scan eight logs with 12

feet long and 12 to 14 inches in diameter, at full length. The images from these logs now provide valuable data for TOPSAW.

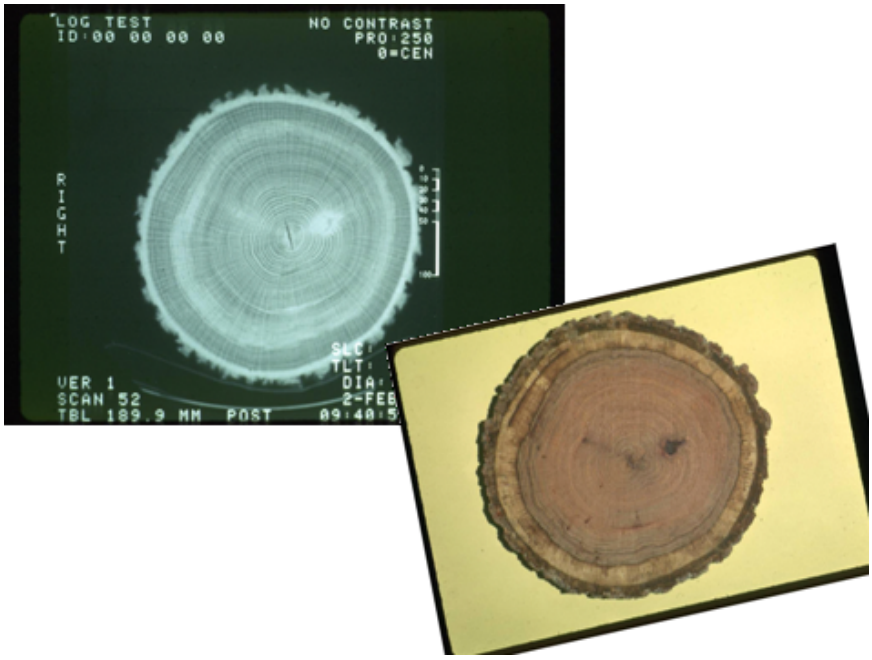


FIGURE 5.8: A matched set of an X-ray CT scan image and a photo picture of a cross section of a red oak log.

Most of the internal defects inside a log can be identified correctly. For example, presented in Figure 5.8 is a matched set of an X-ray CT scan and the actual photo picture of a cross section of a log. As shown in Figure 5.8, X-ray CT scanning of logs provides highly accurate images of not only the external shape of the log but also of internal defects and features. From the image we can clearly see the delineation of the sapwood and heartwood, scar tissue around the 5 o'clock position, a large knot at the 2 o'clock position, a trace of a knot at the 10 o'clock position and a heart check in the center of the log.

The scanning for each log was performed at 0.5-inch intervals. Image acquisition was performed at the rate of one image per second. The cross-section image resolution was 512 by 512 pixels. After combining a series of the CT log scanning images, the 3-D virtual log picture can be reconstructed and the cut-faces of the log can be generated for various rotational angles and at any depth. The software then identifies the internal defects in the log, boxes the defects on the cut-face,

and grades boards following the sequence used in the sawmill. The saw blade positions used in the software were the same as those recorded at the sawmill. Based on this information, TOPSAW can determine the optimal sawing sequence and examine the effects of different sawing decisions. Figure 5.9 gives an example of cutting a 3D virtual log. The green boxes indicate the good condition of this piece of lumber, while red boxes indicate defects.

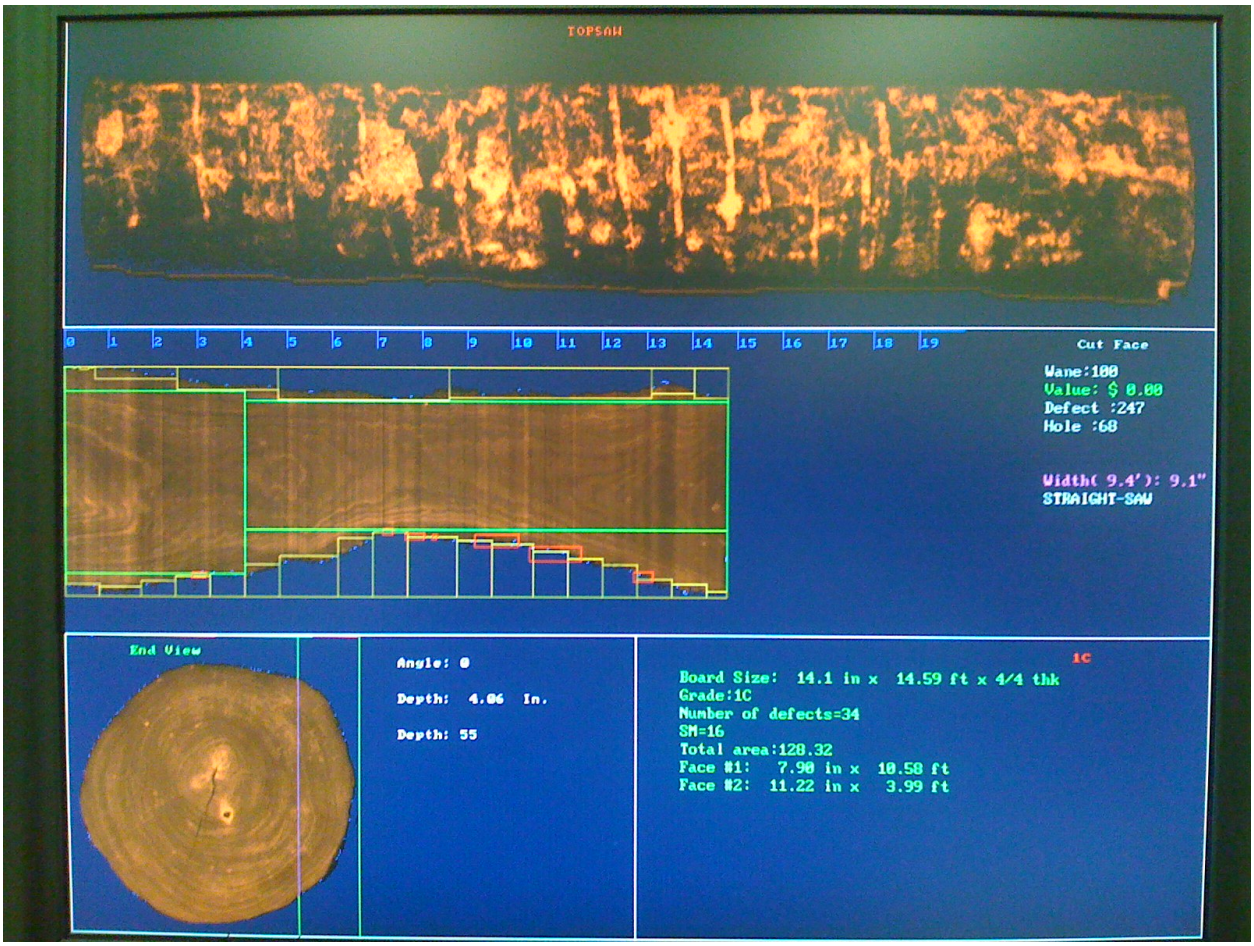


FIGURE 5.9: The image of cutting 3D virtual log.

5.2.1 Challenges

We know, from previous work, that the optimal rotational orientation of the log and the depth of opening cut can significantly affect the value of lumber produced [HWS+91, SWKA93]. Therefore, in order to make the optimal cut, we need to rotate the log to the correct orientation, and then, position the log at the correct depth with respect to the saw. Figure 5.10 shows three ways to do the

FIGURE 5.10: Live, bi-directional, and grade sawing of a log.

sawing. Live sawing simulations were conducted by rotating the virtual logs in 15-degree increments for sawing in 24 different rotational orientations. For each rotational orientation, instead of starting with an opening face of a specific width, we shifted the depth of the opening cut from the edge of the log in 1/16-inch increments for 18 different opening cut depths to determine the best opening cut face. The 18 depths included 16 pixels for the thickness of one 4/4-inch board plus 2 pixels for the 1/8-inch saw kerf. The 432 (24×18) combinations of the rotational orientation $\times$ depth-of-cut for each log provided an exhaustive study of the effects of rotational orientation and depth of the opening cuts on the value of hardwood lumber produced. Simulation results showed that TOPSAW could increase the value of the lumber produced by 15% over actual sawmill production under live sawing. Bi-directional sawing allows the log to be sawed in two directions and produced an additional gain of 8% over just live sawing. Grade sawing permits sawing in four directions; a further gain of 3 to 5% is possible. However, with each log rotating at 15 degrees increment there are 6 different log orientations per log to consider. Within each log orientation, each one of the four sides of the log has 17 different depths of opening cut to consider. All together, there are 501,126 (6×17^4) possible initial sawing combinations. Solving all these combinations to find the optimal solution sequentially on any computer is not realistic. That is the main reason we propose to use grid computing for sawing optimization, as this can keep the cost of computational resources small enough for it to be practical. However, due to the inhomogeneity and variability of the grid, it is challenging to efficiently adapt to this environment. Furthermore, our simulation is a multi-step process: at each step, we need to examine the results and adjust certain parameters

before running the next step. This will result in an unbearably long waiting time during the execution. Therefore, we took advantages of the DA-TC execution model to overcome the challenges in grid computing for our TOPSAW optimization.

5.2.2 System Realization

Before running, TOPSAW needs a set of parameters: the CT scanned data file of the log of interest, the wane percentage, number of blades to be used for sawing, the initial angle of orientation, and the depth of the opening cut. The whole procedure of grid-enabled TOPSAW includes:

1. **Data Preparation**: in this first stage, a local directory is created on the AEA machine. The input CT log data and other input files required by simulation are saved in binary or plain text files in this directory.

2. **Data Staging**: in this second stage, the local directory built in the first stage is replicated on the remote storage sites.

3. **Task Farming**: in this third stage, TCs are assigned to the available computation resources, according to the workload allocation decision.

4. **Job Execution**: in this fourth stage, the required command files are generated and submitted to the remote computational machines when TCs have acquired resources.

5. **Retrieving Results**: in this fifth stage, the computational nodes write their results back to the remote data repository, and the user can find the best solution from all the output files and put it in a file.

6. **Staging Out**: in this sixth stage, this file containing the best optimal solution will be transferred back to the user's local machine for it to be sent to the actual sawmill cutting.

In the DA-TC execution model, once a task container has been allocated resources, it persists as a job on the remote resource and therefore retains the resources until all members assigned

to the container are completed. Thus, all runs assigned to a container have only one queue wait, and dynamic task assignment makes the containers with fast speed execute more tasks (dynamic load balancing); with many members and assimilations per container, this greatly reduces the total queue time. Our experiment shows that the average percentage of turnaround time improvement is up to almost 60%, which significantly reduces our simulation execution time, and makes it applicable in an industry environment.

5.2.3 Results and Discussion

With the help of grid computing and the DA-TC execution model, we are able to analyze the effects of optimal rotational orientation and optimal depth of the opening cut on the final lumber values.

• The Effects of Optimal Rotational Orientation and Depth of the Opening Cut

The maximum and average lumber values produced from all combinations of log orientation $\times$ depth-of-cut for each log are presented in Table 5.2. With 2007 lumber prices, the maximum value significantly exceeded the mean lumber value produced by the average of 14.7% (ρ = 0.0003). Furthermore, the potential gains in lumber value for grade 1 and 2 logs were quite similar; with 14% gain for the former and 15.4% gain for the latter. The results suggest that sawing 4/4-inch lumber with the optimal log orientation and opening cut depth could potentially increase lumber value by as much as 14.7% when compared to a totally random positioning of the log, regardless of log grade.

Given the 14.7% gain in lumber value, it is important to determine whether optimizing rotational orientation or optimizing the depth of the opening cut contributes more to this gain. The following two analyses address this issue.

• The Effects of the Depth of the Opening Cut When the Log is Positioned at Its Optimal Rotational Orientation

To determine the effects of the depth of the opening cut on lumber value when the log is rotated to its optimal orientation, the results from the optimal log orientation were further analyzed. As shown in Table 5.3, given the optimal rotational orientation for each log, the average lumber value

TABLE 5.2: Description of the seven red oak logs

Log No.	Grade	Length (ft)	Avg.lumber ($)	Max.lumber ($)	Difference (%)
1	1	11.8	83.10	94.01	13.1
2	1	12.0	78.62	92.18	17.3
3	1	12.0	99.88	111.98	11.6
4	2	12.7	86.90	99.84	14.9
5	2	12.4	57.68	67.71	17.4
6	2	12.3	47.91	55.67	16.2
7	2	12.3	60.01	67.82	13.0
				Avg.	14.7
				SD	2.3
				p	0.0003

produced from all 18 depths significantly exceeded ($\rho = 0.0237$) that from all 432 combinations of rotational orientations and opening depth by 5.0%. Furthermore, given the optimal log orientation, if the opening cut depth is positioned within $\pm$ 1/8 inch of the optimum, then even more gain can be obtained.

• **The Effects of Log Rotational Orientation Given the Optimal Depth of the Opening Cut**

As shown in Table 5.4, given the optimal opening cut depth, the average lumber value produced from all 24 log rotational orientations was only 0.5% larger than that from all 432 combinations of rotational orientation and depth of the opening cut. This difference was not statistically significant ($\rho = 0.26$). However, given the optimal opening cut depth, if the log can be rotated within $\pm$ 30 degrees of the optimal orientation, we can obtain more lumber value.

The above result suggested that optimal depth-of-cut has insignificant influence on the value of lumber produced unless the log is at or near the rotational orientation producing the maximum lumber value. Our results further showed that when the log is rotated exactly and an opening cut positioned within 1/8 inch of the optimal opening depth, lumber value produced was about 2% to almost 4% greater than the case with an exact opening cut but a suboptimal rotational orientation.

TABLE 5.3: An analysis of the effect of the depth-of-cut on the value of lumber produced when the log orientation is fixed at its optimal orientation.

Log No.	Avg. of 18 depth-of-cut positions ($)	Avg. of 432 combinations ($)	Difference (%)
1	85.29	83.10	2.6
2	82.30	78.62	4.7
3	103.70	99.88	3.2
4	92.15	86.90	6.0
5	60.86	57.68	5.5
6	51.99	47.91	8.5
7	62.52	60.01	4.2
		Avg.	5.0
		SD	2.0
		p	0.0237

5.3 Daymet Acceleration

Daymet is a component of a terrestrial ecosystem modeling system. It is a collection of algorithms and computer software designed to interpolate and extrapolate from daily meteorological observations to produce gridded estimates of daily weather parameters over large regions [Tho05]. A Daymet run requires input data such as digital elevation data and observations of maximum temperature, minimum temperature and precipitation from ground-based meteorological stations. There are approximately 6000 stations in the U.S. National Weather Service Co-op network and the Natural Resources Conservation Service SNOTEL network (automated stations in mountainous terrain) [Tho05]. The output of Daymet is further processed and the generated data can be analyzed by text analysis or visualization tools. Figure 5.11 gives the visualization of the 18-year mean precipitation of the United States from the output of Daymet.

5.3.1 Challenges

In Daymet, the gridded data are subdivided into sections, which are called tiles. The execution of each tile is coordinated by a single-threaded Perl script. Figure 5.12 lists the Daymet execution steps. From this figure, we can see the execution of each tile is a sequential job. However, the exe-

TABLE 5.4: An analysis of the effect of the log's rotational orientation on the value of lumber produced when the optimal opening depth-of-cut is maintained.

Log No.	Avg. of 24 rotational orientations ($)	Avg. of 432 combinations ($)	Difference (%)
1	82.74	83.10	-0.4
2	79.48	78.62	1.1
3	100.26	99.88	0.4
4	87.14	86.90	0.3
5	58.62	57.68	1.6
6	48.27	47.91	0.7
7	59.89	60.01	-0.2
		Avg.	0.5
		SD	0.7
		p	0.26

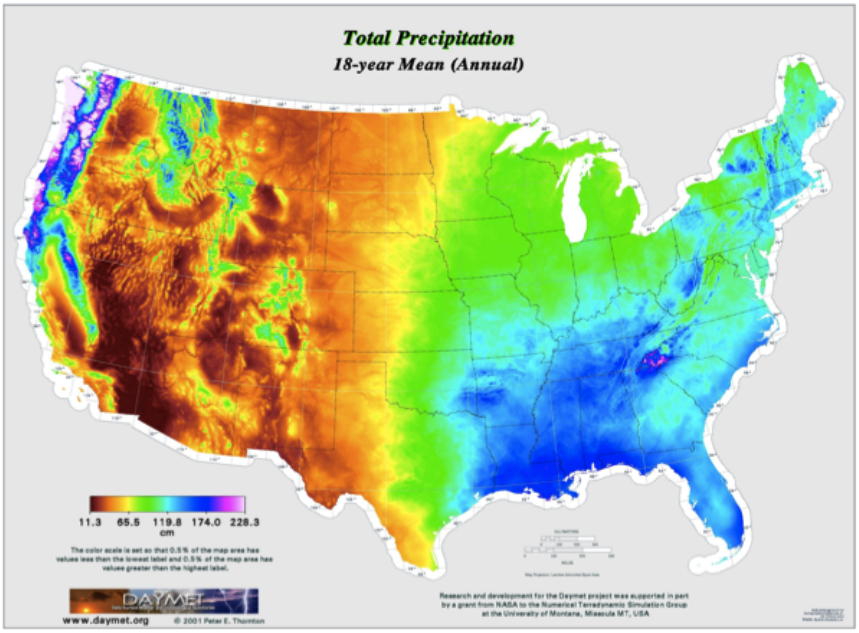


FIGURE 5.11: Mean value of precipitation of 18-year data.

cution of each tile will have variable length. Using Daymet to accommodate simulations of small areas works well, but quickly becomes an overwhelming job for scientists who want to achieve high resolution modeling over a large area, which may include thousands of tasks. Management of these many tasks requires tedious attention to details, including periodically monitoring running

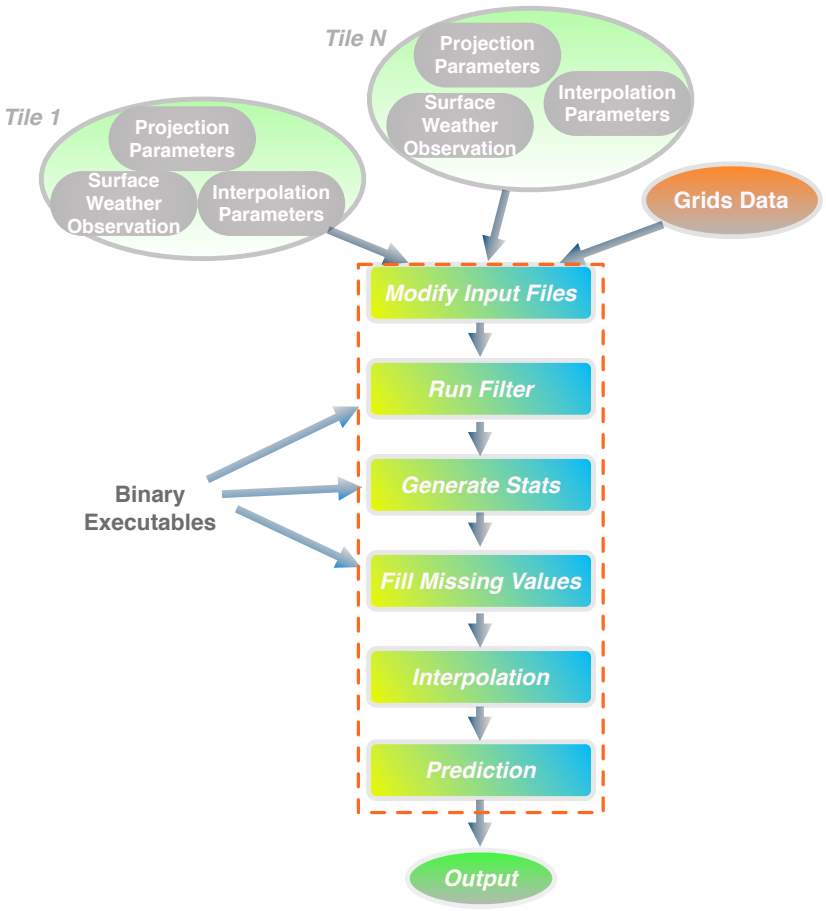


FIGURE 5.12: Daymet execution.

simulations, transferring data, correctly scripting configuration files for each model, and detecting failed simulations and handling the failures as appropriate [CHM+05]. Besides these problems, the total execution time will be extremely long if each task needs to be queued and executed individually.

Due to the nature of these tiles, they can be individually scheduled on different computing resources across multiple administrative boundaries in order to achieve "task level parallelism" to shorten the execution time. However, supporting multiple resources requires excessive resource-specific knowledge and software development experience for the developers. Different resources are administered independently and may have varying performance and characteristics. Metascheduling becomes the burden of the application and gateway developer since static scheduling can not

give good performance due to the heterogeneity of computational performance of each resource, and also because of the variable length of each tile. Any system failure in participating resources will affect execution. The slowest system is the bottleneck for application execution since we will need all the outputs in order to proceed to the next step. We need a system that can not only respond to users' workload but can also achieve reduced turnaround time and enhanced execution reliability.

5.3.2 System Implementation

Although all these challenges make the execution difficult, we can use the DA-TC model to fully remove the difficulties and make the execution efficient. In order to implement the DA-TC model for Daymet, there are several steps involved in system preparation.

1. **Executable Preparation.** Due to the incompatibility of the Daymet executables to different Linux/Unix Operation System, the user needs to compile the source files and provide different versions of the executables to different systems. The DA-TC model can use the system characteristics of the participating clusters to stage-in the compatible executables to the different clusters.

2. **TC Scheduling.** The DA-TC model will do the static scheduling for the task container submission. The users need to specify the resources they intend to use for Daymet execution in the configuration file. The DA-TC information service can provide all the system information for these resources, such as LRMS, Available_CPUs, CPU_speed, System Architecture, Memory_size, and the number of Running and Queued jobs, etc. The user can also use the scheduling algorithms provided to generate a static schedule for the TCs. The scheduling algorithms include Queue Length mode, Weighted Workload Allocation mode, and Shortest Expected Delay mode, etc. The user can also specify the number of TCs submitted to each resource directly by writing into the configuration file.

3. **Data Preparation.** Since the tiles to execute are organized with a discontinuous name pattern, and also to allow the users to specify the tiles they want to skip during execution, we use a file to specify the name of the tiles which will not be included in the Daymet execution. The users also need to give the beginning name of the tiles for the DA-TC executing sequence. Each tile will have a Gzipped file of the Daymet input data and a Perl script coordinating the execution of each tile.

After the system preparation, the DA-TC model can be used to execute these thousands of Daymet tiles automatically. The complete procedure for Daymet execution is:

1. **Binary executables and gridded data staging in.** Since each tile's execution will need the executables and gridded data, these files are transferred to each participating cluster as the first step. If these files are already there, the AEA will skip this step.

2. **Task container submission.** The AEA will submit TCs to each cluster based on the scheduling results or users' specification. The TCs will be treated as normal jobs, waiting in the queue for resource allocation.

3. **Input data and script staging in.** Once a TC is allocated resources, the AEA will assign next available tile to this TC. The corresponding input data file and the Perl script in that tile directory will be staged in to this TC location.

4. **Tile execution.** The TC will take charge of the execution of the tile based on the Perl script.

5. **Results staging out.** After execution, all output data have been generated. The outputs are gzipped and transferred back to the AEA machine into the corresponding tile directory.

6. **Terminating.** After staging out the data, the TC is available for the next tile. If at this time, there are no further tiles that need to execute, the AEA will terminate this TC.

5.3.3 Results

The DA-TC model has been successfully implemented in NCAR's clusters to perform Daymet execution. By using cluster Frost for the AEA and integrating two NCAR evaluation clusters and one external cluster, we are able to finish all 800 tiles within three hours with only eleven TCs. If all these 800 tiles were executed on one CPU, they would take almost 33 hours to finish. This is a huge reduction of the execution time for the Daymet application. By using a larger number of TCs, the total execution time can be further reduced. Through the DA-TC model, the scientists can be insulated from tedious configuration details, thereby increasing their productivity.

5.4 Large-scale Ensemble Subsurface Modeling

The economic impact of inaccurate prediction can be substantial, especially in the petroleum industry, which is notorious for its high-risk investments. Model inversion is important for determining values of model parameters and making relatively accurate predictions [Tar97, Tar82]. It is used to calibrate subsurface properties (e.g., porosity, permeability, and hydraulic conductivity) in a subsurface simulation model. This method leads to computed values of observables, such as rates, pressures (or head), and saturations, at different observation locations that are in reasonable agreement with actual measurements of those quantities. Commonly, engineers manually adjust model parameters to minimize the square of the mismatch between all measurements and computed values. Nowadays, the increase in sensor deployment in oil and gas wells for monitoring pressure, temperature, resistivity, and/or flow rate, has added impetus to continuous model updating. Instead of simultaneously using all recorded data to generate an appropriate reservoir flow model, it has become important to capture reservoir flow information by incorporating these real time data.

The Ensemble Kalman Filter (EnKF) based method [Eve03] is an inversion modeling technique that is used by various scientific and engineering applications, such as uncertainty assessment, subsurface modeling, and data assimilation. It reduces a nonlinear minimization problem in a huge parameter space to a statistical minimization problem in the ensemble space through changing objective function minimization with multiple local minima. It searches for the mean rather than

the mode of the posterior probability density function (pdf). Doing so avoids getting trapped in local minima as happens with gradient methods, making EnKF a promising methodology for various inverse modeling problems. Furthermore, EnKF provides an ideal setting for operational reservoir monitoring and prediction because of its updating features.

However, the EnKF method is processing-intensive because it performs iterative simulations of a large number of subsurface models. Multiple steps are involved in this computation process. A large amount of computation is required in each sequential propagation step, with the next step depending on the results of the previous step(s). Manual inverse modeling for a large-scale application is extremely time consuming. Thus, it is often limited to small-scale applications. Grid computing technologies are needed to provide an effective grid-enabled EnKF solution, to lower the computation cost required by EnKF.

5.4.1 Grid-Enabled EnKF Solution

One of challenging issues for implementing grid-enabled EnKF is efficient simulation synchronization. In each EnKF iteration, simulations are dispatched onto geographically distributed computing resources. Typically, it is very hard to predict the completion time of each simulation due to the inhomogeneity and different consumption of participating resources. It is even more difficult in this EnKF simulation since the problem scale can not be decided until execution time. Filter execution and task assignment for the next iteration have to wait until all simulation results for the current iteration are returned. The DA-TC execution model provides excellent support for simulation synchronization. AEA in the DA-TC model checks the status of each task and task container to decide whether or not the current iteration of EnKF is completed. The dynamic assignment also ensures that each participating cluster will finish execution at almost the same time, so that users do not have to wait longer for the results from slower clusters.

Another issue in implementing grid-enabled EnKF is how to handle waiting time on remote queues for each iteration. In the traditional grid execution model [KJB+05], submitted jobs have to follow scheduling policies on remote sites, where they wait for resource allocation in queues. The

overall execution time of an EnKF process with a large number of iterations would be unbearably long due to waiting in the queues at each iteration. Through the DA-TC model, once the required resources are allocated for the first iteration, the task containers hold the resources until the whole EnKF process is done, so the following iterations can be executed without waiting.

Figure 5.13 illustrates the logic used to implement the grid-enabled EnKF. After each task assignment via the DA-TC model is carried out, the condition of whether or not all tasks are assigned and all task containers are ready is examined by checking the task and container status tables in the DA-TC. If the answer is no, the process of task assignment continues. If the answer is yes, an application-specific Kalman filter provided by the application researchers is invoked. This filter program analyzes the simulation results and decides whether further iterations are needed or not. If the results are not acceptable, new task sets and corresponding data sets are generated, and the whole process is repeated.

The model inversion scenario within the DA-TC is described as follows:

1. Initial ensembles are generated and ensemble state vectors are built for various reservoir models, in the AEA machine.

2. The DA-TC model is used to submit the simulations to a grid.

3. Once a task container acquires resources, it will communicate with the AEA to update its status. The AEA will assign the next task to this container. The container can stage-in the executables and assigned task from the location defined by the AEA. The AEA can dynamically assign tasks to different containers based on the run-time status of these task containers.

4. The TCs keep executing the tasks assigned by the AEA and transfer back the results to the AEA after each execution.

5. The AEA keeps checking the status of all tasks and submitted task containers. If all tasks have “Done” status, the Kalman filter is invoked to check if the results satisfy the pre-defined

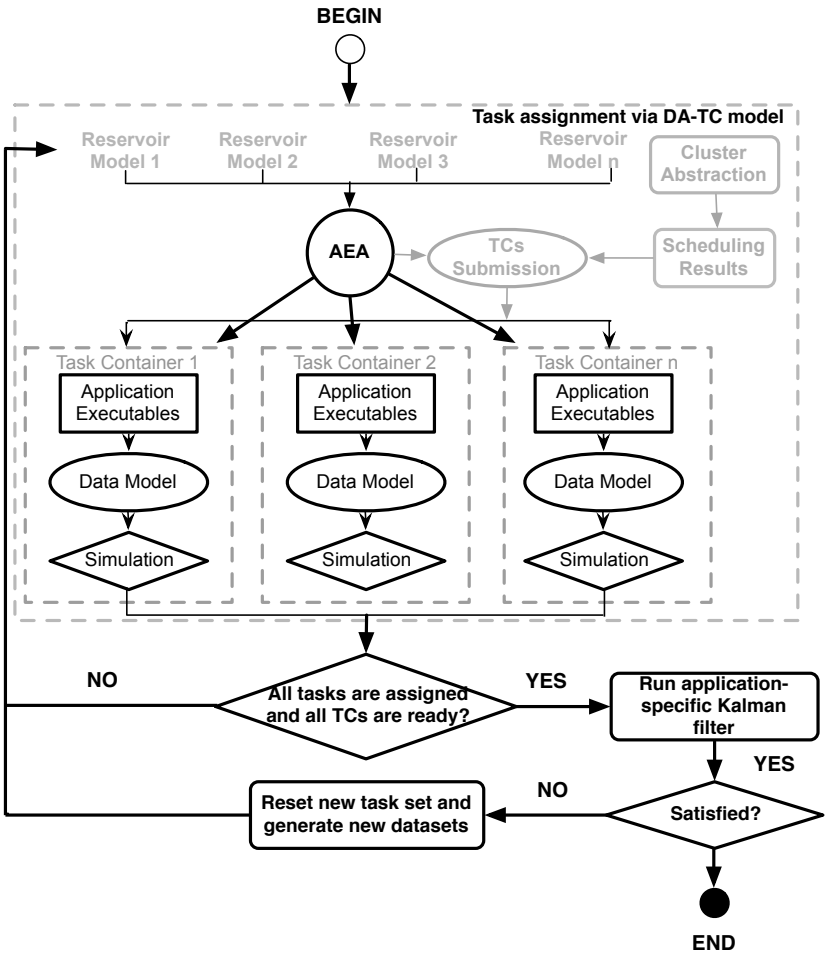


FIGURE 5.13: The logic of the grid-enabled EnKF solution.

requirements. At this time, all the task containers are in "READY" status and no task is executing. If the results are not acceptable, new task sets are generated. The AEA assigns new tasks to task containers to execute. This process repeats until the results are acceptable. If the results are already acceptable, the AEA will terminate all TCs.

6. The user retrieves the results for further analysis.

5.4.2 Computation Cost

Our DA-TC based EnKF workflow has been successfully applied to a 2D water-flooding real-time reservoir model [Eve03, GO05]. The reservoir simulator we used in this model is UTCHEM (University of Texas Chemical Compositional Simulator) [UTC], which is a three-dimensional,

multi-phase and multi-component finite-difference numerical simulator. In this study, the number of iterations is set to 10. Ten iterations are expected to be adequate, based on EnKF for similar models [GO06]. The number of ensemble members is 100. The number of simulation runs is 100×11(1 forecast + 10 iterations)$\times 10$ (assimilation times) $= 11,000$ synchronized updates. UTCHEM's average processor time per simulation is 10 minutes; the computation time is about 1,833 hours (77 days) using 1 processor. In our case, all the simulations are submitted to 3 clusters, which have 256, 15, and 14 processors respectively. We use 5 to 10 containers depending on cluster size; each container uses 1 processor. Using 10 containers for the 256 processor cluster and 5 containers for 15 and 14 processor clusters, the execution time is about 92 hours. Assuming queue wait time is 5 hours for each forecast step, the total time for EnKF processing is $92 + 5 = 97$ hours via DA-TC because there is only one wait time. The total time would increase to $92 + 5 \times 10 = 142$ hours if DA-TC was not used and the queue was re-entered for each forecast step. The queue waiting time depends on the cluster status, and may range from minutes to days. If the production history is quite long, the accumulated queue wait time will increase and negate the computational gains of grid computing.

5.5 Other Applications

Some other applications, such as Monte Carlo Nucleation Simulation, have also been implemented with our DA-TC model to reduce the turnaround time. Nucleation is a basic step in phase transition, which plays a critical role in understanding processes of atmospheric, environmental, and technological importance. Research on nucleation/condensation of water in effecting environmental and atmospheric processes, such as direct involvement in the generation of clathrate hydrates in cloud formation, brings unique insights into the investigation on the effect of the solvation of ions in water.

By combining the method of histogram-reweighing (HR) with the approach of aggregation-volume-bias Monte Carlo with umbrella sampling (AVUS), a new AVUS-HR algorithm was proposed. It allows the calculation of nucleation properties over a wide range of thermodynamic con-

ditions, but the simulation is computationally expensive. For example, relatively short simulation runs of $O(10^7)$ Monte Carlo moves are used for the iterations of the nucleation free energy (NFE) profile, followed by a long production run of 8×10^9 Monte Carlo moves, which takes about 3 days using a Pentium 3.2 GHz machine.

We have shown that the DA-TC execution model optimizes the task management and carries out parallel execution, making the AVUS-HR algorithm more efficient [YKX+07].

5.6 Chapter Summary

In this chapter we presented four different large-scale applications. All these applications require large computational power to reduce the turnaround time. Although a multicluster system provides the possibility to combine the computational power of each cluster for these applications, it is challenging to overcome the autonomy and heterogeneity of these resources. Our toolkit is used to seamlessly integrate these clusters together for application execution. By providing the easy-to-use high level services, our toolkit allows users to focus on their own research without needing to worry about the reliability and application steering during the execution. The turnaround time is greatly reduced, which demonstrates that our toolkit is efficient for assisting application scientists in conducting their research.

Chapter 6
Cloud Computing

Recently, the use of cloud computing through virtualization and the use of on-demand virtual machines (VMs) [Gol74] has become increasingly popular. Originally developed for large centralized computer systems, the concept of a virtual machine consists of three aspects: CPU virtualization, memory virtualization and I/O virtualization. Virtual machines provide virtualizations of physical host machines, upon which a virtual machine monitor (hypervisor) runs. The virtual machine monitor is responsible for capturing and emulating instructions issued by the guest machines, and providing interfaces for operations on VMs. Typically, clients can utilize virtual machines to create an execution environment with certain hardware or software configurations, and deploy it on any resource running hypervisors. Based on virtual machines, the virtual workspace [KFF+05] uses those VM images reflecting a workspace's software requirements to represent an execution environment. But the virtual workspace concept covers a wider territory than the workspace consisting of virtual machines. The virtual workspace includes site-configured workspace, which is installed to support specific needs of specific communities such as TeraGrid [Cat02]. The proposed approach for providing site-configured workspace is to obtain a priori agreement on specific configurations and propagate them, and provide access. Another component is virtual cluster workspaces, which can be constructed using the Cluster-on-Demand (COD) infrastructure [CIG+03]. The virtual cluster is an on-demand integration of multiple VM images that may represent specialized nodes, i.e. compute or head nodes. The nodes of this virtual cluster have the same or similar configuration, but they do not physically belong to the cluster.

Cloud computing is Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand, like the electricity grid [Clo10]. It basically gives users access to compute/data resources that they do not own. Cloud services can

provide the dynamic provision of services and resource pools in a coordinated fashion. Figure 6.1 shows the basic cloud services. By using different virtualization toolkits, the cloud service is able to provision the storage, CPU and network for each individual virtual machine with a service level agreement (SLA). Users can use the cloud client to connect with the cloud services in order to boot the virtual machines and virtual clusters on demand. Once these virtual systems are booted, they can be used immediately for computing without any waiting period. The location of resource usually is irrelevant. However, it may be relevant from the performance perspective if the computing needs a large chunk of data, which will introduces network latency when staging in and out these data. Web interfaces are always provided so that users can get to work anywhere as long as there is Internet connection.

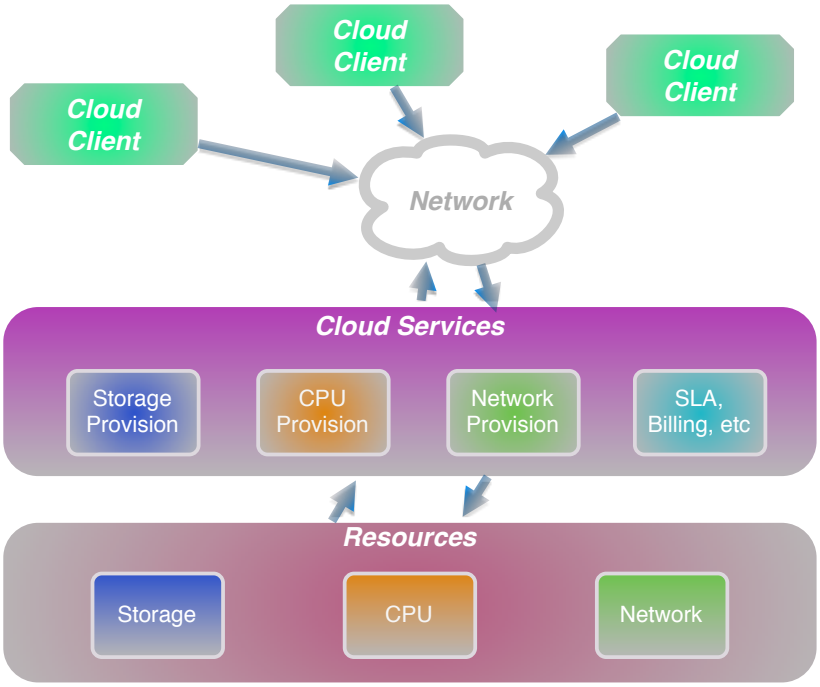


FIGURE 6.1: The cloud services.

There are lots of commercial clouds provided by different companies, such as Amazon, Microsoft, and Google, etc. Some science cloud software are also available, such as *Nimbus* from Argonne National Laboratory & University of Chicago and *Eucalyptus* from University of Califor-

nia at Santa Barbara. Both of them can bring up cloud computing services on clusters so that the clients can lease the remote resources by deploying VMs onto these clusters. They are also compatible to Amazon Web Services [Ama10] and support both KVM [KVM10] and Xen [Xen10] virtualization.

These infrastructure-as-a-service clouds have different advantages than traditional grid systems: users are provided with greater flexibility and have the ability to customize their virtual machine environment. Comparing with the grid systems, where the resources are always shared by large number of users, each virtual machine or virtual cluster booted in the cloud is dedicated to one user. Jobs submitted to grid systems are typically handled by a batch scheduler, whereas in the cloud environment, the jobs can be assigned directly to a virtual machine or virtual cluster by the user. Therefore, jobs can get executed immediately instead of waiting in the queue.

However, how this cloud computing technology can be beneficial to different application scenarios is uncertain. This also leads to the needs to integrate traditional grids and clouds. Developing and running applications in such a hybrid and dynamic computational infrastructure presents new and significant challenges [KeKJP09]. It will need the execution system to be able to support the hybrid execution models, and coordinate and manage the execution in an efficient and scalable manner. How to determine and provision the appropriate mix of grid/cloud resources, as well as dynamically schedule them across the hybrid execution environment to fulfill different performance objectives will be the key issues in this case [LLJ10].

6.1 Cloud Service Toolkit – Nimbus

In order to take advantage of cloud technology for application execution, we deployed the Nimbus science cloud toolkit onto our clusters. The Nimbus toolkit consists of the following components:

- *Infrastructure-as-a-Service (IaaS).* The Nimbus toolkit turns the physical clusters into "Infrastructure-as-a-Service" (IaaS) cloud computing platforms. It gives the administrators the choice to initiate and terminate the IaaS as needed.

• *Cumulus storage service.* Cumulus is a storage cloud implementation compatible with the Amazon Web Services S3 REST API. It provides secure management of cloud disk space, giving each user a "repository" view of VM images they own and images they can launch. Cumulus replaces the Globus GridFTP-based [ABB+01] upload and download of VM images. It is integrated with the Nimbus installation, but can also be installed on its own to manage a storage cloud.

• *Cloud client.* A easy to use end-user tool which provides users the ability to transfer images, check current stored images, launch, query and terminate VMs belongs to that user. Other functions are also available for end users, such as check the information and initiate the grid proxy, query security setups, etc.

• *Workspace service.* The workspace service is composed of a WS front-end and a VM-based resource manager deployed on a site. It supports two front-ends: one based on the Web Service Resource Framework (WSRF) [CFF+04], and one based on Amazon's EC2 Web Services Description Language (WSDL). This service is in charge of the hardware implementation and virtualization for the VMs. It allows a remote client to deploy and manage flexibly defined groups of VMs, and it will dynamically provision resources and environment for each VM. It will also publish the information of each workspace so that user can use the cloud client to easily get all the information of the VMs belong to him, such as IP address, cloud name, time duration, etc. The users can also use this information to directly login to these VMs to perform tasks as if they were physical resources.

• *The workspace control tools*, which are used to start, stop, and pause VMs; implement VM image reconstruction and management; connect the VMs to the network; and deliver contextualization information.

• *Workspace pilot*, which extends existing local resource managers (LRMs) such as Torque [Tor] or SGE [SGE] to deploy virtual machines to allows resource properties to use virtualization without significantly altering the site configuration.

• *Context broker*, which allows a client to deploy a "one-click" functioning virtual cluster as opposed to a set of "unconnected" virtual machines as well as "personalize" VMs.

6.2 Cloud Resources Integration

Our DA-TC execution model has been proved to be efficient for large-scale loosely-coupled applications. It can achieve dynamic load balancing and reduced turnaround time for these applications. However, how to integrate the cloud resources as well as traditional grid resources to collaboratively perform job execution is a challenge. In order to support these hybrid execution resources, we have two different approaches.

The first approach is to treat each virtual machine as a task container. It is quite straightforward. The basic idea behind our DA-TC model is to decouple the resource allocation from resource binding. In the DA-TC model, the resources are bound to each individual task containers instead of each task, so that the queuing time of each task can be reduced. In cloud computing, the resources are bound to each virtual machine instead of each task, which is coherent with our DA-TC model. Once the virtual machine is booted, the AEA can directly assign job to execute on this virtual machine.

Figure 6.2 shows how to launch the virtual machines. Different users can login to the cloud client simultaneously. Users can use the client to query the status of the VMs, check the images already stored at the server side, and transfer in the images they want to boot using Cumulus storage service. Once the users issue the command to launch the VM, the client will contact the Nimbus cloud services on the server side. Users can define the name and duration of this workspace. They can also use a customized XML configuration file to define the memory size and CPU number for each VM. Multiple VMs can be launched at the same time using the same image. The virtual

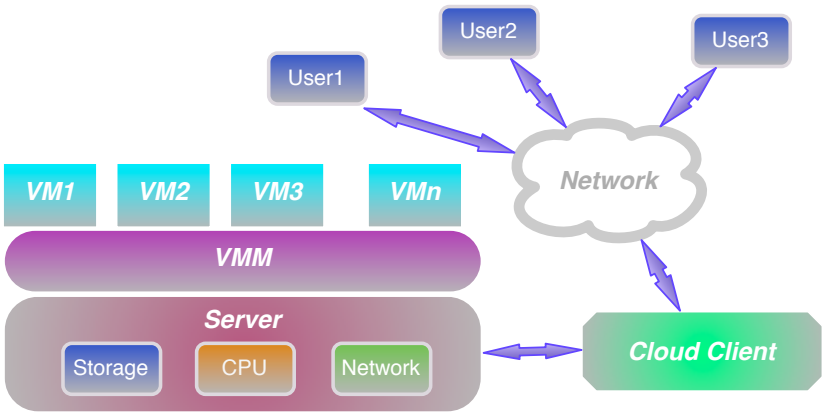


FIGURE 6.2: Launch virtual machines through Nimbus.

machine monitor (VMM, also called the hypervisor) on the server node will provision the resources for the VMs based on the request. Once booted, the information of each VM, such as the beginning time and ending time of the VM, the IP address and name of the workspace, are returned for users' further usage.

Although the VMs are easy to boot, and the VMs can be treated identically as TCs in our DA-TC model, there are still some difference between them. Due to the isolation of each VM on the server node, each VM will need a copy of the binary executables, which will cause more overhead if using multiple VMs. The user also needs to configure the image to make sure the AEA will automatically notice that the VMs begin running once they are booted. The scheduling of VMs is also different from the scheduling of TCs.

Another approach to integrate the cloud resources to the DA-TC model is to use virtual clusters. Users can use a virtual cluster as one of the multiple resources to execute the tasks. This approach will fit our DA-TC paradigm, and the virtual cluster is easy to boot, though it will also need some configurations in order to be used. The big advantage of using a virtual cluster is that once booted, it is a dedicated resource to use, thus there is no queuing time for the TCs anymore.

Figure 6.3 shows how to launch the virtual cluster through Nimbus. The first step is to prepare the image. Some configurations are needed for the image so that once booted, users can submit jobs through the batch scheduler, such as PBS. Users also need to install and configure the context

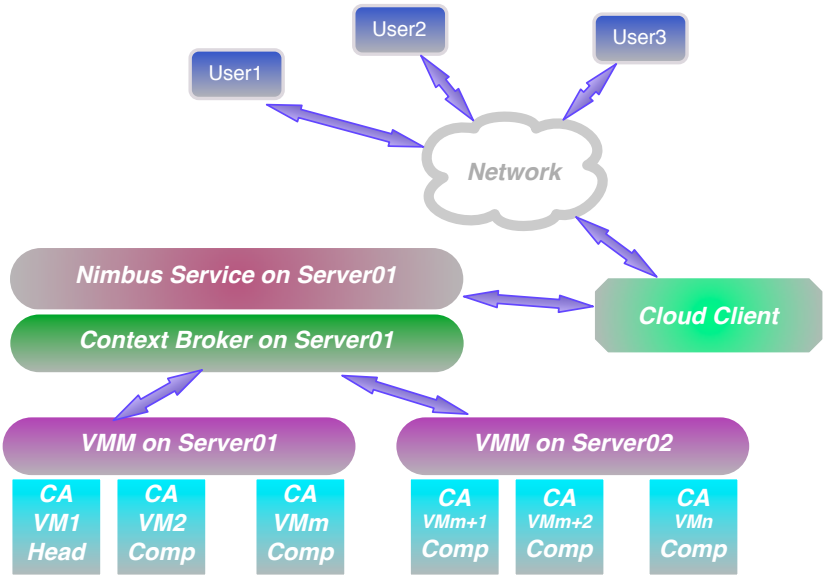


FIGURE 6.3: Launch virtual cluster through Nimbus.

agent on the image before it is booted. Other configurations, such as SSH and name service, are also needed for using the virtual cluster. Once the image is ready, the users can customize the XML configuration file for launching the virtual cluster. Users can define the number of compute nodes, the image for booting each node, and scheduler information in this file. Once configured, the user can use the cloud client to launch the virtual cluster. After booting, the context agent in each VM will contact the context broker on the server node to define all the head node and compute node information. The IP address, hostname and status of each head node and compute node will be returned to the user so that the user can login directly to the head node to submit jobs based on these information. Notice that the head node and compute node can exist on different sites.

We deployed and integrated the virtual machines and virtual clusters into our DA-TC model based on this second approach. Figure 6.4 shows the way in the DA-TC model to integrate virtual cluster and physical clusters for job execution. Once the virtual cluster is ready, the AEA will submit the TCs to it. These TCs will be allocated resources immediately. Then the AEA will assign the centrally queued workload to the TCs dynamically based on the availability.

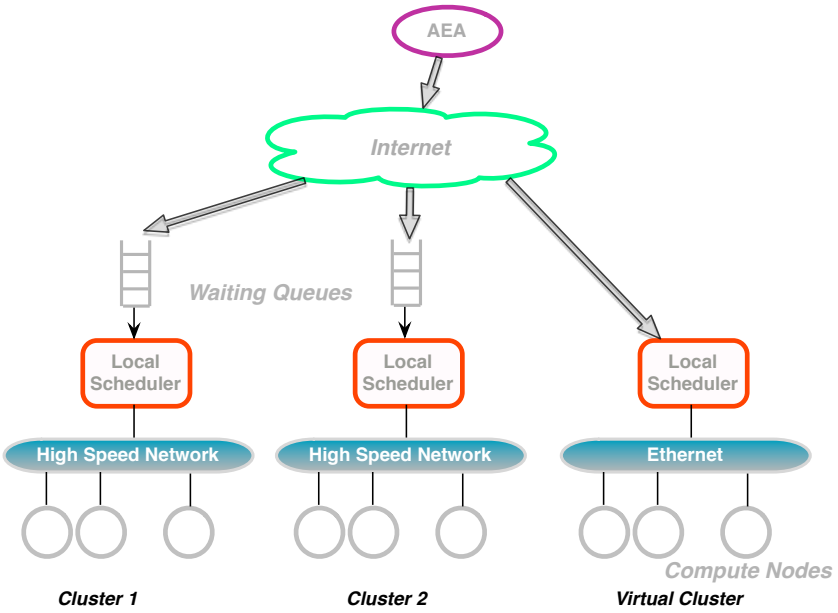


FIGURE 6.4: Integrating hybrid resources for job execution in DA-TC model.

In our experiments, we are able to boot a 4-node PBS cluster with 6 GB for each node in 479 seconds. Compared with TeraGrid [Ter] resources which are share by large number of users with an average queue wait time of 3901 seconds [SY10], the virtual cluster provides a fast and dedicated way for application execution. This virtual cluster can also be integrated into our DA-TC model and works identically as a physical cluster for job execution. Users can configure the images to suit the application execution requirements beforehand, such as the software environment and hardware allocation, so that once booted, the virtual cluster can be used immediately for job execution. Launching the virtual cluster is easy and automatic, and our model is able to dynamically schedule the tasks across this hybrid execution environment to fulfill different performance objectives.

6.3 Chapter Summary

We deployed the Nimbus cloud computing toolkit and demonstrated it with our model. Users can easily turn the clusters into cloud service platforms by initiating the Nimbus service. Users can also launch the virtual machines and virtual clusters by using the cloud client. By configuring the images beforehand, users will be able to boot the virtual cluster easily. Experiments show that by

integrating the virtual clusters into our DA-TC model, we can achieve reduced turnaround time for large-scale applications in this multiple hybrid grid/cloud resources environment.

Chapter 7
Conclusions and Future Work

Multicluster grids offer a promising solution to satisfying the growing computational demands of compute-intensive applications. However, seamlessly integrating all participating clusters in different domains into a single virtual computational platform remains a challenge. In order to fully utilize the capabilities of multicluster grids, computer scientists need to deal with the issue of joining together participating autonomic systems practically and efficiently to execute grid-enabled applications.

In this dissertation, we present the design and evaluation of a multicluster grid management system and an associated toolkit. The toolkit called **Pelecanus** provides user-friendly interfaces, application and data management, and enhanced execution performance. This work is driven the requirements of a number of compute-intensive applications in whose development we have participated. The application developers need to carry out experimental work on a variety of architectures which are autonomic, heterogeneous, and connected by the Internet. Our goal is to address these issues by implementing an easy-to-use toolkit that can provide seamless access to massive computing power and meets various application-specific requirements for these applications.

The performance benefit of this toolkit comes from the novel DA-TC application execution model we implemented in multicluster environments. In this model, there are no specific system configuration or software installation requirements on any participating clusters in order to run this model. This model deploys the task containers to take fully usage of the allocated resources and dynamically assigns individual tasks to the participating clusters according to the runtime status of the task containers, therefore it is able to achieve dynamic load balancing and reduced application turnaround time. Users can easily monitor the application progress by querying the application execution agent because of the centralized table of all the information of each job and

task container. We also discussed the application execution reliability, interaction capability, and system fault tolerance of this model in this dissertation.

Cluster abstraction is another important part of this toolkit. The toolkit can be used as a standalone automated system to monitor and evaluate the clusters' functionality and network performance. Users of this toolkit can use cluster abstraction to view and select appropriate clusters to execute their applications.

We studied the performance of our model by carrying out both a theoretical analysis and experimental evaluation. We investigated the scalability of our model. We also deployed different scheduling policies that have been incorporated into our model. We found that in contrast to the traditional method where scheduling is very important for performance, in our model, the performance of different policies is quite similar. This is because we dynamically assign the tasks to each task container, though the submission of each task container is static. We also compared the performance of our model to the traditional method in different application scenarios. We found that in all cases, our model is able to achieve 30% more reduced turnaround time.

We also investigated how cloud computing technology can benefit our loosely-coupled applications by coupling different grid/cloud resources together. We showed that our model can be easily extended to support dynamic execution across a wide range of heterogeneous infrastructures, from clusters to cloud, whilst preserving the performance.

This **Pelecanus** toolkit has been successfully deployed in a number of applications, and we expect to continue to refine the toolkit and explore a wider range of application domains to take advantages of this toolkit to achieve enhanced performance.

7.1 Major Contributions

We have developed an easy-to-use toolkit to effectively use multiple clusters for application execution. The toolkit provides user-friendly interfaces for application scientists to perform their tasks while hiding the underlying heterogeneity of resources from them. It can not only reduce the application turnaround time but also enhances job monitoring and QoS assurance. This toolkit

is spectacularly efficient for applications with multiple or repeated small tasks, independent of whether each task is a sequential or parallel job. In addition, it allows the user to adapt the workflow according to the runtime status of resources and execution progress.

We have successfully deployed multiple applications from different disciplines in this toolkit. With a little bit of extra time to prepare the application to fit into our model, we can achieve a huge reduction in the turnaround time for each application. It is especially useful when researchers only have access to a variety of different resources that do not share a computational grid, but want to take the advantage of their combined computational power. All the jobs executed belong to each user's account identity, therefore, all per-user resource accounting mechanisms remain the same. This toolkit works as a plug-in service, without any special privilege to support and without any changes to existing infrastructures.

7.2 Future Work

Each system component presented in this work allows a future development path, and a lot of effort could be put into the toolkit optimization and application support. We list below key aspects that we plan to work on.

Resource Monitoring and Discovery. Existing grid monitoring and discovery technologies such as Globus MDS [FK99], Ganglia [MCC04] cluster toolkit and MonALISA [NLG+03] could be integrated into the information service of the cluster abstraction. These services would be very useful when the resources share the same grid middleware.

Resource Matching. Users may have constraints on the placement of jobs due to specific resource requirements. For example, some jobs require a minimum amount of physical memory or disk space, or specific software installation. Currently, our implementation will extract this system information for the users and allow users to choose the resource. We hope in the future that we can supply more detailed software environment information for user to choose.

Fault Tolerance. Although our model is robust against system failure, we want to enable the service to detect and handle different faults. More importantly, if the AEA machine encounters a

problem during execution (e.g., a machine shutdown or network break), we need to clean up the current execution on participating clusters, or migrate to another AEA machine to carry out the execution.

Interface Development. In the current deployment, we provide a single point login service for each user to use the toolkit, assuming the username is the same across different clusters. We want to make this service more user-friendly by using complete information for each user on each resource. We also want to make this toolkit more secure to protect users' data.

Application Studies. Although we have demonstrated that our toolkit is useful in four different application scenarios, we want to explore more scenarios and extend our work to support more diverse applications, in order to test and improve this service.

Bibliography

[ABB+01] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Eighteenth IEEE Symposium on Mass Storage Systems and Technologies (MSS '01)*, page 13, April 2001.

[AD03] J.H. Abawajy and S.P. Dandamudi. Parallel job scheduling on multicluster computing system. In *Proceedings of 2003 IEEE International Conference on Cluster Computing*, pages 11 – 18, Los Alamitos, CA, USA, Dec. 2003. IEEE Computer Society.

[Ama10] Amazon Elastic Compute Cloud. Web page, 2010. `http://aws.amazon.com/ec2`.

[ANS] ANSYS Structura. Web page. `http://www.ansys.com`.

[Aum02] Olivier Aumage. Heterogeneous multi-cluster networking with the Madeleine III communication library. In *Proceedings of 2002 IEEE International Parallel and Distributed Processing Symposium*, volume 2, pages 85–96, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[BAN00] Marcos Barreto, Rafael Avila, and Phillippe Navaux. The multicluster model to the integrated use of multiple workstation clusters. In *Proc. of the 3rd Workshop on Personal Computer-based Networks of Workstations, 2000*, pages 71–80, 2000.

[BAS+05] P. Bogden, G. Allen, G. Stone, J. Bintz, H. Graber, S. Graves, R. Luettich, D. Reed, P. Sheng, H. Wang, and Wei Zhao. The southeastern university research association coastal ocean observing and prediction program: integrating marine science and information technology. In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 803–809 Vol. 1, 2005.

[BBE03] S. Banen, A. I. D. Bucur, and D. H. J. Epema. A measurement-based simulation study of processor co-allocation in multicluster systems. In *Scheduling Strategies for Parallel Processing*, pages 105–128. Springer-Verlag, 2003.

[Ber96] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

[BFT02] S. Bhandarkar, T. D. Faust, and M. Tang. Design and prototype development of a computer vision-based lumber production planning system. *Image Vis. Comput.*, 20(3):167–189, 2002.

[BGJ06] Vandy Berten, Joël Goossens, and Emmanuel Jeannot. On the distribution of sequential jobs in random brokering for heterogeneous computational grids. *IEEE Trans. Parallel Distrib. Syst.*, 17(2):113–124, 2006.

[BNW06] John Brevik, Daniel Nurmi, and Rich Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '06, pages 110–118, New York, NY, USA, 2006. ACM.

[BZ92] Sayed A. Banawan and Nidal M. Zeidat. A comparative study of load sharing in heterogeneous multicomputer systems. In *Proceedings of the 25th annual symposium on Simulation*, ANSS '92, pages 22–31, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[Cat02] C. Catlett. The teragrid: A primer, September 2002. `http://www.teragrid.org/about/TeraGrid-Primer-Sept-02.pdf`.

[CFF+04] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The ws-resource framework, 2004. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`.

[CFM03] Michael Chu, Kevin Fan, and Scott Mahlke. Region-based hierarchical operation partitioning for multicluster processors. In *Proc. of the SIGPLAN '03 Conference on Programming Language Design and Implementation*, pages 300–311, 2003.

[CHM+05] Jason Cope, Craig Hartsough, Sean Mccreary, Peter Thornton, Henry M. Tufo, Nathan Wilhelmi, and Matthew Woitaszek. Experiences from simulating the global carbon cycle in a grid computing environment. In *The Fourteenth Global Grid Forum (GGF 14),Chicago*, 2005.

[CIG+03] 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 *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing*, pages 90 – 100, june 2003.

[CK79] Yuan-Chieh Chow and W.H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Transactions on Computers*, C-28(5):354 –361, May 1979.

[Clo10] Cloud computing. Web page, 2010. `http://en.wikipedia.org/wiki/Cloud_computing`.

[Dow97a] Allen Downey. Using queue time predictions for processor allocation. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 35–57. Springer Berlin / Heidelberg, 1997.

[Dow97b] Allen B. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Symposium on Parallel Processing*, IPPS '97, pages 209–218, Washington, DC, USA, 1997. IEEE Computer Society.

[DRM10] DRMAA. Web page, documentation, 2010. `http://www.drmaa.org/`.

[DSS+05] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia C. Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[Eve03] G. Evensen. The ensemble Kalman filter: Theoretical formulation and practical implementation. *Ocean Dyn.*, 53(4):334–367, 2003.

[Fai10] FairShare. Web page, documentation, 2010. `http://www.adaptivecomputing.com/resources/docs/mwm/6.3fairshare.php`.

[FB87] B. V. Funt and E. C. Bryant. Detection of internal log defects by automatic interpretation of computer tomography images. *Forest Prod. J.*, 37(1):56–62, 1987.

[FK99] I. Foster and C. Kesselman. *Globus: A toolkit-based Grid architecture*, pages 259–278. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, 1999.

[FK08] Timothy Freeman and Katarzyna Keahey. Flying low: Simple leases with workspace pilot. In *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 499–509, Berlin, Heidelberg, 2008. Springer-Verlag.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15:200–222, August 2001.

[Flu] Fluent Flow Modeling Software. Web page. `http://www.fluent.com`.

[FTL+01] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing*, pages 55 –63, 2001.

[GC98] S. Guddanti and S. J. Chang. Replicating sawmill sawing with TOPSAW using CT images of a full-length hardwood log. *Forest Prod. J.*, 48(1):72–75, 1998.

[GO05] Y. Gu and D. S. Oliver. History match of the PUNQ-S3 reservoir model using the ensemble Kalman filter. *SPE Journal*, pages 217–224, June 2005.

[GO06] Y. Gu and D. S. Oliver. The ensemble Kalman filter for continuous updating of reservoir simulation models. *J. of Energy Resources Technology*, 128:79–87, March 2006.

[Gol74] R. Goldberg. A survey of virtual machine research. In *IEEE Computer*, pages 34–45, 1974.

[GR06] C. Geuzaine and J-F. Remacle. Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, 2006. `http://www.geuz.org/gmsh/`.

[Gri] GridFTP speedpage. Project web page. http://speedpage.psc.teragrid.org/speedpage/www/speedpage.php.

[HJS+04] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen, and Graham R. Nudd. Hybrid performance-based workload management for multiclusters and grids. *IEE Proceedings - Software*, 151(5):224–231, 2004.

[HJS+06] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Hong Jiang, Donna N. Dillenberger, and Graham R. Nudd. Allocating non-real-time and soft real-time jobs in multiclusters. *IEEE Trans. Parallel Distrib. Syst.*, 17:99–112, February 2006.

[HJSN04] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, and Graham R. Nudd. Optimising static workload allocation in multiclusters. In *Proceedings of 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS'04)*, pages 26–30. IEEE Computer Society, 2004.

[HWS+91] T. G. Harless, F. G. Wagner, P. H. Steele, F. W. Taylor, V. Yamada, and C. W. McMillin. Methodology for locating defects within hardwood logs and determining their impact on lumber-value yield. *Forest Prod. J.*, 41(4):25–30, 1991.

[INC10] INCA. Project web page, 2010. http://inca.sdsc.edu.

[JSC01] David B. Jackson, Quinn Snell, and Mark J. Clement. Core algorithms of the maui scheduler. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01, pages 87–102, London, UK, 2001. Springer-Verlag.

[KAN+09] Omer Khalid, Richard J. Anthony, Paul Nilsson, Kate Keahey, Markus Schulz, Kevin Parrot, and Miltos Petridis. Enabling and optimizing pilot jobs using xen based virtual machines for the hpc grid applications. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 1–8, New York, NY, USA, 2009. ACM.

[KeKJP09] Hyunjoo Kim, Yaakoub el Khamra, Shantenu Jha, and Manish Parashar. An autonomic approach to integrated hpc grid and cloud usage. In *Proceedings of the 2009 Fifth IEEE International Conference on e-Science*, E-SCIENCE '09, pages 366–373, Washington, DC, USA, 2009. IEEE Computer Society.

[KFF+05] Katarzyna Keahey, Ian Foster, Timothy Freeman, Xuehai Zhang, and Daniel Galron. Virtual workspaces in the grid. In Josë C. Cunha and Pedro D. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 421–431. Springer Berlin / Heidelberg, 2005.

[KJB+05] Daniel S. Katz, Joseph C. Jacob, G. Bruce Berriman, John Good, Anastasia C. Laity, Ewa Deelman, Carl Kesselman, and Gurmeet Singh. A comparison of two methods for building astronomical image mosaics on a grid. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, ICPPW '05, pages 85–94, Washington, DC, USA, 2005. IEEE Computer Society.

[KKNW08] Yang-Suk Kee, Carl Kesselman, Daniel Nurmi, and Rich Wolski. Enabling personal clusters on demand for batch resources using commodity software. *Parallel and Distributed Processing Symposium, International*, 0:1–7, 2008.

[Kle75] L. Kleinrock. *Queueing System*. John Wiley & Sons, 1975.

[KVM10] KVM: Kernel-based Virtual Machine. Web page, documentation, 2010. `http://www.linux-kvm.org/page/Main_Page`.

[LAC+08] Zhou Lei, Gabrielle Allen, Promita Chakraborty, Dayong Huang, John Lewis, Xin Li, and Christopher D. White. A grid-enabled problem-solving environment for advanced reservoir uncertainty analysis. *Concurrency and Computation: Practice and Experience*, 20(18):2123–2140, 2008.

[LHK+06] Zhou Lei, Dayong Huang, Archit Kulshrestha, Santiago Pena, Gabrielle Allen, Xin Li, Christopher White, Richard Duff, John R. Smith, and Subhash Kalla. Resgrid: A grid-aware toolkit for reservoir uncertainty analysis. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '06, pages 249–252, Washington, DC, USA, 2006. IEEE Computer Society.

[LLJ10] André Luckow, Lukasz Lacinski, and Shantenu Jha. Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 135–144, Washington, DC, USA, 2010. IEEE Computer Society.

[LLW+07] Xin Li, Zhou Lei, Christopher D. White, Gabrielle Allen, Guan Qin, and Frank T-C. Tsai. Grid-enabled ensemble subsurface modeling. In *Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 67–72, Anaheim, CA, USA, 2007. ACTA Press.

[LM99] Robert Leslie and Sati McKenzie. Evaluation of loadsharing algorithms for heterogeneous distributed systems. *Computer Communications*, 22(4):376–389, 1999.

[Loa] LoadLeveler. `http://publib.boulder.ibm.com/epubs/pdf/am2ug303.pdf`.

[LON09] LONI: Louisiana Optical Network Initiative. Web page, 2009. `http://www.loni.org/`.

[LSLS05] Chokchai Box Leangsuksun, Lixin Shen, Tong Liu, and Stephen L. Scott. Achieving high availability and performance computing with an ha-oscar cluster. *Future Gener. Comput. Syst.*, 21:597–606, April 2005.

[LYA+08] Zhou Lei, Zhifeng Yun, Gabrielle Allen, Xin Li, Nian-Feng Tzeng, and Christopher White. Improving application execution in multicluster grids. In *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*, pages 163–170, Washington, DC, USA, 2008. IEEE Computer Society.

[Mac07a] J. MacLaren. Co-allocation of compute and network resources using harc. In *Lighting the Blue Touchpaper for UK e-Science: closing conference of ESLEA Project, PoS(ESLEA)016*, 2007.

[Mac07b] Jon MacLaren. Harc: the highly-available resource co-allocator. In *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part II*, OTM'07, pages 1385–1402, Berlin, Heidelberg, 2007. Springer-Verlag.

[MCC04] M.L. Massie, B.N. Chun, and D.E. Cueller. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[NBW07] Daniel Nurmi, John Brevik, and Rich Wolski. Qbets: Queue bounds estimation from time series. In *In Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 76–101, 2007.

[Nel95] R. Nelson. *Probability, Stochastic Processes, and Queueing Theory*. Springer-Verlag, 1995.

[Nil08] P Nilsson. Experience from a pilot based system for atlas. *Journal of Physics: Conference Series*, 119(6):062038, 2008.

[NLG+03] Harvey B. Newman, Iosif Legrand, Philippe Galvez, Ramiro Voicu, and Catalin Cirstoiu. MonALISA : A distributed monitoring service architecture. In *CHEP03*, La Jolla, California, March 2003.

[NWB08a] Daniel Nurmi, Rich Wolski, and John Brevik. Probabilistic advanced reservations for batch-scheduled parallel machines. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, PPoPP '08, pages 289–290, New York, NY, USA, 2008. ACM.

[NWB08b] Daniel Charles Nurmi, Rich Wolski, and John Brevik. Varq: virtual advance reservations for queues. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 75–86, New York, NY, USA, 2008. ACM.

[OGF06] Open Grid Forum. Web page, 2006. `http://www.ogf.org/`.

[Ope] OpenFOAM: The Open Source CFD Toolbox. `http://www.openfoam.com/`.

[PBS10] PBS. Web page, documentation, 2010. `http://www.openpbs.org`.

[Pil] Pilot Jobs. Web page. `http://wiki.egee-see.org/index.php/Grid_Interactivity,_Pilot_Jobs,_and_Job_Pooling`.

[Pil10] Pilot Jobs. Web page, 2010. `http://www.sysadmin.hep.ac.uk/wiki/Pilot_Jobs`.

[Pin03] Christopher Pinchak. Placeholder scheduling for overlay metacomputers. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, Spring 2003.

[PLG02] Christopher Pinchak, Paul Lu, and Mark Goldenberg. Practical heterogeneous placeholder scheduling in overlay metacomputers: Early experiences. In *JSSPP '02: Revised Papers from the 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 205–228, London, UK, 2002. Springer-Verlag.

[Rie06] Frank Rieg. Z88: The fast and compact finite elements program, 2006. `http://www.openchannelsoftware.com/projects/Z88/`.

[RZD+07] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, and Mike Wilde. Falkon: a fast and light-weight task execution framework. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage, and Analysis (SC07*, 2007.

[SAG10] SAGA: A Simple API for Grid Applications. Project web page, 2010. `http://saga.cct.lsu.edu`.

[SCJG00] Quinn Snell, Mark J. Clement, David B. Jackson, and Chad Gregory. The performance impact of advance reservation meta-scheduling. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, IPDPS '00/JSSPP '00, pages 137–153, London, UK, 2000. Springer-Verlag.

[Sfi07] I. Sfiligoi. Making science in the grid world: using glideins to maximize scientific output. In *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, volume 2, pages 1107–1109, 26 2007-Nov. 3 2007.

[SGE] SGE: Sun Grid Engine. Web page, documentation. `http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`.

[ShSV+08] Gurmeet Singh, Mei hui Su, Karan Vahi, Ewa Deelman, Bruce Berriman, John Good, Daniel S. Katz, and Gaurang Mehta. Workflow task clustering for best effort systems with pegasus. In *In MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–8. ACM, 2008.

[SLA96] D. L. Schmoldt, P. Li, and P. A. Araman. Interactive simulation of hardwood log veneer slicing using CT images. *Forest Prod. J.*, 46(4):41–47, 1996.

[STF99] Warren Smith, Valerie E. Taylor, and Ian T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS/SPDP '99/JSSPP '99, pages 202–219, London, UK, 1999. Springer-Verlag.

[SWKA93] P. H. Steele, F. G. Wagner, L. Kumar, and P. A. Araman. The value versus volume yield problem for live-sawn hardwood sawlogs. *Forest Prod. J.*, 43(9):35–40, 1993.

[SY10] Subhashini Sivagnanam and Kenneth Yoshimoto. Teragrid resource selection tools: a road test. In *Proceedings of the 2010 TeraGrid Conference*, TG '10, pages 20:1–20:6, New York, NY, USA, 2010. ACM.

[Tar82] Albert Tarantola. Inverse problems = quest of information. *Journal of Geophysics*, 50:159–170, 1982.

[Tar97] Albert Tarantola. *Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation*. Elsevier, Amesterdam, Netherland, 1997.

[TC00] Xueyan Tang and Samuel T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. *International Conference on Parallel Processing*, 0:373, 2000.

[Ter] TeraGrid. Web page, documentation, publication. `https://www.teragrid.org/`.

[TGSR04] Andrei Tsaregorodtsev, Vincent Garonne, and Ian Stokes-Rees. DIRAC: A scalable lightweight architecture for high throughput computing. *IEEE/ACM International Workshop on Grid Computing*, 0:19–25, 2004.

[Tho04] J. R. Thome. *Preliminary Design Procedures*, pages 226–234. Engineering Data Book III. Wolverine, 2004.

[Tho05] Peter Thornton. Daymet user's guide, 2005. `http://daymet.org/`.

[Tor] Torque. Web page, documentation. `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.

[TTL02] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[TWJ+84] F. W. Taylor, F. G. Wagner, C. W. Mcmillin Jr., I. L. Morgan, and E. F. Hopkins. Locating knots by industrial tomography – a feasibility study. *Forest Prod. J.*, 34(5):42–46, 1984.

[UGE] Univa Grid Engine. Web page, documentation. `http://www.univa.com`.

[UTC] UTCHEM. Web page, documentation, publications. `http://www.cpge.utexas.edu`.

[vNMB+00] Rob van Nieuwpoort, Jason Maassen, Henri E. Bal, Thilo Kielmann, and Ronald Veldema. Wide-area parallel programming using the remote method invocation model. *Concurrency: Practice and Experience*, 12(8):643–666, 2000.

[WF01] Ahuva Mu'alem Weil and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.

[WGLT06] E. Walker, J.P. Gardner, V. Litvin, and E.L. Turner. Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment. In *IEEE Workshop on Challenges of Large Applications in Distributed Environments*, pages 95–103, Paris, France, 2006.

[Wol03] Richard Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

[Xen10] Xen. Web page, documentation, 2010. `http://www.xen.org`.

[Xu01] M. Xu. Effective metacomputing using LSF multicluster. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid(CCGrid01)*, 2001.

[YCL+08] Zhifeng Yun, Sun Joseph Chang, Zhou Lei, Gabrielle Allen, and Ashwin Bommathanahalli. Grid-enabled sawing optimization: from scanning images to cutting solution. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, MG '08, pages 16:1–16:8, New York, NY, USA, 2008. ACM.

[YKA05] Kenneth Yoshimoto, Patricia A. Kovatch, and Phil Andrews. Co-scheduling with user-settable reservations. In *In Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP)*, pages 146–156, 2005.

[YKX+07] Zhifeng Yun, Samuel J. Keasler, Maoyuan Xie, Zhou Lei, and Gabrielle Allen. An innovative simulation approach for water mediated attraction based on grid computing. In *Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences*, IMSCCS '07, pages 204–211, Washington, DC, USA, 2007. IEEE Computer Society.

[Yun10] Zhifeng Yun. Application execution using hybrid resources. Technical report, National Center for Atmospheric Research - Computational and Information Systems Laboratory, August 2010.

[YXZ+08] Zhifeng Yun, Maoyuan Xie, Fuguo Zhou, Gabrielle Allen, Tevfik Kosar, and Zhou Lei. Collaborating mechanical design phases across a grid. In *Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering - Workshops*, pages 65–70, Washington, DC, USA, 2008. IEEE Computer Society.

[ZKC09] Yang Zhang, C. Koelbel, and K. Cooper. Batch queue resource scheduling for workflow applications. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER '09)*, pages 1 –10, Sept. 2009.

Appendix: Nomenclature

Symbol	Description
$A_i(t)$	random variable of the number of jobs submitted between 0 and t to C_i
C_i	local cluster i
c_i	computational capability of the cluster C_i
$D_i(t)$	random variable of the number of jobs left C_i during 0 and t
F_i	fraction of total TCs to cluster C_i
i	local cluster index
J	total number of task containers
J_i	number of task containers submitted to cluster i
K	number of iterations in inverse modeling
k	inverse modeling iteration index
L_i	workload submitted to cluster i
l	job size
m_i	number of CPUs of C_i
N_i	number of jobs in C_i
$\widetilde{N}_i$	number of jobs already finished when TCs in cluster C_n get the resource
$\widetilde{\widetilde{N}}_i$	rest number of jobs to cluster C_n when TCs in cluster C_n get the resource
N	total number of tasks
$\widetilde{N}$	rest number of jobs when TCs in cluster C_n get the resource
n	number of all local clusters
p_0	limiting probability that the system contains 0 members

Symbol	Description
Q_i	average queue size of C_i
R_i	percentage of waiting time to the total response time
s_i	relative speed of C_i
T_i	average job response time of cluster i
T_k	turnaround time in each inverse modeling iteration
T_{total}	turnaround time in inverse modeling
T_n'	turnaround time in rest iteration in DA-TC for inverse modeling
T	turnaround time of whole application
T	application turnaround time
t	simulation duration
X_i	expected job execution time
W_i	expected waiting time in the queue

Greek

λ_i	mean job arrival rate of cluster i
μ	base-line job service rate
μ_i	mean job service rate of a node in cluster i
ρ_i	utilization of the systems

Abbreviations

AEA	Application Execution Agent
DA-TC	Dynamic Assignment with Task Container

Symbol	Description
EnKF	Ensemble Kalman Filter
FCFS	First Come, First Served
GEMS	Grid Execution Management Service
GUI	Graphic User Interface
LRMS	Local Resource Management Systems
NHLA	National Hardwood Lumber Association
TC	Task Container
VCP	Virtual Cluster Pool

Vita

Zhifeng Yun was born in Changzhou, China, in 1981. He graduated from Huazhong University of Science and Technology, Wuhan, in 2004 with a bachelor's degree in the electronics and information engineering with Honors. After graduation he started the masters program in the school of electronics and computer science at University of Southampton, U.K. He earned his M.Sc. degree with Distinction from University of Southampton in January 2006.

He was offered a graduate research assistantship from Louisiana State University and started the doctoral program in the Department of Electrical and Computer Engineering in 2006. He joined the Center for Computation and Technology as a research assistant in 2007. Since then he has worked on several research projects and won student scholarships from both the Center for Computation and Technology and the Department of Electrical and Computer Engineering.

His research is in high-performance and distributed computing with emphasis on resource and execution management. He is also interested in cloud computing, system design, and application exploration in very high end architectures. His articles and publications include:

- Zhou Lei, **Zhifeng Yun**, and Gabrielle Allen,"Grid Resource Allocation," in *Grid Computing: Infrastructure, Service and Applications*, Ed: L. Wang, J. Wei, and J. Chen, CRC Press, pp. 171-190, April 2009 (ISBN: 978-1-4200-6766-8).

- H.-C. Wu, M. Saquib and **Z. Yun**,"Novel automatic modulation classification using cumulant features for communications via multipath channels," *in IEEE Transactions on Wireless Communications*, vol.7, no.8, pp. 3098-3125, August 2008.

- **Z. Yun**, Z. Lei, G. Allen, D. Katz, T. Kosar, S. Jha, and J. Ramanujam, " An Innovative Application Execution Toolkit for Multicluster Grids," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing*, pp.1-4, New Orleans, LA, 2009.

- Z. Lei, **Z. Yun**, G. Allen, X. Li, N. -F. Tzeng, C. White, "Improving Application Execution in Multicluster Grids," in *Proceedings of 11th IEEE International Conference on Computational Science and Engineering*, pp.163-170, Sao Paulo, Brazil, 2008.

- **Z. Yun**, M. Xie, F. Zhou, G. Allen, T. Kosar, and Z. Lei, "Collaborating Mechanical Design Phases Across A Grid," in *Proceedings of 11th IEEE International Conference on Computational Science and Engineering Workshops*, pp.65-70, Sao Paulo, Brazil, 2008.

- **Z. Yun**, S. J. Chang, Z. Lei, G. Allen, and A. Bommathanahalli, "Grid-enabled Sawing Optimization: from scanning images to cutting solution," in *Proceedings of the 15th ACM Mardi Gras Conference*, no. 16, Baton Rouge, LA, 2008.

- **Z. Yun**, S. J. Keasler, M. Xie, Z. Lei, B. Chen and G. Allen, "An Innovative Simulation Approach for Water Mediated Attraction Based on Grid Computing," in *Proceedings of International Multi-Symposiums of Computer and Computational Sciences*, pp. 204-211, Iowa City, Iowa, 2007.

- M. Xie, **Z. Yun**, Z. Lei and G. Allen, "Cluster Abstraction: Towards Uniform Resource Description and Access in Multicluster Grid," in *Proceedings of International Multi-Symposiums of Computer and Computational Sciences*, pp. 220-227, Iowa City, Iowa, 2007.

- A. Bommathanahalli, M. Xie, **Z. Yun**, S. J. Chang, Z. Lei and G. Allen, "TOPSAW Sawing Optimization Analysis Using Grid Computing," in *Proceedings of International Multi-Symposiums of Computer and Computational Sciences*, pp. 228-234, Iowa City, Iowa, 2007.

- **Z. Yun**, Z. Lei, D. Katz, J. Ramanujam, G. Allen, S. Jha, and T. Kosar, "Integrating Multiclusters for Efficient Application Execution," Poster reception, in *Proceedings of the 2009 IEEE/ACM Conference on Supercomputing (SC09)*, Portland, Oregon, USA.

- **Z. Yun**, G. Allen, D. Katz, T. Kosar, and S, Jha, "Pelecanus: An Innovative Application Execution Toolkit for CyberTools," in *Proceedings of Louisiana EPSCoR RII Cybertools/Science Drivers 2009 Symposium*, Baton Rouge, LA, 2009.

- **Z. Yun**, G. Allen, J. Ramanujam, and D. Katz, "Integrating Multiple Clusters for Compute-Intensive Applications," in *Proceedings of Louisiana EPSCoR RII Cybertools/Science Drivers 2010 Symposium*, Baton Rouge, LA, 2010.