ADAPTIVE SCIENTIFIC WORKFLOWS

By

ARZU GOSNEY

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
Department of Computer Science

DECEMBER 2011

To the Faculty of the Washington State University:

The members of the Committee appointed to examine the dissertation of ARZU GOSNEY find it satisfactory and recommend that it be accepted.

John H. Miller, Ph.D., Chair

Christopher Oehmen, Ph.D.

Li Tan, Ph.D.

Carl Hauser, Ph.D.

ACKNOWLEDGEMENTS

Words cannot express how grateful and forever appreciative I am for the guidance, the motivation and the inspiration that each person has provided during the course of this dissertation. Nevertheless I make a sincere attempt to thank everyone who played a part in the completion of this dissertation.

First, I would like to thank both Dr. Christopher Oehmen and Dr. John Miller. It is to them that I owe an awful lot. They both have played a major role, did more than just explaining things to me, and as busy as they are, they both made themselves available when I needed their time the most. Dr. Miller not only explained the steps in what it would take to complete of my degree, but he also followed along with my progress and provided his years of experience and supervision over my research work. If it wasn't Dr. Miller's direction I would have given up in the middle of my research work. Dr. Christopher Oehmen on the other hand is an amazing mentor. He is not only a great scientist full of ideas to improve and present research, but he also sincerely cares about his students' work. He expects the best out of his students and is willing to dedicate his time to improve their research work as well as their conference presentations, papers, etc. If it wasn't for Dr. Oehmen's support at Pacific Northwest National Laboratory, I would not have the funding and the resources to complete my research. Next on the list is Dr. Ian Gorton, who introduced me to the field of high performance data intensive computing and scientific workflows. My discussions with Dr. Gorton on software engineering and software architecture principles as well as scientific workflows gave insight into my research.

My acknowledgement section can never be complete without the mention of my husband, Greg Gosney, who had a major role to play in this dissertation. It was him who believed in me when I did not. It was his never ending, unconditional support which got me through the hardest days. Lastly I acknowledge the support of my family members, my dad, Mehmet Kurtulus and my mom, Emine Kurtulus, who have constantly help us in the most loving way when we needed their help to take care of our beautiful children.

I would also like to acknowledge my funding source. This research was performed and partially funded by Laboratory Directed Research and Development (LDRD) Funds at Pacific Northwest National Laboratory.

ADAPTIVE SCIENTIFIC WORKFLOWS

Abstract

by Arzu Gosney, Ph.D.
Washington State University
December 2011

Chair: John H. Miller

Large computing systems including clusters, clouds, and grids, provide high-performance capabilities that can be utilized for scientific applications. As the ubiquity of these systems increases and the scope of analysis performed on them expand, there is a growing need for applications that do not require users to learn the details of high-performance computing (HPC), and are flexible and adaptive to accommodate the best time-to-solution. In this dissertation we introduce a new adaptive capability for the MeDICi middleware and describe the applicability of this design to a scientific workflow application for biology. This adaptive framework provides a programming model for implementing a scientific workflow using high-performance systems and choosing configuration options at run-time, automatically reacting to HPC load fluctuations.

In production multi-user high-performance (HPC) batch computing environments, wait times for scheduled jobs are highly dynamic. For scientific users, the primary measure of efficiency is wall clock time-to-solution. In high throughput applications, such as many kinds of biological analysis, the computational work to be done can be flexibly scheduled taking a longer time on a small number of processors or a shorter time on a large number of processors. Therefore the capability to choose a platform at run-time based on both processing capabilities and availability (lowest wait time) would be attractive. The goal of our work was to create an adaptive interface to HPC systems that dynamically reschedules high-throughput calculations in response to fluctuating load, optimizing for time-to-solution. This was done by implementing middleware functionality to (1) monitor the resource load on a given compute cluster, (2) generate a plan, checking on the applicability of the plan with the defined goals and (3) adaptively choosing the optimal job dimensions (number of processors and wall-clock time) to provide the best time-to-solution results.

ABBREVIATIONS

HPC	High Performance Computing
MeDICi	Middleware for Data Intensive Computing
MIF	MeDICi Integration Framework
PNNL	Pacific Northwest National Laboratory
BPEL	Business Process Execution Language
ASF	Adaptive Server Framework
JEE	Java Platform Enterprise Edition
MAPE	Monitoring, Analysis, Planning and Execution
SPA	Sense Plan Act
QoS	Quality of Service
SLA	Service Level Agreement
NCBI	National Center for Biotechnology Information

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Dedication

This dissertation is dedicated to my daughter

Melisa Nur Sencer

and my son

Burak Parker Gosney.

Our children are the future to many generations to come.

CHAPTER ONE

1. INTRODUCTION

Promoting collaboration in the business world has been an ongoing activity that has led to the establishment of widely used tools for social and knowledge networks. Bringing a similar level of collaborative working environments to scientists is crucial for numerous research areas as well. However, this new way of sharing and managing information to electronically capture end-to-end science processes and to share them for future reference and reuse, creates many challenges and numerous research areas as scientific data analysis processes massive data volume and requires high- performance computing. Scientific users are increasingly dependent on large-scale HPC (High Performance Computing) architectures for analysis and calculations due to the exponential growth in data generation and the resulting data sets. However, the steep learning curve for utilizing these systems has prohibited many potential users from deriving the greatest value of HPC platforms (1) (2).

In multi-user batch HPC systems, users submit jobs to a central resource manager that keeps track of which jobs are currently running, what resources they are utilizing (usually compute nodes), and the expected run-time of these jobs. On oversubscribed systems, typical of medium- to large-scale HPC installations, there is a process that manages a job queue to enforce usage policies. For instance, if a user’s job is still running when the user’s requested wall-clock time has elapsed, it may be forcibly terminated to ensure that users do not abuse the resources allotted to them. Running jobs may fail, due to programmer errors, input data errors such as failure to have all the correct input files available at run time, or because of hardware faults. The job queue manager may also watch for node failures and “kill” the remainder of a job if one of the nodes fails. To make matters more complex, the queue manager process also tracks the priority of jobs awaiting execution. The scheduler can manage reservations in advance and keep track of short time queue for quick running jobs (3).

For these reasons, jobs of any size may terminate significantly before their intended end time, potentially changing the availability of nodes for other jobs in a dramatic way. As a result, there is a constantly evolving estimate of when each job in a queue is likely to run given the current distribution of available and reserved nodes, the current profile of pending jobs, and the potential for failure of running jobs. This provides a rich opportunity for adaptively changing job run-time requirements so that a given task may run as early as possible (3).

To address this challenge, we aim to simplify the details of utilizing HPC platforms while at the same time improving a user’s time to solution for a given computational task. We achieve this by creating an adaptive middleware solution that is based upon the MeDICi (Middleware for Data Intensive Computing) Integration Framework (4) that has been designed to address the challenges of building high- performance, distributed data intensive applications. MeDICi Integration Framework (MIF) integrates component-based and service-oriented approaches to provide a flexible development and deployment environment for scientific workflows (5).

The remainder of this dissertation presents an implementation of a novel capability for flexible job submission to a dynamic HPC queuing environment based upon MIF, but augmented with adaptive capabilities. We also analyze and quantify the benefits in performance that are achieved by using this adaptive framework.

2. MOTIVATION

Middleware is computer software that connects software components or applications, providing a set of services or interfaces that allow multiple processes running on one or more machines to interact. Previously, researchers at Pacific Northwest National Laboratory (PNNL) implemented MeDICi to address the challenges of building high- performance, data intensive applications. MeDICi integrates component-based and service-oriented approaches to provide a flexible development and deployment environment for scientific workflows (6).

Though MeDICi enables one to construct pipelines using a combination of computing and data management components, these pipelines are static in nature, and therefore unable to easily accommodate dynamic computing environments. The concept of adaptivity is therefore critical for building flexible MeDICi applications that can react to changing user load, system failure, and system availability. Adaptive workflows adjust at runtime usually due to an unexpected event, such as a hardware failure of a node. This is in contrast to static pipelines in which exceptions force a process to fail rather than adjust to change. According to The Workflow Management Coalition (7), the requirements of dynamic adaptive workflows include, 1) contingency management and handoff, which provide mechanisms for dealing with and recovering from expected and unexpected divergence from the intended process; 2) partial execution, which supports creating and executing processes and process fragments (modules) as they are needed, rather than requiring the entire process be rigidly specified ahead of time; 3) dynamic behavior in terms of both execution model and object behaviors provides flexibility to modify workflow paths and executed behaviors at run-time independent of object data and, 4) reflexivity, which allows a workflow component to programmatically examine, analyze, create and manipulate its own process and data as part of automatable tasks during execution (8).

In this dissertation, an adaptive design was proposed for MIF as a next generation scientific workflow. Our design approach takes the Gat's three-layer architecture design and breaks it into a layered implementation making it modular and introduces the adaptivity as a separate framework within MIF. Our architecture with distinct layers is more suitable for integrating with software architectures implemented as frameworks such as MIF. We proposed, designed and implemented an adaptive middleware framework and show the run-time results in a production HPC environment. Furthermore, we also analyze and quantify the benefits in performance that are achieved by using our adaptive MIF framework.

3. OBJECTIVES AND CONTRIBUTIONS

In multi-user batch HPC systems compute jobs are scheduled on non-overlapping resources so that each job is unimpeded by other jobs running at the same time. When a user submits a new job to the resource manager, or scheduler, the user specifies what resources they need (*e.g.* how many processors), and estimates how long these resources will be utilized.

The most straightforward form of priority is first come, first serve. In this approach, the top job in the queue "reserves" resources until they are available, then it runs. Other jobs that can run with leftover resources concurrently with the top priority job may also be allowed to run. At this point, the next job that was submitted to the queue would start reserving resources, and when its required resources are available, it would begin execution. This combination of reservation for the top priority job and "backfilling" for the lower priority jobs often achieves good utilization of the compute resources (3).

However, on many specialized systems, more sophisticated queue policies are used. For instance, some large-scale computing centers prioritize jobs that require a large number of processors. In this policy, a job that has the top priority in the queue can be “skipped” by a more recently submitted job that requires more processors, creating a highly dynamic situation for lower priority jobs (3).

To make matters more complex, running jobs often fail—either because of programmer error such as segmentation fault; input error such as failure to have all the correct input files available at run time; or because of hardware fault. In any case, jobs of any size may terminate significantly before their intended end-time potentially changing the availability of nodes for other jobs in a dramatic way. As a result, there is a constantly evolving estimate of when each job in a queue is likely to run given the current distribution of available and reserved nodes, the current profile of pending jobs, and the potential for failure of running jobs. This provides a rich opportunity for adaptively re-sizing jobs with flexible run-time requirements so that they can run as early as possible given a shifting distribution of available and pending nodes (3).

In this dissertation, adaptive capabilities developed for use with MeDICi to address the need for dynamically scheduling high-throughput scientific calculations on multi-user batch systems are presented and it is shown that this approach can correctly optimize wall clock time-to-solution on a multiuser system using real load data from a large-scale system.

4. RELATED WORK

As the complexity of current software systems and uncertainty in their environmnet is increasing, sofware engineering community is looking for inspiration in diverse related fields (e.g., robotics, artificial intelligence, control theory, etc.) for new ways to design and manage systems and services that are "self-adaptive" (9). As self-adaptation is becoming one of the most promising research direction, the "self" prefix indicates that the system decides how to react to changes at run-time without or with minimal interference (9).

Self adaptive systems can be categorized by their operating principles or multiple dimenstions of properties, such as centralized or de-centralized, top-down or bottom-up approach, environment uncertainty (low/high dynamics of the current environment), etc. A top-down self adaptive system is ofen centralized and operates with the guidance of a central controller or a policy maker, assesses its own behavior in the current surrondings, and adapts itself if the monitoring and analysis warrants it. Such a system operates with an explicit representation of its environment and its global goals. By anaylizing the components of the global goals once can predict the adaptation behavior of the self adaptive system (9).

In contrast, a self adaptive system that is designed with bottom-up principles alone usually employs decentralized components that interact locally with simple rules without a central authority. It is often difficult to analyze the global properties of such self adaptive systems by examining the local interactions of its components (9).

Most engineered self adaptive systems fall somewhere between these two extreme cases of self adaptive system types. There are several software languages, architectures, and scientific workflow applications that try to address adaptivity at some level in their usage, design, and implementations.

4.1. Software languages for adaptivity

BPEL, short for Business Process Execution Language, is an executable language for specifying interactions with web services. BPEL uses generic XML data types to provide flexibility with optional value selections, therefore providing dynamic selection of services at runtime for adaptivity (7).

The ASF (Adaptive Server Framework) is implemented on top of JEE (Java Platform Enterprise Edition) application servers, and uses components to enable MAPE (Monitoring, Analysis, Planning and Execution) based adaptive behavior for JEE based applications (10) (11). ASF provides an extensible framework in which components can monitor and sense the change within a process, analyze the change and decide whether or not to adapt to the change. ASF is a module layered on JEE servers, and importantly provides an approach that is non-intrusive to the application code itself that is being augmented with adaptive behavior.

4.2. Software architectures for adaptivity

The autonomic computing community proposed an architecture known as MAPE, which provides a structure and methodology for developing adaptive systems (12). The MAPE model creates execution plans and revises application behavior in response to external changes in the application's environment.

Dynamic embedding is another adaptation technique utilizing frames and templates to separate control-flow from data-flow. A frame wraps a collection of possible actor implementations and a template specifies a sub-workflow with "holes" that can be filled in at design time or run time with actors or other templates. Dynamic embedding takes the frames and templates approach one step further by allowing actors and control-flow behavior to be selected at workflow runtime.

An alternative adaptive software architecture was proposed by Kramer and Magee (13). They based their proposal for self-managed systems on Gat’s three-layer model (14). Gat’s paper took the early robotics, SPA (sense-plan-act) approach and proposed a three-layer control-sequence-deliberation model that formed the foundation of Kramer and Magee’s three-layer adaptive architecture. The three- layer conceptual model for self-management introduced by Kramer and Magee provides generality to a range of application domain adaptations, and is the basis for the design of our Adaptive MIF technology described in this dissertation. A more detailed description of the theory of the adaptive capabilities used in this application was presented in (3) and the prototype of our design was demonstrated in (15).

4.3. Scientific workflow applications for adaptivity

Pegasus is a scientific workflow application which employs an adaptive workflow model based on the MAPE architecture utilizing loosely coupled, reusable components (12). The e-HTPX project, a scientific workflow application for high-throughput protein crystallography utilizes the standardized workflow language BPEL for adaptation (8). The generic data type “any” in BPEL is used to wrap arbitrary XML fragments that can be linked to implementations. Although this technique provides flexibility to run different code implementations, it puts a greater work load on the web service which must examine the different messages to react appropriately. Furthermore, there is little flexibility with reruns of the wrapped object.

By contrast, adaptivity in KEPLER, another scientific workflow technology, uses dynamic embedding to discover suitable actors within frames and templates (16) (17). In this approach, frames and templates are introduced to separate control-flow from data-flow.

CHAPTER TWO

2. HIGH PERFORMANCE COMPUTING

HPC uses supercomputers and compute clusters to solve advanced computation problems. Scheduling in HPC systems is becoming an increasingly important and difficult task. As an HPC system can have as many as 10^5 multi-threaded processors it is desirable to operate such systems as efficiently as possible.

HPC system at PNNL is based on the Moab® scheduler. Moab® dynamically adapts HPC resources on demand to match workload needs, an essential capability for delivering HPC as a service and HPC cloud (18) (19). Moab® is a complete solution to manage HPC environment with complete support for workload management, job scheduling, and an adaptive OS switcher for Linux & Windows workloads all rolled into one. The diagram below shows the Moab® architecture diagram (18).

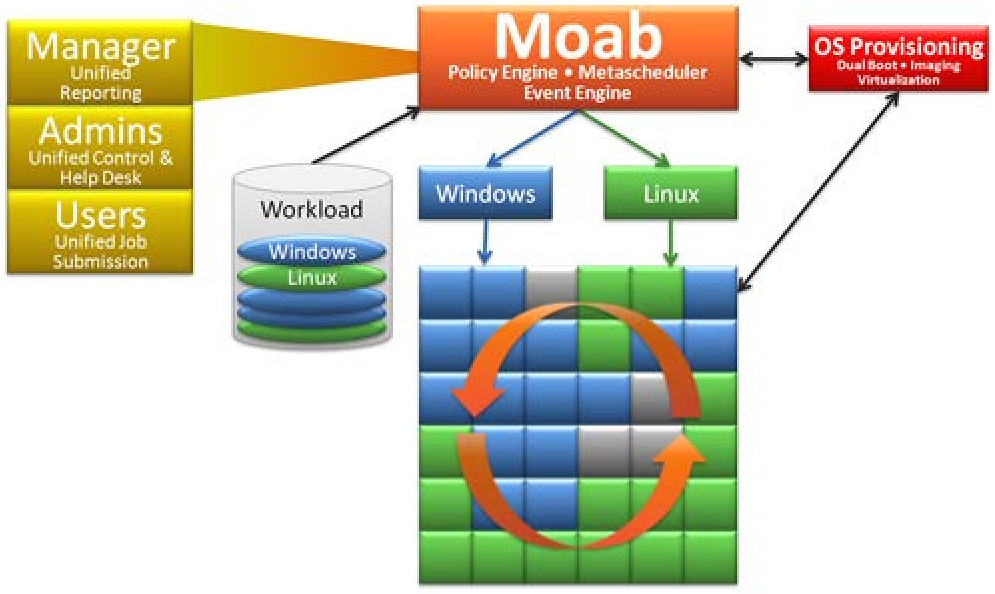


Figure 1. Moab(R) Architecture Diagram

2.1. Moab Policy Engine

Moab® is designed to run thousands of jobs per hour across thousands of nodes supporting various configurations to serve the needs of a typical HPC environment. Moab® enables a system level adaptive HPC environment by allowing the changing needs and failed systems to be automatically fixed or replaced. Moab® applies site policies and extensive optimizations to orchestrate jobs, services, and other workload across the ideal combination of network, compute and storage resources (20). Moab® by itself increases system resource availability, offers extensive cluster diagnostics, delivers powerful QoS/SLA (Quality of Service/Service Level Agreement) features, and provides rich visualization of cluster performance through advanced statistics, reports, and charts (20).

Moab® has a full set of features for job prioritization. It supports priorities based on credentials, resources, usage, and job attributes. Priorities of jobs can be changed while the job is queued and user priorities can be provided at runtime (20). Moab® automatically increases the priorities of jobs based on their queue time to avoid starvation (20). Moab® also provides advanced capabilities for reserving HPC resources for any period of time. It guarantees the availability of the reserved resources when a reservation is started. The advanced reservations enable Moab® to backfill jobs, provide deadline based scheduling, and QoS support. All the flexibility Moab® policy engine brings creates a highly dynamic and unpredictable HPC queue environment.

2.2. Moab Scheduler

Moab® can schedule, monitor, and manage jobs using existing scheduler and resource management technologies deployed to HPC as well as provide a single view to Administrators. While Moab makes the scheduling and allocation decisions, the Resource

Managers provide Moab® with input on current resource availability, but the Resource Manager itself is in charge of orchestrating the actual job staging and job execution.

Moab® supports the specification of various resource parameters during job submission: nodes, memory, cpu, generic resources, wall time, node features, start time, etc. Moab® supports options for passing in runtime parameters to jobs. Moab® provides all the basic job management functions such as start, stop, cancel, hold, restart, suspend/resume. It can also provide the user with the exit status. Moab has an extensive set of scheduling algorithms. It can schedule batch jobs, parallel jobs, and service workload. Extensive user tutorial on how to submit jobs and to use Moab® is outlined in (21).

2.3. Moab Limitation on Adaptive HPC

There is continuing research on the optimization of HPC local storage space and the scheduling algorithms as nodes in HPC clusters usually have processor heterogeneity, load variation and dynamic availability (22). HPC systems can have multiple jobs with different execution priorities and need to address dynamic environment changes such as subsequent workload and system changes (23). Moab is a highly advanced scheduling and management system designed for clusters, grids, and on-demand computing systems. HPC optimal scheduling policy algorithms try to address the cluster wide need to use as many nodes as possible at a time dynamically. However, scientists who have applications that can run on as many parallel nodes as possible need a way to dynamically adjust their run-time parameters based on the availability of HPC cluster nodes to obtain optimal run times. Therefore, a second tier adaptivity within a scientific workflow middleware is needed to help address the challenges of HPC cluster optimization problem (3).

CHAPTER THREE

3. MIF and ADAPTIVE MIF FRAMEWORK COMPONENTS

MIF (Middleware for Data Intensive Computing Integration Framework) allows researchers to build scientific workflows or “pipelines” of heterogeneous software components, each of which performs some analysis of the incoming data and passes on its results to the next software component in the pipeline. As MIF was designed to address the challenges of building high- performance, distributed data intensive applications, today MIF is being used in several data intensive computing software applications such as cyber analytics, proteomics, and text analysis (5) (24). MIF is open source and freely downloadable from http://medici.pnl.gov.

3.1. MIF Architecture

As illustrated in Figure 2, MIF architecture leverages open source middleware technologies and imposes a component-based programming model on the virtual machine provided by the underlying platform. The resulting MIF architecture is described in (15).

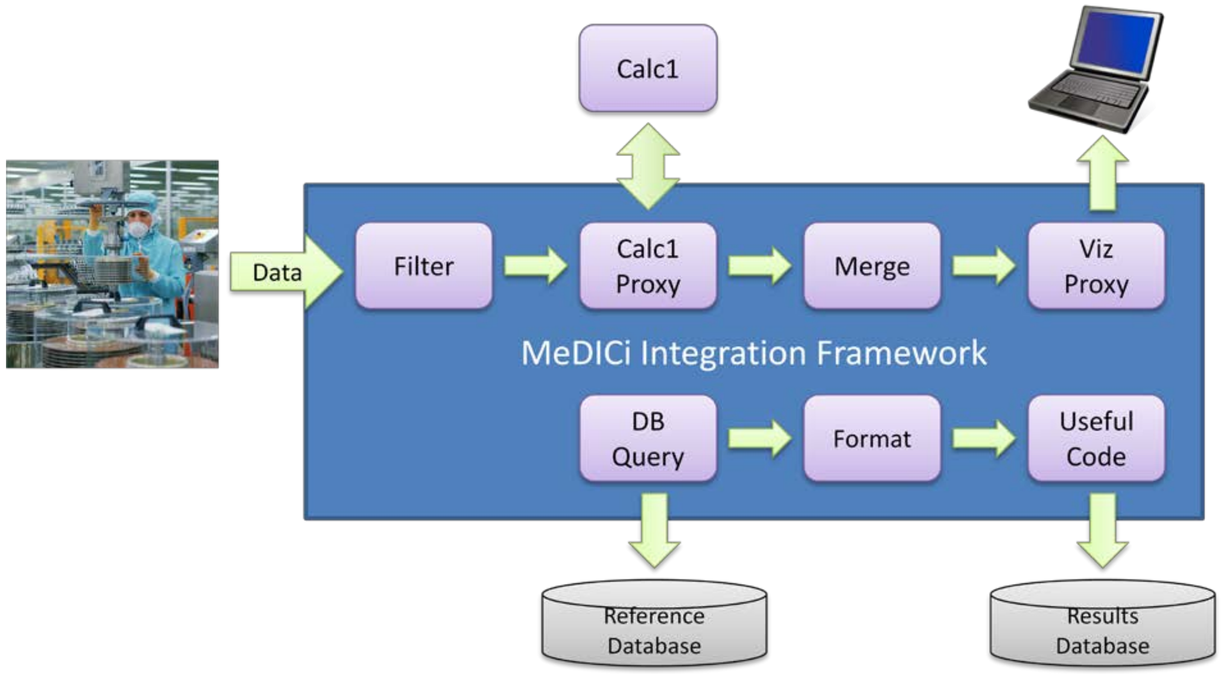


Figure 2. The MeDICi Integration Framework (An Example MeDICi Pipeline)

MIF components are constructed using a Java API that supports inter-component communication using asynchronous messaging. Local components execute inside the MIF container. Remote components support the same programmatic API, and utilize additional MIF facilities to execute component code outside the MIF container. Remote components are used to create distributed solutions and to integrate with non-Java codes (4) (25).

MIF also provides a BPEL-based design and execution environment that integrates with MIF components to provide workflow definition tools and a standards-based recoverable workflow execution engine.

3.2. Adaptive MIF Framework Components

In our work, MIF was enhanced to incorporate adaptivity based on Kramer and Magee’s model reported earlier (13). In this model, the bottom *component layer* (Figure 3) comprising independent control components reports the current status of the monitored application environment to the higher levels in the architecture. If necessary, the component layer also adjusts the operating parameters of the environment. The middle *change management* layer is responsible for analyzing changes that are reported from the controls below or from the new objectives that are reported from the layer above. Once there is a change of action, meaning an adaptation response, this layer communicates with the control components and directs the actions to be taken. The upper *goal management* layer defines the high-level goal and introduces a plan to achieve it.

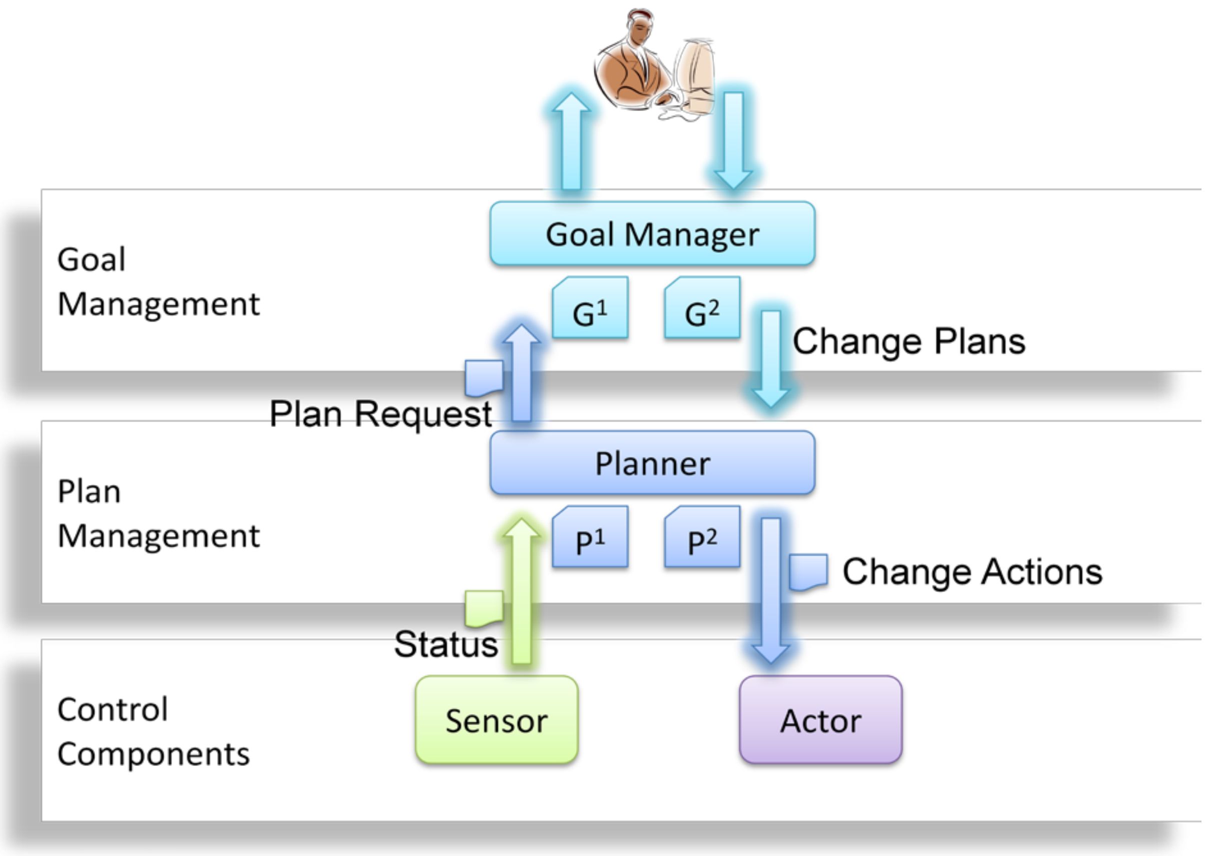


Figure 3. MIF Adaptive Architecture

As the driving use case for our work on introducing adaptivity into MIF, we describe the adaptive MIF design based on a pipeline to efficiently schedule batch jobs on a HPC platform and address the problem described in Chapter 1. The adaptive MIF pipeline is designed to optimize time-to-solution for high-throughput computations such as biological analysis of sequence data, where computing can be a rate limiting step. For such applications, total processing time is directly related to the number of processors used for a calculation. In (26) it was shown that our test application scales linearly to thousands of processors on HPC. Test application that was used for the basis of our runtime results will be discussed in further detail in chapter five. There is flexibility in choosing the number of processors so that a small processor pool can be utilized if this means the queue wait time is significantly reduced. Using MIF components, a pipeline for high-throughput data-intensive analysis was constructed. The initial pipeline design, as shown in Figure 4, includes no adaptivity and simply moves the necessary input files to the HPC execution platform and schedules a batch job on the compute cluster. The pipeline then waits for the job to complete. Once the batch job completes the pipeline generates the batch run output file and moves it back to the remote resource location.

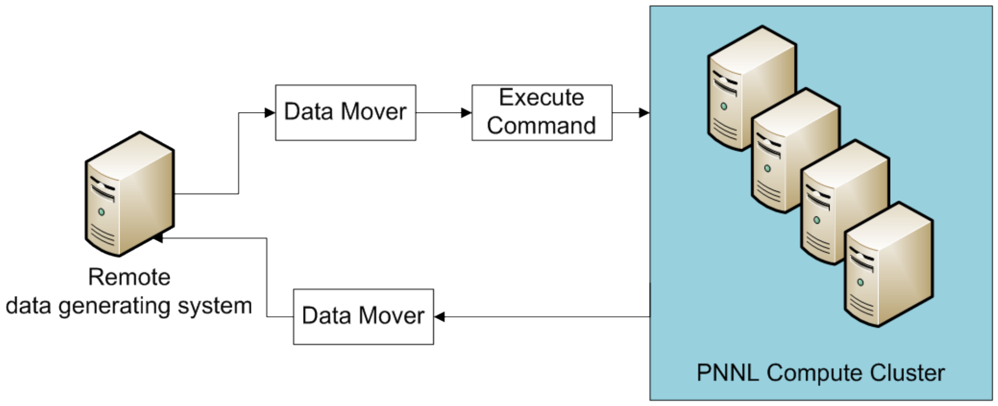


Figure 4. MIF Pipeline Design (Without Adaptivity)

Although this pipeline automates the manual tasks undertaken by the user when submitting jobs to a HPC platform, job execution is still completely controlled by the platform job scheduler, which may not provide a sufficiently rapid turnaround time to obtain results if the job queues are long. Therefore, we augmented the simple MIF pipeline with an adaptive capability to optimize the execution of scheduling the batch job on the compute cluster. Figure 5 depicts the adaptive MIF design, with the new components shown in yellow.

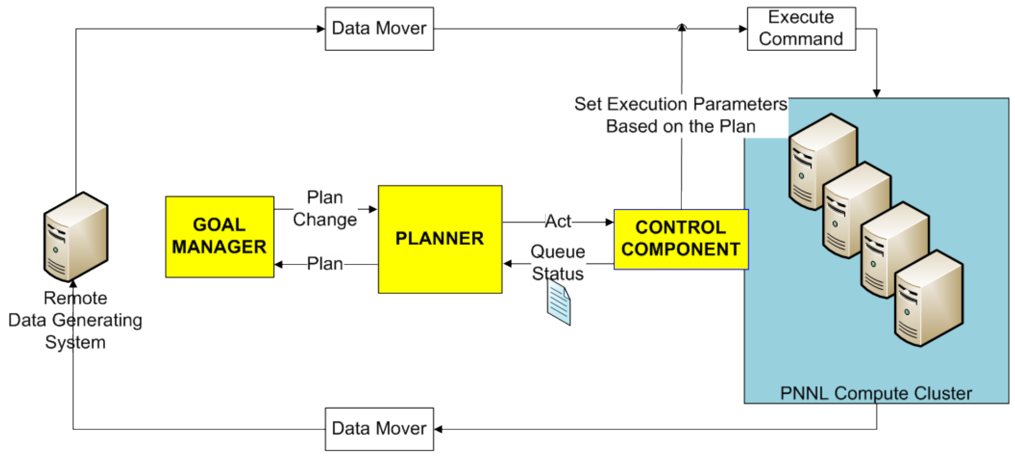


Figure 5. Adaptive MIF Pipeline

As can be seen from Figure 5, the adaptive MIF framework has three layers; the Control component layer, the Planner layer, and the Goal manager layer. These layers communicate with each other via asynchronous messaging.

As described earlier, a system is adaptive if it is able to adjust its behavior in response to its perception of the environment and the system itself (27). The question then becomes how an adaptive system will make adaptation decisions based on these observations. A single iteration of the adaptive MIF framework is shown in the UML sequence diagram in Figure 6. A UML component diagram for adaptive MIF framework can also be found in Appendix D.

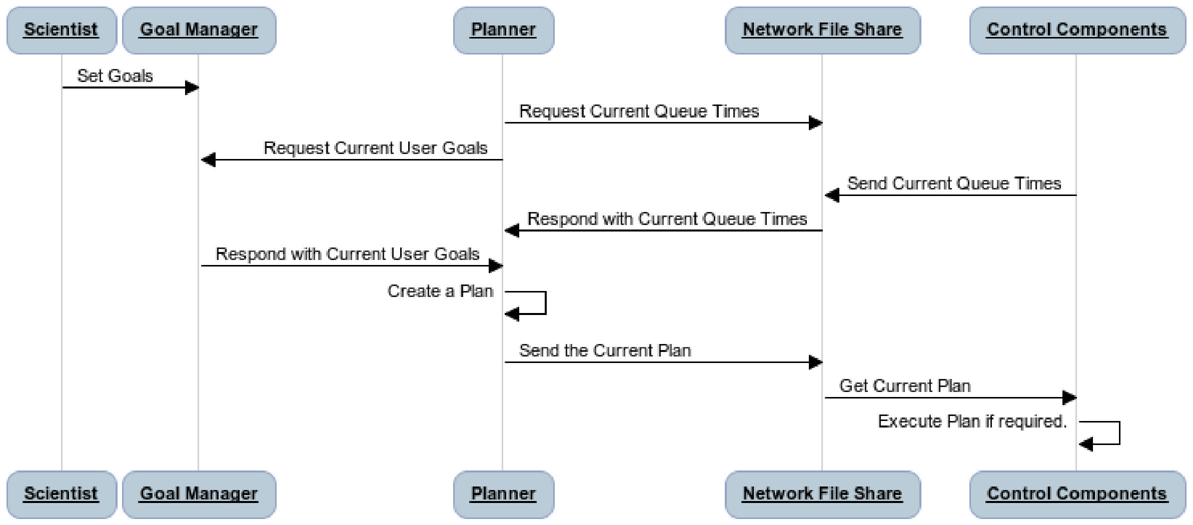


Figure 6. Adaptive MIF Framework UML Sequence Diagram

In our use case, the MIF adaptive framework continuously monitors the status of the cluster queues. The control component layer of our pipeline senses the queue status every 20 seconds and outputs the queue status into a file. It then moves this file to a location for the upper adaptive layer, the planner, to examine. The control component layer is also responsible for receiving messages from the planner and acting on these messages via setting the input parameters for a batch job.

The adaptive planner layer communicates with both the upper goal manager layer, and the lower control components. The planner in an adaptive middleware application encapsulates the core of the specific adaptive strategy. In our pipeline, the planner looks at the compute cluster queue status and makes the decision to pick which queue slot our batch job can best fit into, and notifies the goal manager.

The Goal manager examines the new plan and if the new plan will give better time-to-solution given the constraints of users' goals for the job, it notifies the planner to change the batch run input parameters, number of processors and wall-clock times. The planner then notifies the control component to take action on the cluster queue based on the new plan. The Goal Manager Layer is responsible for directing the overall actions, the goals of the planner and responding to changes in the environment when communicated by the planner. The MIF adaptive pipeline framework continues to monitor the queues as long as the batch job is waiting and terminates once the job is running. Each layer of MIF adaptive architecture will be examined in more detail during the rest of this chapter.

3.2.1. Control Component

The control component layer of our pipeline senses the queue status every 20 seconds and outputs the queue status into a file. It then moves this file to a location for the upper adaptive layer to examine. The control component layer is also responsible for receiving messages from the planner and acting on these messages via setting the input parameters for our batch job. The control component essentially utilizes MIF components to act and sense change. The summary of the control component sensing process is as follows:

Wake up every 20 seconds

Run the queue status script

Write output to a file

Move the output file to a temporary remote location for the planner to examine

The summary of the control component queue manipulation process is:

```
Wake up every 20 seconds
If the batch job is already scheduled but not running and there is a change in the plan
    Cancel the scheduled batch job due to change in plan
    Rerun the preprocessing batch job
    Schedule the batch job using the new parameters
```

3.2.2. Planner

The adaptive planner layer communicates with both the upper; goal manager layer, and the lower; control component layer as depicted in Figure 3. The planner in adaptive middleware usually encapsulates the core of the specific adaptive strategy. In our pipeline, the planner looks at the compute cluster queue status and makes the decision to pick which queue slot our batch job can best fit into, notifies the goal manager. The Goal manager examines the new plan and if the new plan is giving better time-to-solution results, it notifies the planner to change the batch run input parameters, number of processors and wall-clock times. The planner then notifies the control component to take action on the cluster queue based on the new plan.

In the pipeline, the planner gets queue status information from the control component in a text file format. A (shortened) example file from a compute cluster is shown in Table 1. Appendix A shows an example output of a production HPC queue status file.

Table 1. A Simplified Queue Status File

Total Number of Jobs: 6 4 running, 2 pending				
Time	**State**	**Job**	**Nodes Required**	**Nodes Available**
10:00	NOW	-	499	500
10:00	START	**A**	-400	100
10:00	START	**B**	-50	50
10:00	START	**C**	-20	30
10:00	START	**D**	-29	1
10:30	*FINI	**D**	29	30
10:30	START	**E**	-29	1
11:40	*FINI	**E**	29	30
10:40	START	**F**	30	0
11:42	*FINI	**F**	30	30
11:00	*FINI	**C**	20	50
11:30	*FINI	**B**	50	100
12:00	*FINI	**A**	400	500

As it can be seen from Table 1, currently there are a total of 6 batch jobs in HPC, 4 of which are running and 2 out of 6 are pending, waiting in the HPC queue to run. Current time is 10:00 and job A is going to start at time 10:00, right away, reserving 400 nodes out of 500 in the cluster. At time 12:00, job A finishes, giving it a 2 hour estimated run time, resulting with a 2 hour total time to completion as there is no wait time. Job B also runs right away at time 10:00, reserving 50 nodes and finishes at time 11:30, with a total time to solution of 1.5 hours with no wait times. Job C starts at time 10:00 with no wait time as well. Job C reserves 20 nodes with a total time to solution of 1 hour, ending at time 11:00. Job D also runs right away reserving 29 nodes, with a total run time of 30 minutes; on 29 nodes. Job E on the other hand starts at time 10:30 with a wait time of 30 minutes. The run time to run Job E on 29 nodes is 10 minutes. Therefore the total time to solution for Job E is 40 minutes; 30 minutes of wait time plus the 10 minutes of run time. Job F is highlighted in this table since it is the job that was scheduled dynamically. The scheduling of Job F will be discussed with the graphs below.

Given, the output in Table 1, which shows how many compute cluster nodes are available at what day and time, the objective of the planner is to find the best possible gap for our batch job run to fit. Table 2 is a different representation of Table 1 and it shows the same example queue of running jobs as in Table 1 on the compute cluster at the given time.

Table 2. An Example Queue Status

Job	Nodes	RunTime (minutes)	WaitTime (minutes)	TotalTime (minutes)
A	400	120	0	**120**
B	50	90	0	**90**
C	20	60	0	**60**
D	29	30	0	**30**
E	29	10	30	**40**
F	30	2	40	**42**

The total number of nodes in this cluster is assumed to be 500. Job A is utilizing 400 nodes and will be running for the next two hours. Job B is using 50 nodes and will be running for 1.5 hours. Job C is using 20 nodes and will be running for 1 hour. Job D is using 29 nodes and will be running half an hour. Job E is waiting in the queue reserving 29 nodes and will run for 10 minutes.

As an example, assume the planner knows that our batch job will run for 1 hour on 1 node, and scales linearly with the addition of compute nodes. The planner algorithm examines the queue status as illustrated in Figure 6, and evaluates alternative schedule options. It determines that it can reserve 1 node that is available immediately on the compute cluster. However, a better alternative is to reserve 30 nodes after the completion of Job E, since our job will only take 2 minutes to run on 30 nodes. This will give a total runtime of 42 minutes (40 minutes wait time in the queue, 2 minutes runtime), hence the planner takes this action.

Even after the planner schedules a batch job on a highly utilized, dynamic computer cluster, the queue status can change based on several factors. For instance, a job can finish early or another job submitted after ours can be "skipped" by a very large job with high priority altering the node availability timeline. Since the framework keeps checking periodically for change in the compute cluster queue status, it can detect such changes, re-evaluate options and re-submit the job with different run-time parameters to achieve optimal time-to-solution in the updated queue if better options exist.

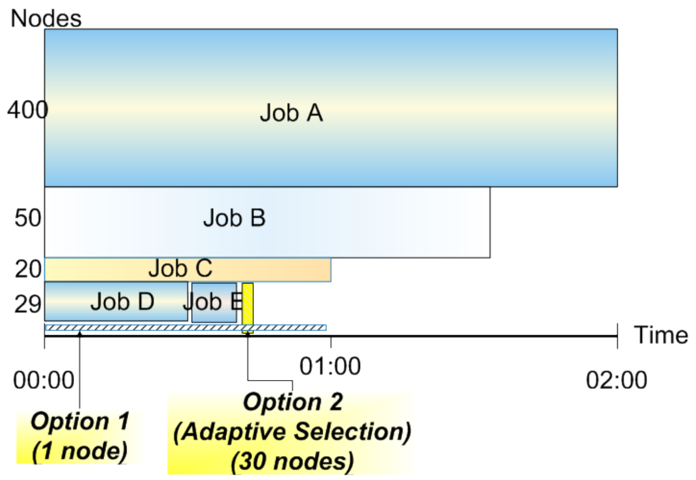


Figure 7. Adaptive Scheduling Given Example Queue Status in Table 2

Further illustrating the need for adaptivity, if Job B fails or finishes earlier than expected, see Figure 7 for option 1 where when there is no adaptivity, the pending jobs simply move ahead in the queue. This scenario gives our batch job a run time total of 12 minutes (10 minutes waiting in the queue, and 2 minutes run time). However, a better alternative is to adaptively adjust the runtime parameters so the batch job runs on 22 nodes which are immediately available, see option 2, therefore, reducing the total job time down to 2.72 minutes.

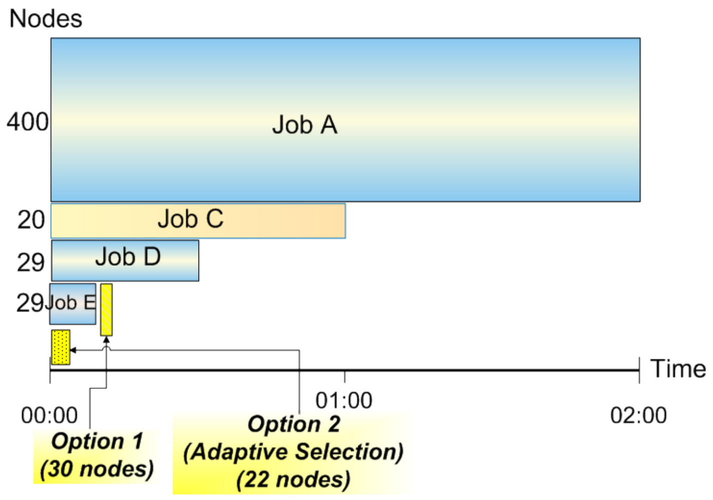


Figure 8. Adaptive Scheduling Given the Changed Queue Status

The summary of the planner process in our adaptive middleware is:

```
Get the currently selected goal from the Goal Manager
Loop through the queue status file reading each line
If (available nodes exist)
    Look ahead in the status file for how long nodes are available (gaps)
    If (# of available nodes * batchruntimepernode<= howlongnodesareavailable)
    AND
    If (the new configuration is within the user's goals that are set previously)
    AND
    If (batchruntimepernode / # of available nodes + waitTime < currentScheduledTotalTime)
        Propose the new queue-slot (gap) to the Goal Manager
        Send an act message to the control component
```

3.2.3. Goal Manager

The Goal Manager Layer is responsible for directing the overall actions of the planner, and responding to changes in the environment when communicated by the planner. Whittle, et al. describes scenarios wherein human input is required in the form of natural language (either speech or text), as a way to trigger adaptations that otherwise could not be achieved (28). In our example, Goal Manager decisions in MIF Adaptive Framework are based on an input file that specifies several options, see Figure 6 below. In the example, there are four types of different possible goals for the application that a user can specify:

1. NODECOUNT: NODECOUNT is used if the user wishes to specify only the number of cluster nodes to be used for processing the request. For example, there may be

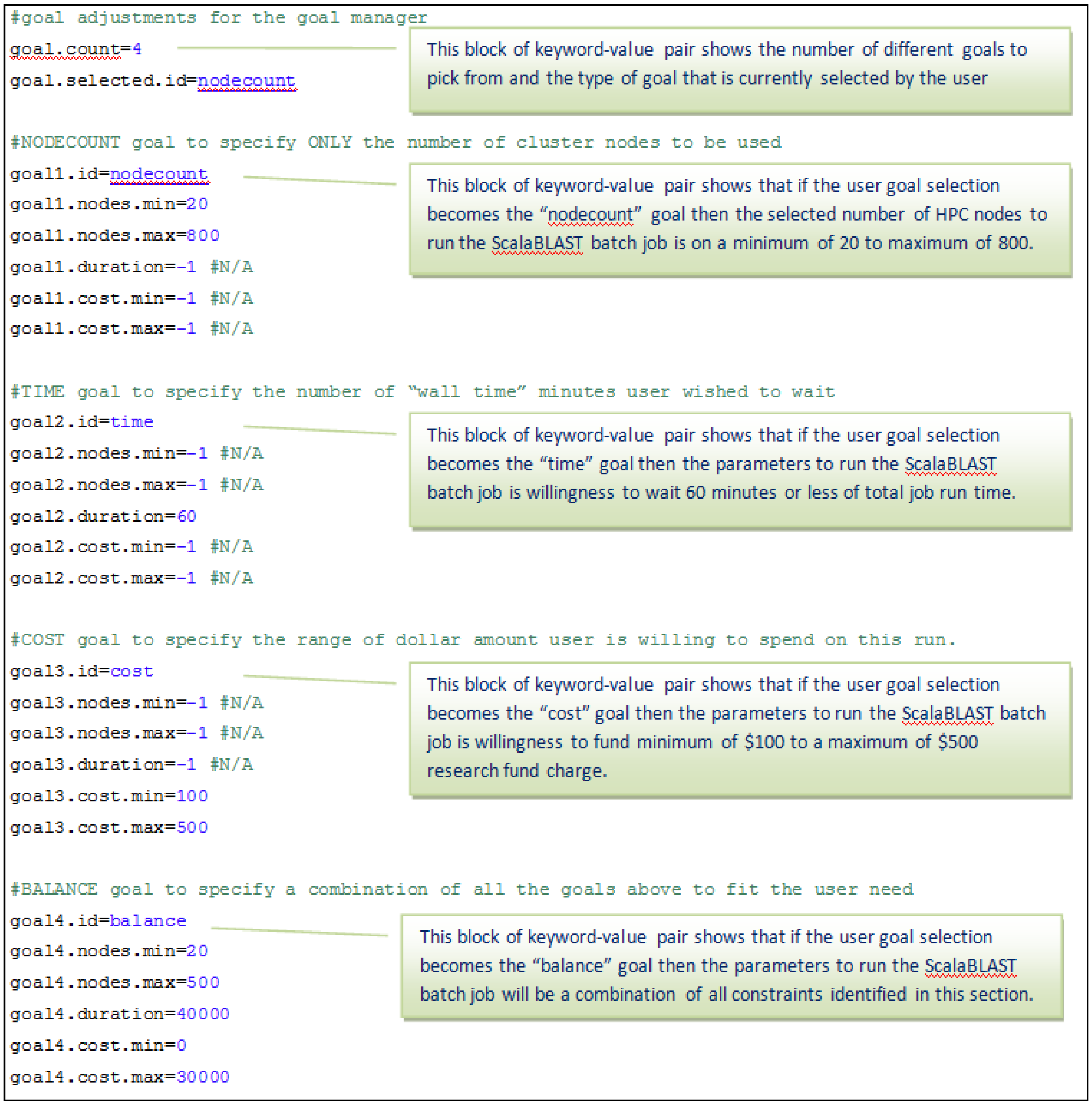
specific design patterns which may have been implemented in the application that benefit from specifying the number of nodes.

2. TIME: Specifying a goal of TIME indicates that the user wishes the job to be completed within a certain amount of “wall time”. The adaptive framework calculates the best plan to achieve the time goal.

3. COST: There is a cost to executing on HPC nodes. If a goal of COST is specified, the adaptive framework attempts to calculate a plan which will not exceed the cost specified. Although at first glance this option may be similar to the TIME goal, a different cost function may be necessary in some cases to validate the applicability of a separate COST goal. HPC environments are getting much more focused on power utilization. While goal of TIME has a simple relationship between wall clock time and number of cores, the power utilization may be much more complex. If the cost of using each node is A + Bt + Cct where A is a startup cost per node, B is the cost per unit time of running on a node, and C is the constant cost of running each core per unit time. In this case “t” is the time and “c” is the number of cores. This cost functionwould favor running longer runs using as many cores as possible on as few nodes as possible and therefore, will provide a different goal than the TIME goal for the scientist.

4. BALANCE: If the BALANCE goal is specified, the user allows the framework to calculate the best balance between NODECOUNT, TIME, and COST. Balancing TIME with COST goals would be ideal for such HPC environments that may constrain usage by complex memory resource “cost”. In such data centers, users use more

memory when they use more nodes and more cores. Goal BALANCE selection for the use in such systems would favor towards longer runs on smaller node sets.



```
#goal adjustments for the goal manager
goal.count=4
goal.selected.id=nodecount

#NODECOUNT goal to specify ONLY the number of cluster nodes to be used
goal1.id=nodecount
goal1.nodes.min=20
goal1.nodes.max=800
goal1.duration=-1 #N/A
goal1.cost.min=-1 #N/A
goal1.cost.max=-1 #N/A

#TIME goal to specify the number of "wall time" minutes user wished to wait
goal2.id=time
goal2.nodes.min=-1 #N/A
goal2.nodes.max=-1 #N/A
goal2.duration=60
goal2.cost.min=-1 #N/A
goal2.cost.max=-1 #N/A

#COST goal to specify the range of dollar amount user is willing to spend on this run.
goal3.id=cost
goal3.nodes.min=-1 #N/A
goal3.nodes.max=-1 #N/A
goal3.duration=-1 #N/A
goal3.cost.min=100
goal3.cost.max=500

#BALANCE goal to specify a combination of all the goals above to fit the user need
goal4.id=balance
goal4.nodes.min=20
goal4.nodes.max=500
goal4.duration=40000
goal4.cost.min=0
goal4.cost.max=30000
```

This block of keyword-value pair shows the number of different goals to pick from and the type of goal that is currently selected by the user

This block of keyword-value pair shows that if the user goal selection becomes the "nodecount" goal then the selected number of HPC nodes to run the ScalaBLAST batch job is on a minimum of 20 to maximum of 800.

This block of keyword-value pair shows that if the user goal selection becomes the "time" goal then the parameters to run the ScalaBLAST batch job is willingness to wait 60 minutes or less of total job run time.

This block of keyword-value pair shows that if the user goal selection becomes the "cost" goal then the parameters to run the ScalaBLAST batch job is willingness to fund minimum of $100 to a maximum of $500 research fund charge.

This block of keyword-value pair shows that if the user goal selection becomes the "balance" goal then the parameters to run the ScalaBLAST batch job will be a combination of all constraints identified in this section.

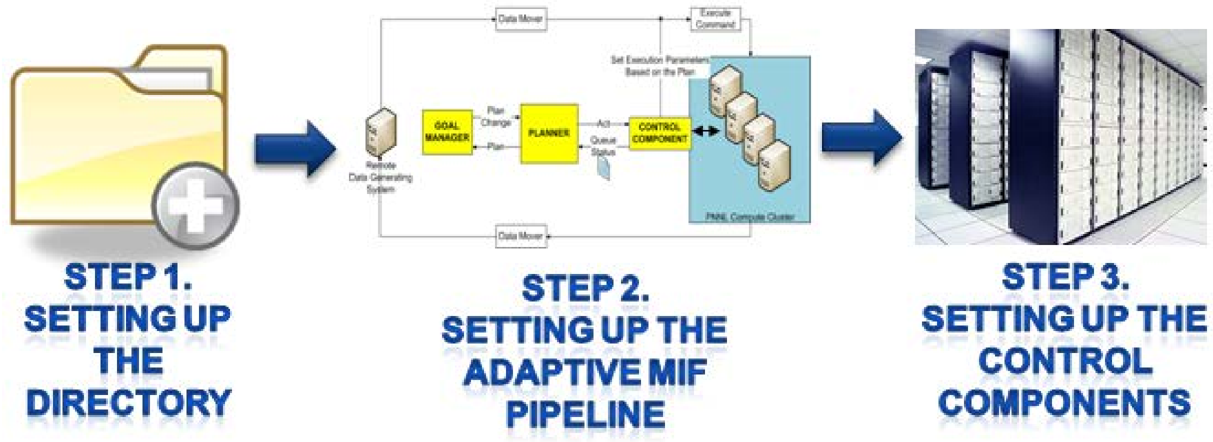
Figure 9. Goal Manager Adjustable Decisions

Once the plan is received from the user, the Planner first verifies that the plan can be achieved. The details of the Planner algorithm were discussed in the previous section.

CHAPTER FOUR

4. RUNNING THE ADAPTIVE MIF PIPELINE

Running an Adaptive MIF pipeline is a three step process shown in Figure 10. This three step process of running an adaptive MIF pipeline is generic and may be applicable to any scientific application. Furthermore, running each of the components of a MIF adaptive pipeline steps alone is possible. However, no adaptive scheduling will be performed if the components run individually and are not synchronized.



Data Mover
Execute Command
Set Execution Parameters Based on the Plan
GOAL MANAGER
Plan Change
Plan
PLANNER
Act
Queue Status
CONTROL COMPONENT
Remote Data Generating System
PNNL Compute Cluster
Data Mover
STEP 1. SETTING UP THE DIRECTORY
STEP 2. SETTING UP THE ADAPTIVE MIF PIPELINE
STEP 3. SETTING UP THE CONTROL COMPONENTS

Figure 10. The Three Step Adaptive MIF pipeline Setup

4.1. Setting up the Directory Structure

For the adaptive MIF pipeline to schedule any type of adaptive job on an HPC cluster the initial communication channels must be established and all of the components of the adaptive MIF architecture must be running. The communication between the Adaptive MIF planner and the control components is done through a shared file location accessible by the user, the MIF planner and the control component. A user account must be established and given access to the network file share. The various components in the adaptive framework communicate via files as follows:

- The ‘act’ folder structure is used to keep data files that are going to be used by the Control Component. This folder is also used to keep the log files for the decision process that adaptive MIF takes during an adaptively scheduled run.

- The “data” folder structure is used to transfer files needed to run the batch job. The transfer of the data files can be done separately ahead of time.

- The “process” folder structure is used to keep files related to runtime activities, such as, start time of adaptive MIF run, end time of adaptive MIF run, wait time of the currently scheduled adaptive job.

- The “sense” folder structure is used to keep the HPC queue status files gathered by the sense control component

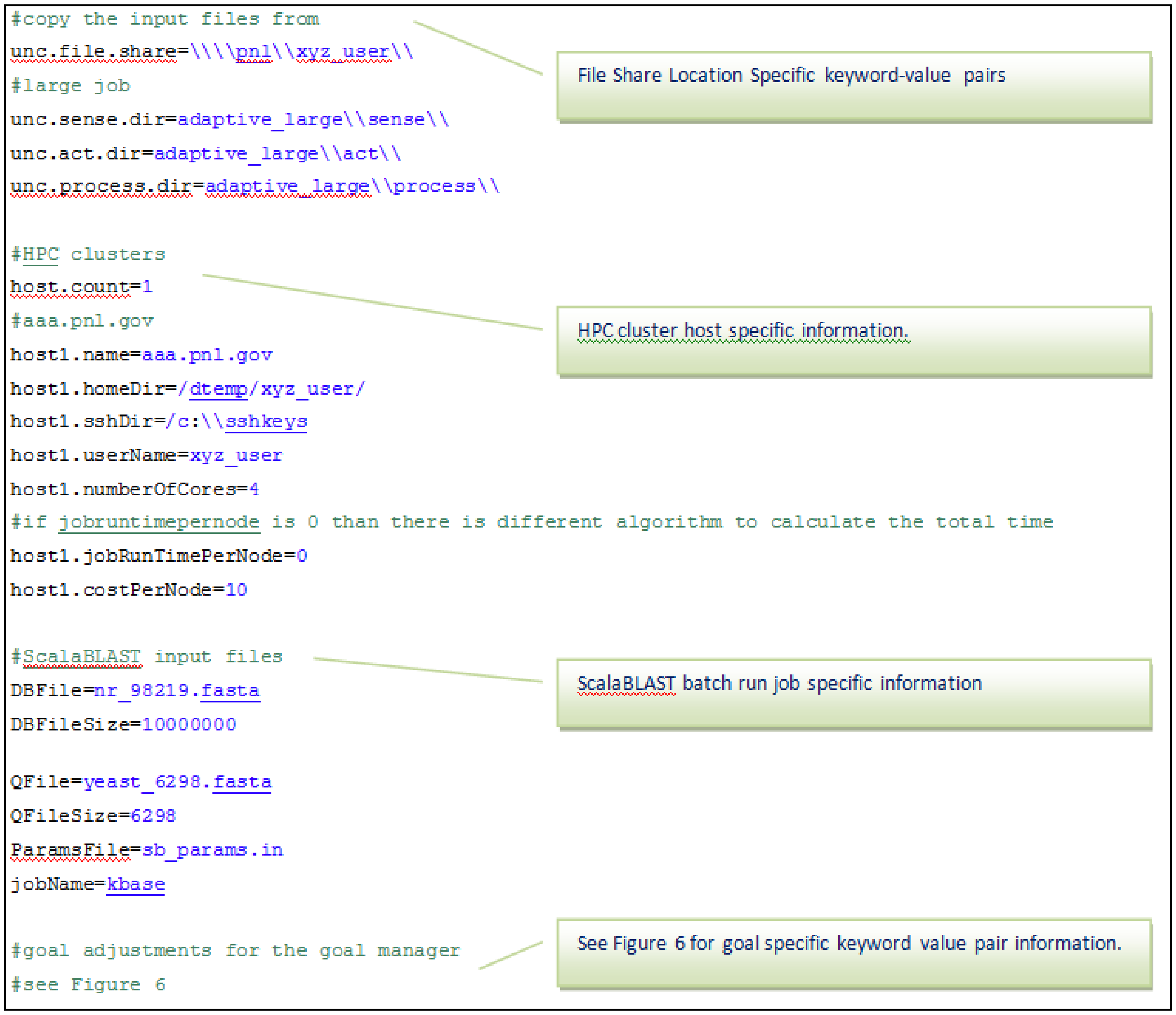
4.2. Setting up the Adaptive MeDICi Pipeline

Setting up the adaptive MeDICi pipeline requires input file setup as well as adding the adaptive module as a component of the MeDICi pipeline to move it from being a static pipeline to adaptivity aware one.

4.2.1. Setting up the input file

The adaptive MIF pipeline interacts with a configuration file that is named "adaptive.properties". This file must include

- MIF's Adaptive Planner Network file share - the file share used for this job
- HPC cluster host specific information, such as ssh directory, user name, cost per node, etc.
- The batch job specific information, in this example scalaBLAST specific information is provided, to calculate correct runtimes, and
- Goal information – Used by the adaptive framework to schedule adaptive jobs, if possible, according to the goals established by the user. A configuration of input file setup example, adaptive.properties file, is shown in Figure 11 in addition to Figure 9.



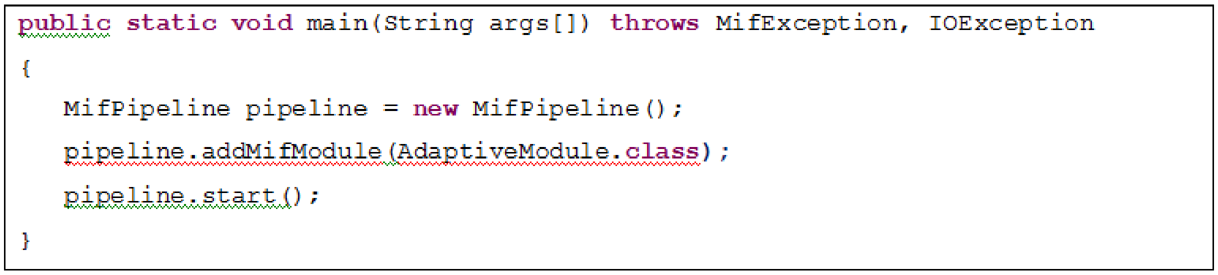
```
#copy the input files from
unc.file.share=\\\\pnl\\xyz_user\\
#large job
unc.sense.dir=adaptive_large\\sense\\
unc.act.dir=adaptive_large\\act\\
unc.process.dir=adaptive_large\\process\\

#HPC clusters
host.count=1
#aaa.pnl.gov
host1.name=aaa.pnl.gov
host1.homeDir=/dtemp/xyz_user/
host1.sshDir=/c:\\sshkeys
host1.userName=xyz_user
host1.numberOfCores=4
#if jobruntimepernode is 0 than there is different algorithm to calculate the total time
host1.jobRunTimePerNode=0
host1.costPerNode=10

#ScalaBLAST input files
DBFile=nr_98219.fasta
DBFileSize=10000000

QFile=yeast_6298.fasta
QFileSize=6298
ParamsFile=sb_params.in
jobName=kbase

#goal adjustments for the goal manager
#see Figure 6
```

File Share Location Specific keyword-value pairs
HPC cluster host specific information.
ScalaBLAST batch run job specific information
See Figure 6 for goal specific keyword value pair information.

Figure 11. Adaptive Framework Input File (adaptive.properties)

4.2.2. Programming the adaptive MeDICi pipeline

Augmenting MIF with adaptive capabilities requires adding the AdaptiveModule to the pipeline. MIF pipeline supports adding MIF Modules that are user defined and can run enhancements or application specific code within the pipeline. As shown in Figure 12, scientists can add the AdaptiveModule constructor for MIF to their scientific workflows by calling the addMifModule function.

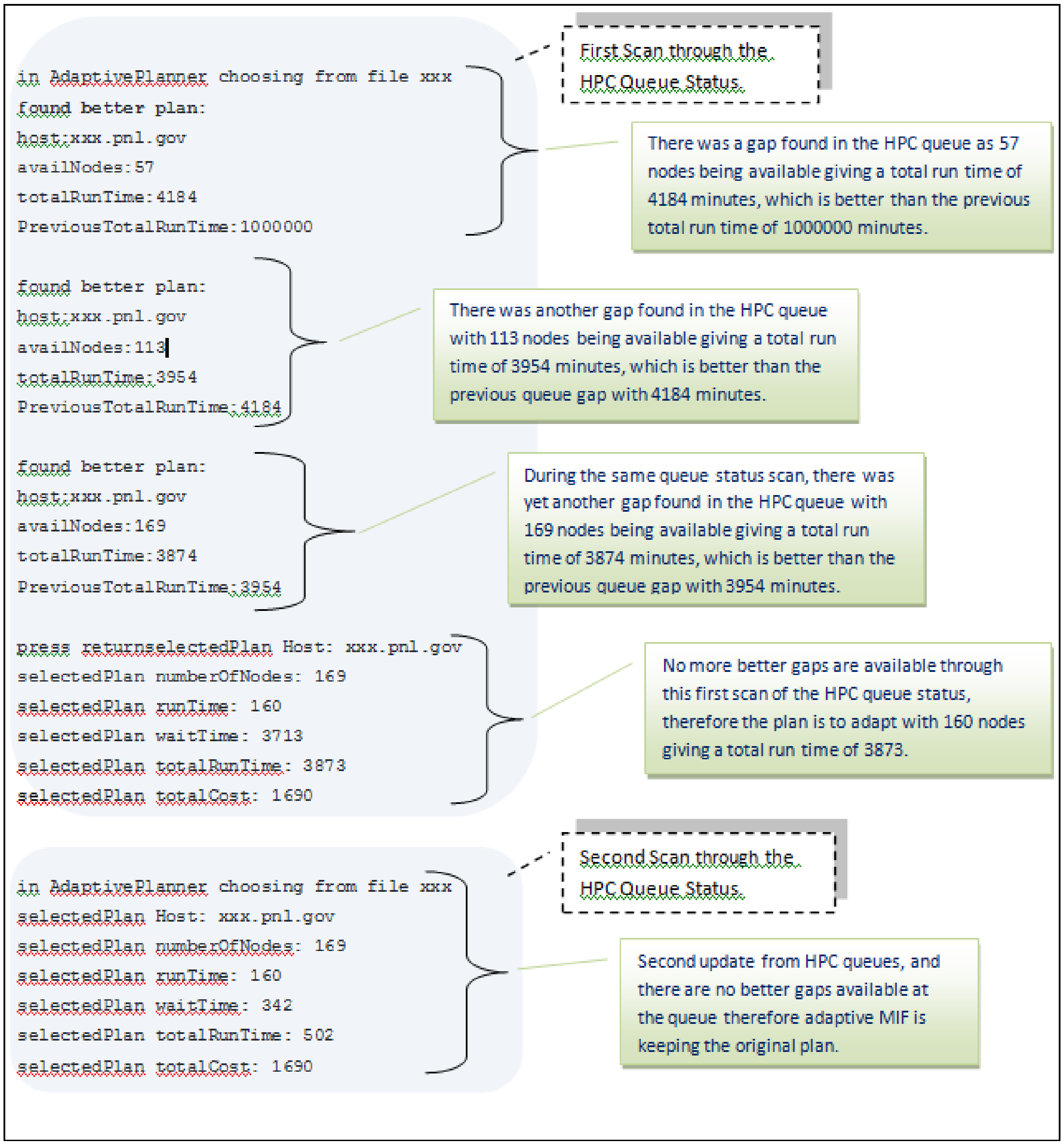


```
public static void main(String args[]) throws MifException, IOException
{
   MifPipeline pipeline = new MifPipeline();
   pipeline.addMifModule(AdaptiveModule.class);
   pipeline.start();
}
```

Figure 12. Adding Adaptive Capabilities to a MIF Pipeline

Once adaptiveModule is added to the MIF pipeline, it instantiates and initializes the Goal Manager, Planner, and Control Component. Then, it verifies that the job has not been started by checking the files that were setup on the communication channel during step 1. If the batch job is not running, it launches Planner to evaluate current environment, which will be constantly changing. Planner, as described above, gets the user's current goals from the Goal Manager, and it then gets the current queue stats from the Control Component. Once the planner has a picture of the current environment, it evaluates the best run-time parameters as described by the pseudo-code outlined in chapter 3.

Figure 13 shows an example output from a running adaptive MIF pipeline. As you can see in the output, a queue status file is being traversed for the next best gap that will outperform the current configuration by total run times. As you can see in the example output, the second scan through the queue status there were no better plans found, therefore no new plan was executed.



```
in AdaptivePlanner choosing from file xxx
found better plan:
host:xxx.pnl.gov
availNodes:57
totalRunTime:4184
PreviousTotalRunTime:1000000

found better plan:
host:xxx.pnl.gov
availNodes:113
totalRunTime:3954
PreviousTotalRunTime:4184

found better plan:
host:xxx.pnl.gov
availNodes:169
totalRunTime:3874
PreviousTotalRunTime:3954

press returnselectedPlan Host: xxx.pnl.gov
selectedPlan numberOfNodes: 169
selectedPlan runTime: 160
selectedPlan waitTime: 3713
selectedPlan totalRunTime: 3873
selectedPlan totalCost: 1690

in AdaptivePlanner choosing from file xxx
selectedPlan Host: xxx.pnl.gov
selectedPlan numberOfNodes: 169
selectedPlan runTime: 160
selectedPlan waitTime: 342
selectedPlan totalRunTime: 502
selectedPlan totalCost: 1690
```

First Scan through the HPC Queue Status.

There was a gap found in the HPC queue as 57 nodes being available giving a total run time of 4184 minutes, which is better than the previous total run time of 1000000 minutes.

There was another gap found in the HPC queue with 113 nodes being available giving a total run time of 3954 minutes, which is better than the previous queue gap with 4184 minutes.

During the same queue status scan, there was yet another gap found in the HPC queue with 169 nodes being available giving a total run time of 3874 minutes, which is better than the previous queue gap with 3954 minutes.

No more better gaps are available through this first scan of the HPC queue status, therefore the plan is to adapt with 160 nodes giving a total run time of 3873.

Second Scan through the HPC Queue Status.

Second update from HPC queues, and there are no better gaps available at the queue therefore adaptive MIF is keeping the original plan.

Figure 13. A Sample run-time output from Adaptive ScalaBLAST

4.3. Setting up the Control Components

The Control Components of adaptive MIF must be deployed on the HPC environment. The sensor control component script is called sense.csh and the actor control component script is called act.csh. The scripts are where the generic adaptive control component logic resides.

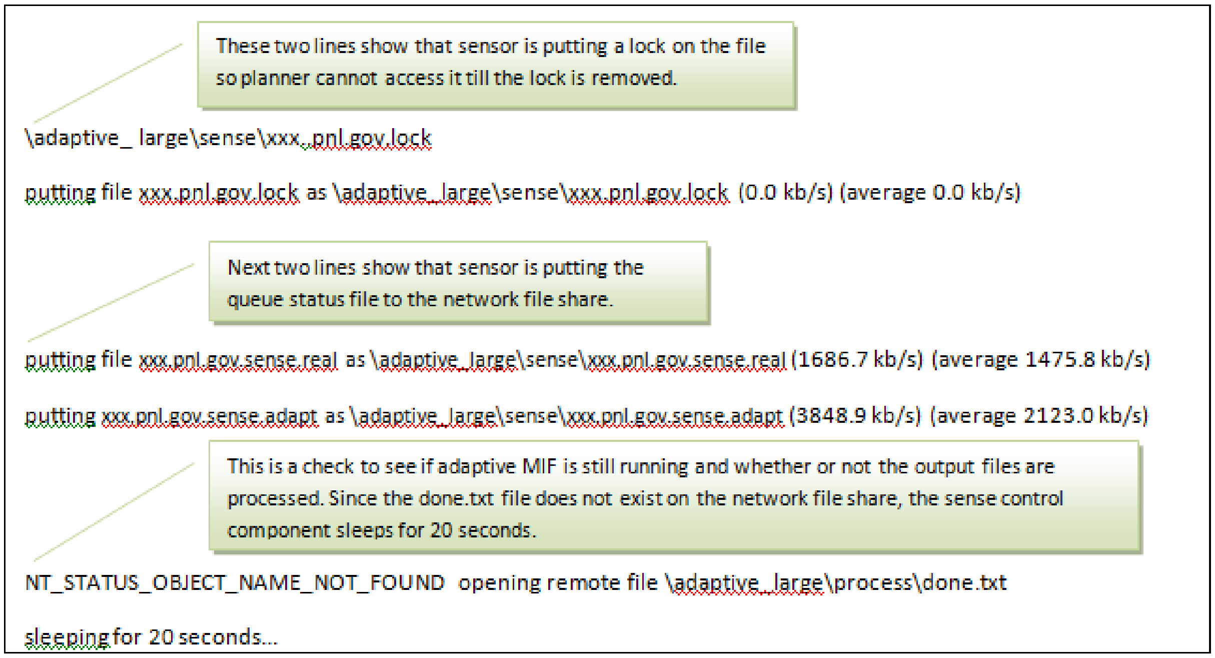
There are also two scripts deployed in addition to sense.csh and act.csh which parameterize the running of the control component scripts with system specific information.

4.3.1. Running the Sense Control Component

The parameterized run of the sense control component is called run_sense.csh. The run_sense.csh script launches the sense control component with the parameters as follows:

- File share server location - Used for communication with the adaptive MIF planner.
- The top level folder name - This top level folder contains the specific directory structure for adaptive MIF communications.
- Host Name - This parameter could be obtained from the operating system but due to differences in operating systems, currently the host name is parameterized.

Once started, the sense script polls the queue status every 20 seconds until the job starts to run. It gets the output of the current HPC queue and moves the output file to the shared location specified. If the file is locked, the Planner is evaluating the plan, and it should not be changed until the Planner has completed this process. The sense control component uses hostnames to identify which HPC cluster the information belongs to in case MIF adaptive framework is utilizing more than one HPC cluster. An example run of sense.csh is given below:



These two lines show that sensor is putting a lock on the file so planner cannot access it till the lock is removed.

```
\adaptive_ large\sense\xxx..pnl.gov.lock

putting file xxx.pnl.gov.lock as \adaptive_large\sense\xxx.pnl.gov.lock (0.0 kb/s) (average 0.0 kb/s)
```

Next two lines show that sensor is putting the queue status file to the network file share.

```
putting file xxx.pnl.gov.sense.real as \adaptive_large\sense\xxx.pnl.gov.sense.real (1686.7 kb/s) (average 1475.8 kb/s)

putting xxx.pnl.gov.sense.adapt as \adaptive_large\sense\xxx.pnl.gov.sense.adapt (3848.9 kb/s) (average 2123.0 kb/s)
```

This is a check to see if adaptive MIF is still running and whether or not the output files are processed. Since the done.txt file does not exist on the network file share, the sense control component sleeps for 20 seconds.

```
NT_STATUS_OBJECT_NAME_NOT_FOUND  opening remote file \adaptive_large\process\done.txt

sleeping for 20 seconds...
```

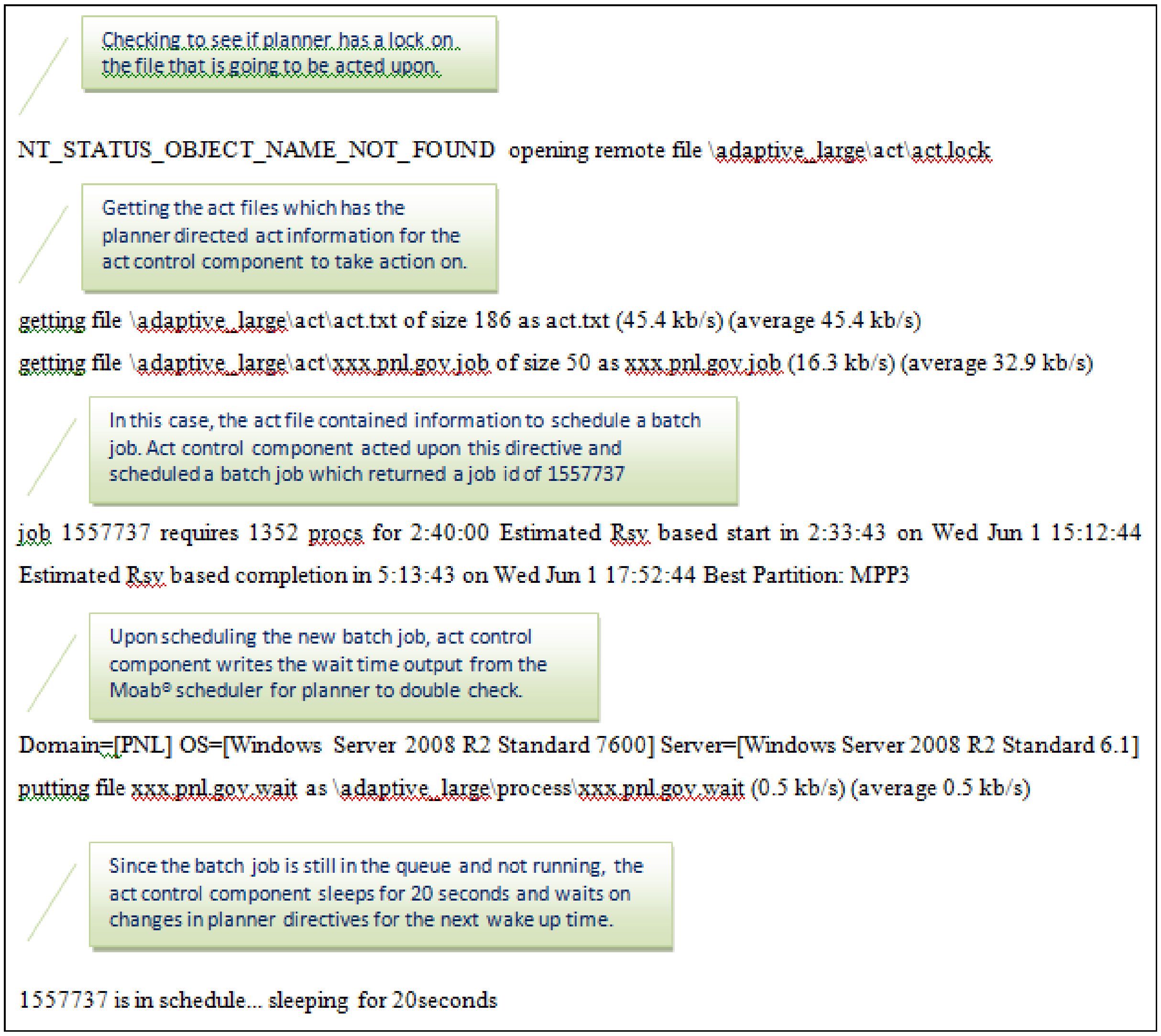
Figure 14. An Example Output of the Sense Control Component

4.3.2. Running the Act Control Component

The parameterized run of an act control component is called run_act.csh. The run_act.csh script launches the act control component with the parameters as follows:

- The server directory to be used for communication with the adaptive MIF planner.
- The directory path that contains the specific directory structure for adaptive MIF communications.
- Host Name. This parameter could be obtained from the operating system but due to differences in operating systems, currently the host name is a parameter.
- The script that will be used to reset the run-time HPC batch job parameters if/when the adaptive MIF changes the number of nodes to be used during an adaptive run.

Once run_act.csh runs, it starts to make decisions based on MIF adaptive planner directives. If a job must be canceled, it cancels that job. If a new job must be scheduled, it schedules that job. It also pushes queue wait time information related to the adaptively scheduled job to the file share for the MIF adaptive planner to access. The Act control component iterates every 20 seconds just like the sense control component. An example run of sense.csh is shown below:



Checking to see if planner has a lock on the file that is going to be acted upon.

```
NT_STATUS_OBJECT_NAME_NOT_FOUND  opening remote file \adaptive_large\act\act.lock
```

Getting the act files which has the planner directed act information for the act control component to take action on.

```
getting file \adaptive_large\act\act.txt of size 186 as act.txt (45.4 kb/s) (average 45.4 kb/s)
getting file \adaptive_large\act\xxx.pnl.gov.job of size 50 as xxx.pnl.gov.job (16.3 kb/s) (average 32.9 kb/s)
```

In this case, the act file contained information to schedule a batch job. Act control component acted upon this directive and scheduled a batch job which returned a job id of 1557737

```
job 1557737 requires 1352 procs for 2:40:00 Estimated Rsv based start in 2:33:43 on Wed Jun 1 15:12:44
Estimated Rsv based completion in 5:13:43 on Wed Jun 1 17:52:44 Best Partition: MPP3
```

Upon scheduling the new batch job, act control component writes the wait time output from the Moab® scheduler for planner to double check.

```
Domain=[PNL] OS=[Windows Server 2008 R2 Standard 7600] Server=[Windows Server 2008 R2 Standard 6.1]
putting file xxx.pnl.gov.wait as \adaptive_large\process\xxx.pnl.gov.wait (0.5 kb/s) (average 0.5 kb/s)
```

Since the batch job is still in the queue and not running, the act control component sleeps for 20 seconds and waits on changes in planner directives for the next wake up time.

```
1557737 is in schedule... sleeping for 20seconds
```

Figure 15. An Example Output of the Act Control Component

CHAPTER FIVE

5. ScalaBLAST: AN ADAPTIVE SCIENTIFIC WORKFLOW

ScalaBLAST was the test subject for the comparative results of adaptive MeDICi framework on HPC. ScalaBLAST is implemented using the MPI parallel programming libraries and the Global Array software for shared memory management (26). Amongst its other techniques, ScalaBLAST employs distributed memory management, latency hiding through pre-fetching database sequences, parallel I/O and multi-level parallelism to achieve higher performance and scalability (26). ScalaBLAST has been shown to scale linearly with respect to the number of processors. ScalaBLAST has high-performance data management capabilities. ScalaBLAST has been built and executed on many different platforms, and is an ideal application for handling large-scale bioinformatics calculations.

Ideally, ScalaBLAST would be run on smaller platforms when the compute tasks required by a given annotation task was small in size, and it would run on a larger platform when the compute task is larger in size. There are several reasons for carefully managing where ScalaBLAST is running. Larger systems tend to have longer queue wait times, so for smaller runs, it does not always pay off to submit them to larger batch systems. Likewise, sending a very large task to a small cluster could overwhelm that resource preventing other users from running on it. Additionally, different compute resources have their own user loads. Multiple users on a system can compete for limited resources (like global file system bandwidth or capacity, or interconnect bandwidth), so the optimal platform for a given sequence analysis task may vary in time as the loads of candidate systems fluctuate. The adaptive MIF workflow model is therefore ideal for developing a flexible ScalaBLAST workflow that dispatches compute tasks in response to new task requests from the adaptive MIF pipeline on the resource that is optimal for the task.

5.1. Results with Adaptive ScalaBLAST

The behavior of our adaptive MIF pipeline for ScalaBLAST jobs was observed over several runs over a two month period on a production HPC system at PNNL. These ScalaBLAST jobs were aligning a set of queries against sequences from databases of varying sizes, and the adaptive MIF pipeline was performing protein comparisons for ScalaBLAST using large dataset sizes. The ScalaBLAST job is considered large if the dataset query size was above 10,000,000 dataset queries, medium size if the database query size was above 1,000,000 but below 10,000,000 and small size ScalaBLAST jobs had over 100,000 but less than 1,000,000 dataset queries. For each ScalaBLAST job, all queries were compared to a public NIH database maintained by The National Center for Biotechnology Information (NCBI), (http://www.ncbi.nlm.nih.gov/) (29) containing a collection of all known sequenced proteins with redundant entries removed. This dataset is known as the nonredundant protein database, or ‘nr’ for short. The version of nr that we utilized in this study contained 10 million protein sequences.

5.2. Run-time Results with Large Datasets

In this section, one example of a large dataset (including over 10,000,000 dataset queries) based adaptive run will be examined. Summary statistics for multiple runs are presented in later sections. An individual analysis of all the adaptive runs is available for interested readers and the summary stats tables are included in the Appendix B-C.

The number of times different HPC nodes were selected over the course of this large dataset based adaptive run is a total of 6 (see Figure 16). Therefore, over the course of this adaptive run, the MIF adaptive planner made decisions based on the changes it sensed in HPC queues and made 6 different adaptive scheduling attempts, which resulted in dynamically selecting a large number of compute nodes that suddenly became available while the original submission was waiting to run.

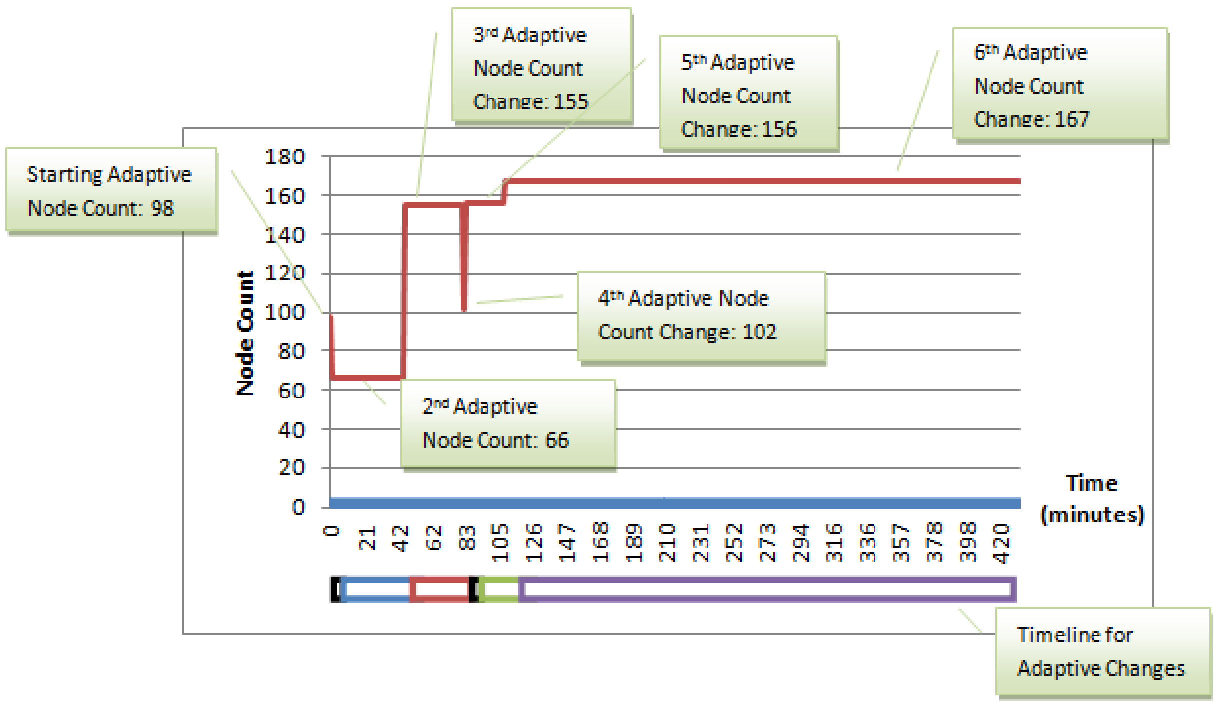


Figure 16. Adaptively Changing Node Count

In Figure 16, we inserted a new timeline constructed from colored rectangles that indicates the adaptive decision making baseline and carried this timeline to the figures related to this adaptive batch run in order to give the reader a better understanding of what is adaptively happening inside the run and how the rest of runtime parameters are being affected by adaptive changes.

Due to the highly dynamic nature of HPC queues, there is a risk that another job gets scheduled ahead of the adaptive run. Bumps in time-to-completion (Figure 17) come from new jobs coming in ahead of adaptive batch job and not because of adaptive decisions. Adaptive MIF planner decision changes always reduce the total time to solution results.

Since the runtimes of the batch job will be different based on the dynamic selection of the number of nodes, Figure 17 below has 6 different runtimes during the course of the adaptive scheduling. Comparing Figure 17 with the adaptive timeline, notice that total runtime changes are directly related to adaptive node count selections. When adaptive MIF reserves more HPC nodes, the runtime of our batch job becomes smaller since there are more processing power of parallel runs over the nodes and vice versa, when fewer nodes get selected to run a batch job, total runtime of the batch job takes longer.

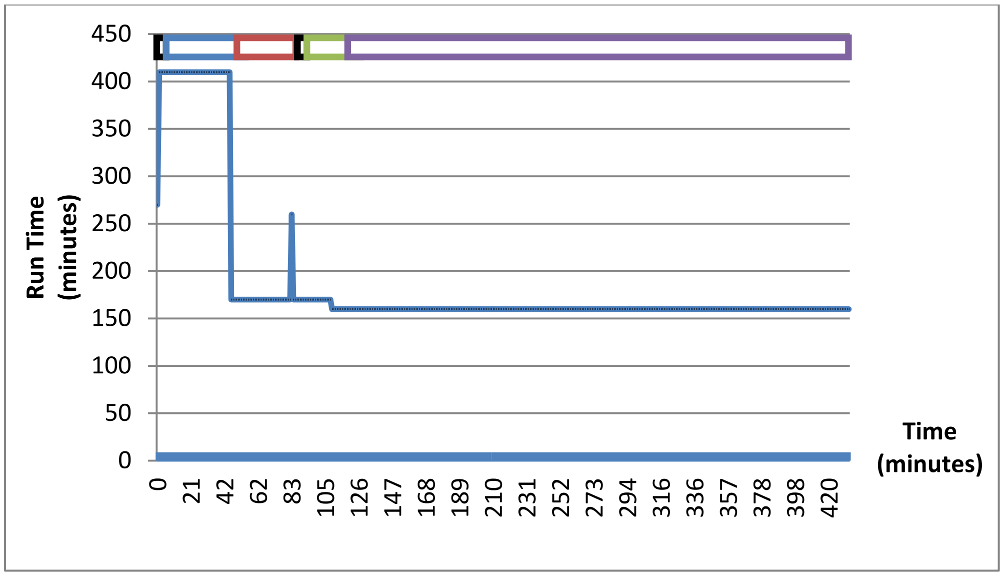


Figure 17. Adaptively Changing Node Count Affects on Run Times

Figure 18 below shows the wait times in the HPC queues for our ScalaBlast batch job. As can be seen from the graph below, even though the number of nodes reserved changes during the adaptive run, the wait times are reducing, which leads to reduced total time-to-completion results. Comparing Figure 18 to the adaptive time line, first adaptive decision leads to a big drop on the reamining wait time. Although there is no adaptive decision is being made during the second adaptive timeline, please note that wait times are changing dramatically, which is an indication of PNNL's highly dynamic HPC environment. The adaptive timeline adaptation only makes a new configuration change when this change results in a new more optional run-time as well as wait-time solution.

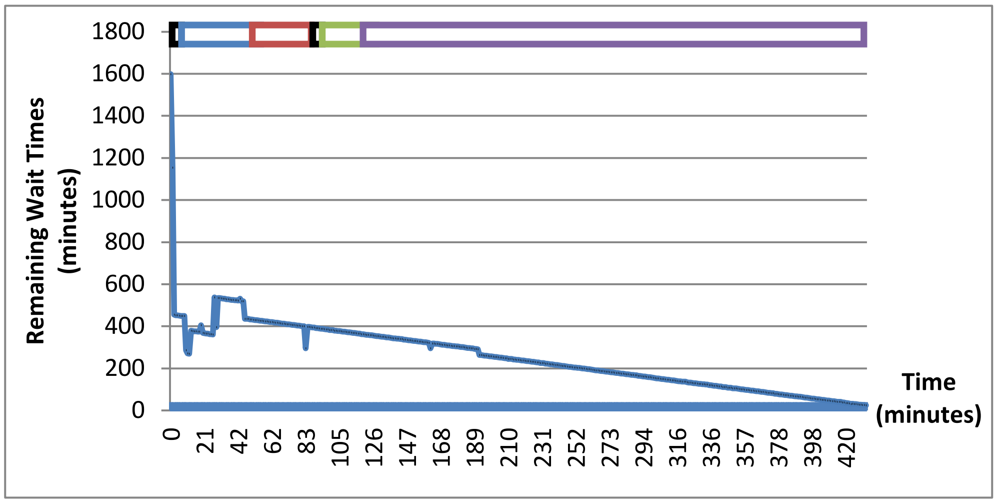


Figure 18. Adaptive Scheduling Impact on Remaining Wait Times

When examining total time-to-completion results (see Figure 19), note that the total time-to-completion results are reducing during adaptive job runs. Due to the highly dynamic nature of HPC queues, there is a risk that another job gets scheduled ahead of the adaptive run. Bumps in the wait times come from new jobs coming in ahead of adaptive batch job due to highly dynamic HPC queue and not because of adaptive decisions. Adaptive MIF planner decision changes always reduces the total time to solution results.

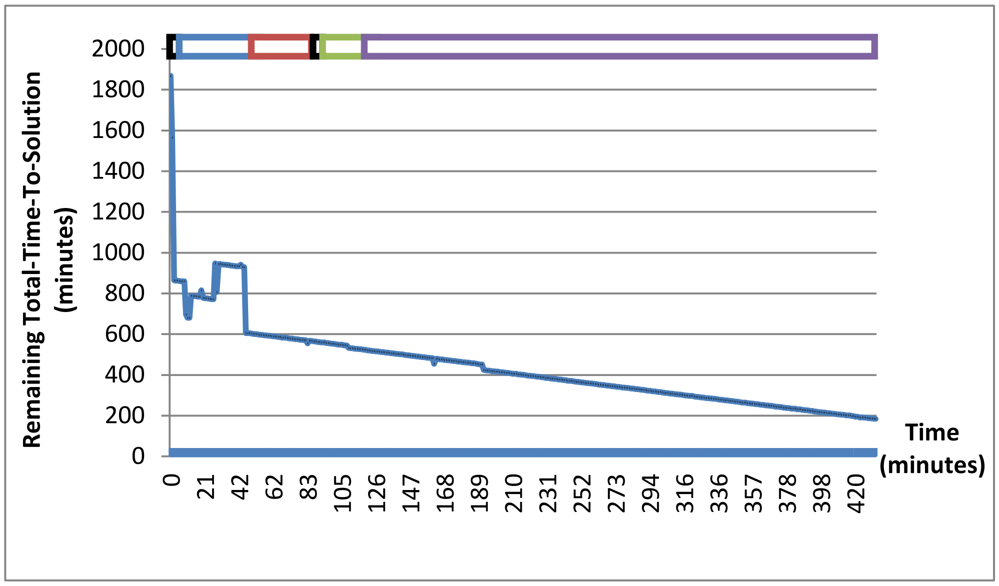


Figure 19. Adaptive Scheduling Impact on Remaining Total-Time-To-Solution

5.3. Results with Small to Medium Datasets Batch Runs

We ran several adaptive batch run jobs with using small to medium datasets. Since small to medium dataset batch run jobs do not require big number of HPC nodes, they get scheduled in the short time queue of HPC. Therefore, adaptive jobs run before they can be rescheduled again adaptively to prove the efficiency of adaptive MeDICi architecture. Nevertheless, adaptive MeDICi takes the guess work out of users' hands and schedules the small batch run jobs with the optimal parameters without having the users adjust the run-time parameters. That alone could save hours of scientist work saving the company soft cost dollars where the scientist can perform their science work and not worry about the implementation details of HPC batch runs.

5.4. Results Comparisons

In order to evaluate that our solution gives better throughput than staticly scheduled jobs in HPC queues, we created a race scenario of adaptive vs. static batch job. Starting the static batch job first with adaptively picked configuration gave our static job priority as well as the best spot in the queue. After scheduling our adaptive batch job, we compared the job submission times, start times, and completion times for both jobs. The comparison between the static and dynamic jobs are based on estimated times (run time, wait time, and total time) at the time of job submission versus the actual times (run time, wait time, and total times). Figure 20 shows one sample result comparing the statically scheduled ScalaBLAST job with the dynamic one, demonstrating the adaptive MIF solution reduces the total time to completion by 409 minutes, a %41 improvement.

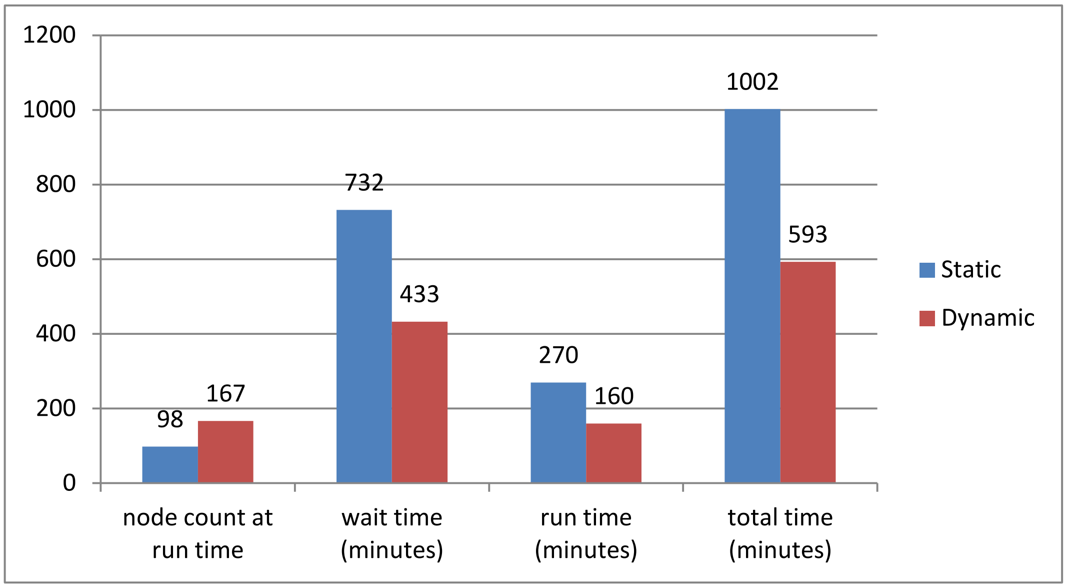


Figure 20. Static vs. Dynamic ScalaBLAST Batch Run Comparison

When MIF adaptive scheduling is started, job parameters (number of nodes and wait times) are optimally selected given the HPC queue status and user goals (see the job parameter selection algorithm in section 3.2.2.). There is no quarantee that there will be gaps and/or changes in the HPC queue status that will enable better total time-to-solution results. Obviously if the adaptive job does not reschedule itself, the static job that has given initial priority will run ahead of our dynamicly scheduled job. Table 3 shows that adaptation produces significant improvement in total time to completion in 5 out of 20 cases and marginal improvement in 4 additional jobs.

Table 3. Static vs. Dynamic ScalaBLAST Batch Run Comparison

Job Run	% improvement on total time with dynamic Scheduling	% improvement on total time with Static Scheduling	% Improvement Difference
1	68.25	46.36	21.90*
2	73.66	44.42	29.24*
3	86.55	88.83	-2.29#
4	79.90	80.04	-0.14@
5	84.89	85.03	-0.14@
6	87.98	82.71	5.27#
7	78.42	76.18	2.24#
8	96.69	85.72	10.97*
9	64.34	45.92	18.41*
10	75.59	75.63	-0.04@
11	92.67	88.36	4.31#
12	77.59	77.59	0.00@
13	61.67	61.74	-0.07@
14	83.74	84.53	-0.79@
15	90.89	90.93	-0.04@
16	86.03	86.03	0.00@
17	48.73	41.21	7.52#
18	25.28	25.73	-0.45@
19	90.38	90.78	-0.40@
20	60.62	-13.97	74.59*
*	Significant (>10%) – all improvements		
#	Moderate (1% to 10%) – all but one are improvements		
@	Insignificant (<1%)		

In Table 3, % improvement on total time to completion is the percentage of the difference between the actual total-time-to-completion and estimated total-time-to-completion at the time of first submission therefore includes the wait time as well as the run times. Significant improvement is defined as % improvement difference between a static and a dynamic batch job being above 10% and moderate improvement is defined as % improvement difference between static and dynamic job being between %1 to %10 and insignificant improvement is %improvement being below 1%. These results clearly show that adaptive scheduling enables a good chance for significantly improving the total job run times on HPC.

Over 20 runs, the jobs scheduled adaptively had 16% better time-to-completion that the equivalent statically scheduled jobs. As seen in Figure 21,there are a few results where the dynamic job resulted in a longer time to solution. In the instances that the dynamic job couldn’t perform better runtime results, the adaptive planner was not able to reschedule the batch job, meaning, there were no better spot in the queue and since the priority was given to the static job from the first time scheduling, the static batch job simply ran ahead of the dynamic job as the dynamic job did not get a chance to be rescheduled. Adaptive MIF job scheduling has several advantages even if the rescheduling doesn’t happen. First of all, users never need to figure out what the best parameters are to run their jobs given the current status of the queues. This step alone saves a scientist considerable preparation time. In addition adaptive scheduling gives a scientist a way to interact and change the limits of run time parameters for the batch job while the adaptive job is waiting in HPC queues. Furthermore, initially both the static and dynamic jobs pick the perfect parameters given the synapses of the queue, therefore, if the queue synapses do not change, dynamic job will not get a chance to get ahead of the static job but yet will still run with the best runtime parameters at a given time on the HPC queue.

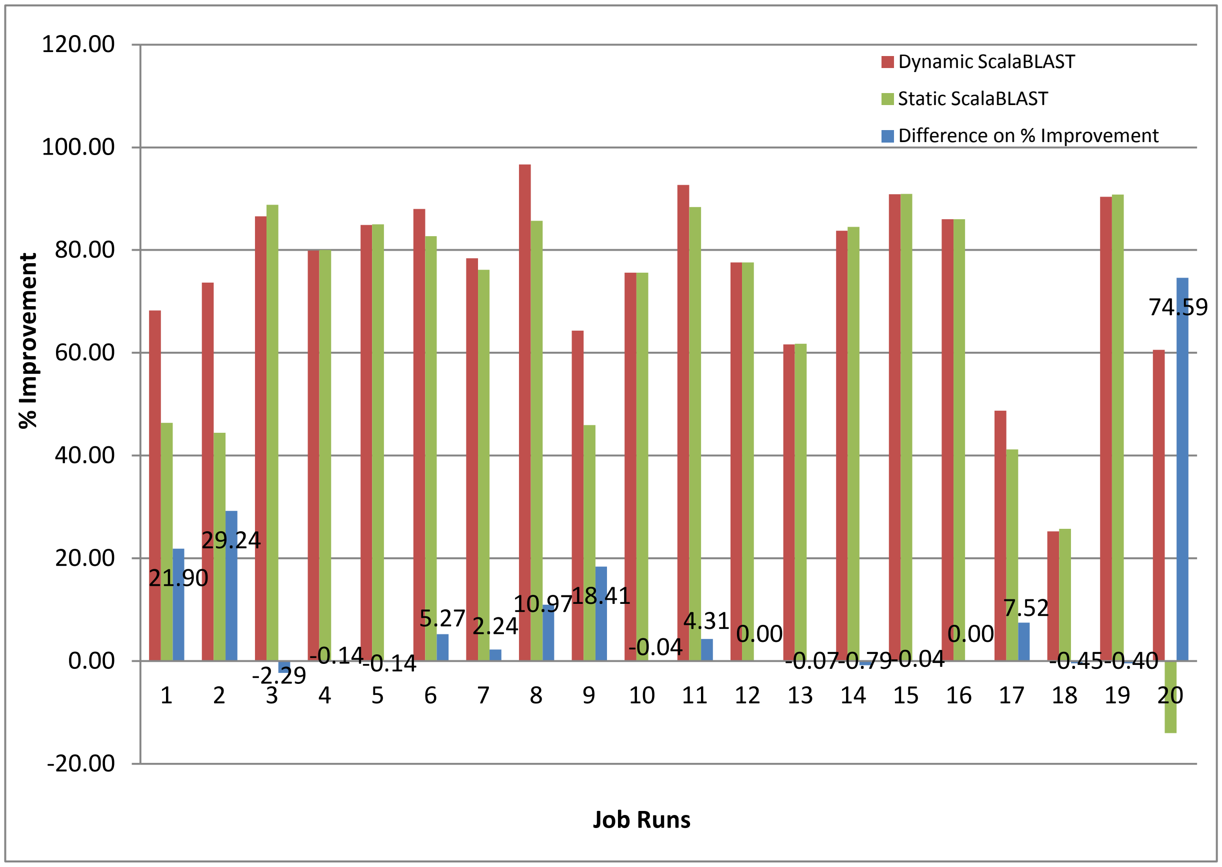


Figure 21. Dynamic vs. Static ScalaBLAST Batch Run Comparison Graph

Considering that the static job got the initial priority, or favoritism, of the HPC job scheduler, as a result of the historic runs shown in the results section, researchers can expect to see a 25% significant improvement on total time to completion of their jobs using the adaptive MIF framework running on an HPC system. We ran several adaptive batch run jobs with using small to medium datasets. Since small to medium datasets do not require a large number of HPC nodes, they were scheduled after a short wait in the queue. In fact, adaptive jobs were often run before the MIF planner had time to inspect the queue status. Nevertheless, the adaptive MIF technology alleviates the user from estimating job size to request when submitting the job. This simplifies their work load, enabling them to focus on their science and ignore the implementation details of HPC batch runs.

CHAPTER SIX

6. CONCLUSIONS and FUTURE WORK

6.1. Conclusions

In this dissertation, the software design and architecture of the MIF adaptive middleware framework for scientific workflows is presented. The MIF adaptive middleware framework was inspired by Kramer and Magee's proposed adaptive software architecture model. The MIF adaptive middleware framework was demonstrated to be successful in adapting the scheduling of large batch jobs on a highly dynamic HPC queue environment. Furthermore, the proposed software design and architecture was put to use and the benefits of using the adaptive middleware framework for scheduling jobs on a production HPC system was experimentally evaluated. Using this framework, often one can realize a substantial improvement in wall-clock time to solution on large-scale multi-user systems over static job scheduling because the adaptive scheduler aggressively takes advantage of volatility in the job queue.

When dealing with clusters, fully loaded machines are desirable because it leads to better and more efficient use of HPC resources. Adaptive MIF architecture maximizes cluster utilization and throughput by allowing dynamic jobs lower in the queue to run ahead of a job waiting at the top of the queue, as long as the job at the top is not delayed as a result. Revisiting adaptive decision process presented in Figure 8, a small modification to this figure showing the resource utilization is illustrated in Figure 22. The figure below shows the important concept of backfill windows of batch HPC environment with four running jobs and a reservation of a fifth dynamic job. The present time is represented by the leftmost end of the box with the future moving to the right. The light gray boxes represent currently idle nodes that are eligible for backfilling but cannot be backfilled by the HPC scheduling backfill algorithm alone. The dynamic scheduling utilizes the idle nodes in the backfill window that gives the best time to solution results. With only a single adaptive client, this can increase system utilization and improve throughput of large-scale calculations.

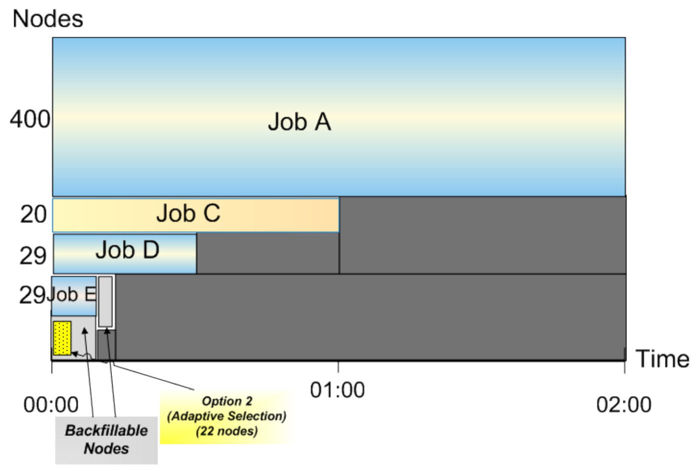


Figure 22. Adaptive Decision Process with Backfill Windows

In conjunction with MIF adaptive middleware framework, scientific workflows and the high-performance BLAST implementation, ScalaBLAST, this dissertation shows the software architecture to fill in the gap of achieving best run time solutions on a given HPC system eliminating the scientists having to learn the implementation details.

6.2. Future Work

Several areas of future work are being considered based on this research:

1. Further investigation of the proposed model to

    a. Study the behavior and performance when there is more than one adaptive job scheduled in a given HPC queue. If an HPC system starts promoting the adaptive scheduling to many users, adaptively scheduled jobs could start competing with each other. This scenario would be another research topic of an improvement on the current adaptive architecture.

    b. Study the performance when applied to other scientific workflow applications that are computationally intense, such as NWChem.

    c. Study other approaches for managing the communication with control components and MIF adaptive planner. Although the current communication method of file input/output and file locking mechanism works well to support different software language integrations for future work, using a queue or database to manage communication could provide a more robust and scalable solution.

2. Expansion of the proposed model to support the utilization of multiple HPC sites.

Only one target HPC system was used throughout this project. However our adaptive MIF architecture is generic and the components could be extended to adaptively schedule jobs on multiple HPC systems. By selecting amongst multiple potential execution sites, it may be possible to provide substantially better times-to-completion for jobs on HPC platforms.

3. Application of MIF adaptive framework (Kramer and Magee model) to other adaptivity problems where adaptive middleware is needed.

Different adaptation scenarios can be grouped into mapping and scheduling adaptations. In this dissertation, MIF adaptive framework was applied to a particular scheduling adaptation problem where increasing/decreasing the level of parallelism of a service was the main concern to achieve best total-time-to-solution results. The adaptation scenarios can be expanded to move services between different execution sites (different HPC queues), possibly located in different states or possibly changing the ScalaBLAST processing steps based on the changes in the queue, etc.

4. Applying different software architecture models to MIF adaptive framework based on applicability.

MIF adaptive framework model could be expanded to select different adaptive architecture (see section 4.2) models as needed based on changes in environment and/or the selection of a particular scientific workflow application that is being the subject of adaptivity.

5. Considerations on a graphical user interface.

Furthermore, writing a graphic user interface for the adaptive MeDICi framework should be a research topic. Interaction with the goal manager and users, setting up MeDICi parameters and showing the progress graphically may improve the user experience.

REFERENCES

1. *A roadmap towards sustainable self-aware service systems.* **Schahram Dustdar, Luciano Baresi, Giacomo Cabri, Cesare Pautasso, Franco Zambonelli.** New York, NY, USA : s.n., 2010. SEAMS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 10-19.

2. *Design patterns for developing dynamically adaptive systems.* **Andres J. Ramirez, Betty H.C. Cheng.** New York, NY, USA : s.n., 2010. SEAMS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 49-58.

3. *An Adaptive Middleware Framework for Optimal Scheduling on Large Scale Compute Clusters.* **Arzu Gosney, John H. Miller, Ian Gorton, Christopher Oehmen.** 2011. 2011 Eighth International Conference on Information Technology: New Generations. pp. 713-718.

4. *Components in the Pipeline.* **Ian Gorton, Adam Wynne, Yan Liu, Jian Yin.** 3, s.l. : IEEE Software, May/June 2011, Vol. 28.

5. *A Flexible, High Performance Service-Oriented Architecture for Detecting Cyber Attacks.* **Adam Wynne, Ian Gorton, Justin Almquist, Jack Chatterton, Dave Thurman.** Los Alamitos, California : IEEE Computer Society, 2008. Proceedings of the 41st Annual Hawaiian International Conference on Systems Sciences (HICSS 2008). p. 263.

6. *The MeDICi Integration Framework: A Platform for High Performance Data Streaming Applications".* **Ian Gorton, Adam Wynne, Justin Almquist, Jack Chatterton.** Vancouver, Canada : s.n., 2008. Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008). pp. 95-104.

7. *The Workflow Management Coalition.* [Online] http://www.wfmc.org.

8. *Evaluation of BPEL to Scientific Workflows.* **Asif Akram, David Meredith, Rob Allan.** 2006. Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid. pp. 269-274.

9. *Engineering Self-Adaptive Systems through Feedback Loops.* **Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Muller, Mauro Pezze, Mary Shaw.** 2009, Self-Adaptive Systems, LNCS 5525, pp. 48-70.

10. *Enabling adaptation of J2EE applications using components, web services and aspects.* **Liu, Yan.** s.l. : ACM Press, New York, NY, 2006. Proceedings of the 5th Workshop on Adaptive and Reflective Middleware (ARM '06) Melbourne, Australia. Vol. 190.

11. *Implementing Adaptive Performance Management in Server Applications.* **Yan Liu, Ian Gorton.** 2007. Proceedings of the 2007 International Workshop ion Software Engineering for Adaptive and Self-Managing Systems (SEAMS07) IEEE Computer Society Press. p. 12.

12. *Adaptive Workflow Processing and Execution in Pegasus.* **Kevin Lee, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro A.A. Fernandes, Gaurang Mehta.** 2009. Concurrency and Computation: Practice and Experience. Vol. 21, pp. 1965-1981.

13. *Self-Managed Systems: An Architectural Challenge.* **Jeff Kramer, Jeff Magee.** 2007. Future of Software Engineering, International Conference on Software Engineering. pp. 259–268.

14. *"Three-layer Architectures", Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.* **Erann Gat.** s.l. : MIT Press, 1998.

15. *An adaptive middleware framework for Scientific Computing at extreme scales.* **Arzu Gosney, Christopher Oehmen, Adam Wynne, Justin Almquist.** Las Vegas, NV : s.n., 2010. Information Reuse and Integration (IRI), 2010 IEEE International Conference on Information Reuse and Integration. pp. 232 - 238 .

16. *Flexible Scientific Workflow Modeling Using Frames, Templates, and Dynamic Embedding.* **Anne H. Ngu, Shawn Bowers, Nicholas Haasch, Timothy Mcphillips, Terence Critchlow.** 2008. Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008, Hong Kong, China. pp. 566-572.

17. *Kepler: an extensible system for design and execution of scientific workflows.* **Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, Steve Mock.** 2004. 16th International Conference on Scientific and Statistical Database Management. pp. 423-424.

18. Adaptive Computing. *Moab Adaptive HPC Suite.* [Online] http://www.adaptivecomputing.com/products/moab-adaptive-hpc-suite.php.

19. *Adaptive Computing.* [Online] http://www.adaptivecomputing.com.

20. *Review of Moab HPC Suite.* [Online] http://rnirmal.com/review-of-moab-hpc-suite.

21. Using Moab. [Online] https://computing.llnl.gov/tutorials/moab/.

22. *An efficient adaptive scheduling policy for high performance computing.* **Jemal H Abawajy.** 2009, Future Generation Computer Systems, Vol. 25, pp. 364-370.

23. *Adaptive data-aware utility-based scheduling in resource-constrained systems.* **David Vengerov, Lykomidis Mastroleon, Declan Murphy, Nick Bambos.** 9, 2010, Journal of Parallel and Distributed Computing, Vol. 70.

24. *Services + Components = Data Intensive Scientific Workflow Application with MeDICi.* **Ian Gorton, Jared Chase, Adam Wynne, Justin Almquist, Alan Chappell.** 2009, Lecture Notes in Computer Science, Vol. 5582, pp. 227-241.

25. *Model-Driven Application Development for the MeDICi Integration Framework: An Experience Report.* **Ian Gorton, Adam Wynne, Justin Almquist, Jack Chatterton.** s.l. : Conference on Component-Based Software Engineering, 2008.

26. *ScalaBLAST: A scalable implementation of BLAST for High Performance Data-Intensive Bioinformatics Analysis.* **Christopher Oehmen, Jarek Nieplocha.** 2006, IEEE Trans. Parallel. Dist. Sys., Vol. 17, pp. 740-749.

27. *Software Engineering for Self-Adaptive Systems: A Research Roadmap.* **Betty H.C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, Jeff Magee.** 2009, Software Engineering for Self-Adaptive Systems, pp. 1-26.

28. *On the role of the user in monitoring the environment in self-adaptive systems: a position paper.* **Jon Whittle, Will Simm, Maria-Angela Ferrario.** New York, NY, USA : s.n., 2010. SEAMS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 69-74.

29. *National Center for Biotechnology Information.* [Online] 8.18.2011. http://www.ncbi.nlm.nih.gov/.

APPENDIX

A. SAMPLE QUEUE OUTPUT FILE

The HPC user specific information, such as userid is non-disclosed.

```
num_jobs = 140
104 running jobs
32 pending jobs
4 deferred jobs
Year  Mo.  dt  Hr mn  Day  State   Job No   Userid  Sn   An  Sa      Aa
2011  Jun   8   9:52  Wed  NOW           0     all   8   23   >   8   23
2011  Jun   8   9:52  Wed  START   1560892     ###   0   -8   >   8   15
2011  Jun   8   9:52  Wed  START   1560893     ###   0   -8   >   8    7
2011  Jun   8  10:21  Wed  *FINI   1560943     ###   0    1   >   8    8
2011  Jun   8  10:21  Wed  START   1560894     ###   0   -8   >   8    0
2011  Jun   8  11:26  Wed  *FINI   1560877     ###   0   32   >   8   32
2011  Jun   8  11:26  Wed  *FINI   1560756     ###   0    6   >   8   38
2011  Jun   8  11:26  Wed  START   1560896     ###   0  -16   >   8   22
2011  Jun   8  11:26  Wed  START   1560897     ###   0   -8   >   8   14
2011  Jun   8  11:26  Wed  START   1560900     ###   0   -8   >   8    6
2011  Jun   8  11:38  Wed  *FINI   1560813     ###   0   24   >   8   30
2011  Jun   8  11:38  Wed  START   1560898     ###   0  -16   >   8   14
2011  Jun   8  11:38  Wed  START   1560899     ###   0   -8   >   8    6
2011  Jun   8  12:13  Wed  *FINI   1560882     ###   0  191   >   8  197
2011  Jun   8  12:13  Wed  START   1560902     ###   0   -8   >   8  189
2011  Jun   8  12:13  Wed  START   1560905     ###   0  -16   >   8  173
2011  Jun   8  12:13  Wed  START   1560923     ###   0  -32   >   8  141
2011  Jun   8  12:13  Wed  START   1560925     ###   0   -8   >   8  133
2011  Jun   8  12:13  Wed  START   1560924     ###   0   -8   >   8  125
2011  Jun   8  12:13  Wed  START   1560929     ###   0   -8   >   8  117
2011  Jun   8  12:13  Wed  START   1560928     ###   0   -8   >   8  109
2011  Jun   8  12:13  Wed  START   1560927     ###   0   -8   >   8  101
2011  Jun   8  12:13  Wed  START   1560926     ###   0   -8   >   8   93
2011  Jun   8  12:13  Wed  START   1560930     ###   0   -8   >   8   85
2011  Jun   8  12:13  Wed  START   1560944     ###   0   -8   >   8   77
2011  Jun   8  12:13  Wed  START   1560895     ###   0  -50   >   8   27
2011  Jun   8  12:40  Wed  *FINI   1560828     ###   0   16   >   8   43
2011  Jun   8  12:40  Wed  START   1560885     ###   0  -32   >   8   11
2011  Jun   8  12:50  Wed  *FINI   1560297     ###   0    8   >   8   19
2011  Jun   8  12:50  Wed  START   1560904     ###   0  -16   >   8    3
2011  Jun   8  13:16  Wed  *FINI   1560875     ###   0   64   >   8   67
2011  Jun   8  13:16  Wed  START   1560888     ###   0  -32   >   8   35
```

2011	Jun	8	13:17	Wed	*FINI	1560757	###	0	6	>	8	41
2011	Jun	8	13:18	Wed	*FINI	1560456	###	0	8	>	8	49
2011	Jun	8	13:20	Wed	*FINI	1560457	###	0	8	>	8	57
2011	Jun	8	14:14	Wed	FINIS	1560944	###	0	8	>	8	65
2011	Jun	8	14:48	Wed	*FINI	1560174	###	0	16	>	8	81
2011	Jun	8	15:32	Wed	*FINI	1560871	###	0	16	>	8	97
2011	Jun	8	15:42	Wed	FINIS	1560894	###	0	8	>	8	105
2011	Jun	8	15:45	Wed	*FINI	1560870	###	0	16	>	8	121
2011	Jun	8	16:06	Wed	*FINI	1560791	###	0	6	>	8	127
2011	Jun	8	16:15	Wed	*FINI	1560383	###	0	16	>	8	143
2011	Jun	8	16:15	Wed	*FINI	1560384	###	0	16	>	8	159
2011	Jun	8	16:15	Wed	*FINI	1560385	###	0	16	>	8	175
2011	Jun	8	17:27	Wed	FINIS	1560900	###	0	8	>	8	183
2011	Jun	8	17:39	Wed	FINIS	1560899	###	0	8	>	8	191
2011	Jun	8	17:53	Wed	FINIS	1560892	###	0	8	>	8	199
2011	Jun	8	17:53	Wed	FINIS	1560893	###	0	8	>	8	207
2011	Jun	8	18:00	Wed	SHRTP	0	released	-8	8	>	0	215
2011	Jun	8	18:14	Wed	FINIS	1560902	###	0	8	>	0	223
2011	Jun	8	18:19	Wed	*FINI	1560794	###	0	6	>	0	229
2011	Jun	8	18:34	Wed	*FINI	1560845	###	0	5	>	0	234
2011	Jun	8	18:34	Wed	*FINI	1560843	###	0	5	>	0	239
2011	Jun	8	18:34	Wed	*FINI	1560844	###	0	5	>	0	244
2011	Jun	8	18:34	Wed	*FINI	1560846	###	0	5	>	0	249
2011	Jun	8	18:34	Wed	*FINI	1560849	###	0	5	>	0	254
2011	Jun	8	18:34	Wed	*FINI	1560848	###	0	5	>	0	259
2011	Jun	8	18:34	Wed	*FINI	1560847	###	0	5	>	0	264
2011	Jun	8	18:46	Wed	*FINI	1560850	###	0	5	>	0	269
2011	Jun	8	18:47	Wed	*FINI	1560528	###	0	4	>	0	273
2011	Jun	8	18:55	Wed	*FINI	1559996	###	0	8	>	0	281
2011	Jun	8	19:03	Wed	*FINI	1560880	###	0	49	>	0	330
2011	Jun	8	19:10	Wed	*FINI	1560851	###	0	5	>	0	335
2011	Jun	8	19:10	Wed	*FINI	1560852	###	0	5	>	0	340
2011	Jun	8	20:14	Wed	FINIS	1560923	###	0	32	>	0	372
2011	Jun	8	20:33	Wed	*FINI	1560771	###	0	40	>	0	412
2011	Jun	8	20:38	Wed	*FINI	1559999	###	0	8	>	0	420
2011	Jun	8	20:43	Wed	*FINI	1560000	###	0	8	>	0	428
2011	Jun	8	20:53	Wed	*FINI	1560278	###	0	16	>	0	444
2011	Jun	8	20:53	Wed	START	1560931	###	0	-8	>	0	436
2011	Jun	8	20:54	Wed	*FINI	1560805	###	0	8	>	0	444
2011	Jun	8	21:20	Wed	*FINI	1560002	###	0	8	>	0	452
2011	Jun	8	21:28	Wed	*FINI	1560430	###	0	16	>	0	468
2011	Jun	8	21:46	Wed	*FINI	1560337	###	0	2	>	0	470
2011	Jun	8	21:46	Wed	*FINI	1560335	###	0	2	>	0	472

2011	Jun	8	21:54	Wed	*FINI	1560279	###	0	16	>	0	488
2011	Jun	8	21:54	Wed	START	1560932	###	0	-8	>	0	480
2011	Jun	8	22:26	Wed	*FINI	1560789	###	0	32	>	0	512
2011	Jun	8	22:42	Wed	*FINI	1560687	###	0	50	>	0	562
2011	Jun	8	22:42	Wed	*FINI	1560003	###	0	8	>	0	570
2011	Jun	8	22:50	Wed	*FINI	1560004	###	0	8	>	0	578
2011	Jun	8	23:05	Wed	*FINI	1560005	###	0	8	>	0	586
2011	Jun	8	23:23	Wed	*FINI	1559988	###	0	255	>	0	841
2011	Jun	8	23:46	Wed	*FINI	1560006	###	0	8	>	0	849
2011	Jun	9	0:06	Thu	*FINI	1560007	###	0	8	>	0	857
2011	Jun	9	0:14	Thu	*FINI	1560008	###	0	8	>	0	865
2011	Jun	9	0:20	Thu	*FINI	1560009	###	0	8	>	0	873
2011	Jun	9	0:37	Thu	*FINI	1560800	###	0	8	>	0	881
2011	Jun	9	0:41	Thu	*FINI	1560567	###	0	16	>	0	897
2011	Jun	9	0:41	Thu	*FINI	1560604	###	0	16	>	0	913
2011	Jun	9	0:41	Thu	*FINI	1560010	###	0	8	>	0	921
2011	Jun	9	0:42	Thu	*FINI	1560615	###	0	16	>	0	937
2011	Jun	9	1:04	Thu	*FINI	1560861	###	0	8	>	0	945
2011	Jun	9	1:39	Thu	FINIS	1560898	###	0	16	>	0	961
2011	Jun	9	2:33	Thu	*FINI	1560011	###	0	8	>	0	969
2011	Jun	9	4:51	Thu	FINIS	1560904	###	0	16	>	0	985
2011	Jun	9	5:43	Thu	*FINI	1560170	###	0	32	>	0	1017
2011	Jun	9	5:54	Thu	*FINI	1560386	###	0	16	>	0	1033
2011	Jun	9	6:12	Thu	*FINI	1560350	###	0	2	>	0	1035
2011	Jun	9	6:12	Thu	*FINI	1560352	###	0	2	>	0	1037
2011	Jun	9	6:13	Thu	*FINI	1560353	###	0	2	>	0	1039
2011	Jun	9	6:14	Thu	*FINI	1560255	###	0	32	>	0	1071
2011	Jun	9	6:14	Thu	*FINI	1560256	###	0	32	>	0	1103
2011	Jun	9	7:00	Thu	SHRTP	0	activate	8	-8	>	8	1095
2011	Jun	9	7:05	Thu	*FINI	1560012	###	0	8	>	8	1103
2011	Jun	9	7:05	Thu	*FINI	1560013	###	0	8	>	8	1111
2011	Jun	9	7:05	Thu	*FINI	1560014	###	0	8	>	8	1119
2011	Jun	9	7:05	Thu	*FINI	1560015	###	0	8	>	8	1127
2011	Jun	9	7:05	Thu	*FINI	1560016	###	0	8	>	8	1135
2011	Jun	9	7:14	Thu	*FINI	1560865	###	0	16	>	8	1151
2011	Jun	9	7:19	Thu	*FINI	1560866	###	0	16	>	8	1167
2011	Jun	9	7:38	Thu	*FINI	1560867	###	0	16	>	8	1183
2011	Jun	9	7:47	Thu	*FINI	1560868	###	0	16	>	8	1199
2011	Jun	9	11:14	Thu	FINIS	1560905	###	0	16	>	8	1215
2011	Jun	9	11:46	Thu	*FINI	1560462	###	0	128	>	8	1343
2011	Jun	9	12:14	Thu	FINIS	1560925	###	0	8	>	8	1351
2011	Jun	9	12:14	Thu	FINIS	1560924	###	0	8	>	8	1359
2011	Jun	9	12:14	Thu	FINIS	1560929	###	0	8	>	8	1367

2011	Jun	9	12:14	Thu	FINIS	1560928	###	0	8	>	8	1375
2011	Jun	9	12:14	Thu	FINIS	1560927	###	0	8	>	8	1383
2011	Jun	9	12:14	Thu	FINIS	1560926	###	0	8	>	8	1391
2011	Jun	9	12:14	Thu	FINIS	1560930	###	0	8	>	8	1399
2011	Jun	9	12:14	Thu	FINIS	1560895	###	0	50	>	8	1449
2011	Jun	9	12:14	Thu	START	1560937	###	0	-8	>	8	1441
2011	Jun	9	12:14	Thu	START	1560935	###	0	-8	>	8	1433
2011	Jun	9	12:14	Thu	START	1560938	###	0	-8	>	8	1425
2011	Jun	9	12:14	Thu	START	1560936	###	0	-8	>	8	1417
2011	Jun	9	12:14	Thu	START	1560939	###	0	-8	>	8	1409
2011	Jun	9	12:14	Thu	START	1560940	###	0	-8	>	8	1401
2011	Jun	9	12:14	Thu	START	1560942	###	0	-8	>	8	1393
2011	Jun	9	13:11	Thu	FINIS	1560885	###	0	32	>	8	1425
2011	Jun	9	14:18	Thu	*FINI	1560316	###	0	16	>	8	1441
2011	Jun	9	15:22	Thu	*FINI	1560573	###	0	8	>	8	1449
2011	Jun	9	15:55	Thu	*FINI	1560638	###	0	4	>	8	1453
2011	Jun	9	15:55	Thu	*FINI	1560628	###	0	4	>	8	1457
2011	Jun	9	18:00	Thu	SHRTP	0	released	-8	8	>	0	1465
2011	Jun	9	18:15	Thu	FINIS	1560942	###	0	8	>	0	1473
2011	Jun	9	18:20	Thu	*FINI	1560314	###	0	32	>	0	1505
2011	Jun	9	18:20	Thu	*FINI	1560317	###	0	16	>	0	1521
2011	Jun	9	18:24	Thu	*FINI	1560569	###	0	255	>	0	1776
2011	Jun	9	18:24	Thu	*FINI	1560643	###	0	4	>	0	1780
2011	Jun	9	18:49	Thu	*FINI	1560605	###	0	64	>	0	1844
2011	Jun	9	18:50	Thu	*FINI	1560631	###	0	64	>	0	1908
2011	Jun	9	18:50	Thu	*FINI	1560632	###	0	8	>	0	1916
2011	Jun	9	18:50	Thu	*FINI	1560655	###	0	32	>	0	1948
2011	Jun	9	20:54	Thu	FINIS	1560931	###	0	8	>	0	1956
2011	Jun	9	21:24	Thu	*FINI	1560578	###	0	32	>	0	1988
2011	Jun	9	21:24	Thu	*FINI	1560606	###	0	32	>	0	2020
2011	Jun	9	21:54	Thu	*FINI	1560785	###	0	32	>	0	2052
2011	Jun	9	21:55	Thu	FINIS	1560932	###	0	8	>	0	2060
2011	Jun	9	22:49	Thu	*FINI	1560564	###	0	27	>	0	2087
2011	Jun	9	23:42	Thu	*FINI	1560797	###	0	4	>	0	2091
2011	Jun	10	2:31	Fri	*FINI	1560811	###	0	8	>	0	2099
2011	Jun	10	4:27	Fri	*FINI	1560821	###	0	32	>	0	2131
2011	Jun	10	5:12	Fri	*FINI	1560839	###	0	8	>	0	2139
2011	Jun	10	7:00	Fri	SHRTP	0	activate	8	-8	>	8	2131
2011	Jun	10	7:10	Fri	*FINI	1560862	###	0	32	>	8	2163
2011	Jun	10	7:10	Fri	*FINI	1560863	###	0	6	>	8	2169
2011	Jun	10	8:13	Fri	*FINI	1559977	###	0	2	>	8	2171
2011	Jun	10	11:27	Fri	FINIS	1560896	###	0	16	>	8	2187
2011	Jun	10	11:27	Fri	FINIS	1560897	###	0	8	>	8	2195

2011	Jun	10	12:15	Fri	FINIS	1560937	###	0	8	>	8	2203
2011	Jun	10	12:15	Fri	FINIS	1560935	###	0	8	>	8	2211
2011	Jun	10	12:15	Fri	FINIS	1560938	###	0	8	>	8	2219
2011	Jun	10	12:15	Fri	FINIS	1560936	###	0	8	>	8	2227
2011	Jun	10	12:15	Fri	FINIS	1560939	###	0	8	>	8	2235
2011	Jun	10	12:15	Fri	FINIS	1560940	###	0	8	>	8	2243
2011	Jun	10	13:17	Fri	FINIS	1560888	###	0	32	>	8	2275
2011	Jun	10	18:00	Fri	SHRTP	0	released	-8	8	>	0	2283
2011	Jun	11	0:27	Sat	*FINI	1560263	###	0	8	>	0	2291
2011	Jun	11	7:00	Sat	SHRTP	0	activate	8	-8	>	8	2283
2011	Jun	11	18:00	Sat	SHRTP	0	released	-8	8	>	0	2291

B. STATIC SCALABLAST BATCH JOB RUN TIME RESULTS

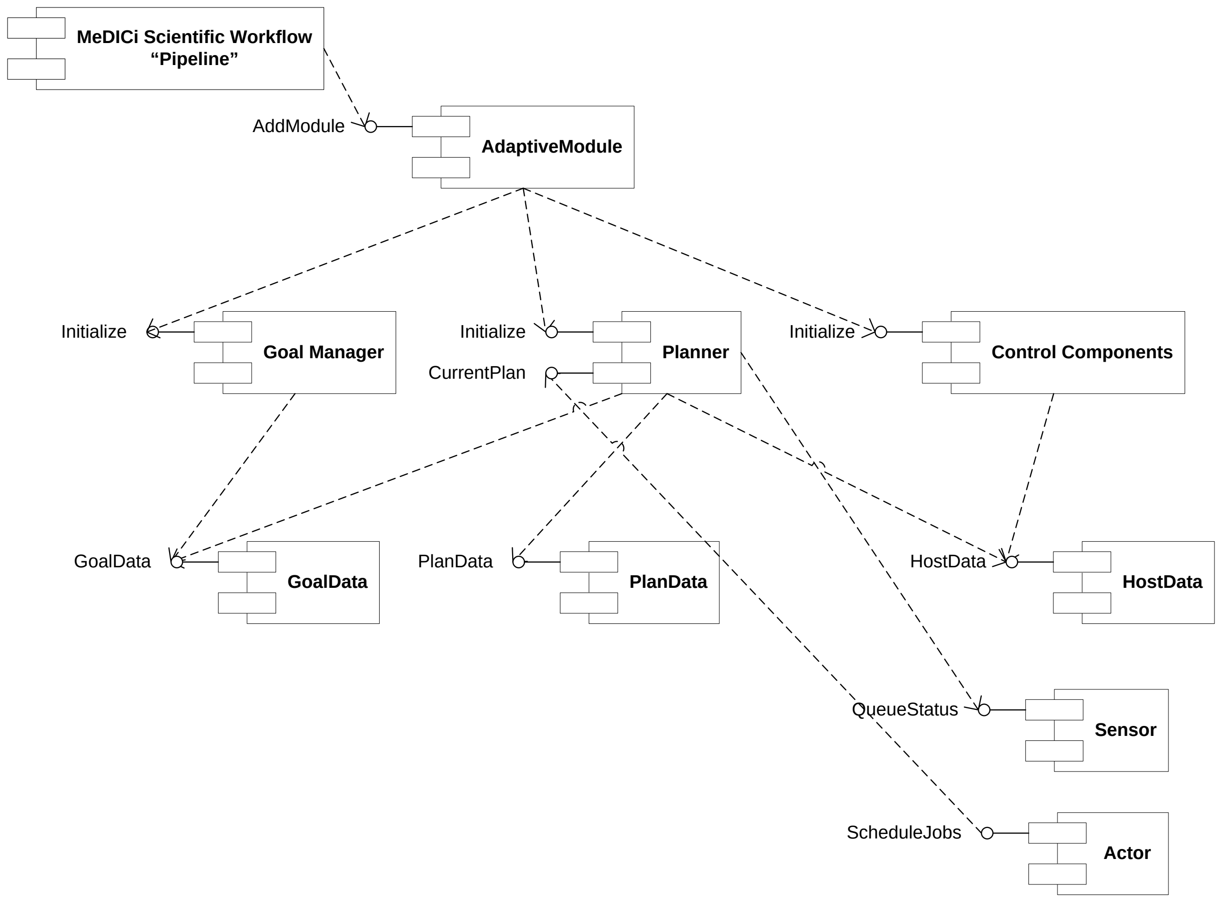
							STATIC			
Job Run	source	date	start job id	end job id	node count at runtime	total run time	submission time	job start time	job end time	total time to completion
1	kbase_7275	5/5/2011	1546022	1546022	98	4:30:00	8:37:13 AM	8:49:34 PM	1:19:34 AM	16:42:21
2	kbase_2832	5/9/2011	1547950	1547950	110	4:00:00	10:44:21 AM	9:26:54 PM	1:26:54 AM	14:42:33
3	kbase_5455	5/9/2011	1548090	1548090	227	2:00:00	2:40:28 PM	5:34:04 PM	7:34:04 PM	4:53:36
4	kbase_5417	5/10/2011	1548482	1548482	268	1:40:00	2:25:50 AM	5:37:02 PM	7:17:02 PM	16:51:12
5	kbase_4288	5/11/2011	1548859	1548859	186	2:30:00	1:07:20 PM	2:07:49 PM	4:37:49 PM	3:30:29
6	kbase_7070	5/11/2011	1548922	1548922	176	2:30:00	2:58:57 PM	4:58:42 PM	7:28:42 PM	4:29:45
7	kbase_2865	5/19/2011	1552496	1552496	321	1:30:00	3:24:55 PM	9:43:42 PM	11:13:42 PM	7:48:47
8	kbase_1382	5/20/2011	1552767	1552767	60	7:30:00	11:18:56 AM	2:37:09 PM	10:07:09 PM	10:48:13
9	kbase_4944	5/23/2011	1553579	1553579	220	2:00:00	8:06:09 AM	9:58:35 AM	11:58:35 AM	3:52:26
10	kbase_6672	5/24/2011	1554603	1554603	175	2:40:00	8:28:55 AM	3:07:23 PM	5:47:23 PM	9:18:28
11	kbase_2161	6/1/2011	1557692	1557692	345	1:20:00	9:23:14 AM	3:34:23 PM	4:54:23 PM	7:31:09
12	kbase_6457	6/1/2011	1557736	1557736	169	2:40:00	10:41:10 AM	10:29:43 PM	1:09:43 AM	14:28:33
13	kbase_1447	6/2/2011	1558252	1558252	191	2:20:00	12:12:13 PM	7:01:13 PM	9:21:13 PM	9:09:00
14	kbase_3737	6/3/2011	1558756	1558756	450	1:00:00	9:06:18 AM	12:40:56 PM	1:40:56 PM	4:34:38
15	kbase_9048	6/3/2011	1559021	1559021	148	3:00:00	1:03:23 PM	2:14:36 PM	5:14:36 PM	4:11:13
16	kbase_1737	6/3/2011	1559109	1559109	132	3:20:00	3:23:31 PM	7:06:57 PM	10:26:57 PM	7:03:26
17	kbase_5188	6/6/2011	1560060	1560060	145	3:10:00	10:13:34 AM	7:03:04 PM	10:13:04 PM	11:59:30
18	kbase_6571	6/7/2011	1560445	1560445	318	1:30:00	8:25:20 AM	12:27:38 PM	1:57:38 PM	5:32:18
19	kbase_6584	6/7/2011	1560669	1560669	440	1:00:00	4:15:39 PM	6:21:36 PM	7:21:36 PM	3:05:57
20	kbase_7078	6/8/2011	1560880	1560880	49	9:10:00	8:34:05 AM	9:52:12 AM	7:02:12 PM	10:28:07

C. DYNAMIC SCALABLAST BATCH JOB RUN TIME RESULTS

DYNAMIC

Job Run	source	start job id	end job id	node count at runtime	total run time	submission time	job start time	job end time	to completion
1	kbase_72	1546023	1546099	167	2:40:00	8:37:16 AM	3:50:39 PM	6:30:39 PM	9:53:23
2	kbase_28	1547951	1547952	190	3:10:00	10:44:22 AM	2:33:17 PM	5:43:17 PM	6:58:55
3	kbase_54	1548091	1548103	151	3:00:00	2:40:29 PM	5:34:04 PM	8:34:04 PM	5:53:35
4	kbase_54	1548483	1548483	268	1:40:00	2:25:51 PM	5:38:57 PM	7:18:57 PM	4:53:06
5	kbase_42	1548860	1548860	186	2:30:00	1:07:21 PM	2:09:33 PM	4:39:33 PM	3:32:12
6	kbase_70	1548923	1548947	384	1:10:00	2:58:58 PM	4:56:46 PM	6:06:46 PM	3:07:48
7	kbase_28	1552497	1552565	212	3:32:00	3:24:56 PM	6:57:03 PM	10:29:03 PM	7:04:07
8	kbase_13	1552768	1552778	269	1:40:00	11:18:58 AM	12:09:04 PM	1:49:04 PM	2:30:06
9	kbase_49	1553580	1553619	312	1:30:00	8:56:10 AM	9:59:30 AM	11:29:30 AM	2:33:20
10	kbase_66	1554604	1554604	175	2:40:00	8:29:01 AM	3:08:54 PM	5:48:54 PM	9:19:53
11	kbase_21	1557693	1557725	123	3:40:00	9:23:15 AM	10:27:36 AM	2:07:36 PM	4:44:21
12	kbase_64	1557737	1557737	169	2:40:00	10:41:11 AM	10:29:44 PM	1:09:44 AM	14:28:33
13	kbase_14	1558253	1558253	191	2:20:00	12:12:34 PM	7:02:51 PM	9:22:51 PM	9:10:17
14	kbase_37	1558982	1558982	338	1:20:00	9:06:19 AM	12:34:58 PM	1:54:58 PM	4:48:39
15	kbase_90	1559022	1559022	148	3:00:00	1:03:24 PM	2:16:20 PM	5:16:20 PM	4:12:56
16	kbase_17	1559110	1559110	132	3:20:00	3:23:32 PM	7:06:58 PM	10:26:58 PM	7:03:26
17	kbase_51	1560061	1560298	273	1:40:00	10:13:35 AM	7:01:18 PM	8:41:18 PM	10:27:43
18	kbase_65	1560446	1560446	318	1:30:00	8:25:21 AM	12:29:22 PM	1:59:22 PM	5:34:01
19	kbase_65	1560670	1560675	149	3:00:00	4:15:40 PM	4:28:49 PM	7:28:49 PM	3:13:09
20	kbase_70	1560881	1560882	191	2:20:00	8:34:06 AM	9:51:32 AM	12:11:32 PM	3:37:26

D. UML COMPONENT DIAGRAM FOR ADAPTIVE MIF



E. SOURCE CODE

The source code attached in this appendix is for future reference only and future documentation and refinement is reserved.

SENSE.CSH

```
#!/bin/csh -f
#$1 = server share //xyz.pnl.gov/xxx
#$2 = server folder
#$3 = host name
set JOB_FILE=$3.job
set BLAST_OUT_FILE=blast.out
set BLAST_END_FILE=blast.end
set SENSE_FILE=$3.sense.real
set SENSE_ADAPT_FILE=$3.sense.adapt

if ( -e done.txt ) then
   rm done.txt
endif

while (1)
  putsense:
   rm $3.lock
   rm $SENSE_ADAPT_FILE
   ./qtime.pl > $SENSE_FILE
   if ( -e $JOB_FILE ) then
      set jobLine = `cat $JOB_FILE`
      set jobHostName = `echo $jobLine | awk '{split($0,a,":"); print a[1]}'`
      set jobid = `echo $jobLine | awk '{split($0,a,":"); print a[2]}'`
      set jobNumNode = `echo $jobLine | awk '{split($0,a,":"); print a[3]}'`
      set jobCalDuration = `echo $jobLine | awk '{split($0,a,":"); print a[4]}'`
      set jobWaitTime = `echo $jobLine | awk '{split($0,a,":"); print a[5]}'`
      set nchar = `echo $jobid | awk '{print length($0)}'`
      if ( $nchar != 0 ) then
         set process = `cat $3.sense.real | grep " $jobid " | grep " START " | grep -v grep`
         set nchar = `echo $process | awk '{print length($0)}'`
         #waiting in the queue to run
         if ( $3 == $jobHostName && ! -e $BLAST_OUT_FILE && ! -e $BLAST_END_FILE && $nchar !=
0 ) then
            ./qtime.pl $jobid > $SENSE_ADAPT_FILE
         endif
      endif
   endif

   if ( ! -e $SENSE_ADAPT_FILE ) then
      cp $SENSE_FILE $SENSE_ADAPT_FILE
   endif

   smbclient $1 -A .smbclient -c "cd $2/sense; get $3.lock; exit;"
   if ( -e $3.lock ) then
      sleep 5 #sleep for 5 seconds
      goto putsense
   endif
   touch $3.lock
   smbclient  $1  -A  .smbclient  -c  "cd  $2/sense;  put  $3.lock;  put  $SENSE_FILE;  put
$SENSE_ADAPT_FILE; rm $3.lock; cd ../; cd process; get done.txt; exit;"
   if ( -e done.txt ) then
      break
   endif
   echo "sleeping for 20 seconds..."
   sleep 20 #sleep for 20 seconds
end

echo "exiting sense.csh gracefully..."
exit 0
```

ACT.CSH

```
#!/bin/csh -f
#$1 = Server file share //xyz.pnl.gov/xxx
#$2 = server folder
#$3 = hostname
#$4 = job to run pre_process_blast.csh
#$5 = userid

#module purge
#module load precision/i4
#module load intel/10.1.015
#module load hpmpi/2.3.1
module load moab

set ACT_FILE=act.txt
set JOB_FILE=$3.job
set BLAST_OUT_FILE=blast.out
set BLAST_END_FILE=blast.end
set BLAST_RUNNING_FILE=$3.running
set BLAST_WAITTIME_FILE=$3.wait
set BLAST_ERROR_FILE=$3.error
set userid = $5
set raceBit=1

rm -f $BLAST_OUT_FILE
rm -f $BLAST_END_FILE
rm -f $BLAST_RUNNING_FILE
rm -f $BLAST_ERROR_FILE

while (1)
getact:
   rm -f $ACT_FILE
   rm -f $JOB_FILE
   rm -f act.lock
   smbclient  $1  -A  .smbclient  -c  "prompt;  cd  $2/act;  get  act.lock;  get  $ACT_FILE;  get
$JOB_FILE; exit;"
   if ( -e act.lock ) then
      sleep 5 #sleep for 5 seconds
      goto getact
   endif
   if ( ! -e $ACT_FILE ) then
      if ( -e $JOB_FILE ) then
         set line = `cat $JOB_FILE`
         set hostname = `echo $line | awk '{split($0,a,":"); print a[1]}'`
         set jobid = `echo $line | awk '{split($0,a,":"); print a[2]}'`
         set numNode = `echo $line | awk '{split($0,a,":"); print a[3]}'`
         set calDuration = `echo $line | awk '{split($0,a,":"); print a[4]}'`
         set waitTime = `echo $line | awk '{split($0,a,":"); print a[5]}'`
         if ( $hostname == $3 ) then
            set process = `./qtime.pl | grep " $jobid " | grep " $userid " | grep -v grep`
            set nchar = `echo $process | awk '{print length($0)}'`
            if ( $nchar != 0 ) then
               canceljob $jobid
            endif
jobFileSense1:
            rm $JOB_FILE.lock
            smbclient $1 -A .smbclient -c "prompt; cd $2/act; get $JOB_FILE.lock; exit;"
            if ( -e $JOB_FILE.lock ) then
               sleep 5
               goto jobFileSense1
            endif
            smbclient $1 -A .smbclient -c "cd $2/act; rm $JOB_FILE; exit;"
         endif
      endif

   else

      set actLine = `cat $ACT_FILE`
      set actHostName = `echo $actLine | awk '{split($0,a,":"); print a[1]}'`
      set actNumNode = `echo $actLine | awk '{split($0,a,":"); print a[2]}'`
      set actCalDuration = `echo $actLine | awk '{split($0,a,":"); print a[3]}'`
      set actWaitTime = `echo $actLine | awk '{split($0,a,":"); print a[4]}'`
      set qFile = `echo $actLine | awk '{split($0,a,":"); print a[5]}'`
      set dbFile = `echo $actLine | awk '{split($0,a,":"); print a[6]}'`
      set paramsFile = `echo $actLine | awk '{split($0,a,":"); print a[7]}'`
      set cancelJobID = 0

      if ( -e $JOB_FILE ) then
         set jobLine = `cat $JOB_FILE`
         set jobHostName = `echo $jobLine | awk '{split($0,a,":"); print a[1]}'`
         set jobid = `echo $jobLine | awk '{split($0,a,":"); print a[2]}'`
         set jobNumNode = `echo $jobLine | awk '{split($0,a,":"); print a[3]}'`
         set jobCalDuration = `echo $jobLine | awk '{split($0,a,":"); print a[4]}'`
         set jobWaitTime = `echo $jobLine | awk '{split($0,a,":"); print a[5]}'`

         #act action is the same as the job action for THIS host AND
         #the job is actually scheduled and running
         set process = `./qtime.pl | grep " $jobid " | grep " $userid " | grep " START " |
grep -v grep`
         set nchar = `echo $process | awk '{print length($0)}'`

         #waiting in the queue to run
         if ( ($actHostName == $jobHostName && $actNumNode == $jobNumNode && $nchar != 0))
then
            set process = `showstart $jobid`
            set wMinStr = `echo $process | awk '{split($0,a,":"); print a[4]}'`
            @ wMin = $wMinStr
            set wHStr = `echo $process | awk '{split($0,a,":"); print a[3]}'`
            set wHStrLen = `echo $wHStr | awk '{print length($0)}'`
            @ wHSubStr = $wHStrLen - 1
            if ( $wHSubStr >= 32 ) then
               set wHourStr = `echo $wHStr | awk '{print substr($0,32,2)}'`
               set wHStrLen = `echo $wHourStr | awk '{print length($0)}'`
               if ( $wHStrLen > 2 ) then
                  sleep 5 #sleep for 5 seconds
                  goto getact
               endif
               @ wHour = $wHourStr
               echo "hour=$wHour"
               @ waitTime = $wMin + ($wHour * 60)
            else
               @ waitTime = $wMin
            endif

            echo $waitTime > $BLAST_WAITTIME_FILE
            smbclient $1 -A .smbclient -c "prompt; cd $2/process; put $BLAST_WAITTIME_FILE;
exit;"
            echo "$jobid is in schedule... sleeping for 20seconds"
            sleep 20
            goto getact
         endif

         set process = `./qtime.pl | grep " $jobid " | grep " $userid " | grep " FINIS " |
grep -v grep`
         set nchar = `echo $process | awk '{print length($0)}'`
         echo "not in queue anymore... nchar=$nchar"

         #running...
         if ( $actHostName == $jobHostName && $actNumNode == $jobNumNode && $nchar != 0 ) then
            echo "$jobid is running... sleeping for 5 minutes"
            echo "$jobid is running" > $BLAST_RUNNING_FILE
            smbclient $1 -A .smbclient -c "prompt; cd $2/process; put $BLAST_RUNNING_FILE;
exit;"
            sleep 300 #wait for 5 minutes
            goto getact
         endif

         #ran successfully
         if ( ($actHostName == $jobHostName && $actNumNode == $jobNumNode && $nchar == 0))
then
            echo "$jobid completed successfully."
            smbclient $1 -A .smbclient -c "prompt; cd $2/process; put $BLAST_OUT_FILE; put
$BLAST_END_FILE; exit;"
            break
         endif

         #ran with errors
         if ( $actHostName == $jobHostName && $nchar == 0 && -e $BLAST_OUT_FILE && ! -e
$BLAST_END_FILE ) then
         #our job ran but didn't finish all the way, there was an error, so quit.
            echo "$jobid ran with errors -- reporting" > $BLAST_ERROR_FILE
             smbclient $1 -A .smbclient -c "prompt; cd $2/process; put $BLAST_ERROR_FILE;
exit;"
             break
          endif


          #different host is picked or
          #different runtime parameters are picked
          #cancel the job
          if ( $actHostName != $jobHostName || ( $actHostName == $jobHostName && $actNumNode
!= $jobNumNode )) then
             set process = `./qtime.pl | grep " $jobid " | grep " $userid " | grep -v grep`
             set nchar = `echo $process | awk '{print length($0)}'`
             echo "$jobid needs to be canceled"
             if ( $nchar != 0 ) then
                #canceljob after schedule
                set cancelJobID = $jobid
             endif
jobFileSense2:
             rm $JOB_FILE.lock
             smbclient $1 -A .smbclient -c "prompt; cd $2/act; get $JOB_FILE.lock; exit;"
             if ( -e $JOB_FILE.lock ) then
                sleep 5
                goto jobFileSense2
             endif
             smbclient $1 -A .smbclient -c "cd $2/act; rm $JOB_FILE; exit;"
          endif

      endif  #if jobfile exists.

      #if this host is still adaptivehost picked
      #schedule the job
      if ($actHostName == $3) then
         #run pre_process_blast.csh
         ./$4 $qFile $dbFile $paramsFile $actNumNode $actCalDuration
         echo $qFile.msub
         if ( -e $qFile.msub ) then
            #schedule the static job before the dynamic one.
            if ( $raceBit == 1 ) then
               msub ./$qFile.msub >& raceBit.out
               set raceBit = 0
            endif

            #get the jobid from the run
            msub ./$qFile.msub >& j.out
            #canceljob after scheduling the new one
            if ( $cancelJobID != 0 ) then
               ./qtime.pl > beforecancel_qtime.out
               showq > beforecancel_showq.out
               sinfo > beforecancel_sinfo.out
               showres > beforecancel_showres.out
               showres -f > beforecancel_showresf.out
               canceljob $cancelJobID
               ./qtime.pl > aftercancel_qtime.out
               showq > aftercancel_showq.out
               sinfo > aftercancel_sinfo.out
               showres > aftercancel_showres.out
               showres -f > aftercancel_showresf.out
            endif
            set working_variable=`cat j.out | sed -e '/^$/d'`
            echo "Step 1: set the working_variable to $working_variable"
            rm j.out

            set filtered_variable=`echo $working_variable | grep -E '^[0-9]+$'`
            echo "Step 2: set filtered_variable to $filtered_variable"

            if ($working_variable != $filtered_variable) then
               echo "** ERROR ** working_variable is not just an integer"
            else
               set jobid = $working_variable
               echo "$jobid is the NEW job scheduled."

               #rewrite jobfile
               echo "$actHostName" > $JOB_FILE
               echo ":" >> $JOB_FILE
               echo "$jobid" >> $JOB_FILE
               echo ":" >> $JOB_FILE
               echo "$actNumNode" >> $JOB_FILE
               echo ":" >> $JOB_FILE
               echo "$actCalDuration" >> $JOB_FILE
               echo ":" >> $JOB_FILE
               echo "$actWaitTime" >> $JOB_FILE
               #put the jobfile on the file share
jobFileSense3:
               rm $JOB_FILE.lock
               smbclient $1 -A .smbclient -c "prompt; cd $2/act; get $JOB_FILE.lock; exit;"
               if ( -e $JOB_FILE.lock ) then
                  sleep 5
                  goto jobFileSense3
               endif
               smbclient  $1  -A  .smbclient  -c  "prompt;  cd  $2/act;  put  $JOB_FILE;  put
raceBit.out; exit;"
            endif
         endif
      endif #if schedule is necessary.
   endif  #if act.txt file exists

   echo "sleeping for 20 seconds..."
   sleep 20 #sleep for 20 seconds.
end
echo "exiting act.csh gracefully..."
exit 0
```

PRE_PROCESS_BLAST.CSH

```
#!/bin/csh -f
#$1 = queryFile
#$2 = DBFile
#$3 = ParamsFile
#$4 = numberOfNodes
#$5 = totalTime

set FILE='/dtemp/xxx'

@ n = (4 * $4)
@ tgs = $n - 1

@ hours = $5 / 60
@ minutes = $5 % 60
set wallTime = "${hours}:${minutes}:00"
rm -f ${FILE}/$1*out*
rm -f ${FILE}/$1*log*
rm -f $1.blast
rm -f done.end

while (! -e $FILE/$1)
   sleep 1
end

while (! -e $FILE/$2)
   sleep 1
end

echo "files are copied"

#*****************************
#create the sb_params.in file
#*****************************

touch ${FILE}/sb_params.in
echo "LOCAL_DIR        ${FILE}/" > ${FILE}/sb_params.in
echo "GLOBAL_DIR       ./" >> ${FILE}/sb_params.in
echo "MAX_FASTA_CHUNKS 64" >> ${FILE}/sb_params.in
echo "MAX_FASTA_LINE_LENGTH 1000000" >> ${FILE}/sb_params.in
echo "REF_BUFF_SIZE    671088640" >> ${FILE}/sb_params.in
echo "REF_META_SIZE    33554432" >> ${FILE}/sb_params.in
echo "QUERY_BUFF_SIZE  4194304" >> ${FILE}/sb_params.in
echo "QUERY_META_SIZE  524288" >> ${FILE}/sb_params.in
echo "MAX_NUM_QUERIES  11000000" >> ${FILE}/sb_params.in
echo "DISK_GROUP_SIZE  $n" >> ${FILE}/sb_params.in
echo "TASK_GROUP_SIZE  $tgs" >> ${FILE}/sb_params.in
echo "FIRST_SUBMANAGER 1" >> ${FILE}/sb_params.in
echo "SEQ_PER_QUIT_CHECK 1" >> ${FILE}/sb_params.in

echo "created the parameters file ${FILE}/sb_params.in file!"
chmod a+r ${FILE}/sb_params.in

#******************************
#create the msub file
#******************************

touch ${FILE}/$1.msub
chmod a+w ${FILE}/$1.msub
echo "${FILE}/$1.msub"
echo '#\!/bin/csh' > ${FILE}/$1.msub
echo '#MSUB -A gc35595' >> ${FILE}/$1.msub
echo "#MSUB -l nodes=${4}:ppn=8,walltime=$wallTime,nodesetdelay=0" >> ${FILE}/$1.msub
echo "#MSUB -o $FILE/%j.out" >> ${FILE}/$1.msub
echo "#MSUB -e $FILE/%j.err" >> ${FILE}/$1.msub
echo "#MSUB -N $1" >> ${FILE}/$1.msub
echo '#MSUB -V' >> ${FILE}/$1.msub
echo 'module purge' >> ${FILE}/$1.msub
echo 'module load precision/i4' >> ${FILE}/$1.msub
echo 'module load intel/10.1.015' >> ${FILE}/$1.msub
echo 'module load hpmpi/2.3.1' >> ${FILE}/$1.msub
echo 'module load moab' >> ${FILE}/$1.msub

echo 'setenv OMP_NUM_THREADS 1' >> ${FILE}/$1.msub
echo 'setenv ACML_NUM_THREADS 1' >> ${FILE}/$1.msub
echo 'limit' >> ${FILE}/$1.msub
echo "cd $FILE/" >> ${FILE}/$1.msub
echo "touch ${FILE}/blast.out" >> ${FILE}/$1.msub
echo "mpirun -srun -n $n -N $4 /dtemp/oehmen/bin/Scalablastall.ak -p blastp -d $FILE/$2 -i
$FILE/$1 -o $FILE/$1.out" >> ${FILE}/$1.msub
echo "touch ${FILE}/blast.end" >> ${FILE}/$1.msub

echo "foreach item ($1.out.*)" >> ${FILE}/$1.msub
set str = 'cat $item'
echo "$str >> blast.out" >> ${FILE}/$1.msub
echo "end" >> ${FILE}/$1.msub

echo "created the msub ${FILE}/$1.msub file!"

chmod a+x ${FILE}/$1.msub

exit 0
```

PlanData.java

```
package adaptive;

public class PlanData {
        private GoalData selectedGoal;
        private String selectedHost;
        private Integer planJobID;
        private Integer waitTime;
        private Integer numberOfNodes;
        private Integer runtime;
        private Integer totalRunTime;
        private Integer totalCost;

        public void setPlanJobID(Integer planJobID) {
                this.planJobID = planJobID;
        }
        public Integer getPlanJobID() {
                return planJobID;
        }
        public Integer getTotalCost() {
                return totalCost;
        }
        public void setTotalCost(Integer totalCost) {
                this.totalCost = totalCost;
        }
        public GoalData getSelectedGoal() {
                return selectedGoal;
        }
        public void setSelectedGoal(GoalData selectedGoal) {
                this.selectedGoal = selectedGoal;
        }
        public String getSelectedHost() {
                return selectedHost;
        }
        public void setSelectedHost(String selectedHost) {
                this.selectedHost = selectedHost;
        }
        public Integer getWaitTime() {
                return waitTime;
        }
        public void setWaitTime(Integer waitTime) {
                this.waitTime = waitTime;
        }
        public Integer getNumberOfNodes() {
                return numberOfNodes;
        }
        public void setNumberOfNodes(Integer numberOfNodes) {
                this.numberOfNodes = numberOfNodes;
        }
        public Integer getRuntime() {
                return runtime;
        }
        public void setRuntime(Integer runtime) {
                this.runtime = runtime;
        }
        public Integer getTotalRunTime() {
                return totalRunTime;
        }
        public void setTotalRunTime(Integer totalRunTime) {
                this.totalRunTime = totalRunTime;
        }

}
```

GoalData.java

```java
package adaptive;

public class GoalData {
	private String id;
	private Integer minNodes;
	private Integer maxNodes;
	private Integer duration;
	private Integer minCost;
	private Integer maxCost;
	private String qFile;
	private Integer qFileSize;
	private String dbFile;
	private Integer dbFileSize;
	private String paramsFile;
	private String jobName;

	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public Integer getMinNodes() {
		return minNodes;
	}
	public void setMinNodes(Integer minNodes) {
		this.minNodes = minNodes;
	}
	public Integer getMaxNodes() {
		return maxNodes;
	}
	public void setMaxNodes(Integer maxNodes) {
		this.maxNodes = maxNodes;
	}
	public Integer getDuration() {
		return duration;
	}
	public void setDuration(Integer duration) {
		this.duration = duration;
	}
	public Integer getMinCost() {
		return minCost;
	}
	public void setMinCost(Integer minCost) {
		this.minCost = minCost;
	}
	public Integer getMaxCost() {
		return maxCost;
	}
	public void setMaxCost(Integer maxCost) {
		this.maxCost = maxCost;
	}
	public String getqFile() {
		return qFile;
	}
	public void setqFile(String qFile) {
		this.qFile = qFile;
	}
	public String getDbFile() {
		return dbFile;
	}
	public void setDbFile(String dbFile) {
		this.dbFile = dbFile;
	}
	public String getParamsFile() {
		return paramsFile;
	}

	public void setParamsFile(String paramsFile) {
```

```
                this.paramsFile = paramsFile;
        }
        public void setqFileSize(Integer qFileSize) {
                this.qFileSize = qFileSize;
        }
        public Integer getqFileSize() {
                return qFileSize;
        }
        public void setDbFileSize(Integer dbFileSize) {
                this.dbFileSize = dbFileSize;
        }
        public Integer getDbFileSize() {
                return dbFileSize;
        }
        public void setJobName(String jobName) {
                this.jobName = jobName;
        }
        public String getJobName() {
                return jobName;
        }
}
```

HostData.java

```
package adaptive;

public class HostData {
	private String hostName;
	private String homeDir;
	private String userName;
	private String sshDir;
	private String senseCmd;
	private String actCmd;
	private Integer costPerNode;
	private Integer jobRunTimePerNode;
	private Integer numCores;
	private Boolean metGoal;

	public Boolean getMetGoal() {
		return metGoal;
	}
	public void setMetGoal(Boolean metGoal) {
		this.metGoal = metGoal;
	}
	public Integer getJobRunTimePerNode() {
		return this.jobRunTimePerNode;
	}
	public void setJobRunTimePerNode(Integer jobRunTimePerNode) {
		this.jobRunTimePerNode = jobRunTimePerNode;
	}
	public Integer getCostPerNode() {
		return this.costPerNode;
	}
	public void setCostPerNode(Integer costPerNode) {
		this.costPerNode = costPerNode;
	}
	public String getHostName() {
		return hostName;
	}
	public void setHostName(String hostName) {
		this.hostName = hostName;
	}
	public String getHomeDir() {
		return homeDir;
	}
	public void setHomeDir(String homeDir) {
		this.homeDir = homeDir;
	}
	public String getUserName() {
		return userName;
	}
	public void setUserName(String userName) {
		this.userName = userName;
	}
	public String getSshDir() {
		return sshDir;
	}
	public void setSshDir(String sshDir) {
		this.sshDir = sshDir;
	}
	public String getSenseCmd() {
		return senseCmd;
	}
	public void setSenseCmd(String senseCmd) {
		this.senseCmd = senseCmd;
	}
	public String getActCmd() {
		return actCmd;
	}
	public void setActCmd(String actCmd) {
		this.actCmd = actCmd;
	}
	public void setNumCores(Integer numCores) {
		this.numCores = numCores;
	}
	public Integer getNumCores() {
		return numCores;
	}
}
```

TestAdaptiveDriver.java (MeDICi Adaptive Scientific Workflow - Pipeline)

```
package adaptive;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import gov.pnnl.mif.MifException;
import gov.pnnl.mif.MifPipeline;
import org.apache.log4j.Logger;

public class TestAdaptiveDriver {
   static Logger log = Logger.getLogger(TestAdaptiveDriver.class);
   protected static String tmpDir = System.getProperty("java.io.tmpdir");
   protected static String sep = System.getProperty("file.separator");
   public static void main(String args[]) throws MifException, IOException
   {
      //Aim of this pipeline is to test ONLY the adaptive component.
      //run the act.csh and sense.csh scripts manually on the hosts
      //that provides the best time to solution results.
      Map<String,Object> props = new HashMap<String,Object>();
      System.out.println(System.getProperty("user.dir"));

      Properties global = new Properties();
      HashMap<String, HostData> hosts = new HashMap<String, HostData>();
      Integer hostCount;

      try {
         File aFile = new File("adaptive.properties");
         System.out.println(aFile.getAbsolutePath());
         global.load(new FileInputStream("adaptive.properties"));
         props.put("fromURI", global.getProperty("from.uri").toString());
         props.put("toURI", global.getProperty("to.uri").toString());
         props.put("postCmd", global.getProperty("postprocess.command").toString());
         props.put("QFile", global.getProperty("QFile").toString());
         props.put("QFileSize", global.getProperty("QFileSize").toString());
         props.put("DBFile", global.getProperty("DBFile").toString());
         props.put("DBFileSize", global.getProperty("DBFileSize").toString());
         props.put("ParamsFile", global.getProperty("ParamsFile").toString());
         props.put("OutputFile", global.getProperty("OutputFile").toString());
         props.put("senseCmd", global.getProperty("sense.command").toString());
         props.put("senseDir", global.getProperty("unc.sense.dir").toString());
         props.put("actCmd", global.getProperty("act.command").toString());
         props.put("actDir", global.getProperty("unc.act.dir").toString());
         props.put("jobDir", global.getProperty("unc.process.dir").toString());
         hostCount = Integer.parseInt(global.getProperty("host.count").trim());

         for(int i=1; i <= hostCount; i++)
         {
            HostData aHostData = new HostData();
            aHostData.setHostName(global.getProperty("host" + Integer.toString(i) + ".name"));
            aHostData.setHomeDir(global.getProperty("host"     +     Integer.toString(i)     +
".homeDir"));
            aHostData.setSshDir(global.getProperty("host" + Integer.toString(i) + ".sshDir"));
            aHostData.setUserName(global.getProperty("host"     +     Integer.toString(i)     +
".userName"));
            aHostData.setCostPerNode(Integer.parseInt(global.getProperty("host"             +
Integer.toString(i) + ".costPerNode").trim()));
            aHostData.setJobRunTimePerNode(Integer.parseInt(global.getProperty("host"        +
Integer.toString(i) + ".jobRunTimePerNode").trim()));
            aHostData.setNumCores(Integer.parseInt(global.getProperty("host"                +
Integer.toString(i) + ".numberOfCores").trim()));
            String cmd;
            cmd = global.getProperty("sense.command");
            aHostData.setSenseCmd(cmd);
            cmd = global.getProperty("act.command");
            aHostData.setActCmd(cmd);
            aHostData.setMetGoal(false);
            hosts.put(aHostData.getHostName(), aHostData);
         } //end of for
      } catch (IOException e) {
         System.out.println("Cannot Read the adaptive.properties file");
         throw e;
      }

      if                      ((props.get("DBFile").toString().isEmpty()                     ||
props.get("QFile").toString().isEmpty()) || props.get("ParamsFile").toString().isEmpty())
      {
         System.out.println("Please specify a database or a query file to run blast on...
Quit...");
         throw new IOException("Please specify a database or a query file to run blast on...
Quit...");
      }

      MifPipeline pipeline = new MifPipeline();
      /*
       * this is for the AdaptiveComponent.
       *
      */
      pipeline.addMifModule(AdaptiveModule.class,    "stdio://in?promptMessage=press    return",
"stdio://stdout");
      pipeline.start();
   }
}
```

AdaptiveModule.java

```
package adaptive;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.RandomAccessFile;
import gov.pnnl.mif.user.MifProcessor;
import java.io.Serializable;
import java.text.DateFormat;
import java.util.Locale;
import java.util.Scanner;
import java.util.Date;

public class AdaptiveModule implements MifProcessor {
    private String adaptiveHost;
    protected static String sep = System.getProperty("file.separator");

    void copy(File src, File dst) throws IOException {
        InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dst);
        // Transfer bytes from in to out
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();
    }

    @Override
    public Serializable listen(Serializable name) {
        System.out.println("in AdaptiveProcessorComponent");
        GoalManager aGoalManager = new GoalManager();
        Planner aPlanner = new Planner();
        ControlComponent aControlComponent = new ControlComponent();
        aGoalManager.initialize();
        aPlanner.initialize(aGoalManager);
        aControlComponent.initialize();

        Integer i = 1;

        String actFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + "act.txt";
        File actFile = new File(actFileStr);
        if (actFile.exists()) actFile.delete();
        String lFile = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + "act.log";
        File logFile = new File(lFile);
        if (logFile.exists()) logFile.delete();

        String jobFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + aPlanner.getSelectedPlan().getSelectedHost() + ".job";
        File jobFile = new File(jobFileStr);
        String errFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + aPlanner.getSelectedPlan().getSelectedHost() +
".error";
        File errFile = new File(errFileStr);
        String runFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + aPlanner.getSelectedPlan().getSelectedHost() +
".running";
        File runFile = new File(runFileStr);
        String waitFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + aPlanner.getSelectedPlan().getSelectedHost() +
".wait";
```

```
        File waitFile = new File(waitFileStr);
        String outFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + "blast.out";
        File outFile = new File(outFileStr);
        if (outFile.exists()) outFile.delete();
        String endFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + "blast.end";
        File endFile = new File(endFileStr);
        if (endFile.exists()) endFile.delete();
        String doneFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + "done.txt";
        File doneFile = new File(doneFileStr);
        if (doneFile.exists()) doneFile.delete();
        String startFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + "start.txt";
        File startFile = new File(startFileStr);
        if (startFile.exists()) startFile.delete();
        try {
            startFile.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }

        Boolean canAchieveGoal = false;
        while (true)
        {
            if (errFile.exists() || runFile.exists() || outFile.exists())
            {
                doneFile = new File(doneFileStr);
                try {
                    doneFile.createNewFile();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                break;
            }

            if (jobFile.exists())
            {
                Scanner s = null;
                String aJobLockFile = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + jobFile.getName() + ".lock";
                File jobLockFile = new File(aJobLockFile);
                try {
                    s = new Scanner(jobFile);
                    jobLockFile.createNewFile();
                    StringBuilder line = new StringBuilder();
                    if
(jobFile.getName().contains(aPlanner.getSelectedPlan().getSelectedHost()) )
                    {
                        while (s.hasNextLine())
                        {
                            line.append(s.nextLine());
                        }
                    }
                    if (line.length() > 0)
                    {
                        String[] jobid;
                        jobid = line.toString().split(":");

aPlanner.getSelectedPlan().setPlanJobID(Integer.parseInt(jobid[1].trim()));
                }
                } catch (FileNotFoundException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                finally{
                    if (!s.equals(null))
```

```
                        s.close();
                    if (jobLockFile.exists())
                        jobLockFile.delete();
                }
            }
            else {
                aPlanner.getSelectedPlan().setPlanJobID(-1);
            }

            if (waitFile.exists())
            {
                //it used to get the waittime from showstart comment.
                //the current direction is to use qtime.pl output to update the waittime.
                //therefore the code to get the showstart output is now commented out but
                //waitFile is still used to indicate that our job is still waiting in the
queue.

                String senseLockFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_sense_share() +  Planner.getSelectedPlan().getSelectedHost() +
".lock";
                File senseLockFile = new File(senseLockFileStr);
                try {
                    while (senseLockFile.exists())
                    {
                        Thread.sleep(5000);  //5 seconds.
                        senseLockFile = new File(senseLockFileStr);
                    }
                    try {
                        senseLockFile.createNewFile();
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

                String sFile = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_sense_share() + aPlanner.getSelectedPlan().getSelectedHost() +
".sense.real";
                File senseFile = new File(sFile);
                String copyFile = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_sense_share() + aPlanner.getSelectedPlan().getSelectedHost() +
".sense.real.copy";
                File aCopySenseFile = new File(copyFile);
                if (senseFile.exists())
                {
                    try {
                        copy(senseFile, aCopySenseFile);
                    } catch (IOException e2) {
                        // TODO Auto-generated catch block
                        e2.printStackTrace();
                    }
                    try {
                        //src = new Scanner(new FileReader(aFile));
                        String line;
                        String month;
                        String nowMonth;
                        Integer day = 0;
                        Integer nowDay = 0;
                        Integer hour = 0;
                        Integer nowHour = 0;
                        Integer minute = 0;
                        Integer nowMinute = 0;
                        String monthChanged = "";
                        Integer addMonths = 0;
                        Integer prevDay = 31;
                        Integer jobno = -1;
                        RandomAccessFile raf = new RandomAccessFile(copyFile, "r");
                        try {
                            i = 0;
                            while (raf.getFilePointer() < raf.length())
                            {
                                line = raf.readLine();
                                if (line.contains(">") && !line.contains("Reservation"))
                                {
                                    month = line.substring(6, 9);
                                    day = Integer.parseInt(line.substring(10, 12).trim());
                                    hour = Integer.parseInt(line.substring(13, 15).trim());
                                    minute = Integer.parseInt(line.substring(16, 18).trim());
                                    if (i.equals(0))
                                    {
                                        nowMonth = month;
                                        monthChanged = month;
                                        nowDay = day;
                                        nowHour = hour;
                                        nowMinute = minute;
                                        i = 1;
                                    }
                                    //16 characters for the jobno.
                                    try {
                                        jobno = Integer.parseInt(line.substring(28, 
46).trim());
                                        if 
(jobno.equals(aPlanner.getSelectedPlan().getPlanJobID()) && line.contains("START"))
                                        {
                                            // update waittime according to the latest qtime 
and quit
                                            if (!month.equalsIgnoreCase(monthChanged))
                                            {
                                                addMonths = addMonths + prevDay;
                                                monthChanged = month;
                                            }
                                            prevDay = day;
                                            Integer waitTime;
                                            waitTime = ((((addMonths + day - nowDay) * 24 + 
hour) - nowHour) * 60) + (minute - nowMinute);
                                            aPlanner.getSelectedPlan().setWaitTime(waitTime);
                                            aPlanner.getSelectedPlan().setTotalRunTime( 
aPlanner.getSelectedPlan().getRuntime() + waitTime );
                                            raf.seek(raf.length());
                                        }
                                    } catch (NumberFormatException e) {
                                        // TODO Auto-generated catch block
                                    }
                                } //if line contains '>'
                            } //end of while
                        }
                        catch (IOException e1) {
                            // TODO Auto-generated catch block
                        }  //end of while for file row iteration

                        try {
                            raf.close();
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }

                    } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        System.out.println("Cannot open source file, first run " + copyFile);
                        //e.printStackTrace();
                    }  //end try for openning randomaccessfile.
                }//if sensefile exists.
                senseLockFile.delete();  //read from the sense file is done, release the 
resource.
            }

            try {
                canAchieveGoal = aPlanner.evaluateGoal(aGoalManager, aControlComponent, i);
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            System.out.println("selectedPlan Host: " +
aPlanner.getSelectedPlan().getSelectedHost());
            System.out.println("selectedPlan numberOfNodes: " +
aPlanner.getSelectedPlan().getNumberOfNodes().toString());
            System.out.println("selectedPlan runTime: " +
aPlanner.getSelectedPlan().getRuntime().toString());
            System.out.println("selectedPlan waitTime: " +
aPlanner.getSelectedPlan().getWaitTime().toString());
            System.out.println("selectedPlan totalRunTime: " +
aPlanner.getSelectedPlan().getTotalRunTime().toString());
            System.out.println("selectedPlan totalCost: " +
aPlanner.getSelectedPlan().getTotalCost().toString());

            this.adaptiveHost = aPlanner.getSelectedPlan().getSelectedHost();
            if (!canAchieveGoal)
                aGoalManager.updateGoal(canAchieveGoal);

            errFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + this.adaptiveHost + ".error";
            errFile = new File(errFileStr);
            runFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + this.adaptiveHost + ".running";
            runFile = new File(runFileStr);
            jobFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + this.adaptiveHost + ".job";
            jobFile = new File(jobFileStr);
            waitFileStr = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_process_share() + this.adaptiveHost + ".wait";
            waitFile = new File(waitFileStr);
            outFile = new File(outFileStr);

            try {
                Thread.sleep(60000);  //sleep for one minute.
                aGoalManager.updateGoal(true);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        String str = "Adaptive Module Exiting Gracefully...";
        if ( logFile.exists() )
        {
            try {
                File f =
File.createTempFile(aGoalManager.getGoals().get(aGoalManager.getSelectedGoalID()).getJobName()
+ "_",".log");
                String filename = f.getName();
                f.delete();

                String aLogFile = aControlComponent.getUnc_file_share() +
aControlComponent.getUnc_act_share() + filename;
                File storeFile = new File(aLogFile);
                FileInputStream l_in = new FileInputStream(logFile);
                FileOutputStream f_out = new FileOutputStream(storeFile, true);

                byte[] buf = new byte[1024];
                int len;

                while ((len = l_in.read(buf)) > 0){
                    f_out.write(buf, 0, len);
                }
                l_in.close();
                f_out.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        return str;
    }

    public void setAdaptiveHost(String adaptiveHost) {
        this.adaptiveHost = adaptiveHost;
    }

    public String getAdaptiveHost() {
        return adaptiveHost;
    }
}
```

Planner.java

```
package adaptive;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.RandomAccessFile;
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class Planner {
    protected static String tmpDir = System.getProperty("java.io.tmpdir");
    private PlanData selectedPlan;
    public PlanData getSelectedPlan() {
        return selectedPlan;
    }
    public void setSelectedPlan(PlanData selectedPlan) {
        this.selectedPlan = selectedPlan;
    }
    public void initialize(GoalManager aGoalMgr) {
        selectedPlan = new PlanData();
        selectedPlan.setSelectedGoal(aGoalMgr.getSelectedGoal(aGoalMgr.getSelectedGoalID()));
        selectedPlan.setSelectedHost("xxx");
        selectedPlan.setPlanJobID(-1);
        selectedPlan.setNumberOfNodes(0);
        selectedPlan.setRuntime(500000);
        selectedPlan.setWaitTime(500000);
        selectedPlan.setTotalRunTime(selectedPlan.getRuntime() + selectedPlan.getWaitTime());
        selectedPlan.setTotalCost(2000000);
    }

    public Integer getHowLongNodesAreAvailable(RandomAccessFile r, String month, Integer d,
Integer h, Integer m, Integer availNodes)
    {
        Integer howLongNodesAreAvail = 0;
        Integer tempAvailNodes;
        Integer tempDay = d;
        Integer prevDay = 0;
        Integer tempHour = h;
        Integer tempMin = m;
        String tempMonth;
        String monthChanged = month;
        Integer addMonths = 0;
        String line;
        try {
            while (r.getFilePointer() < r.length())
            {
                line = r.readLine();
                if (line.length() < 19)
                    return 0;
                tempMonth = line.substring(6, 9).trim();
                tempDay = Integer.parseInt(line.substring(10, 12).trim());
                if (!tempMonth.equalsIgnoreCase(monthChanged))
                {
                    addMonths = addMonths + prevDay;
                    monthChanged = tempMonth;
                }
                prevDay = tempDay;
                tempHour = Integer.parseInt(line.substring(13, 15).trim());
                tempMin = Integer.parseInt(line.substring(16, 18).trim());
                tempAvailNodes = Integer.parseInt(line.substring(line.length()-4,
line.length()).trim());
                if (tempAvailNodes < availNodes)
                {
                    //that many nodes are no longer available
                    //quit and return the howlongNodesAreAvail
                    break;
                }
            }
        } catch (NumberFormatException e) {
            // TODO Auto-generated catch block
            System.out.println("*****returned a numberformatexception in get duration ***");
//e.printStackTrace());
            return -1;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            System.out.println("*****returned an IOException in get duration ***");
//e.printStackTrace();
            return -1;
        }

        if (addMonths != 0)
            howLongNodesAreAvail = ((((addMonths + tempDay - d) * 24 + tempHour) - h) * 60) +
(tempMin - m);
        else
            howLongNodesAreAvail = ((((tempDay - d) * 24 + tempHour) - h) * 60) + (tempMin -
m);
        return howLongNodesAreAvail;
    }

    public Integer getJobRunTime(Integer jobRunTimePerNode, Integer availNodes, Integer
nCores, Integer qSize, Integer dbSize)
    {
        Integer jobRunTime = 0;
        if (jobRunTimePerNode != 0)
        {
            jobRunTime = jobRunTimePerNode / availNodes;
        }
        else
        {
            Integer workerCores = (availNodes * nCores) - 2;
            //6 queries (qfilesize), per million items (dbfileSize), for 1 core, takes 1
minute.
            jobRunTime = 10 * (((qSize / 6) * (dbSize / 1000000)) / workerCores) + 10;
            //if jobRunTime is smaller than 10 minutes, set it to 10, cause it won't run
eitherwise.
            if (workerCores <= 32 && jobRunTime >= 30)
                jobRunTime = 0;
            if (jobRunTime < 10)
                jobRunTime = 10;
        }
        return jobRunTime;
    }

    void copy(File src, File dst) throws IOException {
        InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dst);

        // Transfer bytes from in to out
        byte[] buf = new byte[1024];
        int len;
        while ((len = in.read(buf)) > 0) {
            out.write(buf, 0, len);
        }
        in.close();
        out.close();
    }

    public Boolean evaluateGoal(GoalManager aGM, ControlComponent aCC, Integer j) throws
IOException {
        selectedPlan.setSelectedGoal(aGM.getSelectedGoal(aGM.getSelectedGoalID()));
        //check if there is a current selected plan
        //if there is a current selected plan, does it still meet the goal
        //if so, do nothing.
        //if the current selected goal does no longer meet the goal, reset it.
        Boolean recyclePlan = false;
        if (!selectedPlan.getSelectedHost().equalsIgnoreCase("xxx"))
        {
            if (selectedPlan.getSelectedGoal().getId().equals("nodecount") ||
selectedPlan.getSelectedGoal().getId().equals("x.nodecount"))
            {
                    if (selectedPlan.getNumberOfNodes() <
selectedPlan.getSelectedGoal().getMinNodes() ||
                        selectedPlan.getNumberOfNodes() >
selectedPlan.getSelectedGoal().getMaxNodes()
                        )
                        recyclePlan = true;
            }
            else if (selectedPlan.getSelectedGoal().getId().equals("time") ||
selectedPlan.getSelectedGoal().getId().equals("x.time"))
            {
                if (selectedPlan.getTotalRunTime() >
selectedPlan.getSelectedGoal().getDuration())
                    recyclePlan = true;
            }
            else if (selectedPlan.getSelectedGoal().getId().equals("cost") ||
selectedPlan.getSelectedGoal().getId().equals("x.cost"))
            {
                    if (aCC.getHosts().get(selectedPlan.getSelectedHost()).getCostPerNode() *
selectedPlan.getNumberOfNodes() < selectedPlan.getSelectedGoal().getMinCost() ||
                        aCC.getHosts().get(selectedPlan.getSelectedHost()).getCostPerNode() *
selectedPlan.getNumberOfNodes() > selectedPlan.getSelectedGoal().getMaxCost()
                        )
                        recyclePlan = true;
            }
            if (selectedPlan.getSelectedGoal().getId().equals("balance") ||
selectedPlan.getSelectedGoal().getId().equals("x.balance"))
            {
                if (selectedPlan.getTotalRunTime() >
selectedPlan.getSelectedGoal().getDuration() ||
                    selectedPlan.getNumberOfNodes() <
selectedPlan.getSelectedGoal().getMinNodes() ||
                    selectedPlan.getNumberOfNodes() >
selectedPlan.getSelectedGoal().getMaxNodes() ||
                    aCC.getHosts().get(selectedPlan.getSelectedHost()).getCostPerNode() *
selectedPlan.getNumberOfNodes() < selectedPlan.getSelectedGoal().getMinCost() ||
                    aCC.getHosts().get(selectedPlan.getSelectedHost()).getCostPerNode() *
selectedPlan.getNumberOfNodes() > selectedPlan.getSelectedGoal().getMaxCost()
                    )
                    recyclePlan = true;
            }
        }

        if (recyclePlan)
        {
            System.out.println("Recycling the plan, it no longer meets the goal...");
            selectedPlan = new PlanData();
            selectedPlan.setSelectedGoal(aGM.getSelectedGoal(aGM.getSelectedGoalID()));
            selectedPlan.setSelectedHost("xxx");
            selectedPlan.setPlanJobID(-1);
            selectedPlan.setNumberOfNodes(0);
            selectedPlan.setRuntime(500000);
            selectedPlan.setWaitTime(500000);
            selectedPlan.setTotalRunTime(selectedPlan.getRuntime() +
selectedPlan.getWaitTime());
            selectedPlan.setTotalCost(2000000);
        }

        for (String aHost : aCC.getHosts().keySet()) {
            String senseLockFileStr = aCC.getUnc_file_share() + aCC.getUnc_sense_share() +
aHost + ".lock";
            File senseLockFile = new File(senseLockFileStr);
            try {
                while (senseLockFile.exists())
                {
                    Thread.sleep(5000);  //5 seconds.
                    senseLockFile = new File(senseLockFileStr);
                }
                senseLockFile.createNewFile();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            String aFile = aCC.getUnc_file_share() + aCC.getUnc_sense_share() + aHost +
".sense.adapt";
            File aSenseFile = new File(aFile);
            String copyFile = aCC.getUnc_file_share() + aCC.getUnc_sense_share() + aHost +
".sense.adapt.copy";
            File aCopySenseFile = new File(copyFile);
            if (aSenseFile.exists())
            {
                copy(aSenseFile, aCopySenseFile);
                aCC.getHosts().get(aHost).setMetGoal(false);
                Boolean betterGoal = false;
                try {
                    RandomAccessFile raf = new RandomAccessFile(copyFile, "r");
                    String month;
                    String nowMonth;
                    Integer day = 0;
                    Integer nowDay = 0;
                    Integer hour = 0;
                    Integer nowHour = 0;
                    Integer minute = 0;
                    Integer nowMinute = 0;
                    Integer availNodes = 0;
                    Integer jobno = -1;
                    Integer i = 0;
                    Integer totalRunTime;
                    Integer totalCost = 2000000;
                    String monthChanged = "";
                    Integer addMonths = 0;
                    Integer prevDay = 31;
                    String line;
                    Long filePos;
                    totalRunTime = selectedPlan.getRuntime() + selectedPlan.getWaitTime();
                    try {
                        while (raf.getFilePointer() < raf.length())
                        {
                            line = raf.readLine();
                            if (line.contains(">") && !line.contains("Reservation"))
                            {
                                month = line.substring(6, 9);
                                day = Integer.parseInt(line.substring(10, 12).trim());
                                hour = Integer.parseInt(line.substring(13, 15).trim());
                                minute = Integer.parseInt(line.substring(16, 18).trim());
                                if (i == 0)
                                {
                                    nowMonth = month;
                                    monthChanged = month;
                                    nowDay = day;
                                    nowHour = hour;
                                    nowMinute = minute;
                                    i = 1;
                                }
                                availNodes = Integer.parseInt(line.substring(line.length()-4,
line.length()).trim());
                                //still need to look forward to see a better gap with possibly
more nodes
                                //but keep track of how long the wait is according to qtime...
                                //never schedule a job on one node
                                //always schedule five nodes less than what is available to
secure your spot cause
                                //there are nodes that are reserved for short jobs and won't
be picked up...

                                if (availNodes > 2 &&
                                (selectedPlan.getSelectedGoal().getId().equals("nodecount") ||
selectedPlan.getSelectedGoal().getId().equals("x.nodecount") ||
                                    selectedPlan.getSelectedGoal().getId().equals("balance")
|| selectedPlan.getSelectedGoal().getId().equals("x.balance")) &&
                                    availNodes >= selectedPlan.getSelectedGoal().getMinNodes()
&& availNodes <= selectedPlan.getSelectedGoal().getMaxNodes()
                                )
                                {
                                    availNodes = availNodes - 5;
                                    filePos = raf.getFilePointer();
                                    Integer howLongNodesAreAvailable;
                                    //always think there is 5 minute less time than what is
shown in qtime
                                    //the reason is it takes a while for moab to pick up the
new job
                                    //and by the time it picks up the new job, 2-5 minutes
passed already..
                                    //I am setting this to 10 minutes to be safe at this time.
                                    howLongNodesAreAvailable =
getHowLongNodesAreAvailable(raf, month, day, hour, minute, availNodes) - 10;
raf.seek(filePos);
                                    if (!month.equalsIgnoreCase(monthChanged))
                                    {
                                        addMonths = addMonths + prevDay;
                                        monthChanged = month;
                                    }
                                    prevDay = day;
                                    Integer waitTime;
                                    waitTime = ((((addMonths + day - nowDay) * 24 + hour) -
nowHour) * 60) + (minute - nowMinute);
                                    //can we run our job within this timeframe with the
availNodes?
                                    betterGoal = false;
                                    Integer myJobRunTime = 0;
                                    myJobRunTime =
getJobRunTime(aCC.getHosts().get(aHost).getJobRunTimePerNode().intValue(), availNodes,
aCC.getHosts().get(aHost).getNumCores(), selectedPlan.getSelectedGoal().getqFileSize(),
selectedPlan.getSelectedGoal().getDbFileSize());
                                    if (availNodes != 0 &&  myJobRunTime <=
howLongNodesAreAvailable && myJobRunTime != 0)
                                    {
                                        if
(selectedPlan.getSelectedGoal().getId().equals("nodecount") ||
selectedPlan.getSelectedGoal().getId().equals("x.nodecount"))
                                        {
                                            if (availNodes >=
selectedPlan.getSelectedGoal().getMinNodes() && availNodes <=
selectedPlan.getSelectedGoal().getMaxNodes())
                                            {
                                                aCC.getHosts().get(aHost).setMetGoal(true);
                                                if (myJobRunTime + waitTime <
selectedPlan.getTotalRunTime())
                                                betterGoal = true;
                                            }
                                        }
                                        else if
(selectedPlan.getSelectedGoal().getId().equals("time") ||
selectedPlan.getSelectedGoal().getId().equals("x.time"))
                                        {
                                            if (myJobRunTime + waitTime <=
selectedPlan.getSelectedGoal().getDuration())
                                            {
                                                aCC.getHosts().get(aHost).setMetGoal(true);
                                                if (myJobRunTime + waitTime <
selectedPlan.getTotalRunTime())                           betterGoal = true;
                                            }
                                        }
                                        else if
(selectedPlan.getSelectedGoal().getId().equals("cost") ||
selectedPlan.getSelectedGoal().getId().equals("x.cost"))
                                        {
                                            if (aCC.getHosts().get(aHost).getCostPerNode() *
availNodes >= selectedPlan.getSelectedGoal().getMinCost() &&
aCC.getHosts().get(aHost).getCostPerNode() * availNodes <=
selectedPlan.getSelectedGoal().getMaxCost())
                                            {
                                                aCC.getHosts().get(aHost).setMetGoal(true);
                                                if (aCC.getHosts().get(aHost).getCostPerNode()
* availNodes < selectedPlan.getTotalCost())
                                                betterGoal = true;
                                            }
                                        }
                                        if
(selectedPlan.getSelectedGoal().getId().equals("balance") ||
selectedPlan.getSelectedGoal().getId().equals("x.balance"))
                                        {
                                            if (myJobRunTime + waitTime <=
selectedPlan.getSelectedGoal().getDuration() &&
                                                availNodes >=
selectedPlan.getSelectedGoal().getMinNodes() && availNodes <=
selectedPlan.getSelectedGoal().getMaxNodes() &&
                                                aCC.getHosts().get(aHost).getCostPerNode() *
availNodes >= selectedPlan.getSelectedGoal().getMinCost() &&
aCC.getHosts().get(aHost).getCostPerNode() * availNodes <=
selectedPlan.getSelectedGoal().getMaxCost()
                                                )
                                            {
                                                aCC.getHosts().get(aHost).setMetGoal(true);
                                                if (myJobRunTime + waitTime <
selectedPlan.getTotalRunTime())
                                                    betterGoal = true;
                                            }
                                        }
                                    }

                                    //plan changes for the better
                                    //however if i have 5 minutes or less to run, do not
change the selected plans.
                                    if (betterGoal  && selectedPlan.getWaitTime() > 5)
                                    {
                                        System.out.println("found better plan: " + line);
                                        System.out.println("host:" + aHost);
                                        System.out.println("waitTime:" + waitTime);
                                        System.out.println("availNodes:" + availNodes);
                                        System.out.println("howLongNodesAreAvailable:" +
howLongNodesAreAvailable);
                                        Integer arzu = myJobRunTime + waitTime;
                                        System.out.println("totalRunTime:" + arzu);
                                        System.out.println("PreviousTotalRunTime:" +
totalRunTime);


                                        selectedPlan.setSelectedHost(aHost);
                                        selectedPlan.setWaitTime(waitTime);
                                        selectedPlan.setNumberOfNodes(availNodes);
                                        selectedPlan.setRuntime(myJobRunTime);
                                        totalRunTime = selectedPlan.getRuntime() +
selectedPlan.getWaitTime();
                                        selectedPlan.setTotalRunTime(totalRunTime);
                                        totalCost = availNodes *
aCC.getHosts().get(aHost).getCostPerNode();
                                        selectedPlan.setTotalCost(totalCost);
                                        //this.selectedPlanChanged = true;
                                    }
                                    /*
                                    day = tempDay;
                                    hour = tempHour;
                                    minute = tempMinute;
                                    availNodes = tempAvailNodes;
                                    */
                                }  //if availNodes is bigger than 1.
                            }  //if line contains '>'
```

```
                } //end of while  raf.getFilePointer() < raf.length()
            } catch (NumberFormatException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }  //end of while for file row iteration

            //src.close();
            try {
                raf.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            System.out.println("Cannot open source file, first run " + copyFile);
            //e.printStackTrace();
        } //end try for openning randomaccessfile.
    }  //if sense file exists;
    senseLockFile.delete();  //read from the sense file is done, release the resource.
}  //for each host loop

String actFileStr = aCC.getUnc_file_share() + aCC.getUnc_act_share() + "act.txt";
File actFile = new File(actFileStr);

if (selectedPlan.getSelectedHost().equalsIgnoreCase("xxx"))
{
    if (actFile.exists())
        actFile.delete();
    return false;
}
String aLockFile = aCC.getUnc_file_share() + aCC.getUnc_act_share() + "act.lock";
File actLockFile = new File(aLockFile);
actLockFile.createNewFile();

if (actFile.exists())
{
    String lFile = aCC.getUnc_file_share() + aCC.getUnc_act_share() + "act.log";
    File logFile = new File(lFile);
    FileInputStream in = new FileInputStream(actFile);
    FileOutputStream out = new FileOutputStream(logFile, true);

    int c;
    while ((c=in.read()) != -1)
        out.write(c);

    out.write('\n');
    in.close();
    out.close();
    actFile.delete();
}

Locale locale = Locale.US;
Date date = new Date();

FileWriter actWriter = new FileWriter(actFileStr);
String jobLine;
jobLine = selectedPlan.getSelectedHost() + ":" +
selectedPlan.getNumberOfNodes() + ":" +
      selectedPlan.getRuntime() + ":" +
      selectedPlan.getWaitTime() + ":" +
      aGM.getGoals().get(aGM.getSelectedGoalID()).getqFile() + ":" +
      aGM.getGoals().get(aGM.getSelectedGoalID()).getDbFile() + ":" +
      aGM.getGoals().get(aGM.getSelectedGoalID()).getParamsFile() + ":" +
      "jobid=" + selectedPlan.getPlanJobID() + ":" +
      "gID=" + selectedPlan.getSelectedGoal().getId() + ":" +
      "gMinNode=" + selectedPlan.getSelectedGoal().getMinNodes() + ":" +
      "gMaxNode=" + selectedPlan.getSelectedGoal().getMaxNodes() + ":" +
              "gMinCost=" + selectedPlan.getSelectedGoal().getMinCost() + ":" +
              "gMaxCost=" + selectedPlan.getSelectedGoal().getMaxCost() + ":" +
              "gDuration=" + selectedPlan.getSelectedGoal().getDuration() + ":" +
              DateFormat.getDateInstance(DateFormat.SHORT, locale).format(date) + ":" +
              DateFormat.getTimeInstance(DateFormat.DEFAULT, locale).format(date);
        actWriter.write(jobLine);
        actWriter.close();
        actLockFile.delete();
        if (selectedPlan.getWaitTime() > 0)
            selectedPlan.setWaitTime(selectedPlan.getWaitTime() - 1);
        else
            selectedPlan.setWaitTime(selectedPlan.getWaitTime());

        selectedPlan.setTotalRunTime(selectedPlan.getRuntime() + selectedPlan.getWaitTime());
        return true;

    }  //end of evaluatePlan function

}  //end of class
```

GoalManager.java

```
package adaptive;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.Properties;

public class GoalManager {
   private HashMap<String, GoalData> goals = new HashMap<String, GoalData>();
   private String selectedGoal;
   public HashMap<String, GoalData> getGoals() {
      return goals;
   }
   public void setGoals(HashMap<String, GoalData> goals) {
      this.goals = goals;
   }
   public String getSelectedGoalID() {
      return this.selectedGoal;
   }
   public void setSelectedGoalID(String selectedGoal) {
      this.selectedGoal = selectedGoal;
   }
   public GoalData getSelectedGoal(String selectedGoal) {
      return this.goals.get(selectedGoal);
   }
   public Boolean updateGoal(Boolean canAchieveGoal)
   {
      Properties global = new Properties();
      try {
         FileInputStream fin = new FileInputStream("adaptive.properties");
         global.load(fin);
         fin.close();
         if (canAchieveGoal)
         {
            goals.clear();
            int goalCount = Integer.parseInt(global.getProperty("goal.count").trim());
            int i = 0;
            for(i=1; i <= goalCount; i++)
            {
               GoalData aGoalData = new GoalData();
               aGoalData.setId(global.getProperty("goal" + Integer.toString(i) + ".id"));
               aGoalData.setDuration(Integer.parseInt(global.getProperty("goal"              +
Integer.toString(i) + ".duration").trim()));
               aGoalData.setMinCost(Integer.parseInt(global.getProperty("goal"             +
Integer.toString(i) + ".cost.min").trim()));
               aGoalData.setMaxCost(Integer.parseInt(global.getProperty("goal"             +
Integer.toString(i) + ".cost.max").trim()));
               aGoalData.setMinNodes(Integer.parseInt(global.getProperty("goal"            +
Integer.toString(i) + ".nodes.min").trim()));
               aGoalData.setMaxNodes(Integer.parseInt(global.getProperty("goal"            +
Integer.toString(i) + ".nodes.max").trim()));
               aGoalData.setqFile(global.getProperty("QFile").toString().trim());
              aGoalData.setqFileSize(  Integer.parseInt(global.getProperty("QFileSize").trim())
);
               aGoalData.setDbFile(global.getProperty("DBFile").toString().trim());
               aGoalData.setDbFileSize(                                       Integer.parseInt(
global.getProperty("DBFileSize").trim() ) );
               aGoalData.setParamsFile(  global.getProperty("ParamsFile").toString().trim()  );

               aGoalData.setJobName(global.getProperty("jobName").toString().trim());
               goals.put(aGoalData.getId(), aGoalData);
            }

            String selectedID;
            if (global.getProperty("goal.selected.id").startsWith("x"))
               selectedID   =   global.getProperty("goal.selected.id").toString().replace("x.",
"");
            else
```

```
               selectedID = global.getProperty("goal.selected.id").toString();

            setSelectedGoalID(selectedID);
         }
         else
         {
            if (!this.selectedGoal.startsWith("x"))
            {
               this.selectedGoal = "x." + this.selectedGoal;
               global.setProperty("goal.selected.id", this.selectedGoal);
               FileOutputStream fout = new FileOutputStream("adaptive.properties");
               global.store(fout, null);
               fout.close();
            }
         }

      } catch (IOException e) {
         System.out.println("Cannot Read/Write the adaptive.properties file");
      }
      return true;
   }

   public    Boolean    changeGoal(String    kbaseHost,    Integer    kbaseWaitTime,    Integer
kbaseNumberOfNodes, Integer kbaseRunTime) {
      updateGoal(false);
      return true;
   }
   public void initialize() {
      updateGoal(true);
   }
}
```