Methods Toward Automatic Configuration of Computing Environments for Application Execution

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy

Computer Science

by

Karolina Sarnowska-Upton

May 2013

Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science

Karolina Sarnowska-Upton (Author)

This dissertation has been read and approved by the examining committee:

Andrew Grimshaw (Advisor)

John Knight (Chair)

Paul Reynolds

Wesley Weimer

John Hawley

Accepted for the School of Engineering and Applied Science:

James H. Aylor (Dean, School of Engineering and Applied Science)

May 2013

Abstract

Computing resources are now ubiquitous and computational research techniques permeate all disciplines. However, exploiting available resources can be a much more complicated proposition. There is no guarantee that one can simply use a compute resource with no more effort than copying binaries and data. As computing resources are usually heterogeneous in both hardware and software configurations, many requirements be matched to execute a computation on a new resource in a new environment. The difficulties increase when dealing with parallel computations, which add a layer of dependencies related to the Message Passing Interface (MPI) standard libraries.

Unfortunately, managing the migration process by using existing techniques is inadequate or requires a non-trivial amount of effort and experience. In particular, schedulers are not generally designed to capture a computation's software-related requirements and, thus, depend on users to configure such dependencies. Additionally, the set of possible sites where computations could be scheduled is limited to where the computations are known to be able to run – a determination that in the current state of the art is performed manually by the user. This process, which requires enumerating dependencies, checking and making them available in new environments, and potentially recompiling the computation, can take many hours of labor. The difficulty is compounded by the fact that many researchers in disciplines that were previously not traditionally compute-heavy may not have experience with configuring a single environment, let alone with migrating a computation from one environment to another.

An ideal solution for providing deployment and, therefore, scheduling freedom would allow any computation to quickly and easily be run on computing resources with tuned performance. Before addressing the difficult but secondary issues of automatic recompilation and tuning, the first

step on the path toward an ideal solution is to consider how additional scheduling freedom could be achieved with minimum interaction from the user but without modification of the computation code. Such a solution would automatically enable application binaries to quickly and easily be run on new resources. Our hypothesis is that methods for automatically gathering information about execution requirements and composing site-specific instructions that configure the requirements at target environments are more efficient than manual methods for the preparation of multiple shared computing environments for the execution of MPI binaries. Achieving this initial solution alone can dramatically improve the ability of researchers to take advantage of the variety of computing resources available to them and, as a result, carry out more and better research.

Due to the additional requirements that arise when using MPI, our research specifically focuses on enabling the deployment and, therefore, scheduling freedom of parallel computations encoded using the MPI standard on high performance computing clusters. Specifically, to determine if a binary will be able to run without modification, how to form predictions about execution readiness was modeled and what execution-blocking issues could be resolved without recompilation was assessed. The effectiveness of these methods was examined by testing their implementation. The assumed difficulty of the migration process was also investigated by measuring how long researchers take to get computations running at new computing sites. This baseline was used to quantify the cost savings of the presented solution in terms of time.

The work presented in this dissertation is a first step toward an ideal solution of automatically enabling the usage of various computing resources for computation. The evaluation demonstrated the validity of the solution by providing correct predictions of execution readiness more than 90% of the time and enabling 41% more successful executions via generation of site-specific configurations. The effort analysis in terms of time exerted to use the solution predicts that the solution is, in the best case, an order of magnitude more efficient over current manual methods and, in the worst case, no less efficient.

Acknowledgments

I have completed this dissertation with the support of a village of individuals. I would like to thank everyone who was part of the process. In particular:

To my advisor Andrew Grimshaw: thank you for all your invaluable teaching and guidance. From teaching me parallel programming and grid computing to guiding me toward becoming a researcher and professional, it's been a pleasure working with you.

To my husband Dan: thank you for your patience, encouragement, understanding, and love.

To my parents, Beata Sarnowska and Krzysztof Sarnowski and grandparents, Barbara and Adam Habdas: thank you for your unconditional love, guidance, and support and for always encouraging me to push on.

To John Knight, Paul Reynolds, Malathi Veeraraghavan, and Mitch Rosen: thank you for your mentoring.

To the Grimshaw research group and UVACSE team: thank you for your constructive criticisms and support.

To all the many friends I made in Charlottesville: thank you for helping to make my time in graduate school so enjoyable. A special thanks to Tom Tracy and Chih-hao Shen for always being willing to lend a helping hand.

I greatly appreciate everyone's support. Thank you all again!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As computing advances have kept up with the pace predicted by Moore's Law, computing resources have become ubiquitous. Simultaneously, computational research techniques now permeate all disciplines, from engineering and the sciences to economics and literature. As a result, researchers are surrounded by computational resources, from laptops and department machines to university clusters and national supercomputers. However, even though today's computing infrastructures, such as the Extreme Science and Discovery Environment (XSEDE), provide researchers with easy access to a wide variety of computing resources, exploiting available resources can be a much more complicated proposition.

In order to use various resources, computations must be able to execute at different sites. However, there is no guarantee that one can simply move from one resource to another with no more effort than copying binaries and data. Computing resources are usually heterogeneous in both hardware and software configurations. Even when keeping the processor and operating system consistent, the environment, run-time libraries, and third-party software are likely to vary between resources. Since computations often depend on the availability of such libraries and software, these configurations must be matched to execute a computation on a new resource in a new environment.

The difficulties with exploiting resources increase when dealing with parallel computations. When researchers want to maximize performance, they often turn to parallel computing to perform more computations in a given amount of time or to perform a given amount of computation in less time. Typically, parallel computations employ the Message Passing Interface (MPI) standard,

the standard for running parallel programs on distributed memory systems. Indeed, many community applications use MPI to enable more efficient larger scale experiments. Preparing to run computations that use MPI adds a layer of dependencies related to the MPI libraries. Additionally, virtualization technologies are especially inadequate for dealing with MPI parallel codes due to the associated performance overhead.

A variety of tools and approaches exist to help deploy computations across various resources. In particular, schedulers enable automatic selection of resources for running computations based on user-specified requirements and preferences. Schedulers accomplish this by first assessing a set of *candidate hosts* that meet the specified requirements and then scheduling the computation on one of these matching hosts. Schedulers may also prepare the site for execution by staging in files and data. However, schedulers are generally not designed to capture a computation's software-related requirements and, thus, depend on users to configure such dependencies. Additionally, the set of possible sites where computations could be scheduled is limited to where the computations are known to be able to run – a determination that in the current state of the art is performed manually by the user. This process, which requires enumerating dependencies, checking and making them available in new environments, and potentially recompiling the computation, can take many hours of labor. The difficulty is compounded by the fact that many researchers in disciplines that were previously not traditionally compute-heavy may not have experience with configuring a single environment, let alone with migrating a computation from one environment to another. Unfortunately, managing the migration process by using other existing techniques, described in Chapter 7, also requires a non-trivial amount of effort and experience on the part of the researcher. Thus, while more and faster resources should allow for more and better research to be carried out, the increase in resources can just as easily stymie progress.

To illustrate the impact of the gap between accessible and exploitable resources on deployment and, thus, also scheduling freedom, Figure 1.1 presents three diagrams. The diagrams depict the difference in size between the set of accessible and exploitable resources. In the initial case (depicted in diagram (a)), the set of resources usable for application execution (i.e. the candidate host) is just the resource on which an application currently runs. To approach the ideal case (depicted diagram

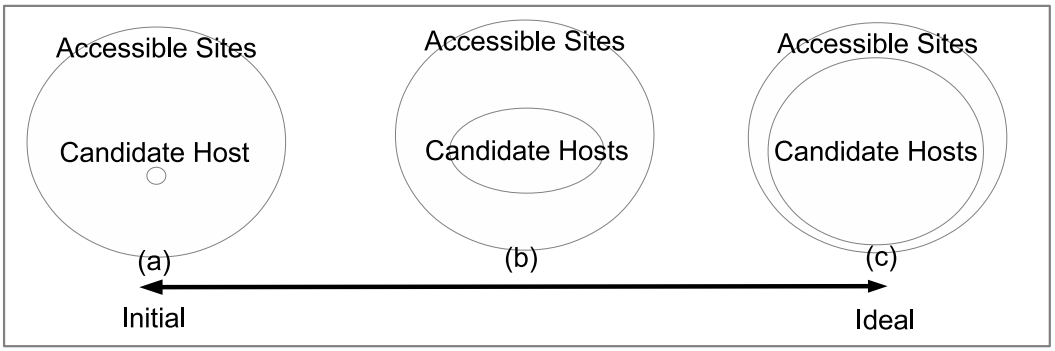


Figure 1.1: Spectrum of scheduling freedom. These three diagrams depict the difference in size between the set of accessible and exploitable resources. Initially – as shown in (a) – the set of exploitable resources only contains the initial site where a computation is run. There may be many more accessible sites but without configuration, they are not ready to run the computation. Ideally – as show in (c) – the set of exploitable resources would coincide with the set of accessible resources. One of the goals of this work was to explore automated methods to expand the candidate host set from (a) to (b).

(c)) where the set of exploitable resources approaches the set of all accessible resources, currently the user must manually prepare each execution site for execution. As a result, job management systems can also only schedule applications to run at sites prepared by the user. Thus, deployment and scheduling freedom is limited until the complications associated with exploiting resources are handled. A solution that could relieve some of this effort from having to be performed manually would more efficiently approach the ideal case and increase the number of resources where a job could be executed to enable greater scheduling freedom (depicted in diagram (b)).

An ideal solution for providing deployment and, therefore, scheduling freedom would allow any computation to quickly and easily be run on computing resources with tuned performance. This would involve enabling a computation to run on resources with minimal interaction from the user and running a version of the code tuned to perform well at each particular site. Reaching the ideal is limited by the ability to automatically recompile and restructure source code to create tuned binaries. Before addressing these difficult but secondary issues, the first step on the path toward an ideal solution is to consider how additional scheduling freedom can be achieved with minimum interaction from the user but without modifying the computation code. Such a solution would automatically enable application binaries to quickly and easily be run on new resources. *Our*

hypothesis is that methods for automatically gathering information about execution requirements and composing site-specific instructions that configure the requirements at target environments are more efficient than manual methods for the preparation of multiple shared computing environments for the execution of MPI binaries. Achieving this initial solution alone would dramatically improve the ability of researchers to take advantage of the variety of computing resources available to them and, as a result, carry out more and better research.

Due to the additional requirements that arise when using MPI, our research specifically focuses on enabling the deployment and, therefore, scheduling freedom of parallel computations encoded using the MPI standard across high performance computing clusters. Specifically, to determine if a binary will be able to run without modification, how to form predictions about execution readiness was modeled and what execution-blocking issues could be resolved without recompilation was assessed. The effectiveness of these methods was examined by testing their implementation. The assumed difficulty of the migration process was also investigated by measuring how long researchers take to get computations running at new computing sites. This baseline was used to quantify the cost savings of the presented solution in terms of time.

1.1 Contributions

This dissertation is a first step toward an ideal solution of automatically enabling the usage of computing resources for computation. The four specific contributions of this dissertation are summarized in this section.

1.1.1 Investigating Migration Effort

Experiences with users and administrators of shared computing resources revealed that the challenges encountered when running computations on new resources can be a barrier to usability. However, there was no experimental data documenting the effort that users put forth migrating their computations to new computing sites. *The first contribution of this dissertation is a user study designed and carried out to quantify in terms of time the scale of the challenge that researchers face*

when attempting to use new resources. The study provided a baseline of the time required for the current manual migration approach carried out before new resources can be used directly or via schedulers to run computations.

The study results emphasize the time consuming nature of preparing to run computations on multiple new resources and the extra challenge encountered by less experienced users. It took the 25 participants on average 2.5 hours spread over three days to migrate one computation to a new resource. The majority of the time was spent on learning, compiling, and debugging tasks. Less experienced users took on average over three weeks to migrate their MPI applications to four new computing sites. This was almost 50% longer than the results of expert participants.

1.1.2 Modeling Execution Readiness

The first step to using a new resource – whether directly or via a scheduler – involves learning about the configurations of the new computing site. Indeed, this behavior was exhibited by all participants in the user study. Once compatible settings for running a computation are discovered, the computation can either be submitted for execution or, if source code is available, can first be recompiled at the new site before attempting execution. Compilation takes more effort than simply running a binary. However, if a binary is not well matched for a computing site, uncovering the issues by debugging, often cryptic, output errors can be as time consuming as setting and debugging compilation configurations. Thus, the cost of simply finding out whether a computation will run on new resources can discourage utilization of otherwise easily accessible sites. If it were known whether recompilation will be needed before execution, researchers and schedulers could better choose between multiple new computing locations and computations could be run more quickly at sites where recompilation is not required. *The second contribution of this dissertation is a model for predicting whether an MPI binary is ready to execute at a computing site.*

The model is based on examining four factors - MPI stack, shared libraries, hardware architecture, and system configuration - related to MPI binaries and computing environments. The model describes the relevance of these factors and outlines - independently of implementation and without running the binary - what information needs to be gathered to make a determination about execution

readiness.

1.1.3 Implementing Execution Prediction

To test the execution prediction model, an implementation called FEAM (a Framework for Efficient Application Migration) was created for Unix-based computing environments. Unix-based environment were chosen as the focus as these dominate shared computing clusters and are most commonly used for running computations parallelized with MPI. FEAM uses Unix-based tools to gather information about the execution factors identified by the model. In addition to predicting execution readiness, the gathered information is used to compose site-specific configurations that incorporate a scheme for resolving execution-blocking issues. *The third contribution of this dissertation is FEAM, a tool that automates the prediction of execution readiness and composition of site-specific configurations.* FEAM allows researchers and schedulers to increase scheduling freedom by automatically determining whether sites are ready to execute MPI binaries.

1.1.4 Evaluation Techniques

The effectiveness of the proposed solution was determined by evaluating the implementation using multiple criteria. The accuracy of FEAM's predictions and resolution techniques was measured. FEAM's time and space requirements were also calculated. Finally, how much more efficiently FEAM enables the preparation of environments for application execution was analyzed by comparing the effort to use FEAM versus manual methods. *The fourth contribution of this dissertation is an evaluation of the model for predicting execution readiness and of the methods for composing site-specific configurations.*

The evaluation demonstrated the validity of the execution prediction model with FEAM correctly predicting over 90% of execution failures. The resolution scheme composed by FEAM was found to automatically enable 41% more successful executions. Overall, running FEAM was assessed to require minimal disk space and compute time. Finally, the analysis of the effort required to use FEAM found that utilizing FEAM is, in the best case, an order of magnitude more efficient over manual methods and, in the worst case, no less efficient. The efficiency analysis put into perspective

the savings in terms of time associated with using FEAM. The analysis simultaneously captures by how much using FEAM decreases the barriers that otherwise must be manually performed before new computing sites can be used to execute applications.

1.2 Outline

The remainder of the dissertation is organized as follows. Chapter 2 presents the design and results of the study of the process of migrating MPI applications to multiple new computing sites. Chapter 3 presents the model for predicting whether an MPI binary can execute on a resource without recompilation. Chapter 4 describes FEAM, a UNIX-based implementation of the model that incorporates resolution of execution-blocking issues. Chapters 5 and 6 present the evaluations of the solution in terms of accuracy, effectiveness, overhead, and efficiency. Chapter 7 discusses related work. Finally, Chapter 8 offers conclusions and presents directions for future work.

Chapter 2

The Study

This chapter presents a user study of the process of migrating MPI applications. The goal of the study was to gather experimental data about the process that researchers go through to run MPI applications at new computing sites. The motivation for the study was to quantify in terms of time the scale of the challenge that researchers face when attempting to use new resources so that this baseline data could be used for the evaluation of the proposed solution. The study additionally identified how much time researchers spent performing various types of tasks and how the amount of their computational experience affected migration time. The study results confirmed that migrating to multiple new computing sites is time consuming and that experience decreases migration time. To our knowledge, this is the first study of this kind conducted on supercomputing resources. The study results serve as a baseline for the time required (also referred to as the duration) for the process of migrating MPI applications. The study design serves as a template for other studies with large time commitments averaging tens of hours spread over many days per participant.

The study design is discussed in Section 2.1. Study participants and migrated MPI applications are profiled in Sections 2.2 and 2.3. Migration results are presented in Section 2.4.

2.1 Study Design

This section presents the various aspects of the study design. The targeted participants and migration targets are described. The quantities the study was designed to measure are discussed. The

protocol for each of the four study phases is outlined along with the compensation scheme used to incentivize study participation and completion.

2.1.1 Participant Selection

In order to also study the impact of experience on migration time, the aim was to recruit participants with different levels of computing and MPI experience. Thus, recruitment for the study targeted undergraduate and graduate students as well as professors and research scientists. The only prerequisite for participants was that they be end-users of an MPI application. This did not have to be an application they had authored but it did need to be a non-toy application that they had experience executing. By already being users of MPI applications, participants had access to a familiar application to migrate for the study. Also, by having previously run an MPI application, participants were guaranteed to have the basic computing skills to complete the tasks associated with the study.

To increase the likelihood of finding participants with various computing backgrounds, recruiting was done not only at the University of Virginia. The study was also announced to the national computational science community via the XSEDE Campus Champions and XSEDE Student networks [19]. This recruiting tactic also geographically diversified the participant pool.

In order to produce meaningful results, the aim was to recruit at least 25 participants. As summarized by Macefield, power analyzes of comparative studies recommend using 8 to 25 participants to likely produce statistically significant findings [36]. The study's estimated time commitment of at least three to five hours distributed over multiple days was anticipated to be a limiting recruiting factor. Thus participation was incentivized with a compensation scheme commensurate to the study's time commitment. The solicitation for participation received over 50 responses, 34 of which met the screening criteria of being MPI end-users of real applications. Of these 34 eligible responses, 29 agreed to participate in the study while 25 actually completed the study.

Table 2.1: Migration site characteristics. This table lists basic information about the six XSEDE computing sites used as migration destinations for the study.

SYSTEM NAME	INSTITUTION where located	CORES	OS
Blacklight	Pittsburgh Supercomputing Center	4,096	SUSE ES
Forge	National Center for Supercomputing Applications	576	RedHat ES
Gordon Compute	San Diego Supercomputing Center	16,384	Linux/Rocks
Kraken	National Institute for Computational Sciences	112,896	Cray Linux
Ranger	Texas Advanced Computing Center	62,976	CentOS
Steele	Purdue University	6,496	RedHat ES

2.1.2 Migration Resources

While participants selected the MPI applications to migrate during the study, all participants used the same migration destinations. For easier management of access to multiple resources, the set of migration destinations was composed of computing sites from the XSEDE infrastructure, which includes resources at over ten supercomputing centers accessible via a web-based access point [21]. Computing sites were chosen from different institutions to create a mix of diverse computing environments. The study was awarded computing cycles on six XSEDE resources: PSC's Blacklight [1], NCSA's Forge [2], SDSC's Gordon Compute [6], NICS's Kraken [8], TACC's Ranger [15], and Purdue's Steele [14]). The basic characteristics of each site are listed in Table 2.1. To use these resources, participants applied for XSEDE Portal accounts and then managed access to all six resources via one central administration point.

2.1.3 Measurement Gathering

The goal of the study was to investigate the effort involved in migrating MPI applications and the impact of experience on the process. The study was designed to quantify migration effort in terms of time while assessing participants' experience. Correlations between levels of experience and migration time were investigated along with any other characteristics shared by participants with similar migration times per site. This section discusses how the study was designed to collect these quantities.

MIGRATION LOG

```
TIME (minutes & date)
====

TYPE (consult list)
====

DESCRIPTION (provide a clear idea of what you were doing)
===========

RATINGS (1 low - 5 high)
=======
Difficulty:
Tediousness:
```

Figure 2.1: Migration log template. Study participants documented the tasks they performed while migrating MPI applications using the template outlined in this figure.

Migration Task Categories

```
LEARNING: XSEDE portal, new computing site
COMPILING: application
DEBUGGING: errors related to compilation/execution
ENVIRONMENT SETTING: setting modules/compilers/libraries/variables
REQUESTING ASSISSTANCE: colleagues, online help
SUBMITTING: creating submit script, submitting jobs
TESTING: application, submission script
OTHER: explain
```

Figure 2.2: Migration task categories. Study participants used the types in this figure to categorize the migration tasks they documented in log entries.

2.1.3.1 Migration Durations

To measure migration time, participants documented the migration process. Participants created log entries about the activities they performed during migration. An example log entry is shown in Figure 2.1. Each log entry described performed tasks, categorized tasks (according to the types in Figure 2.2), and documented the time spent on tasks. Log entry descriptions were used to confirm that the selected activity type(s) matched the performed tasks. The logged activity descriptions were used to identify the common tasks involved in the migration process. The logged timing information was used to determine how long participants took to complete migration tasks. For each log entry, participants also rated the perceived tediousness and difficultly of the associated tasks. The ratings were used to identify particularly difficult and tedious tasks.

Table 2.2: Experience categories. This table presents the guidelines used to categorize participants' experience level (as *advanced* or *novice*) based on reported computational experiences.

EXPERIENCE CATEGORY	YEARS of COMPUTATIONAL RESEARCH EXPERIENCE	NUMBER of COMPUTING SITES where MPI USED
Advanced	3+	4+
Novice	1-2	1-3

2.1.3.2 Computational Experiences

To quantify computational experience, participants completed an evaluation of as well as self-assessed their abilities. A questionnaire (located in Appendix A.2) asked information such as since when, how often, and how confidently participants performed various computational tasks. A test (located in Appendix A.5) asked multiple choice questions about MPI, queuing systems, and running jobs.

Based on the reported experiences, participants were categorized as having different levels of experience as outlined in Table 2.2. Participants were classified as having an *advanced* level of experience if they had at least three years of computational research experience and had run MPI applications at more than three computing sites. The participants who did not meet this criteria were classified as a *novice* level of experience. The threshold of three years and four computing sites was chosen to ensure familiarity with running MPI computations in various environments while having experienced the changes that occur as machines and MPI changes over a multiple year period.

The scheme for categorizing participant's levels of experience resulted in nine advanced and sixteen novice participants. Only these categories were used for grouping of participants in order to still extract likely statistically significant findings from the subgroups per Macefield's recommendation of having subgroups of size no less than eight [36].

2.1.3.3 Other

The study did not only measure migration times and assess participants' computing experiences. It also collected background information about participants such as their place of employment,

gender, and job title. Participants described the application they migrated in terms of qualities such as what it computes, who created it, what MPI type it uses, and what programming language it was written in. This information was used to create general profiles of participants and applications.

2.1.4 Phase Protocol

The study protocol was designed as a sequence of four phases to be completed using any computer with an internet connection. Participants were emailed instructions as they progressed through each phase at their own pace. This section describes the tasks performed during each phase. (All described study materials were distributed and returned as email attachments. They appear in the appendix.)

2.1.4.1 Phase One: Experience Assessment

The goal of the first phase of the study was to gather information about participants' computational experiences and their MPI application. Participants self-assessed their experiences in one questionnaire and described the MPI application they would be migrating in a second questionnaire. Participants also took a timed pretest about their computational experiences.

2.1.4.2 Phase Two: Migration Logging

The goal of the second phase of the study was to document the migration process as described in Section 2.1.3. Participants were asked to migrate their MPI applications to at least two computing sites. Participants choose how many migrations to perform from the set of available computing sites described in Section 2.1.2. To mitigate fatigue effects, participants were restricted to attempting at most two migrations per day. To investigate whether earlier migrations took participants more time to complete than later migrations independent of migration location, participants were asked to migrate to sites in a predefined order that was randomly selected for each participant.

2.1.4.3 Phase Three: Study Reflections

The goal of the third phase of the study was to assess participants' migration experiences and gather background information about participants. Participants provided this information in two questionnaires. Additionally, participants were asked to take a posttest that evaluated the same concepts as the pretest to determine whether participants learned any of the tested concepts while completing the study.

2.1.4.4 Phase Four: Concluding Discussion

The goal of the final phase of the study was to hold a concluding discussion with participants. The discussion provided an opportunity to ask tailored questions specific to participant experiences as well as any additional questions added to the study as it progressed. The concluding discussion was held in person, on the phone, or using video conferencing software.

2.1.5 Compensation

Participants were incentivized to complete the study with a compensation scheme that increased with the amount of time spent on the study. E-readers, a popular consumer electronic of the time, were used as the compensation prize. The quality of the e-readers that participants received was increased with the number of migrations completed and the time spent on the study. For example, participants were eligible to receive an e-reader worth $70 for completing two successful migrations or an e-reader worth $200 for completing five successful migrations. The four levels of compensation are outlined in Figure 2.3.

A budget of $5,000 was allocated for compensation prizes to target at least 25 participants. As not all participants completed the maximal number of migrations, the final cost of the compensation prizes was $3670. With a total of 218 hours of documented migrations plus at least an additional hour spent by each participant filling out the questionnaires and tests, the study compensation equated to around $15 per hour.

COMPENSATION OPTIONS

```
Definitions
 Site: resource where application has not previously been run
 Migration Success: application runs correctly
 Migration Attempt: at least 2 hours spent trying to run application

BASIC: successful migration to 2 sites or attempts to run at 4 sites
=====
  Amazon Kindle, Amazon Kindle Touch, or Barnes & Noble NOOK Simple Touch

MID: successful migration to 3 sites or attempts to run at 5 sites
===
  Amazon Kindle Keyboard 3G or Barnes & Noble NOOK Simple Touch with GlowLight

HIGH: Successful migration to 4 sites or attempts to run at 6 sites
====
  Amazon Kindle Touch 3G or Barnes & Noble NOOK Color

FULL: Successful migration to 5 sites or attempts to run at 7 sites
====
  Amazon Kindle Fire or Barnes & Noble NOOK Tablet
```

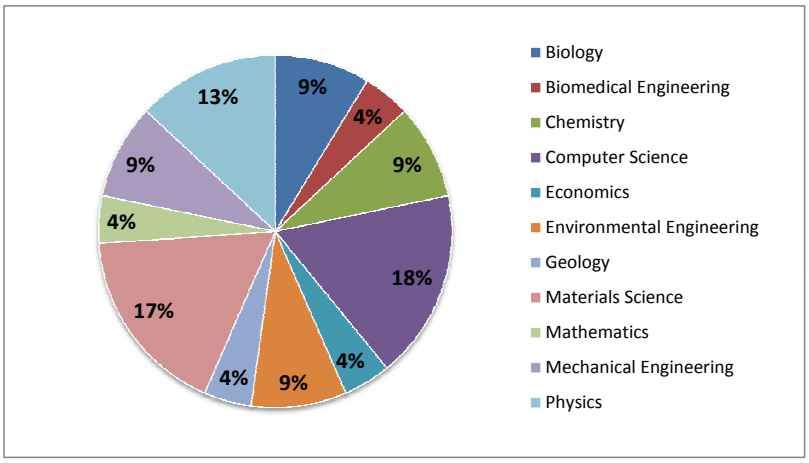
Figure 2.3: Study compensation options. This figure outlines the compensation options that participants were offered for completing the study. Participants were offered compensation at one of four levels depending on the number of successful migrations and migration attempts (of at least two hours). For example, a participant who completed three successful migrations was able to choose a prize at the "mid" (or lower) compensation level.

2.2 Participant Profiles

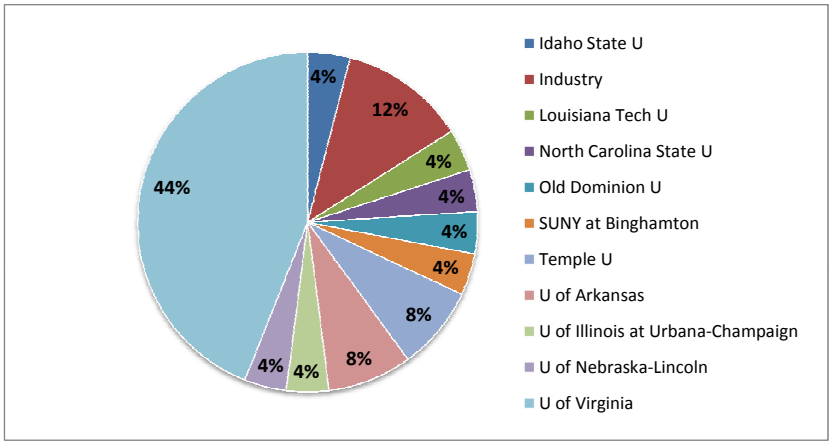
This section highlights information about study participants. The participants are profiled in terms of their academic backgrounds, computational experiences, and views about migration. The full set of results is available the appendix.

2.2.1 Background

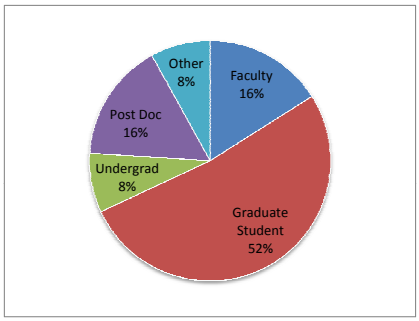
Figure 2.4 highlights participants' general background information summarizing their research areas, places of employment, job titles, ethnicity, and gender. Participants represented 11 disciplines at 13 institutions across 11 states. 44% of the participants were students and faculty at the University of Virginia. Participants consisted of a mix of undergraduate and graduate students as well as professors and scientists with over half of the participants being graduate students. The demographics of the participants were representative of the computing field in general with the majority being male and of white or Asian ethnicity.



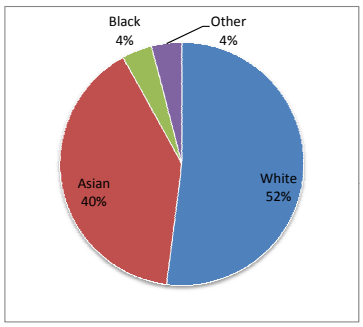
(a) Research Areas



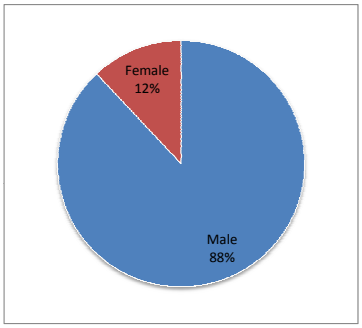
(b) Places of Employment



(c) Job Title



(d) Ethnicity



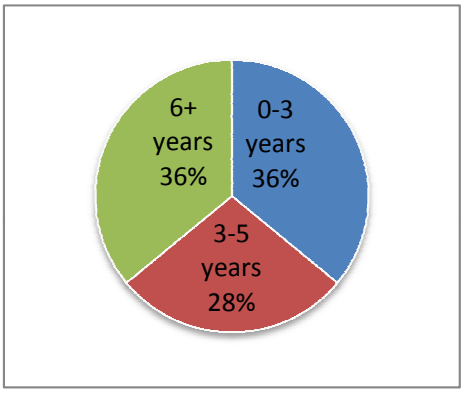
(e) Gender

Figure 2.4: Profile of participants. These figures profile general background information about the participants. Graph (a) presents the participants' areas of research, graph (b) presents the participants' places of employment, graph (c) presents the participants' job titles, graph (d) presents the participants' ethnicities, and graph (e) presents the participants' gender. The quantities are presented as percentages of the total number of participants.

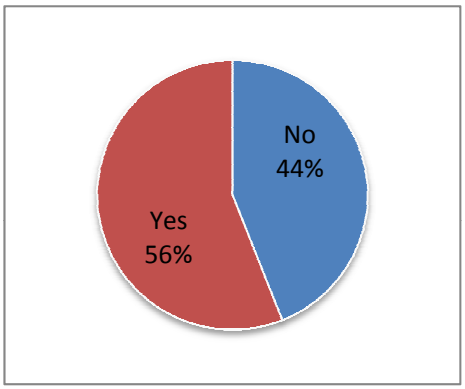
2.2.2 Experiences

Figure 2.5 highlights participants' computational experiences summarizing how participants learned to do computational research, for how many years they have been doing computational research, and whether they had used XSEDE resources previously as well as how many computing sites and what MPI implementations participants use for their research. Participants were well distributed in the number of years of experience they had in doing computational research. About a third reported having less than three years of experience, another third between three and five years of experience, and the remaining third having more than five years of experience. Similarly, participants were well distributed in the number of computing sites they had used for their research. About a third of participants had used only one site, another third used two to three sites, and the remaining third used four or more sites. Only half of participants reported having learned to do computational research in a formal course setting. The majority of participants reported that they had taught themselves, learned informally from others, and consulted online documentation. MPICH was the most widely used MPI implementation with 88% of participants reporting having used this implementation. Around half of the participants had previously used XSEDE resources. One third of the participants had no coding experience.

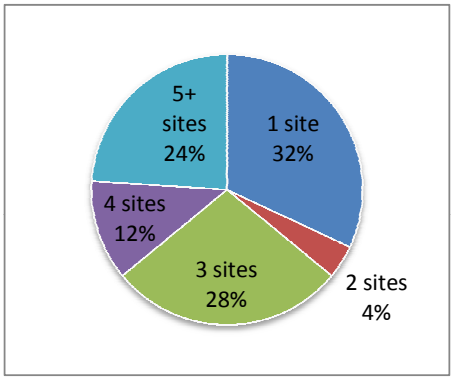
Figure 2.6 summarizes the participants' test scores. Average scores overall and by problem type are presented for all participants and for participants with different experience levels. Regardless of the type of question, the highest scorers were the advanced participants who had more years of experience with computational research and reported being more comfortable performing tasks such as compiling and running parallel applications. On retaking the same test after completing their migrations, participants corrected about a third of the problems they had initially missed. This result suggests that completing the study helped participants learn or relearn some knowledge related to running computations. By examining the answers to questions on practical concepts relating to using shared computing systems, it was found that many of the novice participants were not familiar with the *NIX tools to gather information relevant for execution.



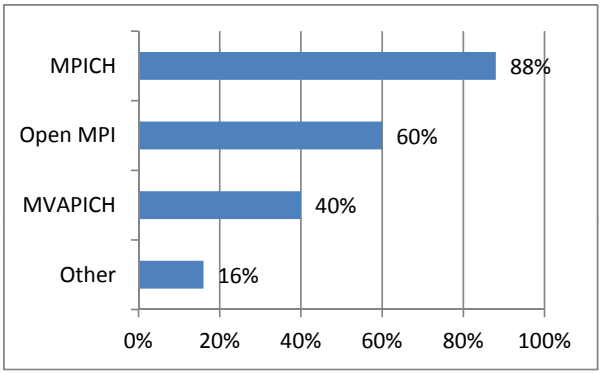
(a) Years of Computing Experience



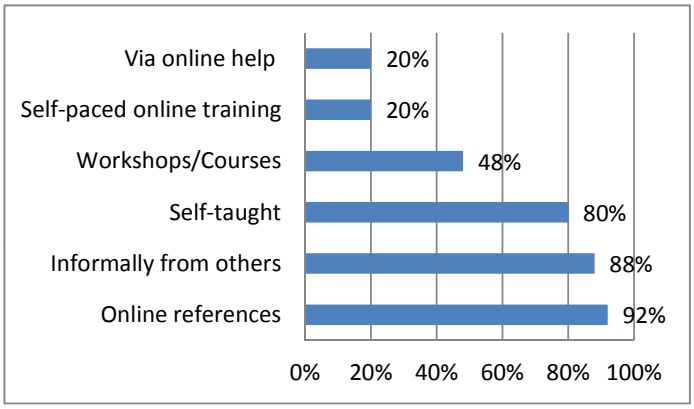
(b) Previous Experience with XSEDE



(c) Number of Computing Sites Used



(d) MPI Implementation Types Used



(e) Computational Learning Methods

Figure 2.5: Computational experiences. The graphs highlight five dimensions of participants' computational experiences. Graph (a) presents the participants' years of computing experience. Graph (b) presents whether participants had previous experience with XSEDE resources. Graph (c) presents the number of sites participants had experience using for computation. Graph (d) presents the MPI implementation types that participants had experience using to run applications in parallel. Graph (e) presents how participants had learned about computation. All quantities are presented as percentages of the total number of participants.

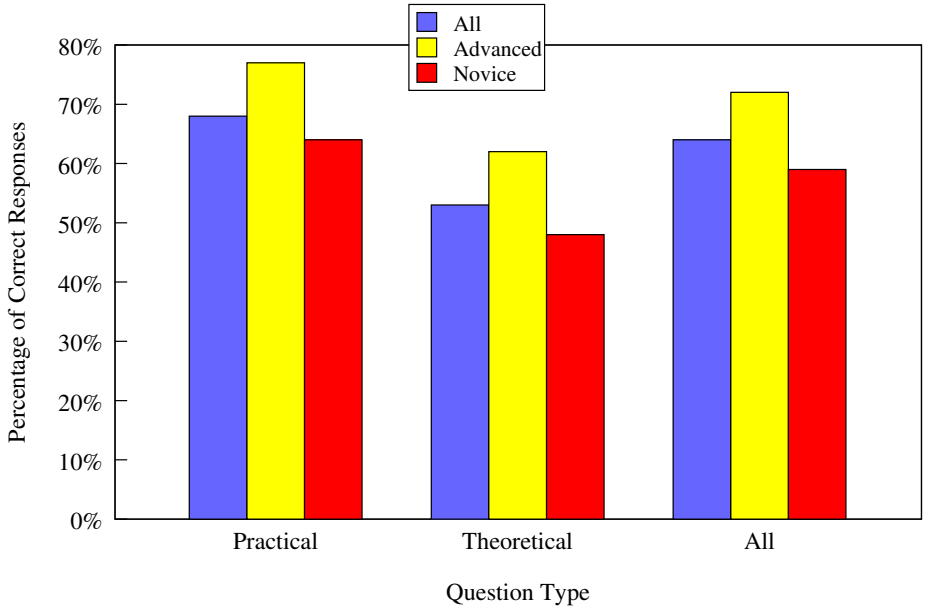


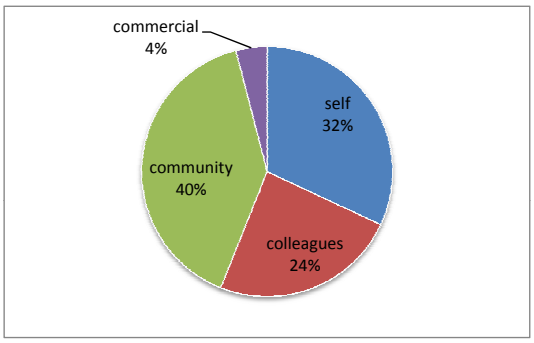
Figure 2.6: Test scores. The graph summarizes participants' scores on a test that assessed participants' knowledge about computational topics such MPI, queuing systems, and running jobs. Average overall scores are presented as well as scores by problem type. The graph also distinguishes how scores differed for participants with different levels of experience.

2.2.3 Perspectives on Migration

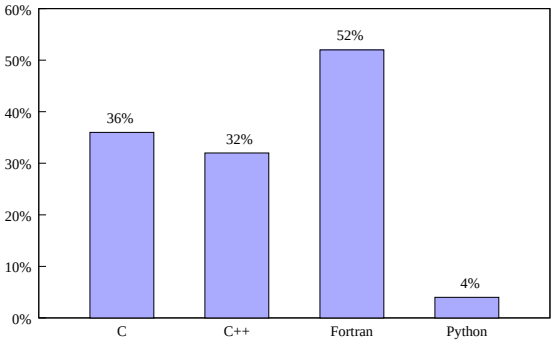
Participants evaluated their previous experiences with and perceptions of migrating applications to new computing sites. The most common reasons listed for migrating applications were running jobs on resources with more compute power or memory, better hardware, and better availability. The most common reasons listed for not migrating applications were the long setup process and large learning curve. The most difficult challenge with migrations listed was debugging issues at new computing sites.

2.3 Application Profiles

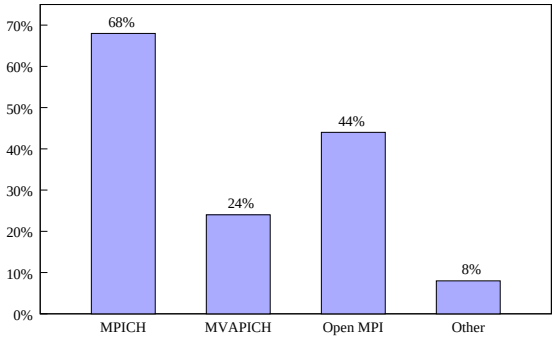
This section presents information about the MPI applications that participants migrated during the study. Figure 2.7 summarizes who wrote the applications, what programming languages were used to write the applications, what MPI implementations were used to run the applications, and at how many sites the applications were previously compiled and run. The authors of applications were



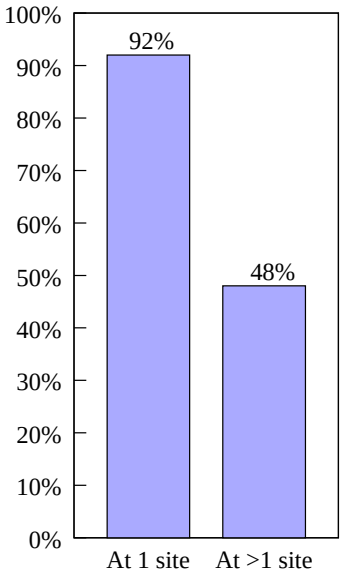
(a) Creators of Applications



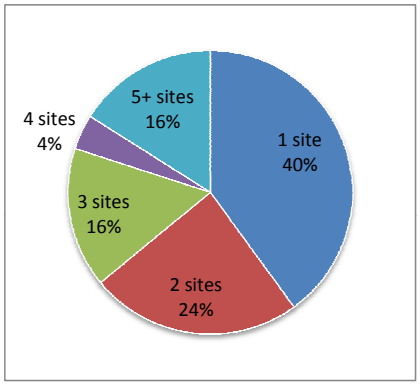
(b) Programming Languages Used



(c) MPI Implementation Types Used



(d) Number of Compilation Sites Used



(e) Number of Execution Sites Used

Figure 2.7: Profile of migrated applications. The graphs highlight five dimensions about the migrated applications. Graph (a) presents who wrote them. Graph (b) presents the programming language in which they were written. Graph (c) presents what MPI implementation was used to run them. Graphs (d) and (e) present at how many sites participants had compiled and run them, respectively. All quantities are presented as percentages of the total number of applications.

well distributed between being written by the participants, by colleagues, or by the community. The programming languages used to write the applications were well distributed between C, C++, and FORTRAN. MPICH was the most widely used MPI implementation by the applications. Almost half of the applications had only been run at one site while the rest of the applications were well distributed between having been run at two, three, and more than four sites. Of the applications that had only been run at one site, the main reason for not using more sites was reported by 60% of the participants to be lack of access to more sites. While all but two participants had compiled their MPI application previously, only around half of the participants had experience compiling their application at more than one site. This is in contrast to only two thirds of the participants having programming experience.

2.4 Migration Results

This section summarizes the results gathered about the migration process. As described in Section 2.1.3, participants documented the migration process using logs. From these logs, information was gathered about the duration of migrations and the types of tasks performed during the process. An analysis of the duration of migrations is followed by an analysis of the tasks performed during migration. The results are presented for all participants and for participants subdivided by experience level. The results are also presented as averages across all migration locations and across individual migration locations.

2.4.1 Migration Durations

The amount of effort associated with migrations is quantified in terms of the amount of time participants spent migrating their applications to different computing sites. To better compare results between participants who performed different numbers of migrations, the time for all migrations as well as the average time per one migration was considered. Timing results are reported as the number of hours spent on migration tasks as well as the number of days over which migrations took

Table 2.3: Migration duration summary. This table summarizes the average migration timing results across all migration sites for all participants as well as by experience levels. The standard deviations are presented along with each statistic. For instance, advanced participants spent on average 1.6 hours spread over 2 active and 3 consecutive days to perform one migration while overall taking on average 8.4 hours spread over 13 consecutive days to complete 5 migrations and succeeded at getting their applications running for 100% of their migrations.

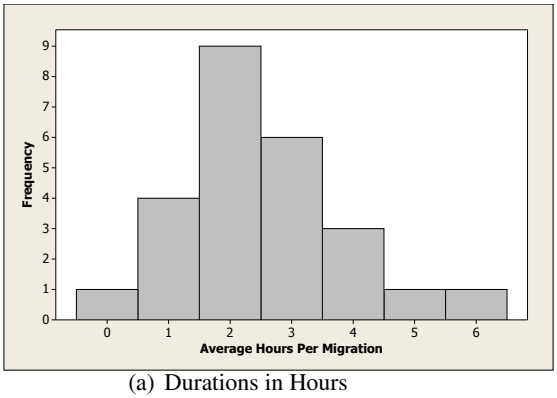
Exper Level	Hours per Migration	Active Days per Migration	Consec Days per Migration	Total Hours	Total Days	Total Migs	Success Rate (%)
All	2.5 $\sigma = 1.3$	3 $\sigma = 2$	7 $\sigma = 6$	8.8 $\sigma = 3.8$	18 $\sigma = 14$	4 $\sigma = 2$	80 $\sigma = .3$
Novice	3.0 $\sigma = 1.3$	4 $\sigma = 2$	8 $\sigma = 7$	9.0 $\sigma = 3.6$	20 $\sigma = 16$	4 $\sigma = 2$	70 $\sigma = .4$
Adv	1.6 $\sigma = 0.8$	2 $\sigma = 1$	3 $\sigma = 3$	8.4 $\sigma = 4.4$	13 $\sigma = 8$	5 $\sigma = 1$	100 $\sigma = .1$

place. Timing results are also reported in relation to each migration location to identify which sites required more time to complete migrations.

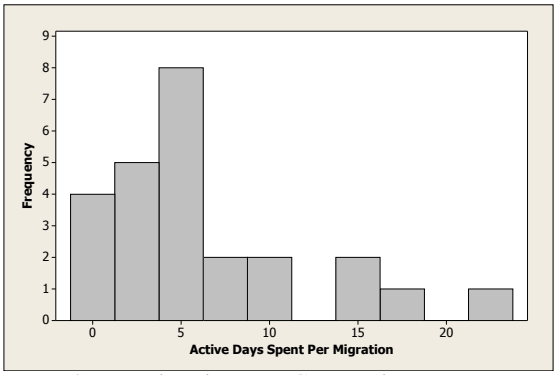
2.4.1.1 Averages over all Migrations

Table 2.3 summarizes the durations of migrations as averages across all migration locations. The distribution of the results is presented in Figures 2.8, 2.9, and 2.10. The migration durations are presented as averages per one migrations and as the totals to complete all migrations. Durations are presented in terms of hours, consecutive days (from the start to end of migrations), and active days (not necessarily consecutive days when migrations were actually being performed). The number of sites to which participants migrated is also presented as the total number of attempted migrations and as the percentage of successful migrations.

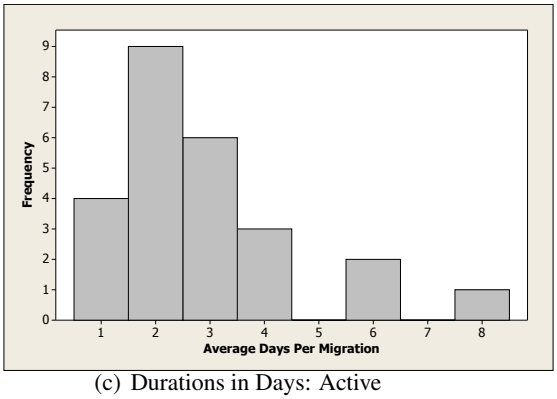
The results show that on average participants spent 18 consecutive days on the migration phase of the study and in that time period spent 8.8 hours performing four migrations. When considering the average time per one migration, participants spent 2.5 hours over 3 (not necessarily consecutive) days to complete one migration. To mitigate fatigue effects, participants had been instructed to perform no more than two migrations per day. Even with this restriction, given the average time per one migration, the entire migration process could have been completed in less than one week instead



(a) Durations in Hours

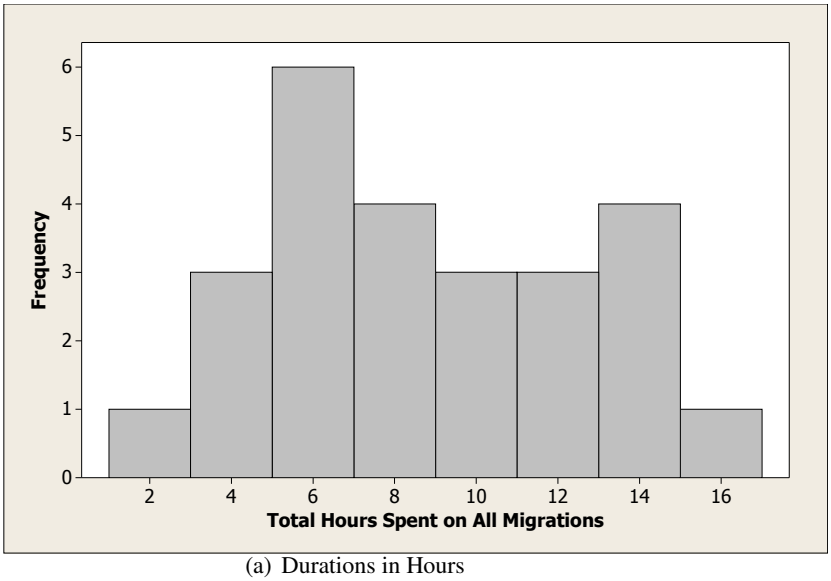


(b) Durations in Days: Consecutive

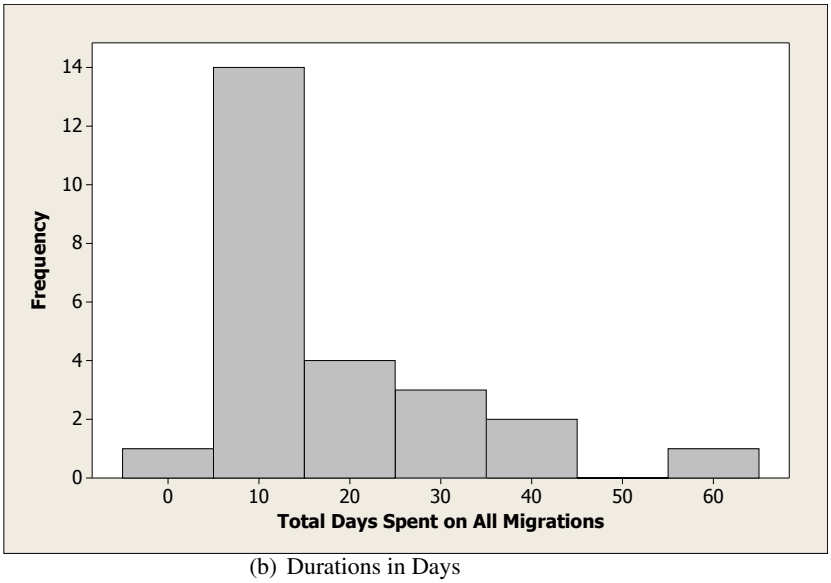


(c) Durations in Days: Active

Figure 2.8: Average durations for one migration. The histogram graphs present the average duration per one migration. Graph (a) presents the durations in terms of hours. Graph (b) presents the durations in terms of the number of consecutive days to perform one migration. Graph (c) presents the durations in terms of the number of active days to perform one migration.

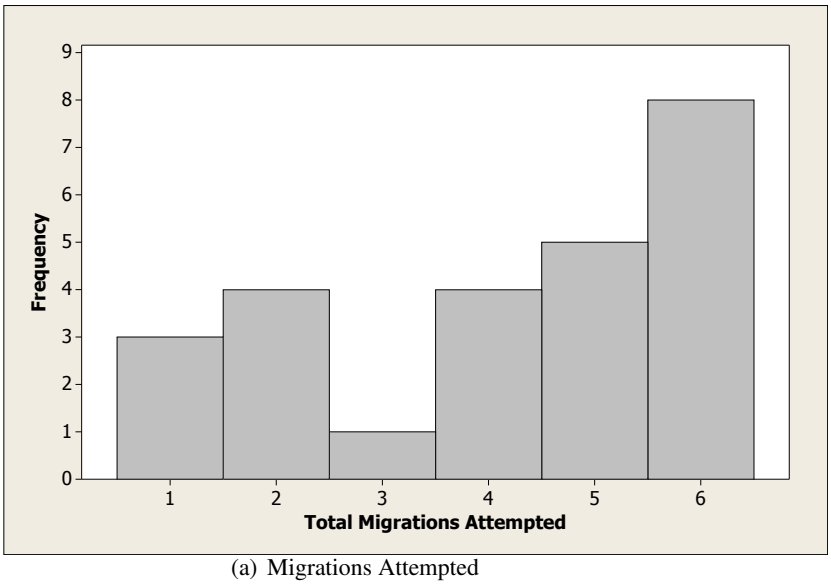


(a) Durations in Hours

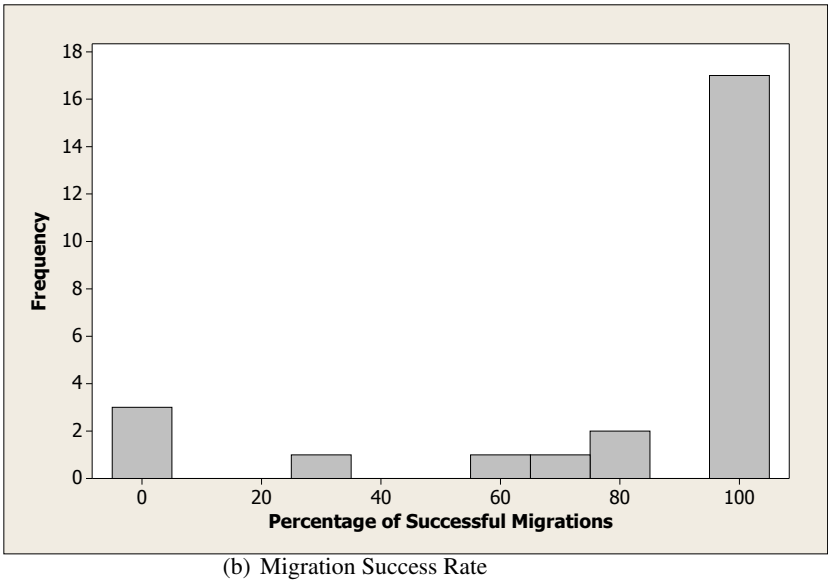


(b) Durations in Days

Figure 2.9: Total durations for all migrations. The histogram graphs present the total durations to complete all migrations. Graph (a) presents the durations in terms of hours. Graph (b) presents the durations in terms of the number of consecutive days to perform all migrations.



(a) Migrations Attempted



(b) Migration Success Rate

Figure 2.10: Number of migrations performed. The histogram graphs present the number of migrations each participant performed. Graph (a) presents the number of total migrations while graph (b) presents the percentage of successful migrations.

of more than two. As less than 25% of migrations were unsuccessful, the results were not greatly skewed toward the two hour minimum migration attempt time counted toward compensation. The overall migration duration results show that while migrations in general did not take many hours, for whatever reason participants spread the work over many days as they managed the process. Thus, the time to get applications running manually at multiple sites can be quite long.

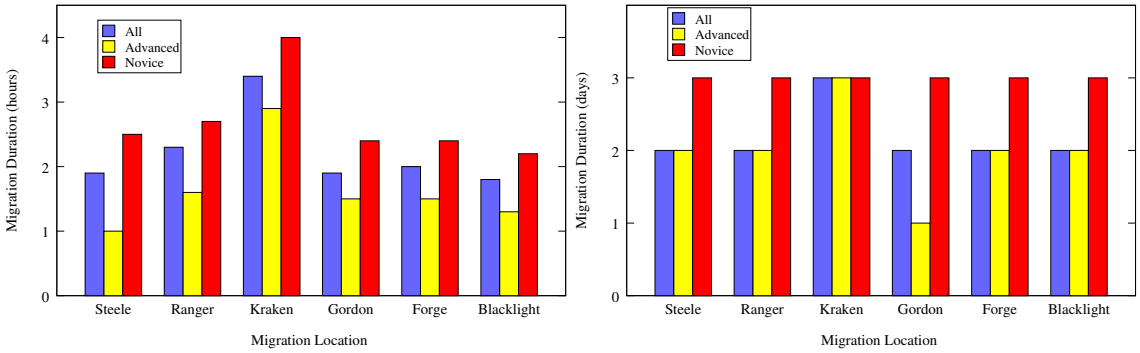
The results also show that the amount of time participants spent on migrations was higher for participants with less experience. Less experienced participants spent more hours to perform each migration and spread the migration process over more days. Compared to advanced participants, novice participants spent 87% more hours per migration (1.4 extra hours over two extra days). Additionally, novice participants spent 53% more total days (an extra week) than advanced participants to complete all their migrations, even while migrating to less sites on average during that time.

2.4.1.2 Averages by Location

To investigate whether any migration location was particularly time consuming, the migration duration results were subdivided by computing site. Figure 2.11 highlights the results with two graphs. Figure 2.11(a) presents the number of hours spent on average on one migration while Figure 2.11(b) presents the number of (not necessarily consecutive) days over which the migration occurred. The migration durations are presented for all participants as well as for participants subdivided by experience level. (At least six data points were collected for each duration average when subdivided by participants' experience level.)

The results spotlight a difficult to use system. On average, participants took closer to two hours over two days to perform one migration at most locations. However, migrations at the Kraken system took around 50% longer at closer to three hours over three day. This result supports Kraken's notoriety for being one of the more difficult to use XSEDE resources.

The results also illustrate the effect of experience on migration durations. At all migration locations novice participants took more hours to perform migrations than advanced participants. Experience had a consistent impact on the migration process.



(a) Durations in Hours

(b) Durations in Days

Figure 2.11: Migration durations by location. These graphs present the average duration of migrations at each migration location. Graph (a) presents how many hours were spent on average on one migration while graph (b) presents over how many days on average those hours were distributed. Results are presented for all participants as well as for participants by experience level.

2.4.1.3 Migration Order

Participants performed four migrations on average. To investigate whether earlier migrations took participants more time to complete than later migrations, the average durations of migrations were grouped by migration number. Participants migrated to sites in a randomly selected, predefined order. Each participant's migrations were numbered in the order in which they were begun. The durations of migrations with the same migration number were averaged to determine how long it took on for participants to complete migrations with the same order number. The results are presented in Figure 2.12. (The figure does not present results by experience level for the sixth migration due to a lack of data points. As only eight participants performed this migration overall, subdividing further would provide few data points for averages by experience level.)

The results show that for novice participants, the effort to start the migration process was especially larger than the effort to perform additional migrations. The first two migrations, independent of location as it was randomly varied for each participant, took novice participants more than 50% longer to perform than later migrations. For advanced participants, the effort to perform migrations did not decrease significantly as more migrations were performed. Overall, the duration of the non-initial migrations averaged at 1.6 hours.

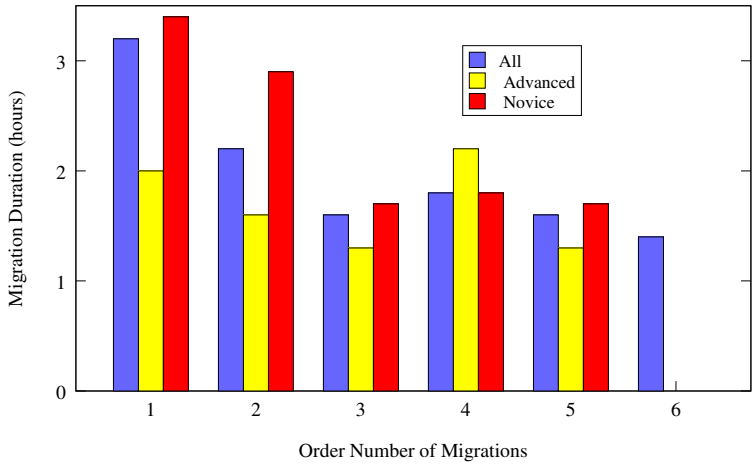


Figure 2.12: Migration durations by migration order. Migration durations were ordered according to when they were begun as each participant migrated to sites in a randomly selected, predefined order. The graph presents the average durations of migrations grouped by migration order.

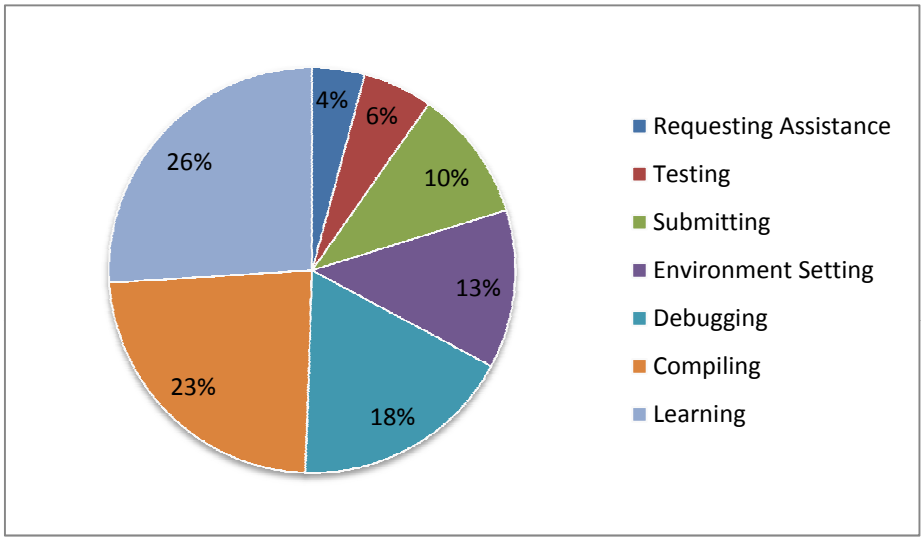
2.4.2 Migration Tasks

The time participants spent on different types of migration tasks was analyzed to identify the most time consuming and common tasks. The duration of different types of tasks when performing one migration was calculated as an average at each migration location and for all locations. Ratings of tediousness and difficulty were also analyzed to identify the most tedious and difficult task types overall and at specific locations.

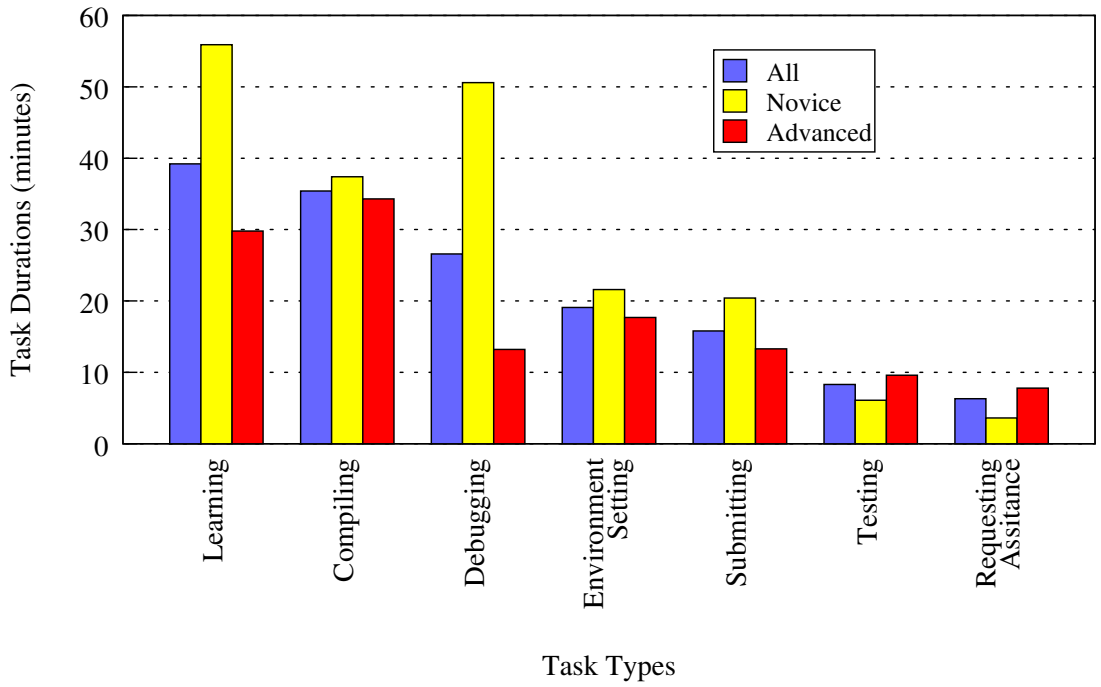
2.4.2.1 Task Durations

The migration timing results were subdivided to analyze the amount of time participants spent on the seven task types described in Figure 2.2. The durations of each migration task are presented in Figures 2.13 and 2.14. Figure 2.13 presents the results averaged across all migration sites while Figure 2.14 presents the results averaged individually for each migration site. Each figure presents the results for participants overall and by experienced level.

When considering the activities of all participants, the results in Figure 2.13(a) show that the majority of each migration was spent performing tasks related to learning, compiling, and debugging while the least amount of time was spent requesting assistance. When considering the activities

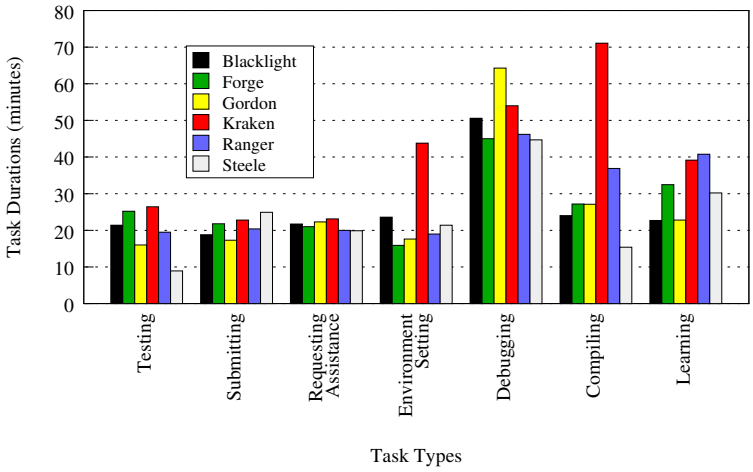


(a) Decomposition of Time Spent on One Migration by Types of Tasks Performed

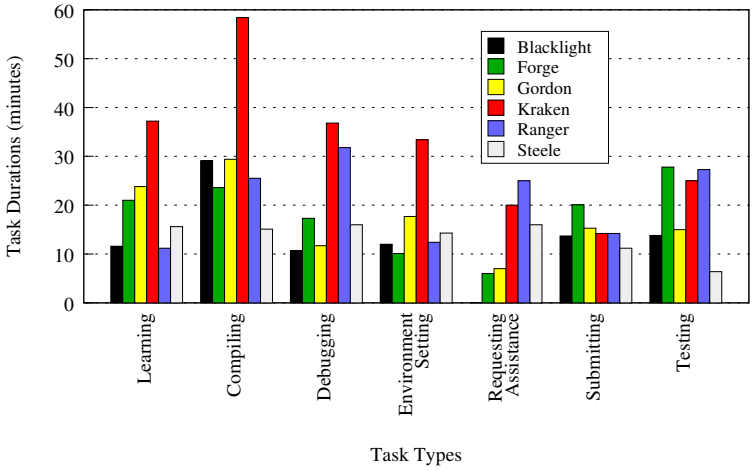


(b) Variance of Task Type Durations by Participants' Experience

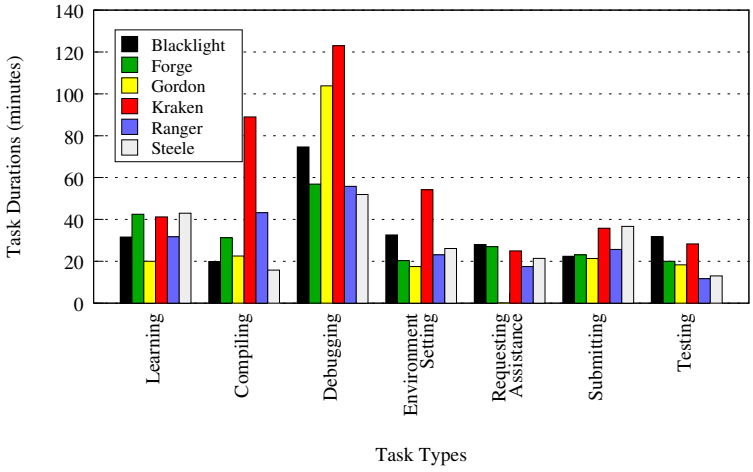
Figure 2.13: Migration task durations. These graphs present the durations of the seven types of migration tasks as averages across all migration locations. Graph (a) shows what percentage of one migration each task type took on average. Graph (b) shows how the average durations of each task type differed for participants with different levels of experience.



(a) Durations of All Participants



(b) Durations of Advanced Participants



(c) Durations of Novice Participants

Figure 2.14: Migration task durations by location and experience. The graphs present the durations of the seven types of migration tasks as averages for participants overall and by experience level at each migration location.

of participants subdivided by experience level, the results in Figure 2.13(b) show that the four most time consuming tasks for participants regardless of level of experiences were learning, compiling, debugging, and environment setting.

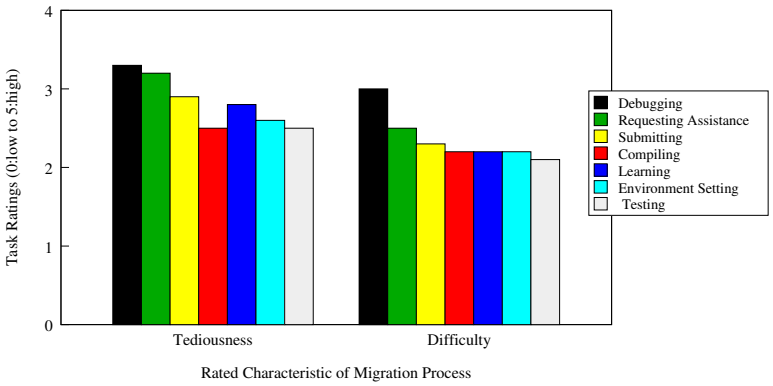
When considering the results by participants' experience levels, the most time consuming tasks for novice participants become especially apparent. While novice participants spent longer than advanced participants on the majority of tasks, they spent almost 100% longer on learning and debugging. Advanced participants spent more than 50% longer on learning and compiling than on other types of migration tasks.

When considering the activities of all participants at different migration locations, the results in Figure 2.14(a) show that, except for two task types at one site, each task type took participants about the same amount of time to perform regardless of location. The two exceptions were again on the Kraken system where compiling and environment setting took almost twice as long. When considering the results in Figures 2.14(b) and 2.14(c) further subdivided by participants' experience, the location where it took the shortest time to complete each task type varied for advanced and novice participants. However, for both the majority of tasks types took the longest to perform on Kraken. This results again supports Kraken's notoriety for being a more difficult to use system.

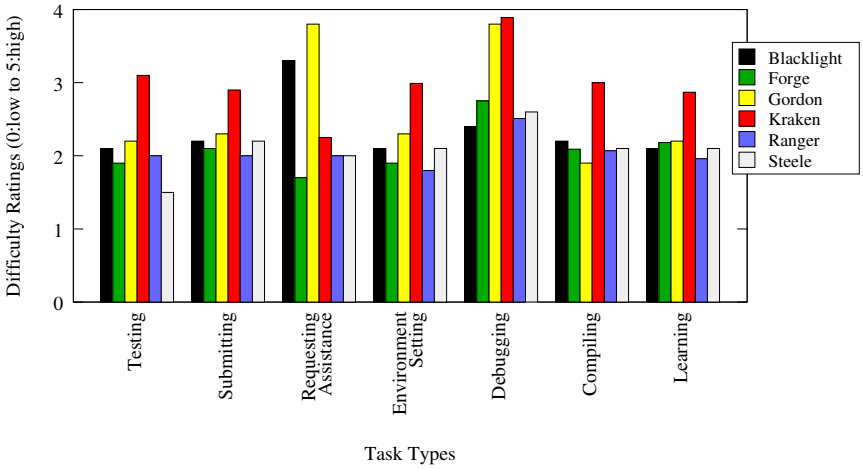
2.4.2.2 Task Ratings

Ratings of the tediousness and difficulty of each of the seven task type documented during the migration process are presented in Figure 2.15. Graph 2.15(a) presents the ratings averaged across all migrations. Graphs 2.15(b) and 2.15(c) present ratings of difficulty and tediousness, respectively, at each migration location.

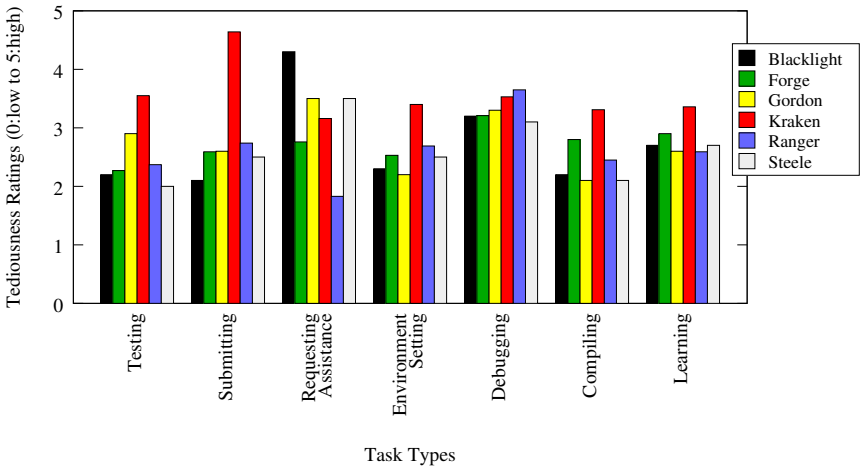
When considering all migrations, the results in Figure 2.15 show that the average difficulty and tediousness ratings of tasks types were similar. While debugging and requesting assistance were rated as slightly more tedious, debugging was rated as 20% more difficult on average than the other task types. When considering the results subdivided by location, tasks of any type were consistently rated as more difficult and tedious at one site in particular. As shown in Figure 2.15(b), the difficulty ratings of all task types were highest for the Kraken system. Similarly, as shown in Figure 2.15(c),



(a) Overall Difficulty and Tediousness Ratings



(b) Difficulty Ratings by Migration Location



(c) Tediousness Ratings by Migration Location

Figure 2.15: Ratings of migrations' tediousness and difficulty. The graphs present the average ratings of the tediousness and difficulty of each task type performed during the migration process. Graph (a) presents the average ratings across all migration locations. Graphs (b) and (c) present the average ratings for different migration locations.

the tediousness ratings for the majority of task types were also highest for the Kraken system. Again, this result supports the notoriety of Kraken as a more difficult to use system.

2.5 Summary

The study aimed to quantify the effort end-users put forth to get their MPI applications running at new computing sites. The study was conducted using XSEDE resources with participants from institutions across the United States. Common tasks involved in the process of migrating applications to new sites were confirmed and how much time was spent on them was quantified. How various levels of experience affect end-users attempts to migrate their MPI applications to new computing environments were also investigated.

The study results indicate that manually migrating applications to multiple targets is a time consuming process. While the results illustrate that the migration process takes less time after one migration is performed, the average durations of migrations did not significantly decrease after the initial migration. The documented duration of one migration averaged two and a half hours but the completion of this work was on average spread over several days and lead to multiple weeks being required to perform multiple migrations. The results also repeatedly showed that less experienced participants spend more time on migrations than more experienced participants. Novice participants spent almost 50% more time per migration than advanced participants both in terms of hours (equating to one and a half extra hours of work) and in terms of the days (equating to one extra week) over which they spread out their migrations. These findings were consistent when analyzing the results by migration location.

The analysis of task types showed that the majority of each migration was spent performing tasks related to learning, compiling, and debugging. The results also clearly illustrated the difficulties associated with one particular migration location. Migrations to the Kraken system took noticeably longer. All participants regardless of experience took longer to perform the majority of task types on Kraken and rated all tasks on Kraken as more difficult and tedious than at other locations.

The study has provided concrete experimental data documenting the migration process for researchers with various levels of experience. The results provide a baseline for the duration of the process that currently must be completed before a new resource can be used directly or via a scheduler to run a computation. The results clearly support that migrating to multiple sites is a time consuming process and that experience impacts migration time. The gathered migration duration results are used in the evaluation of the proposed solution in Chapter 6. Clearly methods that help identify sites where applications can run without much user effort would be useful to decrease the amount of time it takes end-users to get their applications running at multiple sites, especially for users with less computational expertise. Simultaneously, job management systems could benefit from such methods to increase the set of candidate hosts where an application can be scheduled to run without additional user intervention. Such additional scheduling freedom could decrease queue wait times and increase job throughput.

Chapter 3

The Model

This chapter describes the *execution readiness prediction* (ERP) model, a model for predicting whether an MPI binary is ready to execute at a computing site. By knowing whether recompilation will be needed to utilize new resources, researchers and schedulers can better choose execution locations. Job management systems are enabled to schedule computations at a larger assortment of locations if it can be determined without user intervention where the binary can run without recompilation. Also, researchers can directly choose to run on resources where their computations can run with low start-up effort when they know whether recompilation will be needed. To enable this scheduling freedom, a model was formed to describe the relationships between information relevant for execution.

The ERP model serves as a high-level prescription for implementing predictions of execution readiness. *Execution readiness* refers to an application's ability to run at a computing site successfully. The model highlights factors related to applications and computing environments that are highly relevant for execution. The model also outlines the conditions for predicting how these factors influence the execution readiness of an application at a particular computing site.

Migrating binaries instead of source code to new computing sites can be beneficial when tuned performance is not a concern. In this manner, researchers can avoid long compile times or compiling unfamiliar codes like community applications. By knowing that recompilation does not have to occur, researchers can – directly or via schedulers – gain use of new sites quickly enough to react to changing characteristics such as queue delay times.

Section 3.1 begins with a discussion of the requirements that guided the design of the ERP model. Section 3.2 describes the factors that form the model's basis for assessing execution readiness while Section 3.3 presents the guidelines for assessing the influence of each factor on execution readiness. Section 3.4 concludes with a discussion of the order in which the assessments should occur to form the final prediction of execution readiness.

3.1 Model Requirements

The ERP model was designed to meet two basic requirements. First, the model is intended to be simple. A simple model is more easily understood, is applicable to a wider range of computing systems and applications, and helps facilitate its implementation. Second, the model was to be language and operating system independent. The ERP model defines at a high level the factors that influence application execution, and defines when these factors are satisfied, but it does not rely on or prescribe any particular underlying technology for its implementation. Again, this makes the model applicable to more applications and computing sites, and avoids coupling the model with the characteristics of a programming language, architecture, or operating system.

3.2 Execution Factors

Whether an application binary will execute at a particular computing site depends on whether the computing site satisfies the application's execution requirements. Many of an application's execution requirements are formed at compile time. The ERP model identifies four factors as particularly relevant for execution: 1) the MPI stack, 2) shared libraries, 3) the hardware architecture, and 4) the system configuration. This section describes each of these *execution factors*.

The execution factors for the ERP model were identified by performing an exploratory survey of the issues that arise when migrating MPI programs [23]. Six MPI implementations were analyzed across eight computing environments. The computing environments were chosen such that a diverse test set was created in terms of the operating system, hardware architecture, network interconnect, and MPI implementation. Of the systems studied, four were part of XSEDE (Kraken,

Ranger, Lincoln, and Ember), two were part of FutureGrid (India and Xray), and two were from the University of Virginia (ITS Elder and CS Centurion). For each MPI implementation type at each computing site, MPI programs were compiled and observations were made about what issues, if any, arose when attempting to execute the programs at different sites.

3.2.1 MPI Stack Requirements

The MPI stack is an important factor that influences the execution of applications that use the MPI standard. As MPI is only an interface specification, the standard specifies a library interface, not a link level interface. An implementation of the MPI standard consists of a library (i.e. Open MPI) that can have various dependencies especially since each implementation may realize the standardized interfaces using different data strutures and communication protocols. Accordingly, an application compiled with a particular MPI implementation inherits the set of dependencies related to that MPI implementation type. Therefore, in order for a dynamically linked application to execute, the necessary MPI library needs to be available at a computing site along with the MPI implementation related dependencies.

3.2.2 Shared Library Requirements

Shared libraries are collections of object code that can be shared and used by multiple programs. There are two ways in which programs can be linked to shared libraries. In static linking, the object code used by a program is included in the program's executable. In dynamic linking, the object code is loaded when the program is executed. Shared libraries that can be used for static linking are called static libraries and are commonly identified with a *.a* file extension. Shared libraries that can be used for dynamic linking are called dynamically linked libraries and are commonly identified with a *.so* or *.dll* file extension. In the exploratory survey, the MPI configurations at the majority of the computing sites were found to be setup only with shared libraries for dynamic linking. At these sites, the only option for creating MPI binaries was with dynamically linked libraries.

The availability of shared libraries at computing sites influences the execution of any dynamically linked application. If an application's shared library requirements are not met at a new com-

puting site, the application will not be able to execute as it will be missing access to code provided by the shared libraries. In particular, C standard library compatibility is a major determinant in the ability of an application to execute at a new computing site. As a very large and commonly-used shared library, the C standard library is dynamically linked to by most applications, as well as their shared library dependencies.

3.2.3 Hardware Architecture Requirements

Hardware architecture is a very basic determinant for the execution of all applications. Every application is compiled for a specific instruction set architecture (e.g. PowerPC, x86) and a certain word-length (i.e. 32-bit, 64-bit).

3.2.4 System Configuration

Even when all execution factors are satisfied, an application may still fail to execute at a computing site due to system configuration errors. These system errors can be caused by misconfigurations that affect the execution process. For example, an MPI implementation may have been updated incorrectly and cause binaries that depend on that MPI implementation to not be able to execute. The ERP model considers system configurations related to execution as a determining factor for predicting execution readiness.

3.3 Assessing Compatibility

The key to execution prediction is assessing whether an application's execution requirements are satisfied at a computing site. The section provides guidelines for determining whether each of the four factors identified as particularly relevant for execution is met at a computing site.

3.3.1 MPI Stack Compatibility

To assess if an MPI stack requirement is satisfied, the compatibility of MPI implementation types needs to be determined. Only MPI implementations of the same type are considered to be compat-

Table 3.1: Identifying libraries of MPI implementations. The link-level shared library dependencies associated with a specific MPI implementation type can be used to identify an application's MPI implementation type requirement.

MPI Implementation	**Library Dependencies**
MPICH2	libmpich/libmpichf90 (and not other identifiers)
MVAPICH2	libmpich/libmpichf90, libibverbs, libibumad
Open MPI	libnsl, libutil

ible. A MPI application binary has been linked against the dependencies of a specific MPI type. Thus, in order for the application to execute, these dependencies need to be present at a computing site. To determine compatibility, the MPI type with which the application was compiled needs to be matched against the MPI types available at a computing site. For example, an application compiled with Open MPI is compatible with Open MPI but not with MVAPICH.

Matching the MPI implementation is important as computing sites often provide access to multiple MPI stacks. In the exploratory survey, each computing site was found to have multiple versions of two to three MPI implementations available along with at least two compiler options. Thus, relying on a computing site's default MPI configuration will likely not result in a compatible environment state. Indeed, during the exploratory survey tests, applications were able to execute at computing sites using the default MPI implementation settings less than 2% of the time.

To determine compatibility, the ERP model does not consider the version of the MPI implementation type. Different version numbers do not necessarily imply incompatibility or compatibility as there are no universal guidelines regarding backwards compatibility and releases of MPI implementations.

To perform the compatibility assessment, an application's MPI implementation type requirement must be known. In order to not require the user to provide this information, it must be extracted from the binary. A scheme was developed to uniquely identify the MPI implementation type used to compile an application. The link-level shared library dependencies associated with each MPI implementation type were used to make the identification. These MPI implementation identifiers are listed in Table 3.1.

3.3.2 Shared Library Compatibility

Shared library compatibility is assessed by considering name and version conventions as well as word-length (32 vs. 64-bit). All shared libraries have a special name called the *soname*. Files containing shared library code are named using the following convention that consists of the *soname* followed by a period and a version number. This sequence can optionally be followed by another period and a release number.

lib<*soname*>.so.<version#>.<release#>

Following this convention, the shared library name *libmpichf90.so.1.2* indicates that it is the second release of the first version of the MPICH library for Fortran90. As the version number of shared libraries is incremented when non-backward compatible changes occur in a library's application binary interfaces (ABI), shared library compatibility is defined based on this version number. A computing site's shared library is defined to satisfy an application's shared library execution requirement if it has the same *soname*, version number, and word-length format as the required library.

Execution may also be possible with a different version of a shared library than what an application was linked against. The highest version of a particular library that a binary's symbols require can be determined by examining an executable. This is termed the *required version* of a shared library. Execution may be possible if a computing site's shared library version is equal to or greater than an application's required shared library version. This is of particular interest when matching versions of the widely used C standard library. Different versions of the C library are often found at computing sites as a result of the library being updated at different times during the lifetime of systems.

3.3.3 Hardware Architecture Compatibility

To assess hardware architecture compatibility, the format into which an application has been compiled is compared against the format supported by the hardware architecture of a computing site.

The instruction set architecture and binary format such as word-length need to match in order for a binary to execute at the computing site.

3.3.4 System Configuration Compatibility

Whether a computing site is properly configured for execution can be assessed by attempting to run a program that should execute. This process tests the system's configuration of the job submission process and the job management system. To also test whether a particular MPI stack is functioning, a simple MPI program can be created using the MPI stack under investigation and used for testing the execution process. If the process succeeds without errors, then the system can be assessed to have a functioning configuration for the execution of programs that depend on that MPI stack.

3.4 Prediction Forming

To form a final prediction about the execution readiness of an application at a computing site, the order in which the compatibility of each execution factor is assessed matters. An evaluation ordering is created that considers the dependence of factors as well as the complexity of assessment. Hardware architecture compatibility is checked first as it dictates whether an application will be able to execute at a computing site from the basic hardware perspective. The system configuration functionality is checked last as this factor cannot be fully assessed until the relevant site configurations, such as the MPI implementation type, have been assessed. MPI implementation compatibility can be checked before shared library compatibility as it should be a shorter assessment than checking the compatibility of a potentially large number of shared libraries.

A flowchart depicting the ordering of the prediction forming process is presented in Figure 3.1. First, the hardware architecture compatibility is assessed. If an application is not compiled into a format that can be executed at the computing site from the hardware architecture perspective, then the model predicts execution will not be possible. Otherwise, the MPI stack requirements are assessed. Again, if these requirements are not met because no compatible MPI implementation is available at the computing site, then the model predicts execution will not be possible. Otherwise,

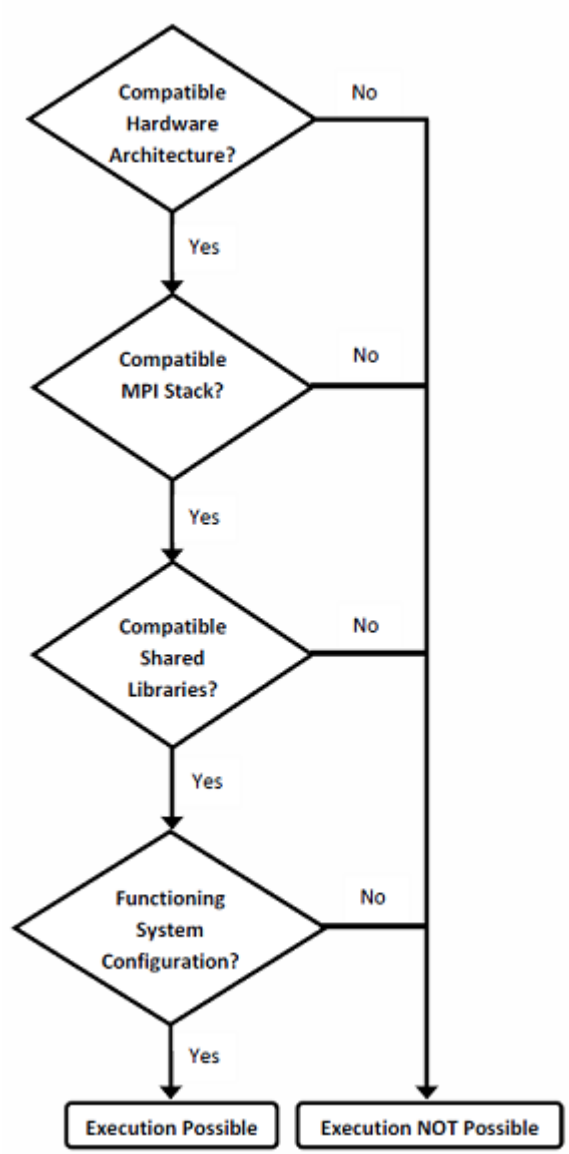


Figure 3.1: Prediction decision tree. This flowchart depicts an ordering when serially assessing each execution factor during the process of forming a prediction of an application's execution readiness. The ordering is based on the magnitude of each incompatibility in terms of affecting execution as well as the complexity of assessing each factor so as to arrive at a prediction more quickly.

the shared library requirements are assessed. If compatible versions of required shared libraries are not present at the computing site, execution is predicted to not be possible. Otherwise, the functionality of the system configuration is lastly assessed. If the configuration is found to not be functional, then execution is predicted to not be possible. Otherwise, execution is predicted to be

possible.

3.5 Summary

This chapter has presented the execution readiness prediction (ERP) model, a model for determining whether an MPI binary is able to execute at a computing site. The model focuses on four execution-influencing factors: MPI stack, shared libraries, hardware architecture, and system configuration. To form a prediction, the ERP model provides guidelines for assessing compatibility between an application's execution requirements and the configurations of a computing site. The prediction is formed without running the application binary.

Automatically forming execution readiness predictions is the basis for enabling deployment freedom. The predictions can provide researchers and schedulers with a determination of whether applications can run at new computing sites without recompilation. In this way, the ERP model provides an approach for automatically determining whether a computing site is a good fit for running a computation.

Chapter 4

Implementation

This chapter describes a concrete implementation of predicting execution readiness for MPI applications on Unix-based computing systems. FEAM, or a Framework for Efficient Application Migration, implements the ERP model described in Chapter 3 while additionally incorporating a scheme for resolving execution-blocking issues. As such, FEAM can be used by users and schedulers to identify computing sites that are ready to execute MPI binaries and form site-specific configurations for running those binaries. An overview of FEAM and its four components is presented in Section 4.1 followed by a description of FEAM's resolution scheme in Section 4.2. Then Sections 4.3 through 4.6 explain how FEAM determines an application's execution readiness and composes a resolution scheme in terms of the work done by each of FEAM's components.

4.1 Overview

FEAM is composed of four components that work together to form an execution readiness prediction and resolution scheme. Two components gather information about the factors relevant for determining execution readiness as outlined by the ERP model. The Binary Description Component gathers information about the application binary, while the Environment Description Component gathers information about the computing site. The third component, the Resolution Component, gathers information for realizing the resolution scheme that FEAM implements in addition to predicting execution readiness. The fourth component, the Target Evaluation Component, assesses

compatibility and forms a final execution readiness prediction according to the guidelines of the ERP model. The Target Evaluation Component also creates a script with site-specific configurations, a description with any detected issues, and a resolution scheme when applicable. For portability between Unix-based sites, FEAM and each of its components is implemented using `bash` scripts. To account for techniques that do not work equally well on different Unix-based systems, FEAM's components use multiple methods to discover relevant information about application binaries and computing sites.

FEAM differentiates between two location types: 1) a target site and 2) a guaranteed execution site. A *target site* is a computing site where an application binary is to be run while a *guaranteed execution site* is a computing site where an application is known to already run successfully. The guaranteed execution site may be the site from where the application is being migrated and/or it may be the site where the application was compiled. To make a prediction, FEAM must be run at a target site. To form a better prediction and to create a resolution scheme, FEAM can be additionally run at guaranteed execution site.

FEAM also differentiates between two computing node types: 1) a head node and 2) a compute node. The *head* node refers to the front end node that users traditionally log into and submit jobs from when using Unix-based clusters. The *compute* node refers to a back end node where jobs are traditionally executed in Unix-based clusters. While FEAM is designed to be always invoked from a head node, FEAM can execute directly on the head node or form a submission script to execute on a compute node. If the head and compute nodes of a system have the same configurations in terms of the filesystem, runtime environment, and architecture, FEAM can execute on the head node to assess the majority of the ERP model's execution factors as well as to form a resolution scheme. However, to evaluate each of the ERP model's execution components, FEAM must execute on a compute node. In particular, the system configuration of a target site can only be evaluated by running on a compute node.

Where FEAM is run determines which FEAM components are executed. Figure 4.1 illustrates which FEAM components are executed in relation to location and node type. To gather information about a new computing environment, FEAM's Environment Description Component must run

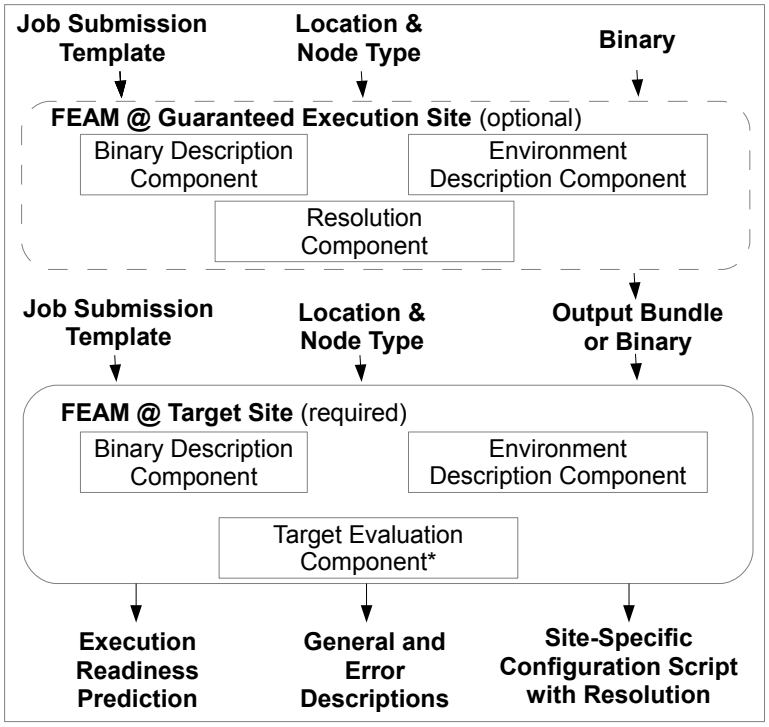


Figure 4.1: FEAM overview. FEAM, a Framework for Efficient Application Migration, implements the Execution Readiness Prediction Model on Unix-based computing systems and incorporates a resolution scheme. This diagram depicts an overview of how FEAM runs at different site locations along with the corresponding inputs and outputs. FEAM must be run at a target site to form a prediction and FEAM may additionally be run at a guaranteed execution site to enable resolution and improve prediction capabilities. (The asterisk indicates the FEAM component that requires to be executed on a compute node. The remaining components may be run on a head node assuming that head and compute node configurations are identical.)

at a target site. To gather information about the application binary, FEAM's Binary Description Component can run at either at a target or guaranteed execution site. To gather information relevant for the resolution scheme, FEAM's Resolution Component must run at a guaranteed execution site. Similarly, to gather information for enhanced system configuration testing, FEAM's Binary Description Component must run at a guaranteed execution site. To make an evaluation of the system configuration execution factor, FEAM's Target Evaluation Component must run on a compute node.

The location where FEAM is run also determines FEAM's inputs and outputs (also depicted in Figure 4.1). In addition to specifying the location and node type, FEAM requires information about 1) the application binary and 2) a site's job submission syntax. The binary location is simply specified as input to FEAM while a site's job submission syntax is specified by filling out a template file. Because it can vary greatly from site to site, the submission format is the only site-related information that is not automatically discovered by FEAM. The general syntax for invoking FEAM follows along with an example invocation. In the example, FEAM is invoked at a guaranteed execution site for a binary located at */path/binary.sh* and a template file called *geSyntax*.

```
./feam <binary/bundle path> <job template path> <site type> [head node only]

./feam /path/binary.sh geSyntax -ge
```

The output of running FEAM at a guaranteed site, as in the above invocation, is a bundle of the gathered information. To enable FEAM to use this information at target sites, this bundle is provided as input to FEAM instead of a binary location. As a consequence, the binary does not need to be made available at target sites. An example invocation of FEAM on the compute node of a target site using a bundle called *feam-bundle.tar.gz* and a template file called *targetSyntax* follows:

```
./feam feam-bundle.tar.gz targetSyntax -t -h
```

The output of running FEAM at a target site, as in the above invocation, is a prediction of execution readiness, a summary of discovered information and errors as well as a site-specific configuration script that incorporates a resolution scheme when possible. Examples of FEAM's outputs are presented during the Target Evaluation Component description in Section 4.6. An example job submission syntax template is presented in Figure 4.2.

4.2 Resolution Scheme

In addition to implementing execution readiness prediction, FEAM incorporates a scheme for resolving some execution-blocking issues. Specifically, FEAM resolves shared library requirements.

Job Submission Template

```
# job submission command
qsub

# script type
bash

# combine stderr and stdout
#PBS -j oe

# name output file
#PBS -N <NAME>

# select 10 minute runtime
#PBS -l walltime=00:10:00

# select 2 nodes
#PBS -l nodes=2

#select queue
#PBS -q batch

#start job in submission dir
cd $PBS_O_WORKDIR

#other
. /opt/Modules/default/init/sh

#mpi command specification
mpiexec <BINARY>
```

Figure 4.2: Example job submission template. A site's job submission syntax is specified by filling out a template file. This figure presents an example of a completed template. Because it can vary greatly from site to site, the submission format is the only site-related information that is not automatically discovered by FEAM.

This scheme was created by considering how the execution factors of the ERP model could be influenced to enable execution. If the application binary is not to be modified, then the only way to affect the compatibility of execution factors is to change the configurations of the target site. Installing a missing MPI stack or emulating a mismatched instruction set is beyond the scope of what most users are allowed to do at a target computing site. In contrast, copying binaries, such as shared libraries, and making them available for use at runtime requires no special privileges or infrastructure.

Resolution of missing shared libraries by copying requires access to shared library binaries that will execute at the target site. Whenever there is access to an application's guaranteed execution site, there is also access to the shared libraries required by the application. Barring licensing issues, these shared libraries can be copied for use at other computing sites. However, these shared libraries will not necessarily be able to execute at a target site for the same reasons that any binary may not be able to run at a new computing site. To determine if a shared library copy can be used for resolution at a target site, a prediction of execution readiness can be formed. This process may include recursively resolving missing shared libraries.

Thus, FEAM incorporates a scheme for determining whether missing shared library requirements are resolvable along with implementing execution readiness predictions. First, FEAM gathers copies of shared libraries at guaranteed execution sites. Second, FEAM determines whether the library copies will execute at the target site. If this analysis predicts that the library copies are ready to execute at the target site, FEAM concludes that the missing shared library requirements are not execution-blocking.

4.3 Binary Description Component

The FEAM Binary Description Component (BDC) gathers information about the application binary to create an application description. An example description is presented in Figure 4.3. The BDC investigates the execution factors (hardware architecture, MPI stack, and shared libraries) outlined by the ERP model as relevant for execution prediction from the perspective of the requirements

APPLICATION DESCRIPTION

```
BINARY 107-in.omp-i.binary
FORMAT ELF
ISA x86
BITS 64
LIBC-REF x
SO libmpi_f90.so.0 mpi_f90 0 x
SO libmpi_f77.so.0 mpi_f77 0 x
SO libmpi.so.0 mpi 0 x
SO libopen-rte.so.0 open-rte 0 x
SO libopen-pal.so.0 open-pal 0 x
SO libdl.so.2 dl 2 x
SO libnsl.so.1 nsl 1 x
SO libutil.so.1 util 1 x
SO libgfortran.so.1 gfortran 1 x
SO libm.so.6 m 6 2.2.5
SO libgcc_s.so.1 gcc_s 1 x
SO libpthread.so.0 pthread 0 x
SO libc.so.6 c 6 2.2.5
LIBC-SO-REF 2.2.5
MPISTACK omp
LANGUAGE c
COMPILER gcc
```

Figure 4.3: Example application description. The FEAM Binary Description Component discovers information about an MPI application binary. The resulting *application description* contains information about the execution factors outlined by the ERP model from the perspective of the application. In this example, the application was found to have 13 shared library dependencies, to be formatted for an x86 64-bit architecture, and to be compiled with Open MPI.

of an application binary. This section presents the techniques the FEAM BDC employs to gather information about the execution factor and for realizing the resolution scheme.

4.3.1 Hardware Architecture Requirements

The FEAM BDC aims to discover for what type of hardware architecture the application binary was created. Two tools are used to investigate a binary's file format: the *NIX `file` utility and the GNU `binutils` [4] program `objdump`. The `file` utility classifies a file's type while `objdump` displays information about object files. Specifically, `objdump` is called with the `-f` option to view the binary file's file header contents. Example output from `file` and `objdump` is displayed in Figure 4.4. From this information, a determination is made into what file format (i.e. the executable and linkable format ELF), for what instruction set architecture (i.e. x86), and for how many bits (i.e. 64) the binary was compiled.

EXAMPLE OUTPUT: FILE

```
ELF 64-bit LSB executable,
AMD x86-64, version 1 (SYSV),
for GNU/Linux 2.6.9,
dynamically linked (uses shared libs),
not stripped
```

EXAMPLE OUTPUT: OBJDUMP

```
file format elf64-x86-64,
architecture: i386:x86-64,
flags 0x00000112: EXEC_P, HAS_SYMS, D_PAGED
start address 0x0000000000404040
```

Figure 4.4: Example analysis of application binary's hardware architecture requirements. The FEAM BDC determines for what type of hardware architecture an application was created by employing the *NIX `file` utility and the GNU `binutils` `objdump` program to examine the format of the binary. In this example, the binary is a 64-bit ELF executable created for the x86 architecture.

EXAMPLE OUTPUT: READELF

```
String dump of section '.comment':
  [     1]  GCC: (GNU) 4.2.4 (Ubuntu 4.2.4-1ubuntu3)
```

Figure 4.5: Example analysis of ELF application binary. The GNU Utility `readelf` is used to potentially determine what operating system and compiler were used to create the application binary. In this example, the binary was compiled on an Ubuntu Linux system.

The application binary is additionally examined to determine information about the operating system and the compiler used to create it. When dealing with binaries in the ELF file format (the executable and linkable format: the standard binary format for Unix-like systems on x86 architectures), the GNU `binutils` `readelf` program is called with the `-p .comment` option to display the binary's comment section header. This optional binary file section may contain compiler and linker specific version control information. Example output produced by `readelf` in this way is displayed in Figure 4.5. Any discovered information is output as a supplemental description of the application binary.

4.3.2 Shared Library Requirements

The FEAM BDC aims to discover an application binary's shared library requirements. Two tools are used to investigate these dependencies: the *NIX utility `ldd` and the GNU `binutils` program `objdump`. The `ldd` utility, when called with the `-v` option, prints the shared libraries required by a program along with their locations in the local filesystem and symbol version information. The

`objdump` program, when called with the `-p` option, prints file format specific information including shared library dependencies and symbol versioning information. Example output from `ldd` and `objdump` called in this way is displayed in Figure 4.6. This information is used to determine the `sonames` and linked-against versions of the shared libraries required by the application binary. In the `objdump` output, the shared library dependencies are listed as "needed components" in the "dynamic section". Information about the version requirements of the binary's shared library dependencies is listed under the "version definition" and "version reference" sections. Each shared library's required version (as defined by the ERP model in Section 3) is determined by locating the newest version of each library, and, in particular, the C standard library, required by the binary's symbols. Analogous information is determined using the `ldd` utility.

4.3.3 MPI Stack

The FEAM BDC aims to discover with what MPI implementation (i.e. MVAPICH) an application binary was compiled. This information is determined by examining the list of the application's shared library dependencies discovered per the methods explained in Section 4.3.2. The shared library dependencies will include the link-level dependencies associated with a particular MPI implementation type. That MPI implementation type is identified by searching for dependencies that are also MPI implementation identifiers as per the identification guidelines presented by the ERP model in Section 3.3.1. For example, in Figure 4.4, the description is of an application that depends on the `nsl` and `util` shared libraries, which are identifiers for the MPI implementation type Open MPI.

4.3.4 System Configuration

The FEAM BDC also aims to gather information that will help determine the functionality of a target site's execution configuration. When FEAM runs at a guaranteed execution site, the BDC compiles test MPI programs using the MPI implementation type required by the application binary. These programs are run by the FEAM Target Evaluation Component (as described in Section 4.6.5)

EXAMPLE OUTPUT: LDD

```
libmpi_f90.so.0 => /opt/openmpi-1.4.3-intel/lib/libmpi_f90.so.0
libmpi_f77.so.0 => opt/openmpi-1.4.3-intel/lib/libmpi_f77.so.0
libmpi.so.0 => /opt/openmpi-1.4.3-intel/lib/libmpi.so.0
libopen-rte.so.0 => /opt/openmpi-1.4.3-intel/lib/libopen-rte.so.0
libopen-pal.so.0 => /opt/openmpi-1.4.3-intel/lib/libopen-pal.so.0
libdl.so.2 => /lib64/libdl.so.2
libnsl.so.1 => /lib64/libnsl.so.1
libutil.so.1 => /lib64/libutil.so.1
libgfortran.so.1 => /usr/lib64/libgfortran.so.1
libm.so.6 => /lib64/libm.so.6
libgcc_s.so.1 => /lib64/libgcc_s.so.1
libpthread.so.0 => /lib64/libpthread.so.0
libc.so.6 => /lib64/libc.so.6

Version information:
 libc.so.6 (GLIBC_2.2.5) => /lib64/libc.so.6
 libm.so.6 (GLIBC_2.2.5) => /lib64/libm.so.6
```

EXAMPLE OUTPUT: OBJDUMP

```
Dynamic Section:
  NEEDED      libmpi_f90.so.0
  NEEDED      libmpi_f77.so.0
  NEEDED      libmpi.so.0
  NEEDED      libopen-rte.so.0
  NEEDED      libopen-pal.so.0
  NEEDED      libdl.so.2
  NEEDED      libnsl.so.1
  NEEDED      libutil.so.1
  NEEDED      libgfortran.so.1
  NEEDED      libm.so.6
  NEEDED      libgcc_s.so.1
  NEEDED      libpthread.so.0
  NEEDED      libc.so.6

Version References:
  required from libc.so.6:
    0x09691a75 0x00 03 GLIBC_2.2.5
  required from libm.so.6:
    0x09691a75 0x00 02 GLIBC_2.2.5
```

Figure 4.6: Example analysis of application binary's shared library requirements. The FEAM BDC determines an application's shared library requirements by employing the *NIX utility `ldd` and the GNU `binutils` program `objdump` to examine the binary's dependencies. In this example, output from the tools shows the `sonames` and the version requirements for each of the binary's 13 shared library dependencies.

to assess the ability of a target site's MPI stack to run a program created by an MPI stack that was able to run the application.

4.4 Resolution Component

The FEAM Resolution Component (RC) aims to collect information to support the resolution of missing shared library requirements at target sites. Specifically, the FEAM RC creates a store of an application's shared library dependencies along with their application descriptions to be later used by the FEAM Target Evaluation Component to form an execution readiness prediction (as explained in Section 4.6).

The FEAM RC gathers copies of an application's shared library dependencies from a guaranteed execution site. (Licensing issues are outside the scope of this work.) To be able to make copies of the libraries, they must be first located in the local filesystem. If available, the libraries are found using the location information provided by the `ldd` utility (as described in section 4.3.2). Otherwise, files with matching filenames are searched for. If available, the `locate` utility is used to perform the search. Alternately, the `find` utility is used to search at specific locations. These include the paths specified in the LD_LIBRARY_PATH environment variable, the list of directories consulted by the dynamic loaders in search of shared objects. Other locations where shared libraries are typically found on *NIX systems (i.e. /lib) are also searched. A list of search locations is specified in a configuration file that can be augmented by the user. The search list includes system-specific shared library locations identified by examining the locations of dependencies of locally-compiled MPI applications.

Once copies of an application's shared library dependencies have been collected, an application description is created for each copy by employing the FEAM BDC described in Section 4.3. When target sites are found to be missing shared library dependencies, these descriptions provide the FEAM Target Evaluation Component information for making execution readiness predictions about the copies according to the resolution scheme (as described in Section 4.6).

ENVIRONMENT DESCRIPTION

```
--Hostname:  i134                                **USER-ENVIRONMENT MANAGEMENT TOOLS
--Binary: mpi.out                                --Environment Modules: V 3.2.8
                                                 --SoftEnv: no
**ARCHITECTURE
--ISA: x86                                       **MPI STACKS
--Bits: 64                                       --Default: none
                                                 --Available:
**OS                                                /opt/mpich2/1.4
--Name: Red Hat Enterprise Linux Server             /opt/mvapich2/1.7a2
--Version: 5.7                                      /opt/openmpi-1.4.3-intel
                                                    /N/soft/openmpi/1.4.2
**LIBC                                              /opt/openmpi-1.4.3-gnu
--Location: /lib64/libc.so.6
--Version: 6                                     **MISSING SHARED LIBRARIES
--Release Version: 2.5                           --Soname: libmpich
--Compiled with CC version: 4.1.2 20080704       --Version: 1.2
 (Red Hat 4.1.2-51)
```

Figure 4.7: Example environment description. The FEAM Environment Description Component discovers information about a computing site. The resulting *environment description* contains information about the hardware architecture, MPI implementations, and shared libraries present at the computing site. For instance, the computing site in this example is described as having an x86 64-bit architecture and supporting three MPI implementation types.

4.5 Environment Description Component

The FEAM Environment Description Component (EDC) gathers information about a computing site to create an environment description. An example description is displayed in Figure 4.7. The EDC investigates the factors (hardware architecture, MPI stack, shared libraries, and system configuration) outlined by the ERP model as relevant for execution prediction from the perspective of a computing site. This section presents the techniques the FEAM EDC employs to gather information about the execution factors.

4.5.1 MPI Stack

The FEAM EDC aims to discover what MPI stack combinations are available at a computing site as well as what MPI stack combination, if any, the computing site is configured for by default. To determine if any MPI stacks match the application binary's requirements, the MPI stacks available at a computing site must be identified. A site's default MPI configuration is investigated in order to form guidelines for how a site's configuration should be modified to prepare it for application

ENVIRONMENT MODULES

```
$module list
1) torque/2.5.5        2) moab/5.4.0          3) openmpi/1.4.3-gnu

$module avail
------------------ /opt/Modules/3.2.8/modulefiles/applications -----------------
R/2.11.1(default)      mpich2/1.4             soapdenovo/1.04
git/1.7.8.3            mpiexec/0.84           velvet/1.0.15
gromacs/4.0.7(default) mvapich2/1.7a2         wgs/6.1
hpcc/1.3.1(default)    ncbi/2.2.23(default)
hsi/3.5.3              ruby/1.9.3
-------------------- /opt/Modules/3.2.8/modulefiles/compilers ------------------
cmake/2.8.1(default)       intel/11.1(default)        java/1.6.0-x86_64(default)
intel/10.1                 java/1.6.0-i586
-------------------- /opt/Modules/3.2.8/modulefiles/libraries ------------------
intelmpi/4.0.0.028(default)  openmpi/1.4.3-multi-threads
mkl/10.2.5.035(default)      otf/1.7.0(default)
openmpi/1.4.2                unimci/1.0.1(default)
openmpi/1.4.3-gnu(default)   vampirtrace/intel-11.1/5.8.2
openmpi/1.4.3-intel
```

Figure 4.8: MPI stack discovery with environment modules. When possible, the EDC employs user-environment management tools, like Environment Modules, to determine a computing site's available and default MPI configurations. In this example, two Environment Module commands reveal that the site supports three MPI implementations and is configured by default for Open MPI.

execution.

The FEAM EDC first attempts to employ user-environment management tools for MPI stack discovery. User-environment management tools, such as Environment Modules [10] and SoftEnv [11], help manage a shell environment and support the discovery of software packages. They provide a means to alter and set environment variables in order to configure the shell for a particular usage, such as compiling an application with a specific compiler or using a specific version of a shared library. To determine if user-environment management tools are present at a computing site, a search is performed for tool-specific configuration files (i.e. `.modules` for Environment Module). If present, the tool's search mechanisms for examining software packages is used to investigate what MPI stacks are available at the computing site in general (i.e. `module avail` for Environment Module). The tool's mechanisms is also used for identifying the current site configuration to investigate if the site is configured for any MPI stack by default (i.e. `module list` for Environment Module). Example output from examining a site's available software packages and default configuration with Environment Modules is depicted in Figure 4.8.

If user-environment management tools are not available at a site, the EDC attempts to discover information about the site's MPI configurations by searching for MPI libraries and wrapper compilers. Shared libraries associated with MPI implementations are searched for in order to find installed MPI implementations. The library search methods are applied to locate the MPI implementation identifying libraries outlined by the ERP model in Table 3.1. Commonly-used wrapper compilers are also identified. Wrapper compilers for MPI programs, such as `mpicc`, transparently augment a compilation command with relevant compiler and linker flags necessary for MPI program compilation and then invoke the specified compiler. The versioning information of these compilers can reveal what MPI implementations are available at a computing site. To determine if any MPI stack is configured by default at a site, the paths specified by the PATH and LD_LIBRARY_PATH environment variables are examined to identify whether they correspond to the locations of any of the MPI identifying libraries and wrapper compilers.

4.5.2 Shared Libraries

The FEAM EDC aims to discover what shared libraries are present at a computing site. However, instead of collecting information about all the shared libraries present at a computing site, the focus is on discovering information about the application binary's shared library dependencies. Specifically, missing libraries are identified along with the versions of available libraries with matching *sonames*. Making an exhaustive list of shared libraries that are available at a computing site is impractical as shared libraries can be installed at various locations and their versions are often updated. However, when using FEAM often at a target site to make execution readiness predictions for various binaries, the search process could be refined to consult a cache of shared libraries previously discovered to exist at the site.

If the application binary is present at a target site, the GNU Utility `ldd` is employed to identify which of the application's shared library dependencies are missing. The tool indicates with a "not found" message when it is unable to locate a dependency in the local filesystem. If the application binary is not available at the target site, the library search methods are applied. In either case, the GNU Utility `objdump` is used to determined the version information of found shared libraries as

EXAMPLE OUTPUT: UNAME

```
Linux i136 2.6.18-308.11.1.el5 #1 SMP x86_64 x86_64 x86_64 GNU/Linux
```

Figure 4.9: Example analysis of a computing site's hardware architecture. The FEAM EDC determines a computing site's hardware architecture configuration by employing the *NIX utility `uname`.

described in Section 4.3.2.

Instead of searching for the C standard shared library (as many libraries begin with the name libc), an investigation is made into what version of the library is running at the computing site. This is done via the C standard library application programming interface. Specifically, the `gnu_get_libc_version` function is called. Alternately, the C library binary is invoked to print general information about the library including its version. In this case, the library binary is located using the library search methods.

4.5.3 Hardware Architecture

The FEAM EDC aims to discover a computing site's hardware architecture configurations. The *NIX utility `uname` is employed for this task. The utility is called with the `-a` option to view all system information. An example of the output produced by `uname` in this way is displayed in Figure 4.9.

Additional information is gathered about a computing site's operating system. By convention in Unix-based systems, information about the operating system type and version can be found in files under the `/proc` and `/etc` directories. Any discovered information is output as a supplemental description of the computing site. If registries of site information are known to exist, they could be used to augment this description. None of this information is necessary for the operation of FEAM. It is gathered as supplemental information for the user.

4.6 Target Evaluation Component

The FEAM Target Evaluation Component (TEC) forms a prediction of execution readiness based on the ERP model and resolution scheme. Compatibility is assessed for the application binary and the execution site according to the ERP model guidelines by analyzing the information captured

in the application and environment descriptions. If FEAM previously executed at a guaranteed execution site, the resolution scheme is applied as appropriate. This section describes the process by which the FEAM TEC arrives at a final prediction of an application's execution readiness.

4.6.1 Hardware Architecture Requirements

An application description contains information that the FEAM BDC discovered (as described in Section 4.3.1) about the hardware architecture of an application binary. An environment description contains information that the FEAM EDC discovered (as described in Section 4.5.3) about the hardware architecture of a target site. To determine if the hardware architecture execution factor is satisfied, the number of bits and the ISA for which an application was compiled is matched against the corresponding computing site information. To be considered compatible, there must be an exact match of the ISA type and the required number of bits must be less than or equal to the available number of bits. If both conditions are met, the hardware architecture requirements are satisfied as per the ERP model compatibility guidelines in Section 3.3.3.

4.6.2 C Library Version Requirements

An application description contains information that the FEAM BDC discovered (as described in Section 4.3.2) about the C library version required by an application binary. An environment description contains information that the FEAM EDC discovered (as described in Section 4.5.2) about the version of the C library installed at a target site. A determination is made about whether the C library version found to be installed at the computing site is equal to or greater than the version found to be required by the application binary. If so, then the C library version requirement is satisfied as per the ERP model compatibility guidelines in Section 3.3.2. A differentiation is made between an exact match of the major versions which guarantees full compatibility and a mismatch between the major versions that may result in some compatibility issues.

4.6.3 MPI Stack Requirements

An application description contains information discovered by the FEAM BDC (as described in Section 4.3.3) about the MPI stack used to compile an application binary. An environment description contains information discovered by the FEAM EDC (as described in Section 4.5.1) about the MPI stacks available at a target site. To determine whether a compatible MPI stack is available at the target site, the MPI implementation type required by the application is matched against the types found to exist at the target site. If at least one matching MPI type is found, the MPI stack requirement is satisfied per the ERP model compatibility guidelines in Section 3.3.1. If no matching MPI stacks are accessible at the target site by default, a note is made about how to make the configuration either via user-management tools or by setting environment variables.

4.6.4 Shared Library Requirements

An environment description lists which of an application's required shared libraries the FEAM EDC found (as described in Section 4.5.2) to be missing at a target site. If no shared library dependencies were found to be missing, the shared library requirements are met as per the ERP model compatibility guidelines in Section 3.3.2. Otherwise, an attempt is made to resolve the missing dependencies. However, resolution schemes can only be applied if FEAM was previously run at a guaranteed execution site and gathered the additional input needed for this task. This input consists of copies and application descriptions of all of an application binary's shared library dependencies. Thus, if the input is available, the TEC locates each missing library copy and determines if it can execute at the computing site by recursively applying the ERP model. If all missing shared library copies are predicted to be ready for execution, the missing shared library dependencies are resolvable and the application's shared library requirements are met by the target site. A note is made of the configuration details for making these libraries accessible for use at runtime by adding their location to the LDD_LIBRARY_PATH environment variable in the site-specific configuration script.

4.6.5 System Configuration Requirements

The functionality of the system configuration for job execution is assessed by running a series of tests. Tests are performed for each compatible MPI stack as described in Section 4.6.3. Test MPI programs are compiled using the compatible MPI stack and execution of each resulting binary is attempted. If execution succeeds, the MPI stack and system configuration requirements are satisfied as per the ERP model compatibility guidelines in Section 3.3.4. If execution fails, the errors are examined for cause indicators that may generalize to any MPI execution.

If an input bundle was provided from running FEAM at a guaranteed execution site, the test MPI programs generated at the guaranteed execution site (as described in Section 4.3.4) are also run. The successful execution of these tests will further support compatibility between the target site's MPI stacks and the application's MPI stack requirements. Similarly, if errors are detected, they are examined for cause indicators that may affect the execution of any binary from that guaranteed execution site.

Running MPI test programs on compute nodes requires knowing the execution command that corresponds to the selected MPI stack. As the syntax of a submission script for parallel execution may vary greatly between computing sites due to customized configurations, this syntax is not automatically detected. Rather, it is specified by the user in a template as described in Section 4.1.

4.6.6 Prediction

The TEC makes the final prediction that an application is ready for execution at a target site if all of the execution factors are found to be satisfied according to the ERP model. Example predictions are illustrated in Figure 4.10. When predicting a binary is unable to execute at a target site, the TEC forms a description of the detected issues. An example error report is presented in Figure 4.11. This information is output along with the general binary and environment descriptions created by the other FEAM components. Additionally, the TEC generates a script with site-specific configuration details that incorporates a resolution scheme when possible. An example configuration script is presented in Figure 4.12.

PREDICTIONS

```
POSITIVE PREDICTION
Ready to execute

NEGATIVE PREDICTION
Not ready to execute:
 Missing shared libraries
```

Figure 4.10: Example predictions. FEAM's TEC forms the final prediction about whether an application is ready to execute at a target site. These examples illustrate the high-level prediction summaries of readiness and failure output by FEAM.

ERROR REPORT

```
*Architecture Bitness Mismatch
--Required: 64
--Available: 32

*MPI Mismatch
--Required: mva
--Available:
mpich2/1.4
openmpi/1.4.3-intel
openmpi/1.4.2(default)
openmpi/1.4.3-gnu

*C Library Version Mistmatch
--Required: 2.12
--Available: 2.11.1

*Missing Shared Libraries
libpthread.so.0
librt.so.1
libc.so.6
```

Figure 4.11: Example error report. FEAM's TEC forms an error report of detected issues when predicting a binary is unable to execute at a target site. This report example presents a mismatch in the architecture, MPI, C library version, and shared library requirements.

SITE-SPECIFIC CONFIGURATION SCRIPT

```
#!/bin/bash

# Job Sumbission Script
# Autogenerated by FEAM's Submission Script Creator
# at HOST

#PBS -j oe
#PBS -o <NAME>
#PBS -l walltime=00:10:00
#PBS -l nodes=2:ppn=1

cd $PBS_O_WORKDIR

module load mpich2/1.4

export LD_LIBRARY_PATH=/path/TEC-resolution:/path/lib

mpiexec <BINARY>
```

Figure 4.12: Example configuration script. In addition to a prediction of the execution readiness of an application binary, FEAM outputs site-specific configuration details for setting up a target site for the execution of the binary.

4.7 Summary

This chapter presented FEAM, an implementation of the execution readiness prediction model for Unix-based systems. FEAM's components employ Unix-based techniques to discover information about application binaries and computing sites. In addition to predicting execution readiness as per the ERP model, FEAM incorporates a scheme for resolving execution-blocking issues related to shared library dependencies. FEAM can be used by users and scheduler to improve scheduling freedom. The next chapters present an evaluation of FEAMs effectiveness.

Chapter 5

Performance Evaluation

The performance of FEAM was evaluated in terms of three metrics: (1) prediction accuracy, (2) resolution effectiveness, and (3) footprint. *Prediction accuracy* measured FEAM's ability to correctly predict execution readiness. *Resolution effectiveness* measured FEAM's ability to enable execution readiness using resolution schemes. FEAM's *footprint* in terms of time and space usage was also examined. The evaluation results indicate that FEAM is a lightweight framework that provides accurate predictions of execution readiness while enabling more successful executions overall via resolution schemes. As such, FEAM was shown to be ready to be of value to users and schedulers to increase deployment freedom.

The applications and computing sites used for the evaluation are discussed first in Section 5.1 followed by a description of the prediction accuracy evaluation in Section 5.2 and the resolution effectiveness evaluation in Section 5.3. FEAM's footprint in terms of time and space requirements is discussed in Section 5.4.

5.1 Test Set Description

The evaluation was conducted using 13 applications across five computing sites. The test set was created to consist of a diverse set of applications and computing sites. Diversity of applications was sought in terms of programming languages, scientific domains, and algorithms. Diversity of computing environments was sought in terms of parallel architecture, operating system, and MPI

implementation. The applications and computing sites selected for the evaluation are described in this section.

Table 5.1: Test site characteristics. The five computing sites used for the evaluation were chosen for their hardware and software configuration diversity. This table lists the sites' parallel architecture type, operating system, C library version, compilers, and MPI implementations.

Computing Site (Type - Cores)	**Operating System**	**C Library & Compiler**	**MPI Type (Compilers:*g*nu, *i*ntel, *p*gi)**
XSEDE Ranger at Texas Advanced Computing Center (MPP - 62,976)	CentOS v4.9	LibC v2.3.4 GNU CC v3.4.6 Intel v10.1	Open MPI v1.3 (*g*,*i*,*p*) MVAPICH2 v1.2 (*g*,*i*,*p*)
XSEDE Forge at National Center for Supercomputing Applications (Hybrid - 576)	Red Had Enterprise Linux Server v6.1	LibC v2.12 GNU CC v4.4.5 Intel v12	Open MPI v1.4 (*g*,*i*) MVAPICH v1.7rcl(*i*)
XSEDE Blacklight at Pittsburgh Supercomputing Center (SMP - 4,096)	SUSE Linux Enterprise Server v11	LibC v2.11.1 GNU CC v4.4.3 Intel v11.1	Open MPI v1.4 (*g*,*i*)
FutureGrid India at Indiana University (Cluster - 920)	Red Hat Enterprise Linux Server v5.6	LibC v2.5 GNU CC v4.1.2 Intel v11.1	Open MPI v1.4 (*g*,*i*) MVAPICH v1.7a2 (*i*) MPICH2 v1.4 (*i*)
ITS Fir at University of Virginia (Cluster - 1,496)	CentOS v5.6	LibC v2.5 GNU CC v4.1.2 Intel v12	Open MPI v1.4 (*g*,*i*, *p*) MVAPICH2 v1.7a (*g*,*i*, *p*) MPICH2 v1.3 (*g*,*i*, *p*)

5.1.1 Computing Sites

The computing site test set consisted of the five systems listed in Table 5.1. Three state-of-the-art high performance computing systems were chosen from the XSEDE infrastructure [20] to represent the main types of parallel architectures found at national supercomputing centers. Ranger [15] at the Texas Advanced Computing Center is a massively parallel processing (MPP) system. Forge [2] at the National Center for Supercomputing Applications was a hybrid CPU/GPU system. Blacklight [1] at the Pittsburgh Supercomputing Center is a symmetric multiprocessing (SMP) system. Two mid-sized university clusters were also selected. India is part of the FutureGrid Project [3] test-bed while Fir is a University of Virginia resource [17].

The test set sites have diverse software configurations. The chosen systems run different versions of three Linux operating system variants: RedHat, CentOS, and SUSE. Their installed standard C library versions ranged from 2.3 to 2.12 with only two of the systems being configured with the same version. Each site supports at least one version of the three most popular open-source MPI implementations (MVAPICH, MPICH, and OpenMPI). The sites also support a wide variety of compilers and shared libraries. Table 5.1 lists more details about the characteristics of the five computing sites used for the evaluation.

5.1.2 Applications

The set of MPI application binaries used for the evaluation consisted of 13 applications from two benchmarks suites: the NAS Parallel Benchmarks [48] suite and the SPEC MPI2007 [39] benchmark suite. The NAS Parallel Benchmarks (NPB) suite consists of applications derived from computational fluid dynamics. The SPEC MPI2007 benchmark suite was developed from native MPI-parallel end-user applications. The evaluation used version 2.4 of the MPI reference implementation of the NPB suite and version 2.0 of the SPEC MPI2007 benchmark suite with medium sized inputs.

To create the test set of MPI application binaries, the benchmark codes were compiled using various MPI stacks at the five selected test sites. The number of binaries created using the benchmark codes varied depending on whether compilation was successful with each MPI stack available at a test site. Binaries that would not execute at their compilation site were discarded. As a result, the final set of binaries used for the evaluation consisted of a subset of the chosen benchmark codes.

In total, the final application test set included 110 NPB and 147 SPEC MPI2007 binaries. From the NPB suite, the final test set consisted of four kernels (integer sort, embarrassingly parallel, conjugate gradient, and multi-grid on a sequence of meshes) as well as three pseudo applications (block tri-diagonal solver, scalar penta-diagonal solver, and lower-upper Gauss-Seidel solver). From the SPEC MPI2007 benchmark suite, the final test set consisted of a quantum chromodynamics code, two computational fluid dynamics codes (`107.leslie3d` and `115.fds4`), a parallel ray tracing code (`122.tachyon`), a molecular dynamics simulation code (`126.lammps`), a weather prediction code (`128.GAPgeofem`), and a 3D Eulerian hydrodynamics code (`129.tera_tf`). Tables 5.2 and 5.3 de-

tail the final application test set in terms of which MPI stacks were used to compile the benchmark codes at each test site.

5.2 Prediction Accuracy

The aim of the prediction accuracy evaluation was to measure FEAM's ability to correctly predict execution readiness. The evaluation considered FEAM's prediction accuracy with and without the incorporation of a resolution scheme. In this section, the setup for the prediction accuracy evaluation is discussed along with the evaluation results.

5.2.1 Methodology

The accuracy of FEAM's predictions was investigated using information from target and guaranteed execution sites. FEAM was used to predict execution readiness and form a resolution scheme when applicable for each binary and site in the test set. Each binary was also executed at each site. Accuracy was calculated by comparing the predictions of execution readiness with the execution results.

To gather execution results, target sites had to be configured with MPI implementations that matched the requirements of each binary. As the MPI implementation often affects the format of a job submission script, execution results could only be gathered after selecting the correct MPI implementation. For example, for binaries compiled with MVAPICH2 at the Ranger site, execution results could only be evaluated at the Forge, India, and Fir sites where MVAPICH2 is available. For sites with no matching MPI implementation, an execution result of failure was recorded. These failure results were confirmed by attempting to run the applications with non-matching MPI implementations. An additional set of execution results was collected to evaluate the accuracy of FEAM's resolution schemes. For the set of tests that were predicted to be able to execute if a resolution scheme was applied, execution results were gathered anew after incorporating the resolution schemes into the job configurations. All of these execution results are presented in Figure 5.1.

Table 5.2: SPEC application test set. This table indicates the MPI stack combinations used to compile the SPEC MPI2007 benchmark suite at five sites to create the binaries used for the evaluations. (C:successful, X: discarded due to compile or execution issues, -: unavailable)

Benchmark Code (Application Area)	**Test Site**	**MVAPICH**			**Open MPI**			**MPICH**		
		g	*i*	*p*	*g*	*i*	*p*	*g*	*i*	*p*
SPEC milc (quantum chromodynamics)	Blacklight	-	-	-	C	C	-	-	-	-
	Fir	C	C	C	C	X	C	C	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	C	C	C	C	C	C	-	-	-
SPEC leslie3d (computational fluid dynamics)	Blacklight	-	-	-	X	C	-	-	-	-
	Fir	C	C	C	C	C	C	C	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	C	C	C	C	C	C	-	-	-
SPEC fds4 (computational fluid dynamics)	Blacklight	-	-	-	X	C	-	-	-	-
	Fir	X	C	C	X	C	C	X	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	X	-	X	X	-	-	X	-
	Ranger	C	C	C	C	C	C	-	-	-
SPEC tachyon (parallel ray tracing)	Blacklight	-	-	-	C	C	-	-	-	-
	Fir	C	C	C	C	X	C	C	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	C	C	C	C	C	C	-	-	-
SPEC lammps (molecular dynamics)	Blacklight	-	-	-	C	C	-	-	-	-
	Fir	C	C	C	C	X	X	C	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	X	X	X	C	C	C	-	-	-
SPEC GAPgeofem (weather prediction)	Blacklight	-	-	-	X	C	-	-	-	-
	Fir	C	C	C	C	C	C	C	C	C
	Forge	-	C	-	C	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	C	C	C	C	C	C	-	-	-
SPEC tera_tf (3D Eulerian hydrodynamics)	Blacklight	-	-	-	X	C	-	-	-	-
	Fir	C	C	C	C	C	C	C	C	C
	Forge	-	C	-	X	C	-	-	-	-
	India	-	C	-	C	C	-	-	C	-
	Ranger	X	C	C	X	C	C	-	-	-

Table 5.3: NAS application test set. This table indicates the MPI stack combinations used to compile the NAS Parallel Benchmark suite at four site to create the binaries used for the evaluations. (C:successful, X: discarded due to compile or execution issues, -: unavailable)

Benchmark Code	**Test Site**	**MPICH**	**MVAPICH**	**Open MPI**	
		g	*g*	*g*	*i*
NAS bt	Fir	C	X	C	C
	India	C	C	C	X
	Lincoln	-	C	X	C
	Ranger	-	C	C	C
NAS cg	Fir	C	X	C	C
	India	C	C	C	C
	Lincoln	-	C	X	C
	Ranger	-	C	C	C
NAS ep	Fir	C	X	C	C
	India	C	C	C	C
	Lincoln	-	C	X	C
	Ranger	-	C	C	C
NAS is	Fir	C	X	C	X
	India	C	C	C	C
	Lincoln	-	C	C	C
	Ranger	-	C	C	C
NAS lu	Fir	C	X	C	C
	India	C	C	C	C
	Lincoln	-	C	C	C
	Ranger	-	C	C	C
NAS mg	Fir	C	X	C	C
	India	C	C	C	C
	Lincoln	-	C	X	C
	Ranger	-	C	C	C
NAS sp	Fir	C	X	C	C
	India	C	C	C	C
	Lincoln	-	C	C	C
	Ranger	-	C	C	C

To gather predictions, FEAM was run at target and guaranteed execution sites (using job submission templates based on the submission scripts used to gather execution results). Two types of predictions were formed: (1) basic and (2) extended. FEAM formed *basic predictions* using information gathered only at target sites. FEAM formed *extended predictions* by additionally incorporating information gathered at guaranteed execution sites. (Each binary's compilation site was

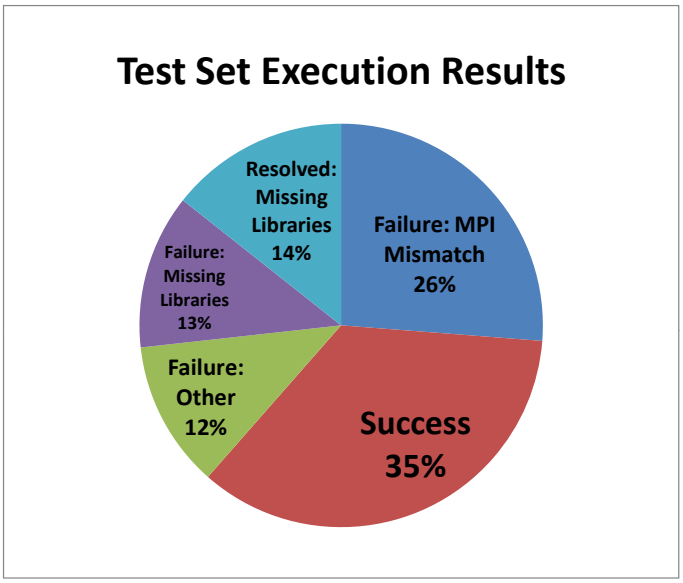


Figure 5.1: Test set execution results. The results of running each test case were recorded. This figure presents the percentage of tests that succeeded as well as the percentage of tests that failed (decomposed by the cause of failure). The tests that initially failed but run successfully using FEAM's resolution scheme are also noted.

used as the guaranteed execution site.) As described in Section 4.1, running FEAM at a guaranteed execution site enables the formation of resolution schemes and enhanced compatibility testing at target sites. In this way, the accuracy evaluation differentiates FEAM's performance for when there is and is not access to a guaranteed execution site, a particularly relevant consideration for users of community applications that may be distributed as binaries. The accuracy of both types of predictions was evaluated by comparing against the corresponding execution result. These accuracy results are presented in Figure 5.2.

5.2.2 Result Analysis

The accuracy evaluation shows that FEAM is able to recognize a variety of execution-blocking issues and, as a result, produce correct predictions of execution readiness. Regardless of whether formed using information from target sites or also from guaranteed execution sites, the prediction accuracy evaluation measured FEAM's predictions as more than 90% accurate. The evaluation also illustrated how execution is influenced by the ERP model's execution factors.

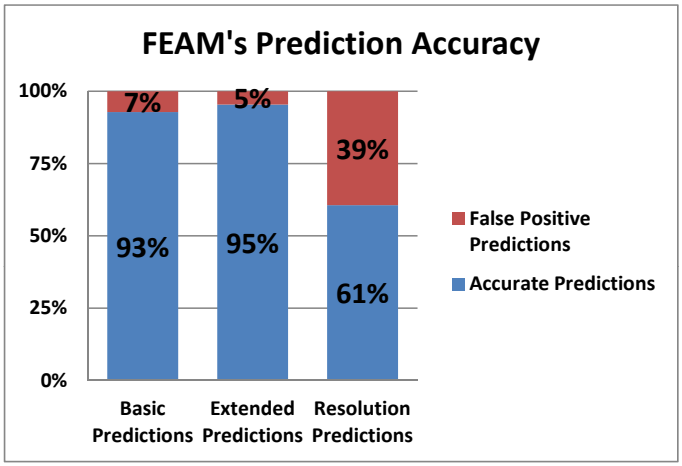


Figure 5.2: FEAM's prediction accuracy. FEAM's accuracy at predicting execution readiness was calculated by comparing predictions with execution results. Basic predictions were formed using information gathered only at target sites while extended predictions were formed by additionally incorporating information gathered at guaranteed execution sites. This figure presents the accuracy of both prediction types as percentages over all tests. The figure also presents the accuracy of resolution schemes in terms of enabling successful executions when applied. These results are listed as percentages of formed resolutions.

The accuracy results (presented in Figure 5.2) show that FEAM correctly predicted execution readiness for more than 90% of the tests regardless of prediction type. Extended predictions produced more accurate results (95% vs 93%). The increase in accuracy was a consequence of the additional compatibility tests that these extended predictions were able to run using information gathered at guaranteed execution sites. For example, by running MPI test programs compiled at guaranteed execution sites, FEAM was able to detect floating point errors and application binary interface (ABI) incompatibilities in shared libraries.

FEAM's inaccurate predictions were all of execution readiness. These false positives were a result of errors that FEAM was not able to recognize. For example, FEAM incorrectly predicted applications would successfully execute when instead they failed due to MPI daemons failing to spawn and communication timeouts. The errors that FEAM was unable to recognized were mostly related to system configurations. A minority of the errors were related to issues with shared libraries. FEAM was 100% accurate at identifying issues related to the MPI stack and hardware architecture requirements.

The evaluation of the accuracy of FEAM's resolution schemes found that FEAM was 39% inaccurate at predicting whether applying the schemes would lead to successful execution. FEAM made these false positive predictions of execution readiness when it did not recognize other causes of failure. These failures are most prevalent in the accuracy evaluation of FEAM's resolution schemes because in this case the other execution factors are all satisfied. When FEAM detects issues with the ERP model's execution factors, it forms an accurate prediction of execution failure regardless of whether there are additional masked failure causes. To predict that a resolution scheme will enable successful execution, FEAM must have already determined that all other execution factors have been satisfied. Thus, the failures that FEAM is not able to detect become apparent. While about half of FEAM's false positive predictions occurred in these circumstances, overall these inaccurate predictions made up less than 5% of the tests.

The execution results used to validate FEAM's predictions of execution readiness (presented in Figure 5.1) show that about a third (35%) of the tests executed without errors when the only configuration in place to gather these results was the selection of a matching MPI implementation. About another quarter (26%) of the tests failed due to target sites lacking matching MPI implementations. The majority of the remaining 39% of the tests failed due to missing shared libraries. Other failure causes included incompatible C library versions, floating point exceptions, and system errors. These results emphasize that choosing a target site based only on the existence of a matching MPI implementation is not enough to guarantee successful execution. To enable the execution of an application binary without recompilation, it is especially important to handle missing shared libraries.

5.3 Resolution Effectiveness

The aim of the resolution effectiveness evaluation was to measure the ability of FEAM's resolution schemes to enable successful executions. FEAM composes resolution schemes upon detecting missing shared libraries during its evaluation of shared library compatibility for execution readiness prediction. Resolution schemes, as described in Section 4.2, determine if missing shared libraries

can be made available at target sites using library copies gathered at guaranteed execution sites. In this section, the setup of the resolution effectiveness evaluation is discussed along with the evaluation results.

5.3.1 Methodology

The effectiveness of FEAM's resolution scheme was evaluated by determining how often the schemes were created and how often they enabled enabled successful execution over the test set. To assess the prevalence of resolution schemes, FEAM's extended predictions were examined. (FEAM composes resolution schemes only when forming extended predictions as scheme creation requires library copies to have been gathered at guaranteed execution sites.) Only full schemes that resolved all missing shared libraries were counted. The success of the schemes was assessed by measuring the results of execution attempts that incorporated the resolution schemes.

The effectiveness evaluation analyzed the formation of FEAM's resolution schemes in relation to the need for resolution schemes. How often FEAM was able to compose full schemes was contrasted with how often shared libraries were missing. (FEAM only considers forming resolution schemes when required shared libraries are identified as missing at target sites.) To express how effective FEAM was at forming schemes to handle the issue of missing shared libraries, a measure of *coverage* was calculated as the percentage of the tests with missing shared libraries for which full resolution schemes were formed. This calculation is expressed per the following equation where the quantity $P_{MissingLibs}$ denotes the number of extended predictions of failure due to missing shared library requirements and the quantity $P_{Resolved}$ denotes the number of extended predictions for which FEAM composed full resolution schemes to resolve missing shared library requirements:

$$\text{Coverage} = P_{Resolvable} / P_{MissingLibs}$$

The effectiveness evaluation also analyzed the success of FEAM's resolution schemes in relation to the execution results. To express how effective FEAM's resolution schemes were at enabling execution, an analysis of *impact* was calculated as the percentage of executions that were impacted by resolution schemes. The impact was additionally calculated in terms of the percentage increase

in the number of successful executions and decrease in the number of failures. These relationships were calculated per the following equations where T_{All} denotes all the executions, $T_{Successful}$ denotes the successful executions, T_{Failed} denotes the failed executions, and $T_{Resolved}$ denotes the tests that were able to execute successfully as a result of applying FEAM's resolution schemes:

$$\text{Overall Impact} = T_{Resolved}/T_{All}$$

$$\text{Success Impact} = T_{Resolved}/T_{Successful}$$

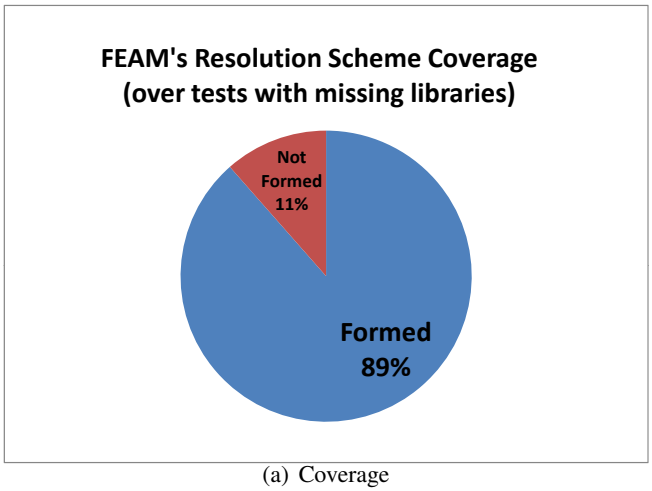
$$\text{Failure Impact} = T_{Resolved}/T_{Failed}$$

Figure 5.3 presents the results of the effectiveness evaluation of FEAM's resolution schemes both in terms of coverage and impact over the MPI application binaries and computing sites in the test set.

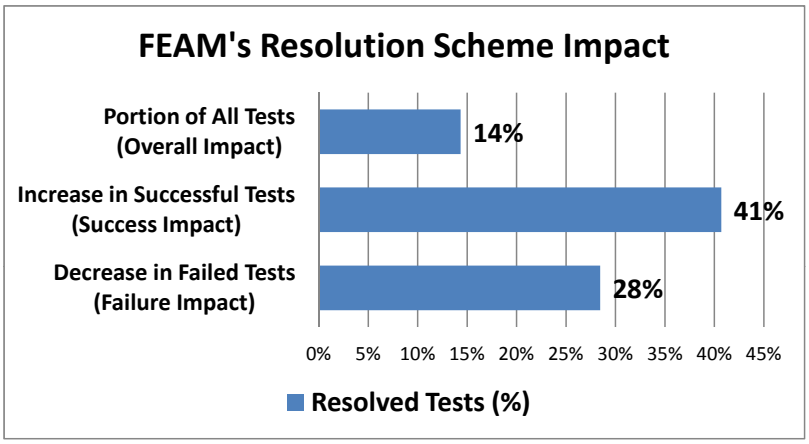
5.3.2 Result Analysis

The evaluation of FEAM's resolution scheme showed that while the scheme had limited effectiveness, it enabled significantly more successful executions over the test set. Overall, 14% of the test cases were impacted favorably by resolution schemes. This equated to a third more successful executions being enabled over the test set by FEAM's resolution schemes.

As the coverage results show, FEAM was able to compose full resolution schemes for 89% of the tests that were missing shared libraries. Resolution schemes were not composed for the other tests with missing libraries due to limitations of the scheme. FEAM's resolution scheme takes advantage of shared library present at guaranteed execution sites but it is also limited by the dependencies of those copies, which are just binaries. The most frequent causes of FEAM not being able to compose full resolution schemes were mismatches between the versions of the C library required by library copies and the versions available at target sites. Note, that FEAM's composition of a full resolution scheme does not equate to FEAM forming a prediction of execution readiness. FEAM may detect other execution-blocking issues, such as an incompatible MPI stack, that will prevent execution regardless of the resolution of missing shared libraries.



(a) Coverage



(b) Impact

Figure 5.3: Resolution effectiveness. The effectiveness of FEAM's resolution scheme was evaluated using two metrics: coverage and impact. The coverage evaluation, presented in figure (a), analyzed the formation of resolution schemes in relation to the need for resolution schemes. The impact evaluation, presented in figure (b), analyzed the success of resolution schemes in relation to the execution results.

As the impact results show, applying FEAM's resolution scheme favorably impacted 14% of the test set by increasing the number of successful executions by 41% and decreasing the number of failures by 28%. These gains were able to occur without any additional effort other than applying the configuration script composed by FEAM's resolution scheme. With only 35% of the tests executing successfully initially (when not applying any configurations related to resolution schemes), a favorable impact of 14% represents a substantial increase in the number of successful executions without having to recompile the application or make any heavyweight changes to the execution environment at the target site.

5.4 Footprint

The aim of the footprint evaluation was to determine FEAM's time and space usage. The time to run FEAM and the amount of disk space used during the process was measured using *NIX operating system commands. The `time` command was used to record the amount of system resources used by FEAM for each test. The command *du* was used to determine the amount of disk space used to run FEAM. This evaluation showed that running FEAM requires on the order of minutes of compute time and MBs of disk space. The timing measurements recorded that FEAM used less than five minutes of compute time to run all of its components on target and guaranteed execution sites. The wall clock time varied with the amount of time spent waiting in the queue to run on a compute node. The space measurements recorded that less than 100 MBs of disk space where needed to store FEAM's inputs, outputs, and installation files as well as the files created during FEAM's execution. The majority of the space was for storing shared library copies used by FEAM's resolution scheme.

5.5 Summary

In this chapter, three evaluations of FEAM, the implementation of the execution readiness prediction model with a resolution scheme, were presented. The *accuracy prediction* evaluation showed that FEAM is able to recognize the vast majority of execution failures to produce a correct prediction of execution readiness with more than 90% accuracy. The *resolution effectiveness* evaluation

showed that while FEAM's resolution schemes positively impacted only 14% of tests, they targeted a prevalent execution-blocking issue whose resolution increased successful executions by 41% and decreased failures by 28%. The *footprint* evaluation showed that FEAM is a lightweight framework that takes on the order of minutes to execute and uses on the order of MBs of disk space. FEAM performed well enough to be useable to increase deployment freedom for users and schedulers.

Chapter 6

Efficiency Evaluation

This chapter presents an evaluation of how much more efficient the preparation of environments for application execution can be when utilizing FEAM, the Framework for Efficient Application Migration. A prediction of the speedup associated with using FEAM is formed by comparing the effort in terms of time of using FEAM versus manual methods. Expenditures of time are analyzed using the Keystroke Level Model [28] as well as data from the user study (from Chapter 2) and the evaluations of FEAM's performance (from Chapter 5). The analysis shows that FEAM provides more than an order of magnitude of speedup in the best case and no significant slowdown over manual methods in the worst case. Additionally, using FEAM potentially eliminates tedious debugging by providing site-specific information about execution factors. The analysis puts into perspective by how much using FEAM decreases the barriers that otherwise must be manually overcome before new computing sites can be used to execute applications.

Section 6.1 first analyzes the effort exerted to use FEAM. Section 6.2 then quantifies the effort spent to manually prepare environments for application execution. Section 6.3 follows by quantifying the overall effort spent to prepare environments for execution when utilizing FEAM. The resulting speedup estimates are presented in Section 6.4 followed by an analysis of the calculations in Section 6.5.

6.1 FEAM Effort

This section presents an analysis of how much effort is exerted to use FEAM in terms of time. Running FEAM involves installing FEAM, preparing a set of inputs, and issuing a start command. This effort is termed T_{FEAM}. An estimate of T_{FEAM} is created by using the Keystroke Level Model (KLM) while gathering experimental data to confirm the efficacy of the encodings. This section first introduces the KLM and then presents KLM encodings of each of the subtasks of the process of using FEAM before presenting an overall estimate of T_{FEAM}.

6.1.1 Keystroke Level Model

Effort is analyzed in terms of time by applying the principles of the Keystroke-Level Model. This model was first developed in the 1980s and, with over six hundred citations, has since been widely used in different variations and extensions to estimate human-computer interaction execution times. The model provides guidelines for predicting the time it takes expert users to perform a specific task on a computer system. Per the KLM, a task is expressed as a set of small, cognitively manageable unit tasks. The KLM provides guidelines to predict the duration of each unit task in terms of the sequence of system commands at the keystroke level used to execute the tasks. The KLM encodes the execution of tasks using four physical-motor operators - keystroking, pointing, homing, and drawing - along with a mental operator and a response operator. The execution time of a task is predicted by summing the execution time of the operators used to encode its execution with constant time estimates used for each operators duration as developed for the KLM. [28]

The analysis of FEAM encodes tasks using the KLM's keystroke (K), response (R), and mental (M) operators. The keystroke operator K refers to keystrokes or button pushes on a keyboard and is determined based on the typing rate. The K operator time is estimated as 0.2 seconds for an average skill typist of 55 wpm. The response operator (R) refers to the amount of time users have to wait for an operation to complete before they can continue performing their task. Each R operator is estimated by measuring the system response time for the task being waited on. The M operator refers to the time users spend mentally preparing to execute an operator such as a keystroke. The M

operator time is estimated using the KLM suggested mental preparation time of 1.35 seconds based on experimental results of expert users. To place M operators in relation to K operators, the KLM's heuristic rules are used. They amount to placing Ms in front of all Ks belonging to one cognitive unit that is not anticipated and not a redundant terminator. Choosing how many Ms to place is the most difficult part of using the KLM. [47]

6.1.2 Method for Task "Install FEAM"

First the time to install FEAM at a computing site is analyzed. The FEAM installation consists of unpacking a downloadable tarball. This task does not need to occur if FEAM was previously installed at a computing site such as after it had been used to predict the execution readiness of another binary. This section models the execution time of the task of installing FEAM at a site by using the *wget* and *tar* commands.

The sequence of commands at the keystroke level to acquire a file using the *wget* command consist of typing the command followed by the URL of the file to be acquired and terminated via the ENTER key. The sequence of commands to extract a tarball consists of typing the *tar* command followed by a set of flags and an argument and terminated via the ENTER key. These operations are encoded using the KLM's K, R, and M operators. For the argument specifying the URL of the FEAM framework, a length of 39 characters is specified as per a possible online location of the tarball. For the argument specifying the location of the downloaded tarball in the local file system, a length of 11 characters is specified as per the name of the tarball. Download and extraction times are also taken into account in the encoding even though during this time the user is not exerting extra effort. These times are too short for the user to overlap them with other tasks. A file sized at less than 10 MB will take less than 90 seconds to download with a 1 Mbps or faster connection while a file sized at less than 10MB will take less than one second to extract. The encoding also includes basic preparatory actions such as creating a new directory for the installation and confirming the files extracted. An estimate of 15 characters is used to specify path locations. Table 6.1 presents the encoding of these operations in sequence along with descriptions and examples.

Using the encoding in Table 6.1 and the operator times proposed by the Keystroke Level Model,

Table 6.1: Method for Task "Install FEAM"

DESCRIPTION	ENCODING	EXAMPLE
Create new directory	2M 22K	mkdir /home/path/new <ENTER>
Navigate to new directory	2M 19K	cd /home/path/new <ENTER>
Start wget command	M 3K	wget
Specify URL	M 39K	www.cs.virginia.edu/ kas9ud/feam.tar.gz
Terminate command	K	<ENTER>
Wait for download	R(90s)	-
Start tar command	M 3K	tar
Specify flags	M 3K	xzf
Specify tarball	M 11K	feam.tar.gz
Terminate command	K	<ENTER>
Wait for tar command to finish	R(1s)	-
Navigate to FEAM directory	2M 9K	cd feam <ENTER>
Confirm installation	M 3K	ls <ENTER>
TOTAL Operators in Encoding	12M 114K R(91)	-

a prediction was formed of the time to execute the method for installing FEAM:

$$T_{InstallFEAM} = 12 * t_M + 114 * t_K + 91 = 130 \text{ seconds}$$

Thus the task of installing FEAM at a new site was predicted to take around 2.2 minutes assuming use of the *wget* and *tar* commands, typing at a speed of 55 wpm, path lengths of 15 characters, and a mental preparation of 1.35 seconds per cognitive unit. The confirmation of the efficacy of this KLM encoding (by measuring the time it took one expert user to perform the same task sequence) found that the time prediction was accurate to within 2 seconds when discounting variance due to a faster download time.

6.1.3 Method for Task "Invoke FEAM"

This section models the time to invoke FEAM while the next two sections consider the time to prepare the inputs for this task. The sequence of commands at the keystroke level to start FEAM consists of typing the FEAM command followed by four arguments and terminated via the ENTER key. This method is encoded using the KLM's K and M operators. The arguments were estimated

Table 6.2: Method for Task"Invoke FEAM'

DESCRIPTION	ENCODING	EXAMPLE
Navigate to FEAM directory	2M 19K	cd /home/path/FEAM <ENTER>
Confirm FEAM directory contents	M 3K	ls <ENTER>
Confirm location of binary/bundle	2M 19K	ls /home/path/binary.sh <ENTER>
Confirm location of template	2M 19K	ls /home/path/template.sh <ENTER>
Call FEAM start command	M 10K	FEAM start
Specify binary/bundle location	M 15K	/home/path/binary.sh
Specify template location	M 15K	/home/path/template.sh
Specify site and node type	2M 4K	ge c
Terminate command	K	<ENTER>
TOTAL Operators in Encoding	12M 105K	-

at 15 characters long to specify the location of a file in a NIX operating system. The encoding also include basic operations such as navigating to the FEAM directory and confirming the locations of the arguments. Table 6.2 lists the encoded operations in sequence along with descriptions and examples.

The duration of FEAM's execution was not incorporated into the encoding as the aim was to only capture the time to invoke FEAM. FEAM's execution time is incorporated into the overall effort estimates of T_{FEAM} in Section 6.1.6.

Using the encoding Table 6.2 and the operator times proposed by the Keystroke Level Model, a prediction was formed of the time to execute the method for invoking FEAM:

$$T_{InvokeFEAM} = 12 * t_M + 105 * t_K = 37.2 \text{ seconds}$$

Thus the task of issuing the command to start FEAM was predicted to take 0.6 minutes assuming typing at 55 wpm, path lengths of 15 characters, and a mental preparation of 1.35 seconds per cognitive unit. The confirmation of the efficacy of this KLM encoding (by measuring the time it took one expert user to perform the same task sequence) found that the time prediction were accurate to within 5 seconds.

6.1.4 Method for Task "Prepare Access to Binary Info"

This section analyzes the time required to compose the input to FEAM that specifies information about the MPI application binary. The exact file provided as input can vary based on the run location (target vs. guaranteed execution site). Access to the MPI application binary can always be specified. Alternately, when FEAM has already run at a guaranteed execution site, the resulting output bundle can be made accessible at target sites instead of the binary. The output bundle includes additional information related to the binary that enables FEAM to form better execution readiness predictions at target sites. Whether in the form of the binary or bundle, this input can be made accessible to FEAM by transferring files from another location. This task does not occur at guaranteed execution sites as the MPI application binary already resides there locally and is ready to be specified as input. This section models the execution time of the task of making the input - whether in the form of a binary or a bundle file - available at target sites by transferring the file using the *scp* command. Using other commandline transfer protocols would result in a similar sequence of commands.

The sequence of commands at the keystroke level to execute file transfer using *scp* consists of typing the command followed by two arguments and terminated via the ENTER key. This method is encoded using the KLM's K, R, and M operators. For the first argument to *scp* specifying the originating location of the file to be transferred, a length of 50 characters is assumed as this string includes the username, hostname, and path to the file. For the second argument specifying the destination of where the file is to be transferred, a length of 15 characters is assumed as this string consists of a location in the local file system. An estimate of 90 seconds is used as the time to complete the file transfer. The encoding also incorporates confirming that the file has transferred. If the time to complete the file transfer is any longer, it is assumed this time will be overlapped with the start of another task. Table 6.3 presents the encoding of these operations in sequence along with descriptions and examples.

Using the encoding in Table 6.3 and the operator times proposed by the Keystroke Level Model, a prediction was formed of the time to execute the method for preparing access to a file:

$$T_{PrepareAccessToFile} = 5 * t_M + 88 * t_K + 90 = 114.35 \text{ seconds}$$

Table 6.3: Method for Task "Prepare Access to Binary Info'

DESCRIPTION	ENCODING	EXAMPLE
Start command	M3K	scp
Specify origin	M50K	user1@host.cs.edu:/home/path/file
Specify destination	M15K	binary.sh
Terminate command	K	<ENTER>
Wait for transfer completion	R(90s)	-
Confirm file transferred	2M19K	ls /home/path/file <ENTER>
TOTAL Operators in Encoding	5M 88K R(90s)	-

Thus the task of transferring a file to a new computing site - be it an MPI application binary or a FEAM output bundle - is predicted to take 1.9 minutes assuming use of the *scp* command, typing at 55 wpm, argument lengths of 50 and 15 characters, and a mental preparation time of 1.35 seconds per cognitive unit. The confirmation of the efficacy of this KLM encoding (by measuring the time it took one expert user to perform the same task sequence) found that the time prediction were accurate to within 2 seconds when discounting variance in download time.

6.1.5 Method for Task "Specify Job Submission Info"

This section analyzed the time required to compose FEAM's input that specifies the location of a template for job submission. As this syntax is site dependent, it must be specified at any location where FEAM is to be run. Because it can vary greatly from site to site due to job manager type and local customized settings, the submission format is the only site information that is not automatically discovered by FEAM. This task does not need to to occur if FEAM was previously installed and used at a computing site as this template – whether created by a user or administrator – will already exist and be ready for use. This section models the execution time of the task of creating such a template using the VI commandline editor. Regardless of what commandline editor is used, the sequence of commands would be similar.

The sequence of commands at the keystroke level to create a template consists of opening the sample template document provided with every FEAM installation and inputting the required information. The template consists of nine pieces of information about job submission as outlined

Table 6.4: Method for Task "Specify Job Submission Info"

DESCRIPTION	ENCODING	EXAMPLE
Open template sample file	M 17K	vi qsub-template <Return>
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify script type	2M 4K	bash
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify node selection	5M 17.5K	#PBS -l select=$NODE
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify queue	4M 6K	#PBS -q batch
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify runtime	5M 23.5K	#PBS -l walltime=10:00:00
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify job starting location	2M 14.5K	cd $PBS_O_WORKDIR
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify error and output joining	4M 9.25K	#PBS -j oe
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify output filename	4M 18.75K	#PBS -N $OUTNAME
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify MPI start command	4M 7.25K	mpiexec $BINARY $ARGS
Navigate to edit location	M 2K	<DownArrow><DownArrow>
Specify other information	3M 20.5K	. /opt/Modules/default/init/sh
Save and quit	M 5K	ESC :wq <Return>
TOTAL Operators in Encoding	44M 161.25K	-

in Figure 4.2. To determine how many keystrokes each operation typically requires, job submission templates were consulted from test set of computing sites (listed in Section 5.1). The average length of the inputs in these templates was used to encode each operation using the KLM's K operator. The average number of logical pieces of the input was used to encode each operation using the KLM's M operator. Table 6.4 presents the encoding of these operations in sequence along with descriptions and examples.

Using the encoding in Table 6.4 and the operator times proposed by the Keystroke Level Model, a prediction was formed the time to execute the method for creating a submission template:

$$T_{PrepareTemplate} = 44 * t_M + 161.25 * t_K = 91.65 \text{ seconds}$$

Thus, the task of creating a submission template was predicted to take around 1.5 minutes as-

suming input lengths as estimated from averages over the computing site test set, typing at 55 wpm, and a mental preparation time of 1.35 seconds per cognitive unit. *This estimate assumes an expert user who does not spend extra time consulting documentation to input the required information into the template.* The time to look up submission syntax information is incorporated into this estimate in Section 6.3. The confirmation of the efficacy of this KLM encoding (by measuring the time it took one expert user to perform the same task sequence) found that the time prediction was accurate to within 20 seconds. While this is a higher degree of fluctuation, the confirmation still attests to the validity of the encoding as providing a prediction at the correct order of magnitude of how long it takes to complete the task.

6.1.6 Method for Task "Use FEAM"

This section analyzes the overall task of using FEAM. A timing estimate is formed for this task by combining the timing predictions of the subtasks involved in the process (i.e. installing FEAM, preparing the two inputs, and issuing the start command) and calculating how many minutes as well as days they will take to complete. How the timing estimate is affected when FEAM is already installed is also considered.

An encoding of the process of using FEAM is presented in Table 6.5. The four subtasks of the process are listed along with their predicted durations (as derived in Sections 6.1.2 to 6.1.5). The table also distinguishes subtasks whose execution depends on FEAM's run location (i.e. target or guaranteed execution site) and whether FEAM has already been installed there. One subtask, issuing the start command, must always be executed to run FEAM. Two of the subtasks, FEAM's installation and the creation of a job submission template, only need to be executed when the framework does not already exists at a site. The fourth subtask of using FEAM is executed depending on the run location. Binary information only needs to be prepared via file transfer when FEAM is executing at target sites where neither the MPI application binary or a FEAM output bundle created for that binary are available.

A typical use of FEAM will involve executing FEAM at a guaranteed execution site followed by executing FEAM at a target site. Thus, the effort to use FEAM, T_{FEAM}, can be described as the

Table 6.5: Method for Task "Use FEAM". The table presents an encoding of the process of using FEAM where the subtasks marked with * are only executed at sites where FEAM is not already installed and the subtask marked with ** is only executed at target sites.

SUBTASK DESCRIPTIONS	DURATION
Install FEAM*	130 s
Prepare access to binary info**	37.2 s
Prepare submission template*	114.35 s
Invoke FEAM	91.65 s

sum of the effort to use FEAM at a guaranteed execution site (which is termed $T_{FEAM:geSite}$) and the effort to use FEAM at a target site (which is termed $T_{FEAM:tSite}$). However, the subtasks involved in $T_{FEAM:geSite}$ and $T_{FEAM:tSite}$ will differ as already described depending on whether FEAM has already been installed at each site. When FEAM has been previously used to predict the execution readiness of other MPI application binaries, it can be assumed that the framework will already be installed at the guaranteed execution site and target sites. These relationships are expressed with the following equations (where T^* is used to denote effort that only occurs when FEAM is not installed):

$$T_{FEAM} = T_{FEAM:geSite} + T_{FEAM:tSite} \text{ where}$$

$$T_{FEAM:geSite} = T^*_{InstallFEAM} + T^*_{PrepareTemplate} + T_{InvokeFEAM} \text{ and}$$

$$T_{FEAM:tSite} = T^*_{InstallFEAM} + T_{PrepareFileAccess} + T^*_{PrepareTemplate} + T_{InvokeFEAM}$$

These equations do not take into account the time for FEAM to complete execution. The footprint evaluation in Section 5.4 found that FEAM used less than five minutes of compute time to run while the wall clock time varied with the amount of time spent waiting in a queue to be scheduled to run. These compute and wait times can be accounted for by estimating the effort to use FEAM at the granularity of days. Often sites will have a special queue with a short wait time for short and small jobs like running FEAM. Queues rarely have wait times longer than a day and typically range between minutes to hours. Thus, in the duration of a day, FEAM should be able to run both at a guaranteed execution site and at a target site with plenty of time to allow for queue delays.

As a natural use of FEAM is to analyze the execution readiness of a computation at various computing sites, the effort of using FEAM at multiple target sites is also estimated. Specifically, the case of migrating to five computing sites is considered as this is the number of sites to which the most advanced participants migrated to on average during the user study. FEAM can be deployed at multiple target sites simultaneously. However, before running at any target site, FEAM must complete execution at the guaranteed site so that the resulting bundle of information about the binary can be used to form predictions. Thus, the effort of using FEAM at five target sites (which is termed $T(5)_{FEAM}$) will still take one day and the effort estimated at the granularity of minutes is expressed by the following equation:

$$T(5)_{FEAM} = T_{FEAM:geSite} + 5T_{FEAM:tSite}$$

Using the equations and the KLM time predictions of the subtasks listed in Table 6.5, estimates were formed of the effort to use FEAM at one and five target site at the granularity of minutes and days. The resulting time predictions, which also distinguish between the case when FEAM is already installed, are summarized in Table 6.6. Running FEAM at five sites takes about three times as much effort as running FEAM at one site. Similarly, running FEAM for the first time also takes three times as much effort as running FEAM using a preexisting installation.

Specifically, it will take 11.8 minutes to use FEAM to predict execution readiness at one computing site when using FEAM for the first time versus 3.7 minutes when using a preexisting installation of FEAM. Similarly, it will take 36.7 minutes to use FEAM to predict execution readiness at five computing sites when using FEAM for the first time versus 12.3 minutes when using a preexisting installation. In either case, the effort to use FEAM is spread over one day to account for queue delays.

The estimates in minutes are closer to a lower bound prediction as the analysis models the time an expert user would take to gather and analyze the information related to execution prediction. As the KLM notes, less experienced users typically spend more time mentally preparing to execute physical operators when doing things like deciding which command to call or manually searching files for information instead of using scripted search commands. At the same time, the estimates at

Table 6.6: FEAM Effort Summary. The table summarizes the estimates of the effort to use FEAM. Estimates are presented at the granularity of minutes and days for one and five target sites with consideration for whether FEAM was previously installed at the sites (denoted by *).

TERM	TARGET SITES	EFFORT (mins)	EFFORT* (mins)	EFFORT SPREAD (days)
$T(1)_{FEAM}$	1	11.8	3.7	1
$T(5)_{FEAM}$	5	36.7	12.3	1

Table 6.7: Manual Effort Summary. The table presents a summary of the estimates of the overall effort exerted by expert users to manually prepare to execute an MPI computation at new computing sites. Estimates are presented in hours and days for one and five target sites.

TERM	TARGET SITES	EFFORT (hrs)	EFFORT SPREAD (days)
$T(1)_{Manual}$	1	1.6	3
$T(5)_{Manual}$	5	8.4	13

the granularity of days is closer to an upper bound as the queue wait time is likely to be on the order of minutes, especially when using a queue for short jobs.

6.2 Manual Effort

This section quantifies the amount of effort exerted to manually prepare to execute computations at new computing sites. This effect is termed T_{Manual}. An estimate of T_{Manual} is formed by using the timing results from the user study in Chapter 2. The study measured how much time users with differing levels of experience spent on various tasks (i.e. learning, submitting, compiling) while migrating MPI applications to new computing sites. As the KLM encoding estimates of the effort involved with using FEAM assumed expert users, the manual effort estimates are also based on the results of the study participants that were classified as experts. The estimates of the effort to manually prepare to execute MPI computations at new computing sites are summarized in Table 6.7.

As reported in Table 2.3, on average it took the most advanced participants 1.6 hours of effort spread over a three day time period to get an MPI application running at one new computing site. This result is used as the estimate for $T(1)_{Manual}$, the effort to manually prepare to run an MPI

computation at one new location.

As the effort of using FEAM was estimated at multiple target sites, the manual effort to migrate to multiple sites is also estimated. To account for the overlapping of effort at the granularity of days as well as general learning effects, the study results are consulted instead of using multiples of the estimate of the effort for one site. As also reported in Table 2.3, on average it took expert participants 8.4 hours of effort performed over a 13 day time period to get an MPI application running at five new computing sites. This result is used as the estimate for $T(5)_{Manual}$, the effort to manually prepare to run an MPI computation at five new computing sites.

6.3 Automated Effort

This section quantifies how much effort is exerted to prepare to execute MPI computations at new computing sites while partially automating the process by utilizing FEAM. This effort is termed $T_{Automated}$. Estimates of $T_{Automated}$ are formed by combining estimates of the time to use FEAM (T_{FEAM} from Section 6.1.6) with results from the user study of application migration (presented in Chapter 2) and the evaluations of FEAM (presented in Chapter 5). How the $T_{Automated}$ effort was quantified is explained before the resulting timing estimates are presented.

6.3.1 Descriptions

The steps to prepare to execute a computation vary depending on FEAM's prediction about the computation's readiness to execute. If FEAM predicts an MPI computation is ready to execute without recompilation (a case termed $T_{Automated:Success}$), then the application binary can simply be submitted for execution with appropriate site-specific settings (that may include a resolution scheme). This submission effort is termed $T_{SubmitComputation}$. Alternately, if FEAM predicts recompilation is required (a case that termed $T_{Automated:Failure}$), then the full manual effort (i.e. T_{Manual}) of preparing to execute a computation at a new site must occur.

Before FEAM can be utilized to make predictions about a computation's execution readiness, the submission script syntax of the target site must be learned (so that it can be provided as input

to FEAM). This effort is termed $T_{LearnSubmissionSyntax}$. When recompilation must occur, this effort will already be accounted for as part of the manual effort estimate (which includes learning to run at a site). However when recompilation is not necessary, learning the submission syntax must be separately accounted for unless FEAM has already been installed and a submission template already exists. This effort only occurs at target sites as users are assumed to already be familiar with how to run jobs at guaranteed execution sites.

The following equations use the defined terms to summarize the effort associated with utilizing FEAM at n target sites in relation to whether recompilation occurs (where T^* indicates effort that only occurs when FEAM is not already installed):

$$T(n)_{Automated:Success} = nT_{LearnSubmissionSyntax} * + T(n)_{FEAM} + nT_{SubmitComputation}$$

$$T(n)_{Automated:Failure} = T(n)_{FEAM} + T(n)_{Manual}$$

An estimate of $T_{LearnSubmissionSyntax}$ and $T_{SubmitComputation}$ is created using one of the tasks identified during the user study. As part of the *submission* task, participants created a submission script at their target computing site. This effort corresponds to $T_{LearnSubmissionSyntax}$. Also as part of the submission task, participants incorporated settings specific for their computation into the submission script to run their application. This effort corresponds to $T_{SubmitComputation}$. Thus, the time participants took to perform the submission task during the study can be used to estimate $T_{LearnSubmissionSyntax}$ and $T_{SubmitComputation}$. As reported in Table 2.13(b), the submission task took expert participants on average 13 minutes to complete per one migration. This result is used as an estimate for $T_{LearnSubmissionSyntax}$ and $T_{SubmitComputation}$ and is termed $T_{Submission}$. In the case when FEAM has already been installed and $T_{LearnSubmissionSyntax}$ does not need to occur as a template submission file already exists, $T_{Submission}$ is estimated as 6.5 minutes, half of the duration of the submission task. Thus, the equation of $T_{Automated:Success}$ can be rewritten using $T_{Submission}$ as follows:

$$T(n)_{Automated:Success} = nT_{Submission} + T(n)_{FEAM}$$

where $T_{Submission} = 6.5$ or 13 mins depending of whether FEAM has or has not been installed

To quantify $T_{Automated}$ in terms of $T_{Automated:Success}$ and $T_{Automated:Failure}$ requires considering how often MPI computations are ready to execute at new computing sites without recompilation. This probability is termed $P_{Recompilation}$. The performance evaluations of FEAM found that the test cases were able to execute without recompilation 49% of the time. Specifically, Table 5.1 presents the findings that 35% of the applications executed with only MPI configuration settings while Table 5.3 presents the results that FEAM's resolution techniques enabled 14% more executions without recompilation. Thus, one estimate of $P_{Recompilation}$ based on the results gathered is 51%. However, how often a MPI computation is ready to execute at new target sites without recompilation can vary drastically depending on the computation's requirements and the site's characteristics. Thus, the analysis of $T_{Automated}$ considers the full range of possibilities with $P_{Recompilation}$ values of 0%, 50%, and 100%. When all computations are predicted to be ready for execution (i.e. $P_{Recompilation} = 0\%$), $T_{Automated}$ equals $T_{Automated:Success}$ and when no computations are predicted to be ready for execution (i.e. $P_{Recompilation} = 100\%$), $T_{Automated}$ equals $T_{Automated:Failure}$. These relationships between $T_{Automated}$ and $T_{Automated:Success}$, $T_{Automated:Failure}$, and $P_{Recompilation}$ are described in the following equation:

$$T(n)_{Automated} = ((1 - P_{Recompilation})T(n)_{Automated:Success}) + (P_{Recompilation}T(n)_{Automated:Failure})$$

6.3.2 Estimates

The equations presented in the previous section are evaluated to quantify the $T_{Automated}$ effort. Additionally, to quantify the effort at the granularity of days, whether the effort of the different parts of the equations can be overlapped is considered. For the no recompilation case (i.e. $T_{Automated:Success}$), the submission related tasks that are performed in addition to using FEAM do not add any extra days of effort. These submission related tasks (i.e. $T_{Submission}$) that $T_{Automated:Success}$ incorporates are short enough to be completed on the same day as when using FEAM. For the recompilation case (i.e. $T_{Automated:Failure}$), using FEAM also does not add any days to the manual effort estimate. Initially invoking FEAM and gathering the prediction that recompilation is necessary can be overlapped with the start of the manual effort.

Table 6.8: Automated Effort Summary. The table presents a summary of the estimates of the effort to utilize FEAM to prepare to execute MPI computations at new computing sites in relation to the probability that the computation is ready to execute without recompilation. Estimates are presented at the granularity of hours and days for one and five target sites using three probabilities of application recompilation with consideration for whether FEAM was previously installed at the sites (denoted by *).

TERM and TARGET SITES	RECOMPILATION PERCENTAGE	EFFORT (hrs)	EFFORT* (hrs)	EFFORT SPREAD (days)
$T(1)_{Automated:Success}$	0%	0.4	0.2	1
$T(1)_{Automated}$	50%	1.1	0.9	2
$T(1)_{Automated:Failure}$	100%	1.8	1.7	3
$T(5)_{Automated:Success}$	0%	1.7	0.7	1
$T(5)_{Automated}$	50%	5.4	4.7	7
$T(5)_{Automated:Failure}$	100%	9.0	8.6	13

The resulting timing estimates of $T_{Automated}$ along with $T_{Automated:Success}$ and $T_{Automated:Failure}$ are presented in Table 6.8. Estimates are listed for one and five target sites and distinguish whether FEAM is being used for the first time or whether the framework was already installed. The calculations predict $T_{Automated}$ to take on average 1.1 hours over 2 days for one target site and 5.4 hours over 7 days for five target sites. If FEAM is already installed, the effort is reduced by around 10% on average. While installing FEAM has a large impact at decreasing the amount of time involved with using FEAM, these savings are dwarfed in the calculations of $T_{Automated}$ by the amount of time associated with submission tasks and manual effort. When no recompilation is necessary, the consideration of varying recompilation probabilities produces estimates as low as 0.4 hours at one site and 1.7 hours at five sites over one day. When recompilation is necessary, the estimates are as high as 1.8 hours over three days at one site and 9.0 hours over 13 days for five sites. The next section considers these calculations in relation to manual effort by estimating the relative speedup achievable by using FEAM.

Table 6.9: Speedup of Using FEAM. The table presents estimates of the speedup of using FEAM to automate the process of preparing to execute MPI computations at new computing sites over performing the task manually. Estimates of speedup are presented at the granularity of hours and days for one and five target sites using three probabilities of application recompilation with consideration for whether FEAM was previously installed at the sites (denoted by *).

TARGET SITES	RECOMPILATION PERCENTAGE	SPEEDUP (over hrs)	SPEEDUP* (over hrs)	SPEEDUP (over days)
1	0%	3.9x	9.4x	3.0x
1	50%	1.4x	1.7x	1.5x
1	100%	.89x	.96x	1.0x
5	0%	5.0x	11.3x	13.0x
5	50%	1.6x	1.8x	1.9x
5	100%	.93x	.98x	1.0x

6.4 Speedup

The amount of effort associated with preparing to execute MPI computations at new computing sites manually as well as when utilizing FEAM to partially automate the process has been quantified so far. This section uses these quantities to calculate the speedup associated with utilizing FEAM. *Speedup* is defined as the ratio of the effort to prepare to execute MPI computations at new computing sites manually (T_{Manual}) versus when utilizing FEAM ($T_{Automated}$):

$$Speedup = \frac{T_{Manual}}{T_{Automated}}$$

Table 6.9 presents the speedup calculations. As with $T_{Automated}$, a range of estimates varying with the probability that the MPI computation has to be recompiled is presented. Speedup for one and five computing sites is presented using effort estimations in days as well as in hours while distinguishing whether FEAM was already installed.

6.5 Analysis

The average speedup of effort in hours is very close for one and five target sites. Even though the estimates of T_{Manual} and $T_{Automated}$ were derived separately for one and five sites, they scaled

similarly with the increase in sites. The resulting calculations predict that utilizing FEAM produces about 1.5x speedup on average. This equates to just under an hour of effort saved per target site. The speedup of effort in days shows 1.5x speedup in the number days needed to prepare to execute at one target site and a larger speedup of 1.9x when preparing to execute at multiple target sites. Thus, utilizing FEAM enables more efficient preparation for execution especially by decreasing the number of days to get a computation running at multiple execution locations.

In the case when FEAM predicts a computation needs to be recompiled to execute, a small overhead of no more than 11% results due to using FEAM before beginning the manual preparation process. This overhead, which equates to less than ten minutes of effort, may be offset by enabling less time to be spent on learning about target sites and their configurations as this information will have been automatically reported on using FEAM. Thus, while in the worst case the speedup calculations estimate a small overhead, in reality the overhead may be eliminated by providing useful site-specific information such that a small speedup may even be produced, especially at sites where the framework is already installed.

In the case when FEAM predicts a computation is ready for execution without recompilation, the speedup calculations show how significantly more efficient utilizing FEAM can be over recompilation. The speedup of the effort in hours ranges from 3.9x to 11.3x depending on whether FEAM is already installed. This speedup equates to at least two hours of effort saved per site. The speedup of the effort in days ranges from 3x to 13x depending on the number of target sites. This speedup equates to at least two days of effort saved per site.

One alternate to using FEAM to predict the execution readiness of MPI computations is for users to directly run their binaries at target sites and check the output to determine if execution occurred without recompilation. Let us term this process *direct execution*. As when using FEAM, direct execution requires the submission script syntax of the target sites to be learned along with any site-specific settings for application execution. According to the estimates of this submission process, direct execution will take less than 20 minutes of effort spread over a day to incorporate queue delay and job execution time. However, if the job fails, the user must spend additional effort to determine the cause, which may be binary or configuration related. This debugging process can

be prolonged by queue delay times and drag over days as the job is resubmitted for execution. If instead an additional 10 minutes of effort are spent to use FEAM (or less than five minutes if FEAM is preinstalled), FEAM will provide a prediction of the computation's execution readiness along with site information and site-specific configurations that can resolve some execution-blocking issues. Thus, in comparison to direct execution, using FEAM adds little effort in terms of minutes and no effort in terms of days while potentially eliminating tedious debugging.

6.6 Summary

This chapter has presented an evaluation of the increase in efficiency associated with using FEAM to automate the process of preparing to execute MPI computations at new computing sites. The increase in efficiency was analyzed by comparing the effort associated with manually performing the process versus when using FEAM. Manual effort was quantified by using data about expert participants from the user study. The effort to use FEAM was quantified by modeling the use of FEAM with the Keystroke Level Model and incorporating learning time from user study data. The resulting calculations predict that using FEAM results in around a 1.5x speedup on average or just under an hour of effort saved per new computing site. However, when considering effort at the granularity of days, using FEAM saves more than two day of effort per site. In cases when recompilation is not required, using FEAM can be an order of magnitude faster over typical manual preparation methods and more insightful than brute-force direct submission attempts. Thus, this chapter's efficiency evaluations show that even for experts utilizing FEAM enables more efficient preparation of computations for execution, decreasing hours of effort and the number of days to get a computation running at multiple computing sites. Overall, the evaluation illustrates how much more efficient automated methods can be over manual methods. These automated methods will lend themselves especially well to being used without user intervention by schedulers to increase the pool of candidate hosts on which applications can be scheduled.

Chapter 7

Related Work

This chapter presents a discussion of technologies related to the recognition, representation, and resolution of factors relevant for application execution. Package managers, virtual machines, and standardized representations, along with grid infrastructures, build tools, schedulers, and programming languages are discussed. However, none of these technologies automatically identify and describe application information related to execution. It is assumed that application developers or users can fully describe these application characteristics. Similarly, none of these technologies automatically compose site-specific instructions for application execution. It is assumed that application developers or users will develop the deployment procedures for target sites. As such, these existing technologies cannot be utilized efficiently by researchers for the preparation of various computing environments for application execution. The one notable exception is the CDE system which can be used to address a part of the problem that is the focus of this dissertation but for a different a set of applications and with a different set of assumptions. This chapter begins with a discussion of the CDE system followed by an overview of the other related technologies.

Currently, there is no one general method used to deploy applications at computing sites. The process is a manually coordinated mixture of methods often specialized for target sites and applications. It was recorded in our study that users spend non-trivial amounts of time distributed over several days learning about and setting site configurations, a process that must be accomplished before most existing deployment techniques can be applied. Performing this process along having with an understanding of application dependencies are barriers to the use of otherwise easily acces-

sible resources. The methods presented in this dissertation free users from being the bottleneck to using new resources. Instead of users manual labor, FEAM can be used to automatically predict whether an application is ready to execute at a computing site and with what site-specific configurations, thus enabling direct usage of resources or more scheduling freedom for job management systems. This chapter compares FEAM to related work.

7.1 CDE

The *Code, Data, and Environment* (CDE) system was created to seamlessly migrate applications to different machines [32]. CDE monitors an application's execution using `ptrace` to identify the code, data, and environment that it uses for execution. The system automatically creates a package containing the components required for an application to run. This package is used at target sites to run the application without having to recompile, reinstall, or setup dependencies. However, CDE does not guarantee that all of a binary's dependencies will be identified and packaged. Also, migration can only occur between target sites with compatible architectures and Linux kernel versions. CDE specifically focuses on Linux binaries compiled for the x86 architecture.

While both CDE and FEAM make migrations of applications between computing sites more efficient, the two solutions differ in many ways. Most notably, CDE can not be used to migrate MPI applications. The MPI stack is not something that can be configured at a computing site by copying files. This is why FEAM determines what available, already configured, MPI stacks are present at target sites. In general, FEAM can be used with a broader set of applications and target sites than CDE as FEAM is not limited to x86 architectures. Also, in terms of usage differences, FEAM does not require the binary to be executed while CDE does to perform its runtime analysis of dependencies.

7.2 Package Managers

Package management systems provide methods related to the representation and resolution of factors relevant for application execution to address a different general problem. Package managers

provide a centralized mechanism for the management of software on computer systems by keeping track of software in the form of packages. This includes providing tools for installing, uninstalling, verifying, querying, and updating packages. The most widely used package format, RPM [25], is associated with the RPM Package Manager for GNU/Linux systems. Packages consist of the piece of software being managed along with metadata such as a version number, description, and list of dependencies on other packages. Meta-package managers, such as Yum [22] and SmartPM [13], can provide dependency resolution by computing dependencies and automatically installing, uninstalling, or updating the relevant set of packages.

Package management systems are not suited for managing many of the applications of researchers due to setup costs and deployment limitations. Only dependencies that are packages can be automatically resolved by package managers. Methods to resolve any other dependencies must be provided by the package creator. However, the target package creators are not application consumers who only have a basic familiarity of the components, such as third-party codes, their applications may be using along with no familiarity with the target environments. Such users may not be aware of what dependencies need to be handled. In contrast, FEAM automatically composes configurations that select the required MPI stack and resolve missing shared libraries while only requiring users to specify the job submission syntax. Package creators not only have to specify every command that is to be issued to install an application and prepare a target environment but they also have to describe every file that is to be installed. Thus, package managers provide automatic recompilation at the cost of requiring the process to be initially fully specified. This involves a great deal of effort with a steep learning curve. For community codes or big science applications, this is an acceptable cost while for many other projects, it is just too expensive. Additionally, using a package manager to install software or initiate package creation requires root privileges for most package managers including RPM. As users of computing resources, researchers do not have root access in their computing environments. In contrast, using FEAM does not require any special privileges.

7.3 Virtual Machines

Virtualization technologies provide methods to handle the representation and resolution of factors relevant for application execution. Instead of trying to homogenize target environments, virtual machines (VMs) can be used to mask the heterogeneity of the underlying system. As an encapsulation of the application in its execution environment, a VM instance is a representation of all an application and all of its execution requirements. The infrastructure that starts a VM instance provides the means of making the execution requirements accessible at target sites. Invigo [24] is one example of a virtualization infrastructure that creates grids of virtual resources. Currently, OpenStack [12] and VCL [18] are commonly used for large-scale cloud deployments.

To be able to use VMs requires not only that target sites support virtualization but that they support the type of VM being used as there is more than one type of VM. Even though VMs mask the heterogeneity of the underlying system, VM users still have to handle the heterogeneity between different VM platforms. FEAM does not require any particular infrastructure to be in place other than the lightweight copy of its scripts on a resources head node. Also, unless an application is developed in a VM environment, the VM environment will still need to be configured for application execution. Not only does a starting VM instance need to be selected but also any missing requirements must be installed, a more involved task than selecting from preinstalled configurations on shared resources. Virtualization technologies do not provide methods for identifying information relevant for application execution to setup a VM. Thus, to setup VMs involves the same time-consuming tasks that are such a barrier to the use of shared resources.

VM technologies have raised performance concerns in the HPC community. [49] measured that parallel programs written in MPI perform similarly in the virtualized and non-virtualized environments only when there is little communication between processes and little IO access. Most parallel programs written using MPI do not satisfy either of these assumptions. [33] measured that near-native performance can be achieved for communication between VMs only when they are based on the same physical host and the VMs are able to communicate via shared memory. This assumption would limit possible target environments to computing systems based on shared-memory archi-

tectures or cores residing on an individual processor. Using virtualization technologies can have performance ramifications for researchers running applications that use MPI. In contrast FEAM does not affect the binarys performance as FEAM performs its evaluation independently without running the binary.

7.4 Representation Standards

The Configuration Description, Deployment, and Lifecycle Management (CDDLM) framework and the Solution Deployment Descriptor (SDD) are examples of technologies that provide a standardized syntax for describing software to enable automatable lifecycle management. These technologies provide different representations of factors relevant for application execution. However, they do not provide any means for discovering the information that is to be described. While FEAM does not encode the information it gathers in any specific representation standard, a translation could be made into such a standard to utilize FEAM as an automatic discovery mechanism.

7.4.1 SDD

SDD [37] is an emerging standard from the Organization for the Advancement of Structured Information Standards. SDD defines an XML-based format for describing characteristics of an installable unit of software and its components as relevant for deployment, configuration, and maintenance. How the information is gathered, how an application is actually deployed, or how that automation occurs is beyond the scope of the SDD.

7.4.2 CDDLM

CDDLM [29] is an Open Grid Forum standard for deployment and lifecycle management of distributed software systems using Web Services. The CDDLM standard defines two description languages for representing application components and configuration parameters. The deployment aspect of the standard that is concerned with consuming the standardized descriptions is discussed in the next section.

7.5 Grid Frameworks

Various grid-based technologies have been developed to aid with application deployment. Each of the technologies uses a unique representation for an application's execution requirements (some based on standards) and provides an infrastructure for resolving the described dependencies. For each of these technologies, all possible target hosts must be managed by a specific framework. This is the way that FEAM works except FEAM is not specific to grids and does not require a heavyweight setup. Also unlike with FEAM, none of the existing technologies provide any means for discovering the information that is to be described. However, some of the frameworks divide the responsibility of providing information about applications between application consumers, system administrators, and software developers. The basic information that FEAM requires (binary name and job submission syntax) could be provided by any of these individuals. For the researcher or a scheduler, the existing technologies do not provide efficient methods for preparing a variety of environments for application execution because of the setup effort and application familiarity required to apply them.

7.5.1 DistributedAnt

DistributedAnt [30] depends on application consumers to provide the information for application deployment. DistributedAnt supports application deployment in grids by automating file transport, installation, and configuration. DistributedAnt runs an extended version of Ant on remote machines. To use Distributed Ant, the steps for application deployment have to be known for all target environments. As with package managers and build tools, DistributedAnt enables automatic recompilation after the process is fully specified. For DistributedAnt, this information is described using extensions to Ant, a widely used Java tool for managing the application build process. Deployment clients then coordinate the deployment process with remote deployment servers based on the description. Thus, the infrastructure to use DistributedAnt is not lightweight.

7.5.2 CDDLM

The CDDLM standard described in the previous section also depends on application consumers to provide the information for application deployment. The CDDLM standard defines a framework, consisting of a deployment API and a component model, for automatically deploying applications on Grid resources. However, CDDLM is not an efficient mechanism for researchers or schedulers todeploy binaries at new sites. In addition to creating a description, the CDDLM user has to implement the functionality associated with each interface to the component model. Once these pieces have been created, any CDDLM compliant framework can be used to deploy the application. Thus, unlike for FEAM, the overhead to begin to use CDDLM is large requiring the specification of the interaction of each step of the deployment process.

7.5.3 GLARE

GLARE [46] depends on application developers as well as consumers to provide information for application deployment. The GLARE framework facilitates the execution of workflows on grids for a variety of target environments by automating the configuration of dependencies needed to execute workflow components. Application developers register their applications with the GLARE framework and provide installation procedures for their application for every possible target environment. By providing this information, GLARE is even able to enable automatic recompilation if the process has been specified as part of a registration. Users identify the types of activities present in their workflows. GLARE then hides deployment details from the user by transparently mapping activity types to deployments. If users are trying to deploy custom or self-created activities that do not have existing registrations, the responsibility of creating the registration falls to the user. With FEAM, such issues do not occur. Regardless of whether administrators, users, or schedulers provide the initial basic information to use FEAM, no extra responsibilities will fall to the user regardless of the type of computation that is to be deployed.

7.5.4 UNICORE IDB

The UNICORE Grid's Incarnation Database (IDB) [16] depends on system administrators as well as application consumers to provide information for application deployment. The IDB maps abstract application locations to site-specific implementations. Client resource requests are checked against the IDB before being scheduled. It is the responsibility of administrators setting up the UNICORE grid to update the IDB with site-specific information. In this way, users can easily access pre-deployed applications. The setup effort is transferred from the user to the administrators. However, this solution is not general like FEAM and does not address deploying other applications.

7.6 Build Tools

Build tools provide methods that handle recognition, representation, and resolution of factors relevant for application execution. Build tools create binaries and, as such, are part of the recompilation process that FEAM does not address. These tools, like the GNU Autoconf [5] package, aid with the building of software on a single system or across multiple platforms. These tools can perform dependency checks for requirements needed by an application to build. This is similar to the discovery process that FEAM performs. However, build tools do not prepare computing environments for application execution and often specify other execution requirements left to be resolved by the user in a README file.

Only applications whose source code is available can be configured using Autoconf. Thus the set of applications with which build tools can be used are different than the set of applications with which FEAM can be used. An application's build dependencies are specified in a set of files that are then transformed into configuration scripts. This process is involved, requiring many steps. The target audience of build tools differs from the target audience of FEAM as build tools are meant to be used by developers not users of codes. The resulting configuration scripts are Autoconf's representation of an applications build requirements. These scripts can be run in target environments to produce customized binaries. Just as FEAM checks if a target meets run dependencies, the scripts check whether a target meets compile dependencies. If any dependency is found to be missing, the

build fails unless some resolution mechanism has been implemented by the Autoconf script creator. As with FEAM, an error report is provided to the user. Autoconf scripts can be extended to perform dependency checking and resolution if application requirements and resolution steps are known. Thus, as with package managers and DistributedAnt, Autoconf enables automatic recompilation after the process is fully specified.

Tools have also been created for determining the build information of program binaries after the compilation process. This is a subset of the type of information that FEAM automatically gathers. Work by Rosenblum *et al.* presents machine learning models for inferring compiler provenance [41]. Work by Le *et al.* presents methods that identify what compilers and libraries a binary uses using signature-based detection [35]. Such methods can be used to help describe a specific subset of dependencies of existing binaries. FEAM does not currently use these methods but could apply them to further diversity how it automatically gathers information.

7.7 Programming Languages

Specialized programming languages are a solution that is applied from computation encoding to deployment. One example of high level languages developed to handle both construction and management of applications are the Module Interconnection Languages (MIL)These programming languages, including Polylith, Conic, Darwin and Olan [26, 27], can be used to describe applications in terms of their software components. The programmer identifies an application's components and describes their interconnections and communications. The description is platform independent so that it can be consumed automatically to deploy applications on environments where MIL runtime support is available. The MILs provide representations of application execution requirements and support for resolving the requirements on targets with MIL runtime environments. Thus, as with other solutions, using these specialized programming languages requires non-trivial setup effort to encode an application and its requirements appropriately as well as to deploy the supporting infrastructure at target sites. Additionally, there is no support for automatically discovering any of the information that is to be described. The responsibility falls to the users, developers, or administra-

tors. By not requiring such detailed descriptions, FEAM addresses a subset of the problem but with minimal user intervention.

7.8 Job Management Systems

Job management systems manage sets of resources and distribute jobs amongst them. These services can automatically select the target environment for a job based on user-specified requirements and preferences. However, these requirements tend to be hardware and not software oriented. With traditional schedulers, such as PBS, Platform LSF, or IBM LoadLeveler, users select a specific target resource [40]. With grid-based schedulers, such as Condor-G, Moab, or GridWay, users can target a variety of resources for job execution [7, 34]. It is the responsibility of the user to ensure that applications are ready to be run at any potential resource as job management systems do not address the resolution of missing dependencies and only provide the possibility to express a limited set of application execution requirements. These tools do not provide matching at a fine enough granularity to be able to determine whether MPI dependencies are met.

The predictions and resolutions provided by FEAM could be used by job management systems to increase the set of potential execution targets without requiring users to ensure each new site is ready for application execution. Instead FEAM could be used to investigate whether a site is ready for execution and provide the site-specific configurations to run at that site. In this way FEAM could work with job management systems to enable greater scheduling freedom so that jobs could run when any matching resource is available, regardless if it has been examined manually by a user.

7.9 User-Environment Management Tools

User-environment management tools, such as Environment Modules [10] and SoftEnv [11], can be used to manage a shell environment and support the discovery of software packages. They provide a means to alter and set environment variables in order to dynamically configure a user's computing environment for a particular usage, such as compiling an application with a specific compiler or using a specific version of a shared library. These tools are deployed by system administrators to

encapsulate all the configuration steps needed to prepare to run particular applications or access particular libraries [38]. The Lmod implementation of environment modules [9] even provides a mechanism for handling dependencies between configurations, which is especially useful for managing access to multiple MPI stacks of shared library and compiler combinations. However, to use user-environment management tools, users must be aware of the dependencies that their software requires or the software must be already supported at a given computing site. Our study found that most participants consulted user-environment management tools to help with site configurations. FEAM also consults user-environment management tools as part of the automated discovery of MPI stacks and shared libraries at target sites. Having discovered information about an applications requirements, FEAM knows what to look for using the tools. User-environment management tools help with one piece of the configuration process when preparing to run applications at computing sites.

Chapter 8

Conclusions and Future Work

This dissertation presented a first step toward an ideal solution providing deployment and, therefore, scheduling freedom to allow computations parallelized with MPI to quickly and easily be run on computing resources. The solution focused on determining if a preexisting MPI application binary could be made to execute on a given resource without recompilation. The hypothesis was that methods for automatically gathering information about execution requirements and composing site-specific instructions that configure the requirements at target environments are more efficient than manual methods for the preparation of multiple shared computing environments for the execution of MPI binaries. The presented work supports this hypothesis. The user study established a baseline of the time to manually coordinate the migration of MPI applications to new computing sites, capturing the current manual state of the art methods for preparing to use sites directly or via schedulers. The Execution Readiness Prediction model described the application and environment information relevant for determining whether binaries will execute at computing sites without recompilation. The resolution methods were able to resolve some of the execution-blocking issues to further broaden the set of potential execution sites. These developed methods were shown be applicable automatically in FEAM, the Unix-based implementation of the solution. FEAM's evaluation demonstrated that the implementation was accurate, effective, and light-weight. A further analysis of FEAM's efficiency applied the baseline gathered from the user study to predict the speedup gains of using FEAM to automate the MPI binary migration process. As such, FEAM can be used by researchers and schedulers to increase scheduling freedom. Thus, the contributions of this disseration

include the first study to collect timing information regarding the effort to migrate MPI applications to new computing sites, a model for predicting execution readiness of MPI application binaries, an implementation that automatically makes predictions and composes site-specific configurations, and an evaluation analysis of efficiency in terms of time expenditures.

We began with a presentation of a study to measure the time required to manually coordinate the migration of MPI computations. The study results gathered from 25 participants quantified the intensiveness of the process that serves as a barrier for deploying applications quickly and easily at new computing sites. While the documented average duration of one migration was 2.5, the process was on average spread over several days and lead to multiple weeks being required to perform multiple migrations. The majority of the time was found to be spent on learning, compiling, and debugging. However, less experienced participants took almost 50% more time and overall spread their effort across more days. Consequently, it took them on average three weeks to migrate their MPI computations to four new computing sites. The study results emphasized that less experienced researchers can especially benefit from techniques for more efficient application migration. More importantly, the study results provide a baseline for the magnitude of the barrier that users face to be able to use new resources when applying the current manually coordinated methods for migrating MPI applications.

Next, the model for predicting whether an MPI application binary can execute at a new computing site without recompilation was presented. The model provides guidelines for evaluating four factors - MPI stack, shared libraries, hardware architecture, and system configuration - related to MPI applications and computing environments that are highly relevant for execution. When tuned performance is not key, being able to quickly identify where an application binary can, or why it cannot, execute provides a more efficient way to get running at multiple new sites. The model outlines what information needs to be gathered to make this determination independently of implementation details and without running the application binary. Automatically gathering this information and forming a prediction is the basis for enabling deployment freedom.

Next, FEAM, the Unix-based implementation of the execution readiness prediction model that additionally composes application specific-site configurations and a scheme for resolving

execution-blocking issues was presented. FEAM discovers information about application binaries and computing environments in an automated fashion and then applies the guidelines of the prediction model to form a prediction about execution readiness. FEAM also utilizes the discovered information about application dependencies and site configurations to develop a scheme for resolving shared library requirements when possible. In this way, FEAM is additionally providing an assessment of how much effort will be required to get an application running at a new computing environment. FEAM, the implementation of the developed solution, can be used by users directly or via schedulers to identify computing sites on which MPI computations are ready to execute. To assess the effectiveness of our methods, FEAMs ability to make predictions and resolve issues was evaluated. The evaluation results of 13 applications across five computing sites showed that the methods and their implementation are accurate, effective, and efficient. With an accuracy of more than 90%, FEAM correctly predicted the majority of execution failures and demonstrated the validity of the prediction model. FEAM's resolution scheme was found to be effective by enabling 41% more successful executions. Running FEAM had a small footprint with execution time on the order of minutes and disk usage on the order of MBs. The evaluation showed that FEAM is ready to be of value to users directly or via schedulers to identity sites that are ready to execute MPI binaries.

The efficiency of applying the developed methods was also analyzed to put into perspective the savings in terms of time associated with using FEAM. It was found that using FEAM is more than an order of magnitude faster in the best case and no significantly slower over manual methods in the worst case. This speedup was calculated by estimating the duration of operations at the keystroke granularity involved in running FEAM versus performing the equivalent tasks manually as measured during the study.

8.1 Future Work

The main direction of the proposed future work stemming from this dissertation deals with expanding the model and implementation while incorporating user feedback to evaluate and fine tune the enabling of scheduling freedom for job management systems. The developed solution could be

deployed on a system that coordinates job submission between various resources. A system with such a meta-queue would be a good testing ground for the existing methods as well as conductive for researching extensions to the work. The framework could be used to determine which resources are ready to execute an MPI job and to provide configuration suggestions for running the job. For example, if a meta-queue consisted of 14 target sites, the queue could sent out short probe jobs that would run FEAM at each site and predict whether a binary could execute there in order to establish a broader set of candidate sites for scheduling the binary. The site-specific configuration script generated by FEAM would then be used to run the job. Users would not need to provide any additional input to take advantage of these features as the binary location and submission template for sites could be determined from information already known by or provided to the meta-queue. Direct user feedback as well as traces of usage gathered by the queue could be collected to determine the usefulness of the framework, areas for improvement, and directions for future expansions. Deployment with a meta-queue could also be used to improve the accuracy of predictions, tune overall performance, and create new implementations of the prediction model as driven by application characteristics and user experience.

Another avenue of future work is to apply insights gained from this research to analyze the difficulties associated with automatic recompilation. Due to a lack of generality, creating an automated model for binary recompilation is difficult. Instead, the types of difficulties users experience during the compilation and debugging process at new computing sites could be more clearly identified. The study found that these two tasks were the most consuming for users when migrating their applications to new computing sites. By modeling the specific types of issues, automatable parts could be determined to make the process more efficient for less experienced users. While the automation of the entire recompilation process may not be feasible in general, research on solutions to common issues could make the task less time consuming while moving closer to an ideal solution of deployment freedom.

Appendix A

Study Materials and Results

This appendix contains the study materials that were distributed as part of the user study of the migration process described in Chapter 2. This includes surveys about the migrated applications and the participants' computational experiences, general background and experiences with the migration process as well as the test of participants' computational experiences. Each section first presents the questions and then presents a summary of the responses for each question for each of the 25 participants. The appendix concludes with a summary of the migration data for each computing site in terms of the duration spent on each task type by each participant.

A.1 Application Survey

This section presents the study survey that asked participants to describe the applications they migrated in terms of qualities such as what it computes, who created it, what MPI type it uses, and what programming language it was written in. This resulting answers were used to profile the applications, as summarized in Figure 2.7.

A.1.1 Questions

1. What is the MPI application's general field of study? (i.e. astronomy, economics, material science, etc.)

2. Briefly describe what the MPI application is computing.

3. Who wrote the majority of the MPI application code?

   (a) myself

   (b) colleagues

   (c) community

   (d) unsure

   (e) other:

4. What programming language(s) is the MPI application written in?

   - C
   - C++
   - Fortran
   - unsure
   - other:

5. List any community codes that you are aware the MPI application uses.

6. List any libraries that you are aware the MPI application uses.

7. What MPI implementations do you or have you used to run this MPI application?

   - MPICH/MPICH2
   - MVAPICH/MVAPICH2
   - OpenMPI
   - unsure
   - other:

8. Are you able to compile this MPI application?

   Yes - I have access to the source code

Have you ever compiled the application at a site other than where you currently run it?

Yes No

No - I only have access to a binary.

No - I have access to the source code but I do not compile the code myself.

unsure

other:

9. At how many computing sites have you run the MPI application?

   1

   What are the reasons, if any, that you are not running at more sites?

   2

   3

   4

   More than 4

10. Please describe what a successful run of this MPI application looks like. Please include example job output. (This description will be used to validate your successes during the migration phase of the study.)

A.1.2 Participant Answers

Table A.1: Application Survey Answers: Participants 1 to 12

	PARTICIPANT ID											
Question	1	2	3	4	5	6	7	8	9	10	11	12
3	a	a	c	c	d	c	b	c	b	d	d	a
4a	x			x		x			x		x	
4b										x	x	
4c		x	x		x			x			x	x
4d							x					
7a		x		x	x	x	x	x				x
7b		x	x		x	x						
7c	x	x			x	x			x	x	x	
7d									x		x	
8	1	1+	1	1+	No	1+		1	1+	1+	1+	1+
9	2	5+	1	4	5+	5+	3	1	2	3	5+	2

Table A.2: Application Survey Answers: Participants 13 to 25

	PARTICIPANT ID												
Question	13	14	15	16	17	18	19	20	21	22	23	24	25
3	d	a	a	b	a	b	c	c	b	b	d	a	a
4a					x	x					x		x
4b	x					x	x	x				x	x
4c		x	x	x	x			x	x				x
4d													
7a	x		x	x	x	x	x		x		x	x	x
7b											x		x
7c	x								x	x			x
7d													
8	1+	1+	1	1	1	1	1	1	1	1+	1	1+	1+
9	2	3	1	1	1	1	1	1	1	2	1	2	3

A.2 Experience Survey

This section presents the study survey that participants completed to self-assess their computational experiences such as since when, how often, and how confidently participants performed various computational tasks. This resulting answers were used to categorize participants' level of experience as outlined in Table 2.2.

A.2.1 Survey Questions

1. How long have you been doing computational research?

   0 - 6 months

   6 months - 1 year

   1 year - 3 years

   3 - 5 years

   More than 5 years

2. How long have you been using shared computing resources (i.e. clusters, supercomputers)?

   0 - 6 months

   6 months - 1 year

   1 year - 3 years

   3 - 5 years

   More than 5 years

3. How long have you been using MPI?

   0 - 6 months

   6 months - 1 year

   1 year - 3 years

   3 - 5 years

More than 5 years

4. How did you learn to do computational research (writing code, using shared resources, etc)? (Check all that apply)

   (a) Self-taught

   (b) Online reference documentation (i.e. user manuals and reference guides)

   (c) Workshops/Webinars/Courses

   (d) Self-paced online training (interactive modules, tutorials, etc)

   (e) Informally from others

   (f) Via online help (information about a specific task delivered on request)

   (g) Other

5. Please indicate your experience with the following tasks on a scale of 1 (Novice) to 4 (Experienced).

   (a) How experienced are you with running serial jobs on a shared computing resource?

   (b) How experienced are you with running parallel jobs on a shared computing resource?

   (c) How experienced are you with using MPI?

   (d) How experienced are you with running serial applications on different shared computing resources?

   (e) How experienced are you with running parallel applications on different shared computing resources?

6. Please indicate how often you perform / have performed the following tasks on a scale of 1 (Never) to 4 (Frequently) or mark “unsure”.

   (a) How often do you install software related to your computational research?

   (b) How often do you run MPI applications via a queue manager?

   (c) How often do you compile MPI applications?

(d) How often do you use environment modules?

7. Please indicate how comfortable you are with performing the following tasks on a scale of 1 (Not Very) to 4 (Very) or mark "unsure".

   (a) How comfortable are you with installing software related to your computational research?

   (b) How comfortable are you with running parallel jobs via a queue manager?

   (c) How comfortable are you with compiling MPI applications?

   (d) How comfortable are you with loading environment modules?

8. What MPI implementations have you used before? Mark all that apply.

   (a) MPICH/MPICH2

   (b) MVAPICH/MVAPICH2

   (c) OpenMPI

   (d) Unsure

   (e) Other

9. What shared resource managers have you used? Mark all that apply.

   (a) Portable Batch System (PBS)

   (b) Sun Grid Engine (SGE)

   (c) Simple Linux Utility for Resource Management (SLURM)

   (d) Moab Workload Manager or Maui Cluster Scheduler with TORQUE

   (e) Platform LSF Workload Manager

   (f) Other

10. How many computing sites have you used for your computational research?

1

2

3

4

More than 4

11. List anything that you find challenging, difficult, tedious, or time consuming about running jobs on shared resources. (Enumerated answers were provided by participants.)

    (a) bad environment

    (b) special compiler settings

    (c) lacking licenses

    (d) queue delay

    (e) compiling/debugging

    (f) failing cluster

    (g) submission details

    (h) lacking resources

12. List any reasons why you may or may not move an MPI application to run at a new computing site. (Enumerated answers were provided by participants.)

    (a) inefficient access

    (b) less powerful

    (c) failure resiliency

    (d) run larger jobs (more compute/memory, newer hardware)

    (e) long setup/steep learning curve

    (f) data movement

    (g) small allocation

(h) better software

13. Please indicate if you have run any jobs before at the following computing sites:

    (a) Blacklight at the Pittsburgh Supercomputing Center:

    (b) Forge at the National Center for Supercomputing Applications:

    (c) Gordon Compute at the San Diego Supercomputing Center:

    (d) Kraken at the National Institute for Computational Sciences:

    (e) Ranger at the Texas Advanced Computing Center:

    (f) Steele at the Rosen Center for Advanced Computing at Purdue University:

14. Have you used the XSEDE or TeraGrid portal before?

    (a) Yes as XSEDE

    (b) Yes as TeraGrid

    (c) No

15. Why have you used more than one site? (Enumerated answers were provided by participants.)

    (a) testing portability/resiliency

    (b) collaboration

    (c) more resources

    (d) better performance

    (e) shorter queue time

    (f) more user friendly environment

    (g) more cost effective

A.2.2 Participant Answers

Table A.3: Experience Survey Answers: Participants 1 to 12, Questions 1 to 8

	PARTICIPANT ID											
Question	1	2	3	4	5	6	7	8	9	10	11	12
1	6+	6+	6+	6+	6+	6+	6+	6+	6+	3-5	3-5	3-5
2	6+	6+	6+	6+	6+	6+	1-3	3-5	6+	3-5	3-5	3-5
3	6+	6+	6+	6+	6+	6+	3-5	0-1	6+	1-3	3-5	3-5
4a	x	x		x		x		x	x	x	x	x
4b	x	x	x	x		x	x	x	x	x	x	x
4c		x					x	x	x		x	x
4d	x										x	
4e	x	x	x	x	x	x	x	x	x	x	x	x
4f			x								x	
5a	4	3	1	4	1	4	2	3	4	2	3	3
5b	4	3	3	4	4	4	1	2	4	3	3	3
5c	4	3	2	2	3	4	2	2	2	2	3	3
5d	4	3	1	4	1	4	1	1	1	1	3	2
5e	4	3	2	3	1	4	1	1	1	2	3	2
6a	4	2	2	4	1	4	4	1	4	2	2	2
6b	4	4	4	2	3	4	2	3	4	4	2	3
6c	4	4	2	2	2	4	1	3	3	2	2	3
6d	4	3	2	3	2	4	1	4	3	3	4	1
7a	4	2	2	4	2	4	3	3	4	3	3	4
7b	4	4	3	4	4	4	3	3	4	3	4	4
7c	4	4	1	3	2	4	1	4	4	2	3	4
7d	4	3	2	4	3	4	1	4	4	3	4	1
8a	x	x		x	x	x	x	x	x	x		x
8b	x	x	x		x	x						
8c	x	x		x		x			x	x	x	
8d												
8e	x								x		x	

Table A.4: Experience Survey Answers: Participants 1 to 12, Questions 9 to 15

	PARTICIPANT ID											
Question	1	2	3	4	5	6	7	8	9	10	11	12
9a	x	s	x	x	x		x	x	x	x		x
9b	x	s							x		x	
9c	x										x	
9d	x									x		
9e				x	x				x			
9f	x								x			
10	5+	4	1	5+	5+	5+	3	2	3	4	5+	4
11a	x											
11b		x		x								
11c					x							
11d								x				
11e			x		x		x		x	x		
11f											x	
11g												
11h												
12a	x											
12b	x											
12c		x										
12d			x					x		x		
12e							x		x			x
12f									x			
12g												
12i												
13a				x								
13b												
13c												
13d						x						
13e		x	x		x	x						
13f												
14a		x	x	x		x						
14b		x	x		x	x						
14c	x						x	x	x	x	x	x
15a	x	x										
15b	x											
15c		x								x		
15d					x							x
15e								x			x	
15f									x			
15g												

Table A.5: Experience Survey Answers: Participants 13 to 25, Questions 1 to 8

	PARTICIPANT ID												
Question	13	14	15	16	17	18	19	20	21	22	23	24	25
1	3-5	3-5	3-5	3-5	1-3	1-3	1-3	1-3	1-3	1-3	0-1	0-1	0-1
2	3-5	3-5	1-3	3-5	1-3	3-5	1-3	0-1	1-3	1-3	0-1	0-1	0-1
3	3-5	3-5	1-3	3-5	1-3	1-3	0-1	0-1	1-3	1-3	0-1	0-1	0-1
4a	x	x	x		x	x	x	x	x	x		x	x
4b	x		x	x	x	x	x	x	x	x	x	x	x
4c	x			x	x			x	x	x			
4d					x				x				x
4e	x	x	x		x	x		x	x	x	x	x	
4f	x				x				x				
5a	4	3	4	1	4	4	2	3	1	3	1	1	3
5b	4	2	3	3	3	3	2	2	3	3	1	3	3
5c	4	2	3	3	3	3	1	1	3	2	1	3	2
5d	3	1	1	1	1	3	1	1	1	3	1	1	2
5e	3	1	1	1	1	2	1	1	1	3	1	1	2
6a	4	2	3	2	4	2	2	1	4	2	1	3	2
6b	3	4	3	3	4	2	4	3	4	2	4	4	3
6c	4	4	3	2	4	2	2	2	2	2	4	4	3
6d	2	2	3	3	4	1	4	2	2	2	2	3	3
7a	4	1	3	2	4	4	1	2	4	2	1	3	3
7b	4	3	2	3	3	4	4	3	3	2	3	3	3
7c	4	3	2	3	2	3	1	1	2	2	2	3	3
7d	4	2	2	3	1	1	4	2	2	3	2	3	3
8a	x		x	x	x	x	x	x	x	x	x	x	x
8b	x	x									x	x	x
8c	x	x		x			x		x	x		x	x
8d													
8e	x												

Table A.6: Experience Survey Answers: Participants 13 to 25, Questions 9 to 15

	PARTICIPANT ID												
Question	13	14	15	16	17	18	19	20	21	22	23	24	25
9a	x	x	x	x		x	x	x	x	x	x		
9b	x	x											x
9c					x								
9d		x					x			x			
9e													
9f						x							
10	5+	3	1	1	1	3	1	1	1	3	1	3	3
11a													
11b				x					x				
11c													
11d						x					x	x	
11e				x	x	x			x		x	x	
11f													
11g	x	x		x									
11h													x
12a				x	x								
12b													
12c													
12d		x									x	x	
12e			x		x			x		x			
12f											x		
12g	x												
12i													x
13a													
13b										x			
13c		x											
13d	x									x			
13e		x						x				x	x
13f													
14a		x		x			x	x		x		x	x
14b	x				x		x			x			
14c			x			x			x		x		
15a													
15b	x												
15c												x	
15d						x							
15e										x			
15f													
15g			x										

A.3 Background Survey

This section presents the study survey that collected background information about participants such as their place of employment, gender, and job title. The results were used to create profiles of the participants as summarized in Figure 2.4.

A.3.1 Survey Questions

1. What is your area of research?

2. What is your job title?

   Faculty

   Researcher

   Post doc

   IT administrator

   Graduate student

   Undergraduate student

   Other:

3. Please describe your place of employment. (Select all that apply)

   Industry

   Government laboratory

   Research focused

   Teaching focused

   Baccalaureate granting institution

   Master's granting institution

   Doctorate granting institution

   Associate college

Minority serving institution

Other:

4. Please estimate the size of your place of employment in terms of number of employees.

5. Gender

6. Race/Ethnicity

   White

   Asian

   Black/African American

   Spanish/Hispanic/Latino

   American Indian/Alaska Native

   Native Hawaiian/Pacific Islander

   Other:

A.3.2 Participant Answers

Table A.7: Background Survey Answers

Job Title	**Place of Employment**	**Gender**	**Race**	**Area**
faculty	university	Male	White	Materials Science
grad	university	Male	White	Materials Science
grad	university	Male	White	Other
grad	university	Male	Asian	Computer Science
post doc	university	Male	Asian	Biology
faculty	university	Male	Asian	Mechanical Engineering
post doc	university	Female	Asian	Chemistry
grad	university	Male	Moroccan	Computer Science
faculty	university	Female	White	Geology
grad	university	Male	White	Materials Science
post doc	university	Male	Asian	Materials Science
grad	university	Male	Asian	Physics
grad	university	Male	White	Computer Science
grad	university	Male	White	Computer Science
grad	university	Male	White	Physics
post doc	university	Male	Asian	Chemistry
grad	government	Male	Asian	Economics
grad	university	Male	Asian	Mechanical Engineering
undergrad	university	Male	Black	Computer Science
engineer	industry	Male	White	Environmental Engineering
undergrad	university	Male	White	Physics
grad	university	Male	Asian	Mathematics
grad	university	Female	White	Biomedical Engineering
faculty	university	Male	White	Biology
mathematician	government	Male	White	Environmental Engineering

A.4 Post Migration Survey

This section presents the study survey that was conducted to assess participants' experiences during the migration process. The results were used to clarify the documented migration log data and direct conversation during the exit interview.

A.4.1 Survey Questions

Questions 1 to 5 ask about your experience at each of the possible migration sites.

1. Did you try to execute your MPI application at this site? If not, please explain why.
   (a) Blacklight:
   (b) Forge:
   (c) Gordon Compute:
   (d) Kraken:
   (e) Ranger:
   (f) Steele:
2. Did your MPI application execute successfully at this site? If not, please explain why.
   (a) Blacklight:
   (b) Forge:
   (c) Gordon Compute:
   (d) Kraken:
   (e) Ranger:
   (f) Steele:
3. If you did not successfully execute your MPI application at this site, did you spend long enough at the site for it to qualify as a "migration attempt". If not, explain why.
   (a) Blacklight:

(b) Forge:

(c) Gordon Compute:

(d) Kraken:

(e) Ranger:

(f) Steele:

4. What task type(s), if any, would you rate as the most difficult to perform at this site? Please use examples.

   (a) Blacklight:

   (b) Forge:

   (c) Gordon Compute:

   (d) Kraken:

   (e) Ranger:

   (f) Steele:

5. What task type(s), if any, would you rate as the most tedious to perform at this site? Please use examples.

   (a) Blacklight:

   (b) Forge:

   (c) Gordon Compute:

   (d) Kraken:

   (e) Ranger:

   (f) Steele:

6. Describe any characteristics of your MPI application that slowed down or made the migration process more difficult. (i.e. parallelization technique, required library, version dependency, lack of source code)

7. Describe any characteristics of the migration sites that slowed down or made the migration process more difficult. (i.e. lack of software, lack of documentation, misconfigured MPI libraries)

8. Describe any other characteristics that slowed down or made the migration process more difficult. (i.e. XSEDE portal interface, delay with assistance, lack of experience)

9. Describe your experience with using the XSEDE portal for the migrations. (What was helpful/difficult/awkward/tedious?)

10. A follow-on to this study could involve us asking for a copy of your code so that we could try migrating it automatically with a tool we are developing. Do you think members of your community would be willing to participate in such a study? If not, please explain to help guide us.

11. Did you find participating in this study useful in any way? Please explain.

A.5 Test

This section presents the test that participants completed to evaluate their computational experiences with MPI, queuing systems, and running jobs. The resulting scores were used to profile participants as summarized in Figure 2.6.

A.5.1 Test Questions

1. What is MPI? (Mark all that apply)

   UNFAMILIAR

   a library

   a programming language

   a specification

2. MPI implementations are: (Check one)

   UNFAMILIAR

   libraries invoked from traditional sequential languages

   programming language extensions

   complete parallel programming models

3. List some MPI implementations:

   UNFAMILIAR

4. Can code compiled with one MPI implementation be run with a different MPI implementation? Why or why not?

   UNFAMILIAR

5. How would you determine the location and version of the MPI implementation your application is using?

   UNFAMILIAR

6. How would you determine what MPI implementations are available at a computing site?

   UNFAMILIAR

7. How would you determine what compilers are available at a computing site?

   UNFAMILIAR

8. What is a queue manager?

   UNFAMILIAR

9. Outline the parts of a file used to submit a job to a queue manager.

   UNFAMILIAR

10. What command is used to submit a job to a queue manager?

    UNFAMILIAR

11. What command is used to check the status of a job?

    UNFAMILIAR

12. What command starts an MPI job?

    UNFAMILIAR

13. Name at least one UNIX shell.

    UNFAMILIAR

14. What are typical parallel programming communication models that differ in how parallel tasks communicate?

    UNFAMILIAR

    shared memory/message passing models

    task/channel models

    data/task models

    Gustafson-Barsis/Karp-Flatt models

15. Which parallel programming communication model is MPI based on?

    UNFAMILIAR

16. What are typical paradigms for paralleling programs that differ in what they focus on distributing across computing nodes?

    UNFAMILIAR

    shared memory/message passing models

    task/channel models

    data/task models

    Gustafson-Barsis/Karp-Flatt models

17. Distinguish between statically and dynamically linked libraries.

    UNFAMILIAR

18. What is a shared library?

    UNFAMILIAR

19. What environment variable(s) influence the paths that are used at runtime to resolve shared library dependencies?

    UNFAMILIAR

20. What command(s) can be used to check the shared library dependencies of a binary?

    UNFAMILIAR

21. What command(s) can be used to check the file format of a binary?

    UNFAMILIAR

22. What are environment modules and how are they used?

    UNFAMILIAR

23. What is an example of an MPI function that can be used to send messages between two specific processes (a.k.a. point to point communication)?

    UNFAMILIAR

24. What is an example of an MPI function that can be used to send messages between all processes in a process group (a.k.a. collective communication)?

    UNFAMILIAR

Consider the following error messages (in questions 25-32) returned by a queue manager in a job output file. Provide a short guess at their possible causes and solutions.

25. “cannot execute binary file”

    UNFAMILIAR

26. “/usr/bin/ld: cannot find -lrdmacm”

    UNFAMILIAR

27. “gcc versions do not match”

    UNFAMILIAR

28. “error while loading shared libraries: libmpich.so.1.2: cannot open shared object file: No such file or directory”

    UNFAMILIAR

29. “load failed: Executable file is not a 32-bit ELF file”

    UNFAMILIAR

30. “Floating point exception”

    UNFAMILIAR

31. “Symbol ‘ompimpicommworld’ has different size in shared object, consider re-linking”

    UNFAMILIAR

32. “mpiexec: Error: gethosts: PBS reports more tasks 4 than TM 4”

    UNFAMILIAR

End Time:

Did you consult references? Yes No

A.5.2 Test Scores

The test scores indicate whether each participant answered each questions correctly (*y*), incorrectly (*n*), or as unsure (*u*).

Table A.8: Test Scores: Participants 1 to 12

	PARTICIPANT ID											
Question	1	2	3	4	5	6	7	8	9	10	11	12
1	y	y	y	y	y	n	y	y	n	n	n	n
2	y	y	n	y	n	n	n	n	y	n	n	y
3	y	y	y	y	y	y	y	y	y	n	n	y
4	y	n	y	y	u	y	y	u	y	u	y	y
5	y	y	y	y	u	y	y	u	u	y	y	y
6	y	y	y	y	y	y	y	y	y	y	y	y
7	y	y	y	y	y	y	y	y	y	y	y	y
8	y	y	y	y	y	y	y	y	u	y	y	y
9	y	y	n	y	y	y	y	y	y	y	y	y
10	y	y	y	y	y	y	y	y	y	y	y	y
11	y	y	y	y	y	y	y	y	y	y	y	y
12	y	y	n	y	y	y	y	y	y	y	y	y
13	y	y	y	y	y	y	y	y	y	y	y	y
14	y	y	u	n	u	u	y	y	y	u	u	y
15	y	y	u	n	u	u	y	u	u	u	u	y
16	n	y	u	n	u	u	n	n	u	n	u	n
17	y	y	u	y	y	y	y	y	y	u	u	y
18	y	n	u	y	y	y	y	y	y	u	y	y
19	y	y	u	y	y	y	y	y	y	u	u	y
20	y	u	y	y	u	y	y	y	y	u	n	u
21	y	u	y	y	u	y	y	u	y	u	u	u
22	y	y	u	y	n	y	y	y	y	y	u	u
23	y	y	y	u	u	u	y	y	y	u	u	y
24	y	y	y	u	n	u	y	y	y	y	u	y
25	y	y	y	y	y	y	y	y	y	u	y	y
26	y	y	y	y	y	y	y	u	y	u	u	y
27	y	y	y	y	y	y	y	y	y	y	y	y
28	y	y	y	y	y	y	y	u	y	y	u	y
29	y	y	n	y	y	y	y	u	y	u	u	u
30	y	y	y	y	y	n	y	u	y	u	y	y
31	y	u	u	y	u	y	y	u	y	u	u	u
32	y	u	u	y	u	n	u	u	u	u	u	u

Table A.9: Test Scores: Participants 13 to 25

	PARTICIPANT ID												
Question	13	14	15	16	17	18	19	20	21	22	23	24	25
1	n	y	y	y	y	n	n	y	y	y	y	n	n
2	n	n	y	y	y	y	y	u	y	y	u	y	n
3	y	u	y	y	y	y	y	y	y	y	y	y	y
4	y	u	u	u	n	y	n	u	y	n	y	y	y
5	y	u	n	u	u	y	y	u	y	y	u	u	u
6	y	u	y	y	u	u	y	y	y	y	u	y	y
7	y	u	y	y	u	y	y	y	y	y	y	y	y
8	y	y	y	y	y	y	y	y	y	y	y	y	y
9	y	y	y	y	u	y	y	y	y	y	y	y	y
10	y	y	y	y	y	y	y	y	y	y	y	y	y
11	y	y	y	y	u	y	y	y	y	y	y	y	y
12	u	y	y	y	y	y	y	y	y	y	u	y	y
13	n	y	y	n	u	y	y	y	y	y	y	y	y
14	u	u	u	y	n	y	y	u	y	y	y	u	n
15	u	u	u	u	n	y	y	u	u	y	u	u	u
16	u	u	u	n	u	u	n	u	n	y	u	u	u
17	u	y	y	y	u	y	u	u	n	y	u	u	y
18	u	y	u	u	u	n	u	u	n	n	u	u	n
19	u	u	u	u	u	y	y	u	y	y	u	y	u
20	u	u	u	u	u	y	u	u	u	u	u	y	u
21	u	u	u	u	u	y	u	u	u	u	u	y	u
22	u	n	u	y	u	u	y	y	y	y	u	n	n
23	u	u	n	y	y	y	y	u	u	y	y	y	y
24	u	u	y	y	u	u	y	y	u	y	y	y	y
25	u	y	y	y	y	y	y	y	y	y	y	y	y
26	u	u	y	y	u	y	y	y	y	y	u	y	u
27	y	y	u	y	u	y	y	u	y	y	u	y	y
28	y	u	y	u	y	u	y	y	y	y	y	y	y
29	u	u	u	u	y	u	u	u	y	y	y	y	y
30	u	y	y	u	y	u	y	u	y	y	y	y	u
31	u	u	u	u	y	u	y	u	u	u	u	u	y
32	u	u	y	u	u	u	y	u	y	u	y	y	u

Table A.10: Migration Durations: Blacklight, Participants 1 to 12

	PARTICIPANT ID											
BLACKLIGHT	1	2	3	4	5	6	7	8	9	10	11	12
Learning	10	2	70		7	10		30		5	18	21
Difficulty	1	2	2.5		5	1		2		1	4	1
Tediousness	1	2	3		5	2		3		3	3	2.9
Compiling	10	1	10		168	10		30		10	18	1
Difficulty	1	2	1		4	1		2		2	4	1
Tediousness	1	2	2		3	1		3		3	3	1
Debugging	5									2		
Difficulty	1									4		
Tediousness	2									4		
Environment Setting	10	2	60		21	15	30	30			18	8
Difficulty	1	2	3.8		5	1	1	2			4	2
Tediousness	1	2	3.8		5	1	1	3			3	1
Requesting Assistance											9	
Difficulty											4	
Tediousness											3	
Submitting	5	8	20		7		30			3	27	26
Difficulty	1	2	1		3		3			4	4	1.5
Tediousness	1	2	2		1		1			4	3	1.5
Testing	5		30		7	40				3		
Difficulty	1		1		3	3				4		
Tediousness	2		2		1	2				4		
Total Time (mins)	45	13	190		210	75	60	90		23	90	56
Consecutive Days	2	1	3		7	1	2	1		9	1	2
Active Days	2	1	3		2	1	2	1		5	1	2
Migration Order Number	1	2	1		1	3	3	2		6	5	1

A.6 Migration Log Summaries

This section presents the summary of the contents of each participant's migration logs (per the example shown in Figure 2.1). The summaries are presented for each migration site (described in Section 2.1.2) in terms of the amount of time (in minutes) spent on each category of task and the average tediousness and difficulty rating (on a scale of 1 (low) to 5 (high)) for that task. Each table also presents the number of consecutive and active days each participant spent on the migration at the particular site and each migration's order number (as defined in Section 2.4.1.3).

Table A.11: Migration Durations: Blacklight, Participants 13 to 25

	PARTICIPANT ID												
BLACKLIGHT	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	20	10		40		28	20	18	30	45	25		
Difficulty	1	2		1.5		3.9	1	1	2	2.0	3		
Tediousness	2	2		2		5	2	2	2	3.3	3		
Compiling	15			10		1	25	8	20	30	20		45
Difficulty	2			4		1	2	2	2	2.0	3		3
Tediousness	2			5		1	3	1	2	3.0	3		1
Debugging	25			80		28			60	160			45
Difficulty	1.8			4		1			2	2.0			3
Tediousness	2.4			5		5			2	4.0			1
Environment Setting	10	10		32		41	35	10					45
Difficulty	1	1		1		1	2	2					3
Tediousness	2	2		4		3	3	1					1
Requesting Assistance		40				16							
Difficulty		5				1							
Tediousness		5				5							
Submitting	20	60				12	5	7	10	20	15		45
Difficulty	1.5	5				1	1	1	2	2.0	1		3
Tediousness	2.5	5				5	1	2	2	0.4	2		1
Testing				50			15						
Difficulty				2			1						
Tediousness				2.3			2						
Total Time (mins)	90	120		212		126	100	43	120	255	60		180
Consecutive Days	1	5		10		22	4	1	3	2	4		1
Active Days	1	3		7		5	2	1	2	2	2		1
Site Order Number	2	3		1		1	3	4	5	1	4		1

Table A.12: Migration Durations: Forge, Participants 1 to 12

	PARTICIPANT ID											
FORGE	1	2	3	4	5	6	7	8	9	10	11	12
Learning		2	25		2	10				15	18	40
Difficulty		2	2		3	1				3	4	
Tediousness		2	3		3	2				3	3	
Compiling	9	1	20	20	75	30				10	27	5
Difficulty	1	2	1	1	3	1				2	4	
Tediousness	1	2	2	3	3	1				3	3	
Debugging				10			30			22		
Difficulty				1			5			3.5		
Tediousness				3			1			3.1		
Environment Setting	3	2	10		3	15	15				18	15
Difficulty	1	2	2		3	1	1				4	
Tediousness	1	2	3		3	1	5				3	
Requesting Assistance					7		10					
Difficulty					1		3					
Tediousness					1		3					
Submitting	9		10	20	10		90			30	27	30
Difficulty	1		1	1.5	3		3.0			3.4	4	
Tediousness	1		2	1.5	3		3.0			2.9	3	
Testing	9		15	10	10	115				13		10
Difficulty	1		1	2	3	3				3.2		
Tediousness	1		2	2	3	2				3.2		
Total Time (mins)	30	5	80	60	107	170	145			90	90	100
Consecutive Days	1	1	2	1	3	1	1			9	1	5
Active Days	1	1	2	1	2	1	1			4	1	3
Migration Order Number	2	3	2	2	3	4	4			1	6	2

Table A.13: Migration Durations: Forge, Participants 13 to 25

	PARTICIPANT ID												
FORGE	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	60	25		30	180		30	5		15		30	
Difficulty	3	2.4		2	2.2		1	1		2.0		2	
Tediousness	1.67	3.6		4	3.3		2.3	1		4.0		5	
Compiling	35			55			90	5	15	30		5	30
Difficulty	3			3.5			3	1	3	2.0		1	2
Tediousness	4			4			4	1	4	4.0		3	3
Debugging	20			160			30	13	45	90			30
Difficulty	5			3			3	2	1	2.0			2
Tediousness	5			4			3	2	4	4.0			3
Environment Setting	15			3			20	15	20	40		30	30
Difficulty	2			1			1	1	3	2.0		2	2
Tediousness	2			1			3	1	3	4.0		3	3
Requesting Assistance	5	45		25	30				25				
Difficulty	1	1.3		1	3				2				
Tediousness	1	3.3		3	4				4				
Submitting	15			5			15	18	10	20		10	30
Difficulty	2			1			1	1.7	3	2.0		2	2.0
Tediousness	3			2			1	1.6	3	4.0		5	3.0
Testing				40								5	
Difficulty				1								1	
Tediousness				3								2	
Total Time (mins)	150	70		318	210		185	56	115	195		80	120
Consecutive Days	2	5		27	18		9	1	2	2		1	1
Active Days	2	3		12	3		2	1	2	2		1	1
Site Order Number	3	4		2	1		4	5	6	2		1	2

Table A.14: Migration Durations: Gordon Compute, Participants 1 to 12

	PARTICIPANT ID											
GORDON COMPUTE	1	2	3	4	5	6	7	8	9	10	11	12
Learning	2	12	20	20	2.4					5	120	9
Difficulty	1	4.5	2	3	3					1	4	1
Tediousness	3	4.5	2	3	3					3	3	2.4
Compiling	5	1	20		78	75				10	60	1
Difficulty	1	1	2.5		3	1.4				2	4	1
Tediousness	2	1	3		3	1.67				3	3	1
Debugging	3		30				230			2		
Difficulty	1		4				5			4		
Tediousness	3		4				1			4		
Environment Setting	5	2	40	25	3.6	30	10				60	6
Difficulty	1	2	2	3	3	1	4.6				4	1
Tediousness	3	2	3	3	3	1	1.8				3	1
Requesting Assistance					7		10					
Difficulty					3		4					
Tediousness					3		3					
Submitting	2	5	10	30	12					10	30	23
Difficulty	1	2	1	2	3					3.3	4	2
Tediousness	3	2	2	4	3					2.6	3	3
Testing	8		10	25	24	5				3	30	10
Difficulty	1		1	2	3	1				4	4	1
Tediousness	2.5		2	4	3	2				4	3	2
Total Time (mins)	25	20	130	100	127	110	250			30	300	49
Consecutive Days	1	1	1	1	5	1	27			6	2	1
Active Days	1	1	1	1	2	1	7			3	2	1
Migration Order Number	3	4	3	1	4	1	5			2	1	3

Table A.15: Migration Durations: Gordon Compute , Participants 13 to 25

	PARTICIPANT ID												
GORDON COMPUTE	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	20	15							25				
Difficulty	1	1.7							2				
Tediousness	1	1.7							2				
Compiling	5			10			10		50				
Difficulty	1			1			2		3				
Tediousness	1			2			2		3				
Debugging	30			60					95				
Difficulty	3			5					4.8				
Tediousness	3			5					3				
Environment Setting	10	20		10			20		5				
Difficulty	1	5		1			2		1				
Tediousness	1	5		1			2		1				
Requesting Assistance		50											
Difficulty		4.4											
Tediousness		4.4											
Submitting	10	50		10					15				
Difficulty	2	3.8		2					2				
Tediousness	2	2.8		2					2				
Testing		30					15						
Difficulty		3					2						
Tediousness		2					4						
Total Time (mins)	75	165		90			45		190				
Consecutive Days	1	8		5			1		2				
Active Days	1	4		3			1		2				
Site Order Number	4	5		3			5		1				

Table A.16: Migration Durations: Kraken, Participants 1 to 12

	PARTICIPANT ID											
KRAKEN	1	2	3	4	5	6	7	8	9	10	11	12
Learning	60		40		5					5	30	88
Difficulty	2		4		5					1	5	1.6
Tediousness	3		4		5					3	3	3.05
Compiling	25		20	210	92					10	45	2
Difficulty	3.6		1	3.5	5					2	5	1
Tediousness	4.8		2	5	5					3	3	1
Debugging	85									2		15
Difficulty	3.9									4		3.33
Tediousness	3.5									4		2.33
Environment Setting	40		75		94						15	8
Difficulty	2.8		3		5						5	1.5
Tediousness	2.9		3.5		5						3	1.5
Requesting Assistance	10										45	
Difficulty	1										5	
Tediousness	2										3	
Submitting			20	15	24					3	15	18
Difficulty			1	3	5					4	5	2.88
Tediousness			2	4	5					4	3	16.33
Testing	60			30	7					3		
Difficulty	2			3	5					4		
Tediousness	3			4	5					4		
Total Time (mins)	280		155	255	222					23	150	131
Consecutive Days	7		2		5					10	2	6
Active Days	2		2	1	2					5	2	4
Migration Order Number	4		4	3	6					3	2	6

Table A.17: Migration Durations: Kraken, Participants 13 to 25

	PARTICIPANT ID												
KRAKEN	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	35	24						58	60		50	15	
Difficulty	2.3	5						1.7	3.5		1.4	2	
Tediousness	2.3	5						2.1	3.5		1.5	5	
Compiling	25							20	20		25	360	
Difficulty	3							3	2		2	5	
Tediousness	3							4	2		2	5	
Debugging	45							123					
Difficulty	5							3.3					
Tediousness	5							2.8					
Environment Setting	10	36						80			60	20	
Difficulty	1	5						2.7			3	1	
Tediousness	2	5						2.3			4	5	
Requesting Assistance	5	60						15	25		15	10	
Difficulty	1	5						1.0	2		2	1	
Tediousness	3	5						2.0	4		2.3	4	
Submitting	10	30						28			65		
Difficulty	1	3						2.0			3		
Tediousness	2	4						2.1			4		
Testing		30						50				5	
Difficulty		3						3.8				1	
Tediousness		4						2.9				2	
Total Time (mins)	130	180						374	105		215	410	
Consecutive Days	2	1						32	8		5	8	
Active Days	2	1						5	3		4	4	
Site Order Number	5	6						1	2		1	2	

Table A.18: Migration Durations: Ranger, Participants 1 to 12

	PARTICIPANT ID											
RANGER	1	2	3	4	5	6	7	8	9	10	11	12
Learning					5		55		35	5	15	11
Difficulty					4		3		7.3	1	2	3
Tediousness					5		3		2	3	3	3
Compiling	3		10		110		60		90	10	15	5
Difficulty	1		1		4		3		2.3	2	2	1
Tediousness	2		2		5		1		1.75	3	3	1
Debugging	50						90		70	12		20
Difficulty	3						4		1	3.2		4
Tediousness	2						4		8.4	3.2		3
Environment Setting	4		30		18				15		15	10
Difficulty	1		1		4				1		2	1
Tediousness	2		2		5				2		3	1
Requesting Assistance							30			25		
Difficulty							2			1		
Tediousness							1.5			1		
Submitting	10		5		22				30	13	15	20
Difficulty	3		1		4				1	3.2	2	3
Tediousness	2		1		5				2	3.2	3	3
Testing	3		5		66				10	13		
Difficulty	1		1		4				1	3.2		
Tediousness	2		1		5				1	3.2		
Total Time (mins)	70		50		221		235		250	78	60	66
Consecutive Days	1		1		1		28		8	6	1	3
Active Days	1		1		1		4		8	3	1	3
Migration Order Number	5		5		2		1		1	4	3	5

Table A.19: Migration Durations: Ranger, Participants 13 to 25

	PARTICIPANT ID												
RANGER	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	20	10	80	15			5	25	35		25		
Difficulty	2	3	0	2			1	1	3		1		
Tediousness	3	3	1.3	2			1	2	3		2		
Compiling	10	30	90	15			120	10	15		5		30
Difficulty	2	4	0	5			2	2	2		1		1
Tediousness	2	3	0	5			4	1	2		1		5
Debugging	45			20			120	5					30
Difficulty	4			2			2	1					1
Tediousness	4			2			3	2					5
Environment Setting	15	15	30				50	10			5		30
Difficulty	2	3.7	1				2	2			2		1
Tediousness	3	3.0	1				5	1			2		5
Requesting Assistance		5											
Difficulty		3											
Tediousness		3											
Submitting	5	60						15	20		20		30
Difficulty	1	3.2						1	2		1		1
Tediousness	1	2.5						2	4		2		5
Testing							20						
Difficulty							2						
Tediousness							2						
Total Time (mins)	95	120	200	50			315	65	70		55		120
Consecutive Days	1	2	15	1			2	1	7		4		1
Active Days	1	2	4	1			2	1	3		2		1
Site Order Number	6	1	1	4			2	2	3		2		3

Table A.20: Migration Durations: Steele, Participants 1 to 12

	PARTICIPANT ID											
STEELE	1	2	3	4	5	6	7	8	9	10	11	12
Learning		17			2	10		33		5	15	15
Difficulty		4			2	1		1.5		1	3	2
Tediousness		4			2	2		1.9		3	3	2
Compiling	5	1			24	20	10	18		10	15	1
Difficulty	1	2			2	1	5	1		2	3	1
Tediousness	2	2			2	1	1	1		3	3	1
Debugging							20			2		
Difficulty					2		5			4		
Tediousness					2		1			4		
Environment Setting		2			4	15	40	33			15	10
Difficulty		2			2	1	2	1			3	2
Tediousness		2			2	1	2	1			3	2
Requesting Assistance					7							
Difficulty					2							
Tediousness					2							
Submitting		5			4		110	43		3	15	10
Difficulty		2			2		4	1		4	3	1.5
Tediousness		2			2		1	1.8		4	3	1.5
Testing	5				4	15		33		3		5
Difficulty	1				2	1		1		4		1
Tediousness	2				2	2		2.1		4		1
Total Time (mins)	10	25			45	60	180	160		23	60	41
Consecutive Days	1	1			2	1	12	7		9	2	2
Active Days	1	1			2	1	4	3		4	2	2
Migration Order Number	6	1			5	2	2	1		5	4	4

Table A.21: Migration Durations: Steele, Participants 13 to 25

	PARTICIPANT ID												
STEELE	13	14	15	16	17	18	19	20	21	22	23	24	25
Learning	45	69		50		75	30	12	15		60		
Difficulty	2	4.7		1		1	1	1.6	2		3.5		
Tediousness	2	5.0		2		3	3	1.4	3		3.5		
Compiling	45			5		1	30	7	10				45
Difficulty	2			5		1	3	1	1				3
Tediousness	2			5		1	3	2	2				2
Debugging	30			50		135	30	15	60		60		45
Difficulty	2			3		1	3	1	2		3		3
Tediousness	2			4		5	5	2	3		4		2
Environment Setting	40	36		15		29	5	12			20		45
Difficulty	3	5		1		1	1	2			2		3
Tediousness	2	5		5		3	3	1			3		2
Requesting Assistance	25	25		10		20	30	22					
Difficulty	1	4.4		1		1	3	1.3					
Tediousness	1	5.0		5		5	5	1.7					
Submitting	30					19	20		15		5		45
Difficulty	2					1	1		2		2		3
Tediousness	3					5	2		2		3		2
Testing						1	5						
Difficulty						1	1						
Tediousness						1	2						
Total Time (mins)	215	130		130		280	150	68	100		145		180
Consecutive Days	1	2		9		22	9	1	2		3		1
Active Days	1	2		4		7	2	1	2		3		1
Site Order Number	1	2		5		2	1	3	4		3		4

Bibliography

[1] Blacklight. pittsburgh supercomputing center. /http://www.psc.edu/machines/sgi/uv/blacklight.php.

[2] Dell nvidia linux cluster forge. national center for supercomputing applications. http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/DellNVIDIACluster.

[3] FutureGrid: A distributed testbed for clouds, grids, and HPC. https://portal.futuregrid.org.

[4] GNU binutils. http://www.gnu.org/software/binutils.

[5] GNU project autoconf. http://www.gnu.org/software/autoconf.

[6] Gordon. san diego supercomputing center. http://www.sdsc.edu/us/resources/gordon/index.html.

[7] Group, scheduling working. automatic resource selection report. http://www.teragridforum.org/mediawiki/images/1/17/Schedwg_AutoResourceSelectReport.pdf.

[8] Kraken. national institute for computational sciences. http://www.nics.tennessee.edu/computing-resources/kraken.

[9] Lmod: A lua based environment module system. http://lmod.sourceforge.net.

[10] Modules - software environment management. http://modules.sourceforge.net.

[11] Msys - the MCS systems administration toolkit. http://www.mcs.anl.gov/hs/software/systems/msys.

[12] OpenStack: Open source cloud computing software. https://www.openstack.org.

[13] Smart package manager. http://labix.org/smart.

[14] Steele. rosen center for advanced computing. purdue university. http://www.rcac.purdue.edu/userinfo/resources/steele.

[15] Sun constellation linux cluster: Ranger. texas advanced computing center. http://www.tacc.utexas.edu/resources/hpc/#constellation.

[16] Unicore's xnjs idb configuration. http://www.unicore.eu/documentation/manuals/unicore6/unicorex/xnjs-idb.html.

[17] University of virginia alliance for computation science and engineering. http://www.uvacse.virginia.edu/resources.

[18] Virtual computing lab (vcl). https://www.vclcloud.org.

[19] Xsede campus champions. https://www.xseded.org/campus-champions.

[20] XSEDE: Extreme science and engineering discovery environment. https://www.xsede.org.

[21] Xsede user portal. https://portal.xsede.org.

[22] YUM: Yellow dog update modifier. http://yum.baseurl.org.

[23] Methods for Automatic Preparation of Computing Environments for Application Execution: Experiences with MPI. TeraGrid'11 Student Poster, 2011.

[24] Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, and Xiaomin Zhu. From virtualized resources to virtual computing grids: the in-vigo system. *Future Gener. Comput. Syst.*, 21(6):896–909, June 2005.

[25] Edware C. Bailey. Maximum RPM: Taking the red hat package manager to the limit. Red Hat, Inc., 2000.

[26] Roland Balter, Luc Bellissard, Fabienne Boyer, Michel Riveill, and Jean yves Vion-dury. Architecturing and configuring distributed application with olan. In *Proc. IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), The Lake District*, pages 15–18. UK, Springer Verlag, 1998.

[27] Luc Bellissard, Luc Bellissard, Slim Ben Atallah, Slim Ben Atallah, Fabienne Boyer, Fabienne Boyer, Michel Riveill, Michel Riveill, and Projet Sirac. Distributed application configuration. In *Proc. of 16th IEEE Intl. Conf. on Distributed Computing Systems, Hong-Kong*, pages 579–585, 1996.

[28] Stuart K. Card, Thomas P. Moran, and Allen Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7):396–410, July 1980.

[29] Ayla Dantas, Guilherme Germoglio, Flávio Santos, Marcelo Iury Oliveira, Sandro Rafaeli, Katia Saikoski, Dejan Milojicic, Walfredo Cirne, and Francisco Brasileiro. Using web services for configuration and deployment accoridng to the CDDLM standard. In *International Conference on Web Services*, Chicago, IL, September 2006.

[30] Wojtek Goscinski and David Abramson. Distributed ant: A system to support application deployment in the grid. In *International Workshop on Grid Computing*, pages 436–443, Pittsburgh, PA, November 2004.

[31] Andrew Grimshaw, Mark Morgan, and Karolina Sarnowska. WS-Naming: location, migration, replication, and failure transparency support for web services. *Concurrency and Computation: Practice and Experience*, 21(8):1013–1028, February 2009.

[32] Philip J. Guo and Dawson Engler. Cde: Using system call interposition to automatically create portable software packages. In *Proc. of the 2011 USENIX Annual Technical Conference*, June 2011.

[33] Wei Huang, Matthew J. Koop, Qi Gao, and Dhabaleswar K. Panda. Virtual machine aware communication libraries for high performance computing. SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 2007.

[34] Eduardo Huedo, Rubn S. Montero, and Ignacio M Llorent. The gridway framework for adaptive scheduling and execution on grids. *Scientific International Journal for Parallel and Distributed Computing*, 5:1–8, 2005.

[35] C.-D. Lu, M. D. Jones, and T. R. Furlani. Automatically mining program build information via signature matching. In *TeraGrid 11 Workshop*, Salt Lake City, UT, July 2011.

[36] Ritch Macefield. How to specify the participant group size for usability studies: A practitioners guide. *Journal of Usability Studies*, (1):34–45, November 2009.

[37] Julia McCarthy and Brent Miller. Solution deployment descriptor SDD, part 1: An emerging standard for deployment artifacts. IBM DeveloperWorks, 2008.

[38] Robert McLay, Karl W. Schulz, William L. Barth, and Tommy Minyard. Best practices for the deployment and management of production hpc clusters. In *State of the Practice Reports*, SC '11, pages 9:1–9:11, New York, NY, USA, 2011. ACM.

[39] Matthias S. Müller, Matthijs van Waveren, Rob Lieberman, Brian Whitney, Hideki Saito, Kalyan Kumaran, John Baron, William C Brantley, Chris Parrott, Tom Elken, Yuiyu Feng, and Carl Ponder. SPEC MPI2007 – an application benchmark suite for parallel systems using MPI. *Concurrency and Computation: Practice and Experience*, 22(2):191–205, 2010.

[40] M. Papakhian. Comparing job-management systems: the user's perspective. *Computational Science Engineering, IEEE*, 5(2):4 –9, apr-jun 1998.

[41] Nathan E. Rosenblum, Barton P. Miller, and Xiaojin Zhu. Extracting compiler provenance from program binaries. In *the 9th ACM SIGPLAN-SIGSOFT workshop*, pages 21–28, 2010.

[42] Karolina Sarnowska-Upton and Andrew Grimshaw. Experiences with MPI program migration. In *Grid Computing: The Next Decade*, Zakopane, Poland, January 2012.

[43] Karolina Sarnowska-Upton and Andrew Grimshaw. Predicting execution readiness of MPI binaries with FEAM, a framework for efficient application migration. In *submission to the International Conference on Parallel Processing*, Lyon, France, October 2013.

[44] Karolina Sarnowska-Upton and Andrew Grimshaw. The value of automation: A study of mpi application migration methods. In *11th IEEE International Symposium on Parallel and Distributed Processing with Applications*, Melbourne, Australia, July 2013.

[45] Karolina Sarnowska-Upton, Andrew Grimshaw, and Erwin Laure. Using standards-based interfaces to share data across grid infrastructures. In *International Conference on Parallel Processing*, pages 254–260, Vienna, Austria, September 2009.

[46] Mumtaz Siddiqui, Alex Villazon, Juergen Hofer, and Thomas Fahringer. GLARE: A grid activite registration, deployment, and provisioning framework. In *International Conference on Supercomputing*, Seattle, WA, November 2005.

[47] David Kieras University and David Kieras. Using the keystroke-level model to estimate execution times, 2001.

[48] Rob F. Van der Wijngaart. NAS parallel benchmarks version 2.4, 2002.

[49] Cong Xu, Yuebin Bai, and Cheng Luo. Performance evaluation of parallel programming in virtual machine environment. In *International Conference on Network and Parallel Computing*, Gold Coast, Australia, October 2009.