# A STUDY ON THE SELECTION OF RESOURCES FROM XSEDE SUPERCOMPUTERS AND THE OPEN SCIENCE GRID

**by**

**MING TAI HA**

**A thesis submitted to the**

**School of Graduate Studies**

**Rutgers, The State University of New Jersey**

**In partial fulfillment of the requirements**

**For the degree of**

**Master of Science**

**Graduate Program in The Department of Electrical and Computer Engineering**

**Written under the direction of:**

**Shantenu Jha**

**Matteo Turilli**

**And approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**October, 2018**

## ABSTRACT OF THE THESIS

# A STUDY ON THE SELECTION OF RESOURCES FROM XSEDE SUPERCOMPUTERS AND THE OPEN SCIENCE GRID

**By Ming Tai Ha**

**Thesis Directors:**
**Shantenu Jha and Matteo Turilli**

The effective selection of resources on supercomputers and grids improves workload scheduling and reduces workload time-to-completion. Lower workload time-to-completions allow scientists to gain scientific insights from simulations more quickly. For example, molecular dynamics (MD) simulations are revolutionizing the development of new therapeutics. In the past 6 years, at least 4 billion core-hours consumed on XSEDE machines were from MD software packages alone. Given the increasing number of MD simulations executed and the increasing amounts of computation they require, there is a need for greater efficiency in resource utilization. Thus, we investigate how resources from XSEDE supercomputers and OSG can be effectively selected to reduce workload time-to-completion.

Effective resource selection on grids and supercomputers is difficult. Grids have heterogeneous and transient pools of resources, whose availability and performance vary over time. This makes it difficult to collect information, such as benchmarking results, application profiles and hardware capabilities, used by techniques like application performance modeling and benchmarking to best select grid resources. While the performance of supercomputing resources are easier to assess for grid resources, the acquisition of these resources requires waiting on a queue, sometimes for long periods of time. However, accurately predicting queue waiting time remains difficult.

In this thesis, we studied how to effectively select resources from XSEDE supercomputers and OSG in the presence of limited information. We developed a formalism that allows us to model the cost of task execution based on the information available from XSEDE supercomputers and XSEDE OSG. On the base of our formalism, we constructed the Limited Information Model (LIM) to predict the execution times of compute-intensive, single-threaded, single-process tasks. We evaluated the accuracy of these predictions and the resources selected using these predictions

to gain insight into what information, if any, would be needed to operate better predictions and resource selections. To overcome the difficulty of selecting resources using queue waiting times, we also developed the resource re-selection process, by which tasks are re-assigned to different, acquired resources at runtime. Resource re-selection uses task execution times and resource acquisition times, but not queue waiting times. We show that tasks can be effectively re-assigned even when using inaccurate execution time predictions.

Experimental validation of LIM shows that LIM's predictions are within 157–171% error on XSEDE supercomputers and 18–31% on OSG. By accounting for the differences in software configurations on XSEDE supercomputers and OSG, LIM's predictions can still be used correctly rank XSEDE supercomputers and OSG. Experiments also show that workloads executed using resources selected with LIM's predictions have 67–78% lower workload time-to-completion than those executed using randomly selected resources. However, executing workloads on resources selected using LIM's predictions contributed to ∼29–99% of the reduction in workload time-to-completion. We found that the queue waiting times largely influences workload time-to-completion and should also be considered to effectively select resources from supercomputers and grids.

Finally, experiments show that performing resource re-selection allows tasks to avoid experiencing large queue waiting times on XSEDE supercomputers by executing on OSG instead. Despite using inaccurate task execution time predictions and no queue waiting time predictions, experiments show that resource re-selection reduces task queue waiting times by up to 99% and workload time-to-completion by up to 73% when the queue waiting time experienced on XSEDE supercomputers are high. However, resource re-selection shows little to no reductions in task queue waiting times and workload time-to-completions when the queue waiting times experienced on XSEDE supercomputers are small.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Scientists from various fields perform computational simulations in order to gain insights into physical systems that are difficult, if not infeasible, to gain with analytical models or experiments alone. For example, molecular dynamics (MD) simulations are used in biochemistry to study how drugs bind to ligands, contributing to advances in the design of new therapeutics [1–3]. According to Ref. [4], at least 4 billion core-hours were consumed on XSEDE machines using community MD software packages, 1 billion-core hours with the package GROMACS alone. This number is expected to grow as scientists run larger simulations to gain more insights into the atomic interactions of complex systems [5].

Traditionally, MD simulations are executed on a specific supercomputer, and are expressed as a collection of tasks, i.e, a workload. Given their strong computational capabilities, supercomputers can more quickly perform the large amounts of computations required by MD simulations. However, organizations like XSEDE provide scientists access to multiple supercomputers as well as the Open Science Grid (OSG), a large pool of transient, heterogeneous resources to run many short-lived, mostly single-threaded and single-core applications. Grids like OSG can offer users up to tens of thousands of CPU cores, and excel at executing large numbers of independent tasks [6–8]. Ref. [9] has shown that distributing the execution of workloads of simulations of identical physical systems (i.e., ensembles), across multiple supercomputers can reduce workload time-to-completion. Access to both supercomputers and grids offers the opportunity to improve resource utilization and the concurrent execution of MD ensembles.

However, a consequence of using different heterogeneous, distributed resources is the **resource selection** problem. This problem can be formulated as: "The selection of a subset of resources available to a user to execute a workload". The resource selection problem requires

answering two main questions: one question is of **resource viability**, which asks "Which resources **can** be used to execute a given workload?"; the other question is of **execution affinity**, which asks "Which viable resources **should** be used to execute a given workload?". While many measures can be used to determine how to select resources, task execution time is one of particular interest to scientists. Scientists often select resources based on task execution times in order to minimize a workload's execution time, which is an important time-component of workload time-to-completion.

The resource viability problem has been addressed by [10], which provided a general method that uses task requirements and resource capabilities to determine whether a resource can execute the task. Addressing the execution affinity problem requires the ability to assess how a resource performs when executing an application. For assessing task execution times on different resources, several techniques have been used to inform scientists on how to effectively select resources. Benchmarking is used to rank different resources using performance metrics indicative of how resources perform when executing different tasks. The performance of a resource is ascertained by executing benchmarks on the resource and measuring its performance with respect to the given performance metrics. Performance modeling and statistical modeling are used to predict execution times. Performance models use benchmarking results, as well as hardware architecture and application profiles, to predict execution times. Statistical models use datasets of similar tasks that were previously executed to predict execution times.

Literature on the aforementioned techniques implicitly assumed that the information required to select resources, like benchmarking results, hardware architecture and historical information, can be easily collected. However, this is not the case for grid resources. Due to the heterogeneity and transience of grid resources, both the hardware architecture of the grid and the capabilities of its constituent resources can vary over time. On some grids, like OSG, users have limited knowledge of the capabilities of OSG resources and little control over their software environment. As such, it is difficult to collect application profiles. Thus, it can be difficult to collect the information required by the aforementioned techniques to accurately assess how grid resources execute different tasks. This negatively impacts the performance of many workload scheduling algorithms as they implicitly perform resource selection and assume accurate knowledge of task execution times.

When selecting resources from supercomputers, queue waiting times, the amounts of time spent acquiring resources, should also be considered. The queue waiting time experienced when executing a workload can be a large, if not the dominant, time-component of workload time-to-completion. Despite many attempts to predict queue waiting times using historical datasets, queue waiting times on production-grade supercomputers remain difficult to predict accurately.

This is partly because sudden and sharps increases in resources requested by individual users, called flurries, can greatly increase the queue waiting times experienced by all users of the supercomputer. Also, the presence of flurries in historical datasets can skew queue waiting time predictions as they are not representative of how resources are normally requested by users of a supercomputer.

## 1.2 Objective

The objective of this thesis is to provide insight into how supercomputing and grid resources can be effectively selected to execute workloads in the presence of limited information. This selection depends on knowing task execution times and resource queue waiting times, both of which can seldom be accurately predicted on both types of infrastructure.

The first contribution of this thesis is a **formalism** that allows us to model task execution costs based on the information available from supercomputers and grids. This formalism provides the flexibility to construct a model based on the limited information available from grids. We show that our formalism can be integrated with the Condor Matchmaking algorithm to address the resource selection problem. Any model created based on our formalism can be used to perform resource selection.

The second contribution is the **Limited Information Model** (LIM). On the base of the formalism previously described, we constructed LIM using information that can be currently collected from both the XSEDE supercomputers (Bridges, Comet, SuperMIC) and OSG. We used this model to predict the execution times of single-threaded, single-process, compute-intensive applications. These predictions are then used to operate resource selection across XSEDE supercomputers and OSG. Evaluating the accuracy of our predictions and the effectiveness of the resource selections, we gained insight into what additional information, if any, would be needed to operate better predictions and resource selection. Further, we also evaluated the effectiveness of resource selection across XSEDE supercomputers and OSG when using only task execution time predictions, but not queue waiting times.

The third contribution of this thesis is the process called **resource re-selection**, by which tasks are re-assigned to different resources that are acquired at runtime. The resource re-selection process uses information of task execution times and resource acquisition times and does not use information of queue waiting times. We show that the resource re-selection process can still be used to correctly re-assign tasks to resources using inaccurate task execution time predictions. We show experimentally that resource re-selection effectively reduces time-to-completions of workload executed across resources from XSEDE supercomputers and OSG.

## 1.3 Overview

The motivation and objective for this thesis is given in Ch. 1. In Ch. 2, we provide a brief overview on the state-of-the-art of the resource selection problem. In Ch. 3, we provide a formalism which allows us to construct a model to predict the execution times of single-threaded, single-process tasks running on a resource based on the information that can be collected from XSEDE supercomputers and OSG. We also show that models constructed using this formalism addresses the resource selection problem. In Ch. 3, we develop the resource re-selection process and show how tasks can be correctly re-assigned to execute on resources even with inaccurate knowledge of task execution times.

In Ch. 4, using a GROMACS simulation of a protein in water, we evaluate the accuracy of the predictions generated by LIM using information available from XSEDE supercomputers and OSG. Then, we compare how selecting resources by using the predicted task execution times, either before or during the execution of a bag-of-tasks workload of identical GROMACS simulations, can reduce the time-to-completion of the workload. We study how the flexibility to re-assign tasks originally assigned to execute on an XSEDE supercomputer to execute on OSG can improve workload execution. While there are other important use-cases, we focus on the use-case of executing GROMACS on XSEDE supercomputers and OSG as it applies to our lab's collaborators and to many users of XSEDE.

In Ch. 5, we discuss how selecting resources at runtime can be used to effectively select resources when limited information is available. We also discuss how the ability to select resources at runtime can be used to further improve the resource selection algorithm given in Ch. 3, as well as workload scheduling algorithms. In Ch. 6, we conclude by reviewing the the contributions of this thesis' research and discuss future work.

# Chapter 2

# Background

The problem of selecting resources in the presence of limited information is only one aspect of the overall resource selection problem. In this chapter, we first provide a brief overview on how the resource viability and execution affinity problems have been already addressed. At the same time, we discuss how the effort required to collect information on grid resources makes it difficult to effectively select resources. Then, we discuss how selecting resources in the presence of limited information affects workload scheduling. Finally, we provide a brief overview on the approaches to selecting resources from supercomputers based on their predicted queue waiting times.

## 2.1 Resource Viability

The resource viability problem focuses on identifying the resources,if any, that can be used to execute a workload. This problem has been actively studied in the Grid Computing community. Due to the heterogeneity and transience of grid resources, much of the research focused on: (i) standardizing how jobs (or tasks) and resources are described [11–16]; (ii) providing ways to discover grid resources [17,18]; and (iii) providing a general algorithm, called matchmaking [19], to match jobs with resources whose capabilities satisfy jobs' requirements. Often, the process of matchmaking is carried out by resource brokers [20–23]. Due to the generality of the work in [19], the solutions developed offer insight into how jobs and resources should be defined to enable resource selection over a diverse set of computing tasks and resources.

## 2.2 Execution Affinity

The execution affinity problem focuses on selecting resources to execute a workload that optimizes a given selection metric. Addressing the execution affinity problem requires the ability to assess how effectively a resource executes a task. Given a task and a set of resources, the resource

which most effectively executes a task is selected. Resources are often selected based on their ability to minimize task's execution time, but can also be selected using other metrics [24, 25].

Consistently, literature on addressing the execution affinity problem primarily focuses on accurately assessing the task's execution time on a resource. It is assumed that input information used to assess task execution times can be easily collected. This is true for many supercomputers, where their hardware capabilities and architecture is well-known and relatively stable over time, i.e., static. On supercomputers, users can control the software environment to some extent. However, collecting the input information used to assess execution affinity on grids is often difficult or infeasible. Focusing on the task execution time as the selection metric, we provide a brief overview of three methods commonly used to identify resources that minimize a task's execution time: (1) performance modeling; (2) benchmarking; and (3) execution time predictions. For each technique, we also discuss the difficulties in collecting the requisite information from grid resources.

### 2.2.1 Performance Modeling

Performance models have been used to predict the execution times of workloads running on resources based on salient characteristics of the workloads and resources. Performance models can be used to gain insight on how different workloads execute on existing or future resources. Thus, they are used not only for execution time predictions, but also for application and resource tuning, the procurement of additional resources, and the design of future systems [47].

There are two main types of performance models: *application-specific* and *trace-driven*. Application-specific performance models focus on (1) expressing the execution times on an application running on a resource in terms of the operations performed by different code sections; (2) parameterizing the execution times of each code section in terms of the application's problem size and the target machine's capabilities. To predict an application's execution time, application-specific models [48–51] use detailed knowledge of the capabilities of the underlying hardware; this knowledge can be collected either from documentation or by running benchmarks. These models have been shown to accurately predict the execution times of different application running on different supercomputers.

Trace-based performance models are an alternative to application-specific performance models. Creating application-specific performance models is labor-intensive and requires extensive guidance from domain scientists. Instead of analyzing code sections by hand, application requirements are expressed in terms of the operations (e.g., floating-point calculations, memory accesses, network communication) performed during execution. Usually, this information is captured by using application profilers [52–56] while resource capabilities are measured using

benchmarks. The performance model proposed in Ref. [57] accurately predicts an application's execution time based on the memory access patterns and MPI function calls made during its execution. There have also been initial work Ref. [58] to extrapolate application traces to predict an application's execution time when using core counts or resources different than those used when profiling the application's execution.

Unlike for supercomputers, it is difficult to construct performance models for grids. This is because grids' constituent resources are heterogeneous and transient, e.g., network topology, number of cores, or amount of memory change over time. Thus, it is difficult to select resources using techniques that depend on making assumptions about the resource capabilities. Moreover, users may have little control over the software environment of grid resources. This makes it difficult to use application profilers to characterize applications executions.

### 2.2.2 Benchmarking

Benchmarking is the act of running a suite of programs, called benchmarks, to assess the performance of a resource when executing a well-defined set of operations. To get results that better correlate with the execution costs of real workloads, benchmarks are often reduced versions of operations commonly performed by real workloads. Given a set of resources, a set of performance metrics and a benchmark suite, resources are ranked by the performance values measured when executing the set of operations of the benchmarking suite. A resource ranking provides a simple way to select resources based on their relative performance. Though first suggested by Ref. [41], Ref. [42] showed that expressing application requirements using representative benchmarks via resource brokers improves resource selection on grids.

In reality, ranking resources via benchmarking offers only an approximation of the relative performance of a set of resources. Often, this approximation may not be sufficiently precise. According to Ref. [43–45], an inversion between two resources occurs when the ordering of two resources determined by their benchmarking results is the opposite of the ordering determined by the execution cost of a real application. The 'correctness' of a ranking constructed using a benchmark can be expressed by the number of inversions which occur. Work by Ref. [46] showed that using simple benchmarks and assessing the performance of a resource using only one metric can produce poor rankings of resources. To improve upon this, Ref. [43–45] showed that better rankings can be constructed when using multiple performance metrics, and when using benchmarks that use the machine hardware in a manner similar to the operations performed by the workload of interest.

Despite these difficulties, there have been efforts in using benchmarks to characterize the capabilities of grid resources to improve resource selection [59, 60]. Unfortunately, the act of

benchmarking grids remains difficult [61]. To maintain an up-to-date characterization of the performance of grid resources, benchmarks must be executed regularly [62]. There have been efforts [63, 64] to benchmark grids using fewer benchmark executions, and to develop frameworks [65–67] to simplify the process of benchmarking. Despite these efforts, the process of benchmarking grid resources still requires extensive manual input [68] and the cooperation of the grid's stakeholders [66].

### 2.2.3 Execution Time Predictions

Various statistical and machine learning models have been constructed to predict the execution time of tasks for different problem sizes. Methods like k-Nearest Neighbors [26, 27], neural networks [28–30], and maximum likelihood estimation and random forests [31] have been proposed. Other methods of predicting the executions time of various tasks running on different resources first classify tasks based on predefined attributes, then use various methods to predict the execution times of similar tasks that have executed in the past [32–37]. Ref. [38] used linear regression, while Ref. [39, 40] used regression trees to classify tasks, then predict the task execution times based on similar tasks that been executed in the past.

However, it can be difficult to collect the historical information needed to use the statistical and machine learning models aforementioned. As seen with benchmarking, the heterogeneous and transient nature of grids makes it difficult to develop a system that can collects reliable and consistent historical information from across the grid resource pool in an automated fashion. The limited ability to collect historical information limits the usefulness of statistical and machine learning models to predict task execution times on grid resources.

Also, there is limited availability to existing datasets that can be used to evaluate the execution time of a task on different resources. While archives like the Grid Workloads Archive [69] and Parallel Workloads Archive [70] provide datasets of jobs that executed on various supercomputers and grids, they cannot be used to study the selection of resources from different supercomputers and/or grids for a particular application. This is because the names of the applications executed on the machines are anonymized [69, 71].

## 2.3 Resource Selection with Limited Information

In the presence of limited information, it can be difficult to accurately assess a task's execution time on a resource. It is well-known that the inability to accurately predict task execution times negatively impacts the performance of many workload scheduling algorithms [72–84]. Many workload scheduling algorithms implicitly perform resource selection when assigning tasks to

execute on specific resources. Inaccurate knowledge of task execution times causes workload scheduling algorithms to incorrectly postpone the execution of tasks [85]; incorrectly postponing the execution of a workload's tasks increases workload time-to-completions [85, 86].

There have been efforts to perform workload scheduling by using inaccurate knowledge of task execution time. One approach is to execute a workload according to an initial schedule, monitor the workload's execution and re-schedule tasks at runtime based on the task execution time measured at runtime [81, 87–91]. Ref. [81] showed that the ability to re-schedule tasks allows workloads to meet their deadlines, even when using execution time predictions with a 20% variance. Work in Ref. [92] assumed no knowledge of task execution time when constructing a workload scheduling algorithm. Ref. [92] showed that when no task execution time prediction was used, greedily scheduling tasks to idle processors yields lower workload execution times and higher maximal resource utilization than scheduling tasks based on their execution ordering.

Still, there has been little work on understanding how inaccurate task execution time knowledge impacts workload scheduling across heterogeneous resources. While the work of Ref. [85] provides a systematic study on how different workload scheduling algorithms perform, it primarily focuses on homogeneous systems. Thus, additional studies are required to understand the impact of inaccurate task execution times when scheduling workloads across heterogeneous resources.

## 2.4 Resource Selection on Supercomputers

While knowledge of task execution times is important, queue waiting times should also be considered when selecting resources from supercomputers. Most supercomputers use resource management systems [**?**, **?**, **?**, **?**] to enable users to submit jobs to request resources, and to allocate resources to users. The acquisition of resources requires waiting on the queue of the resource management system. When supercomputers are heavily utilized, users can experience queue waiting times longer than the execution times of their tasks [**?**]. Refs. [**?**, 98] showed that using queue waiting time predictions to select resources can greatly reduce workload time-to-completion.

One method of selecting resources supercomputers is to select resources from the supercomputer with the lowest predicted queue waiting time. Ref. [95] proposed to first model the execution times of jobs on supercomputers using the uniform-log distribution, then predict the queue waiting times of new jobs based on the probabilities that running jobs will end. Work by Ref. [94] provided upper confidence bounds on queue waiting times by first clustering jobs based on their walltimes, then modeling queue waiting times based on the binomial distribution. Instance-based learning methods [**?**, 96, 97, 99, 100] have also been used to predict queue

waiting times, and has been used by Karnak. [93] to predict queue waiting times on XSEDE supercomputers.

However, accurately predicting queue waiting times on supercomputers remains difficult. Refs. [**?**, **?**] demonstrated that flurries, submissions of similar jobs by a single user within a small period of time that drastically alters the utilization of resources, are commonly found on supercomputers and can greatly increase queue waiting times. Flurries introduce noise in the historical datasets used to create queue waiting time prediction models as they do not reflect the normal usage of resources. Moreover, since each flurry is unique, it is difficult to use historical datasets of past flurries to predict future flurries. While some prediction methods [94, 99] indirectly address this problem by using more recently collected data points to make predictions, it remains difficult to develop models predicting queue waiting times that account for the impact of flurries.

Rather than using queue waiting time predictions to select supercomputers to use, users can request resources from multiple supercomputers and assign tasks at runtime to execute on resources that are acquired first. Doing so allows tasks to avoid experiencing large queue waiting times on heavily utilized supercomputers, thereby reducing workload time-to-completion [9]. Ref. [**?**] developed abstractions representing application requirements and resource availabilities and capabilities. Furthermore, Ref. [**?**] showed that middleware implementing these abstractions can acquire information from application and resources to distribute the execution of workloads across multiple resources at scale. Refs. [**?**, 9] also showed that distributing a workload across resources from three supercomputers based on their acquisition times results in smaller and more stable workload time-to-completion.

# Chapter 3

# Resource Selection

To address the problem of selecting resources with limited information, we define a formalism to evaluate the cost of executing a single-threaded, single-process task on a resource without specifying beforehand the system-resources used to characterize tasks and resources. This provides the flexibility to construct a model to predict task execution costs based on the information that can be collected. This flexibility is important as grids may provided little to no information about the capabilities of their resources. We show that this formalism can be incorporated into the Condor Matchmaking algorithm, thereby providing a solution to the resource selection problem.

Using the defined formalism, we constructed the Limited Information Model (LIM) to predict the execution times of single-threaded, single processes, compute-intensive tasks running on resources. We use information that can be collected from XSEDE supercomputers (i.e., Bridges, Comet, SuperMIC) and from XSEDE OSG, separately. We use predictions generated with LIM to select resources that minimize task execution time. Based on the accuracy of the predictions and the resources selected using these predictions, we evaluate the effectiveness of selecting resources to gain insight into what information, if any, would be needed to operate better predictions and resource selections.

LIM does not take into account code structure or the resource's hardware architecture, and assumes that a task executes the same sequence of instructions identically on any resource. In this way, we trade off between the prediction accuracy and the effort required to collect data for execution cost prediction. Lack of accuracy is acceptable if LIM predictions support effective resource selection for the distributed execution of a given workload over a given set of resources.

Building upon the resource selection process, we developed a process called 'resource re-selection' to re-assign tasks to resources based on task execution times and resource acquisitions times. Resource re-selection re-assigns tasks to execute on resources different from those on

which they were originally assigned. We explore how using task execution time predictions can increase the task's time-to-completion. We show that even inaccurate execution time predictions can be used to re-select resources without increasing the task's time-to-completion.

## 3.1 Task Execution Cost Formalism

The formalism centers around the definition of the consumable, an entity that represents one functionality used by a task during its execution. From this, we define an instruction, which uses different amounts of different consumables in a well-defined manner. Then, we define a task as a sequence of instructions. Similarly, we define a resource to be an entity which offers the use of different consumables at different rates. In this way, we define the cost of executing a task on a resource as the cost of using the consumables specified by the task's constituent instructions at the rates specified by the resource. Cost evaluation can be based on different metrics such as execution time, allocation usage, currency or energy. To simplify the construction of the formalism, we assume: (1) the consumables required by a task is independent from the resource that offers them; (2) the cost of using any consumable offered by a resource is fixed. Here, we define the following terms from first principles:

**Consumable:** An entity representing a unit of work. A consumable has two properties: (1) type, which determines the kind of work the entity can perform; and (2) form, which specifies the conditions that must be satisfied for the consumable to be used to perform work.

**Requirement:** Amount of a consumable, where the amount is assumed to be fixed.

**Instruction:** Set of tuples, each specifying a certain requirement.

**Task:** Sequence of instructions, executed in the order specified by the sequence.

**Workload:** Set of tasks, where all tasks can run concurrently.

**Capability:** Rate at which a consumable is offered, assumed to be fixed.

**Resource:** Set of tuples, each specifying a certain capability.

Formally, a consumable $c$ is a set $\{type, form\}$, where $type$ is a single value while $form$ is a set of pairs. For any pair in $form$, the first element is the attribute $attr$ that uniquely identifies the pair in $form$; the second element of the pair is the condition $cond$, expressed as a set of values that specifies how $c$ can be used.

A requirement $req$ is a tuple $(c, amt)$ where $c$ is a consumable and $amt > 0$ specifies some fixed amount of the consumable $c$. An instruction $ins$ is a set $\{req_1, \ldots, req_n\}$, where each

requirement $req_i$ specifies the amount of consumable required by $ins$. A task $task$ is a sequence of instructions $[ins_1, \ldots, ins_n]$. By the definition of an instruction, we can express a task as a sequence of sets of requirements. For brevity, we use the term 'task requirements' to be set of requirements specified by the instructions that compose a task. We define a workload $W$ as a set of tasks $\{task_1, \ldots, task_n\}$.

We define a capability $cap$ as a set $\{c, rate\}$, where $c$ is a consumable and $rate > 0$ is fixed. $rate$ represents the number of consumables offered per unit of cost (e.g., in terms of time, money, or energy). In this way, we establish a relationship between the use of a consumable and the cost of using a consumable. We define a resource $res$ as a set $\{cap_1, \ldots, cap_n\}$, where each capability $cap_j$ offers the use of a unique consumable at a fixed rate.

Assuming that a given task can run on a given resource, we define the cost $K$ of running a task on a resource as the total cost required to sequentially consume the amounts of consumables specified by the requirements of every instruction of a task at the rate offered by the corresponding resource capabilities. The cost of running the task on a resource is expressed sequentially because our formalism does not consider the order (or concurrency) in which tasks can use consumables.

Let there be a task $task = [ins_1, \ldots, ins_m]$, where each $ins_i = \{req_1, \ldots, req_{p_i}\}$ is set of $p_i$ requirements, and a resource $res = \{cap_1, \ldots, cap_n\}$. Then, $K$ is defined as:

$$K = \sum_{i=1}^{m} \sum_{k=1}^{p_i} \sum_{j=1}^{n} \frac{ins_i.req_k.amt}{cap_j.rate} \mathbb{1}_{\{ins_i.req_k.c = cap_j.c\}} \tag{3.1}$$

where $\mathbb{1}$ is the indicator function, which equals 1 if the consumable $c$ specified by the $k$-th requirement of the $i$-th instruction of $task$ is the same consumable specified by the $j$-th capability of the resources $res$, and 0 otherwise.

## 3.2 Limited Information Model

Using the above formalism, we construct the Limited Information Model (LIM) to predict the execution times of single-threaded, single-process compute-intensive tasks running on a resource based on the information available from XSEDE supercomputers (Bridges, Comet, SuperMIC) and from XSEDE OSG. Though we can collect more information on XSEDE supercomputers, we were able to query the processors of OSG resources and perform profiling using Linux `perf` only on some OSG resources. Using `perf`, we were able to measure the task's execution time, number of cycles used, number of instructions executed, the average number of instructions executed per cycle (i.e., instruction rate) and the average clock speed experienced during execution. We use our formalism to construct LIM to predict the execution time (denoted $T_x$) using the number of

cycles required by a task and the processor clock speed of a resource.

### 3.2.1 Constructing the model

We define a cycle as a compute-type consumable that can only be used on processors supporting x86 ISA. Formally, $cycle = \{\text{“}compute'', \{\text{“}ISA'' : \{\text{“}x86''\}\}\}$. Next, we define a compute requirement as $req = \{num\_cy, cycle\}$, where $num\_cy > 0$. A compute instruction $instr = \{req\}$ specifies its compute requirement. Since tasks are assumed to be compute-intensive, we consider only the computational requirements of a task. Thus, we define a task as a sequence of compute instructions $task = [instr_1, \ldots, instr_m]$.

Similarly, we consider the compute capabilities of a resource, namely its clock speed, as the number of cycles it offers per unit of time. We define a compute capability $cap = \{rate\_cy, cycle\}$, where $rate\_cy$ is a positive number given in units $\frac{cycles}{second}$. Thus, we define a resource as $res = \{cap\}$. Since $rate\_cy$ is given in terms of $\frac{cycles}{second}$, we define the execution time $T_x$ of a task running on a resource using Eq. 3.1. Based on our formalism, $T_x$ is the amount of time required for the task to use the number of cycles required by a task's instructions at the rate at which the resource offers cycles. Let $total\_cy$ be the sum of the number of cycles specified by the compute requirement of each instruction of a task $task$. Then, the execution time $T_x$ of $task$ running on a resource $res$ is:

$$T_x = \frac{total\_cy}{rate\_cy}, \tag{3.2}$$

### 3.2.2 Equations

Since $T_x$, defined using Eq. 3.2, does not take into account any instruction-level parallelism (ILP), we need a way to derive the number of cycles required to execute the task without ILP. We show that this number of cycles can be calculated from the actual number of cycles used during execution and the instruction rate.

We define $\#instr$ as the number of instructions the task executes. Since the only requirement of the task is that it consumes some amount of cycles, we define:

$$\#instr = \#cycles \times instr\_rate, \tag{3.3}$$

where $num\_cy, instr\_rate$ denote the number of cycles used to execute the instructions and the average number of instructions executing per cycle, respectively. When only one instruction uses a cycle at any point in time, $instr\_rate = 1$.

We define $pred\_num\_cycles, act\_num\_cycles$ as the predicted and actual number of cycles

used, respectively. Similarly, we define $instr\_rate\_pred, instr\_rate\_act$ as the predicted and actual number of instructions executed during the period of a cycle. Since we are comparing the execution of the same task, $\#instr$ is fixed. From Eq. 3.3:

$$\frac{instr\_rate\_act}{instr\_rate\_pred} = \frac{pred\_num\_cycles}{act\_num\_cycles}, \tag{3.4}$$

We define $p2a\_cy = \frac{pred\_num\_cycles}{act\_num\_cycles}$ to be the ratio between the predicted and actual number of cycles used. Since LIM assumes that only one instruction uses a cycle, $instr\_rate = 1$:

$$p2a\_cy = instr\_rate\_act \tag{3.5}$$

With Eq. 3.5, we can calculate the number of cycles required to execute a task without ILP by measuring the actual number of cycles used and the instruction rate when the task was executed on a baseline resource. We use Eq. 3.5 to derive the number of cycles necessary to execute a task sequentially on any resource. We define $\varepsilon$ as the percent error between $p2a\_cy$ and $instr\_rate\_act$ to measure how much instruction-level parallelism affects the LIM's prediction error:

$$\varepsilon = \frac{|p2a\_cy - instr\_rate\_act|}{p2a\_cy} \times 100 \tag{3.6}$$

If the prediction error in the number of cycles required is completely due to the instruction-level parallelism, then $\varepsilon = 0$.

In the experiments, we used `perf` to profile the execution of a task on a 'baseline' resource to measure the information necessary to predict the number of cycles required to without ILP. We assume that the predicted number of cycles represents the number of cycles required to execute the task on any machine in the absence of ILP. We use the predicted number of cycles and the clock speed of the resource's processor to predict the task's execution time on each resource. Though we sacrifice prediction accuracy by not accounting for the ILP during the task's execution, we simplify the process the collecting information to predict task execution times. In Sec. 3.4 and Sec. 4.3, we show that even inaccurate task execution time predictions can still be used to effectively select resources.

## 3.3 Resource Selection Process

To develop a formal solution to the resource selection problem, we provide solutions to the two subproblems of the resource selection problem (see Ch. 2). To address the resource selection problem, we adapt the Condor matchmaking algorithm to operate in terms of consumables. We

call the resulting algorithm the 'adapted matchmaking algorithm' (AMA). We show that it is possible, but not necessary, for AMA to select resources based on requirements and capabilities.

### 3.3.1 Resource Viability

To adapt the Condor matchmaking algorithm to operate in terms of consumables, we define the algorithms $SATISFY\_REQ$ and $SATISFY\_TASK$ to determine whether a resource can execute a task. $SATISFY\_REQ$ (Alg. 1) takes as input a requirement and capability and determines whether the consumable specified by the capability can be used by the requirement. $SATISFY\_REQ$ checks: (1) if the types of consumables of the requirement and capability are the same; (2) if for every form attribute in the requirement there is a corresponding form attribute in the capability; and (3) if for each form attribute in the requirement there is a value in the condition of the form attribute of both the capability and the requirement. Note that the comparison operator used to decide whether two values are equal depends on the data type of the values (e.g., integer, float, string), as discussed in several specifications [11, 12].

**Algorithm 1** Check if capability (*cap*) consumable can be used to satisfy requirement (*req*)

**Require:** *req*; *cap*
**Ensure:** **True** or **False**

```
1: procedure SATISFY_REQ(req, cap)
2:     if req.c.type != cap.c.type then
3:         return False
4:     for all (attr, cond) in req.c.form do
5:         if attr not in cap.c.form then
6:             return False
7:         if cap.c.form.cond ∩ cond = ∅ then
8:             return False
9:     return True
```

$SATISFY\_TASK$ (Alg. 2) takes as input a task and resource and checks whether for each requirement of each instruction of the task there is a capability of the resource that can satisfy the given requirement. $SATISFY\_TASK$ uses $SATISFY\_REQ$ to determine whether a capability can be used to satisfy a requirement. If $SATISFY\_TASK$ returns $True$ for a given task and resource, then the task can execute on that resource.

$SATISFY\_TASK$ can then be used determine for each task of a workload whether there is a subset of resources that can execute that task. We call this subset of resources the "viable resources set" of that task. If every task in the workload has a nonempty viable resources set, then the workload can be executed across a subset of the available resources.

**Algorithm 2** Check whether a task ($task$) can execute on a resource ($res$)

**Require:** $res = (cap_1, \ldots, cap_m)$; $task = (ins_1, \ldots, ins_n)$, where $ins_i = (req_1, \ldots, req_{i_k})$
**Ensure:** **True** or **False**

```
1: procedure SATISFY_TASK(res, task)
2:     for all ins in task do:
3:         for all req in ins do:
4:             match ← 0
5:             for all cap in res do
6:                 if SATISFY_REQ(req, cap) = True then
7:                     match ← 1
8:             if match = 0 then
9:                 return False
10:    return True
```

### 3.3.2 Execution Affinity

We assume a workload, a set $vrs$ of viable resources $\{res_1, \ldots, res_n\}$ for each task $task_n$ of that workload, and a function $affinity$ which maps a set of input tuples to a totally order set (e.g., $\mathbb{R}$): The higher the value, the better $res_n$ is for executing $task_m$. Note that for every pair of $(res_n, task_m)$, there is only one input tuple that is used to determine the affinity value of that $(res_n, task_m)$ pair.

Generally, the input of $affinity$ does not necessarily include task requirements or resource capabilities. However, if we want to select resources using only information about task requirements and resource capabilities, we can use Eq. 3.1 to select resources and provide the task requirements and resource capabilities as input to $affinity$.

*RES_SELECT* (Alg. 3) identifies the available resources that gives that highest affinity value. We define the set $vrs\_id = \{res\_id_1, \ldots, res\_id_n\}$ to be the set of unique IDs of every resource in a task's viable resources set. We also define the task input set $TIS = \{input_1, \ldots, input_n\}$, where $input_i$ is the input tuple to $affinity$ associated with $res\_id_i$. *RES_SELECT* takes as input $vrs\_id$, $TIS$ and $affinity$, and returns $res \in vrs\_id$, whose associated input tuple gives the highest affinity value.

**Algorithm 3** Determines the resources on which a task input set ($TIS$) should execute, given the viable set ($vrs\_id$) of each task of $TIS$

**Require:** $vrs\_id = \{res\_id_1, \ldots, res\_id_n\}$; $TIS = \{input_1, \ldots, input_n\}$; *affinity*()
**Ensure:** Resource ID *res* or **NONE**

```
1: procedure RES_SELECT(res_id, TIS, affinity)
2:     best_res ← NONE
3:     best_select_val ← −∞
4:     for i from 1 to n do
5:         select_val = affinity(input_i)
6:         if best_select_val < select_val then
7:             best_select_val ← select_val
8:             best_res ← res_id_i
9:     return best_res
```

$RES\_SELECT$ can be used in conjunction with $SATISFY\_TASK$ to provide a formal solution to the resource selection problem. Given a set of tasks and a set of resources, $SATISFY\_TASK$ can be used to identify the viable resource set of each task. If any viable resource set of any task is empty, then the given resources cannot be used to execute the workload. Otherwise, $RES\_SELECT$ can be used to select the resource that yields the highest affinity value. The set of resources used to execute the given tasks is the set of resources that yielded the highest affinity value for at least one task.

It is important to note that the algorithms described are concerned only with selecting resources. However, $RES\_SELECT$ can be modified to return a list containing the affinity value of each resource. This information can be incorporated into workload scheduling algorithms [72–84, 88–90] to decide which resources should be used to execute the tasks and the order in which they execute. However, a discussion in this direction is beyond the scope of the resource selection problem.

## 3.4 Resource Re-selection

Though important, $T_x$ is only one time-component of a task's overall time-to-completion; a task's time-to-completion, denoted $TTC$, is the total amount of time spent on the execution of a task. Aside from $T_x$, $TTC$ includes other time-components like $T_q$, the time spent acquiring the resources necessary to execute the task. While the $T_q$ experienced when acquiring grid resources is usually low, the $T_q$ experienced when acquiring supercomputing resources varies and can be large. This is because many users are concurrently acquiring and using resources from supercomputers to execute their respective applications. While resources may instead be selected based on both $T_x$ and $T_q$ to minimize the task's $TTC$, it remains difficult to predict accurately $T_q$ on production supercomputers [93–100].

Building upon the resource selection process, we developed a process called 'resource re-selection' to re-assign tasks to execute on resources based on their capabilities and their acquisitions at runtime. To focus on selecting resources based on how their capabilities and acquisitions at runtime affect the task's $TTC$, we define $TTC = T_q + T_x$. The intuition behind performing resource re-selection is to re-assign a task that can execute on a different, acquired resource while waiting for its originally selected resource to be acquired. When the originally selected resource is acquired, the task can decide either to continue executing on its newly assigned resource or restart its execution on its originally assigned resource. This decision depends upon the task's execution times and resource acquisition times of the originally and newly selected resources. It is important to note that re-assigning a task to execute on a different resource implies

performing resource re-selection.

Rather than selecting resources using inaccurate $T_q$ predictions on different supercomputers, we re-assign tasks using knowledge of a task's $T_x$ on different resources and the acquisition times of those resources. After providing the assumptions under which we explore resource re-selection, we discuss how resources are re-selected using actual task execution times. Then, we discuss the consequences of selecting resources using predicted task execution times instead. While re-selecting resources using predicted task execution times can negatively impact a task's $TTC$, resources can still be effectively re-selected using such predictions.

### 3.4.1 Assumptions

We assume the following when performing resource re-selection:

1. The workload, the resources used to execute the workload, and the execution times of tasks running on any of those resources are known and fixed.
2. Resource used to execute the workload are initially unacquired, and can be acquired at different points in time.
3. Tasks can only execute on acquired resources.
4. Each task is initially assigned to a viable resource, and can be re-assigned to execute on other viable resources that have been acquired.
5. A task must restart its execution when re-assigned to execute on a different resource.
6. Re-assigned tasks begin executing on a resource immediately after re-assignment.

Assumptions (2)–(4) captures how users typically acquire and use resources from supercomputers or grids. Users submit job requests to the machine's local resource management system (LRMS), specifying the capabilities of the resources they want to acquire. The requested resources cannot be used to execute a task until the machine's LRMS provides the user with the resources specified by the previously submitted job request. Once the user acquires the desired resources from the machine's LRMS, the resources can be used to execute the user's tasks.

### 3.4.2 Re-selection with Actual Execution Times

Here, we describe the process of performing resource re-selection with knowledge of the actual task execution times. Let:

- $s$ be a task.

- $R$ be a set with at least two resources that can execute $s$ when acquired.
- $t$ denote time.
- $t = 0$ be the time at which all resources from $R$ are unacquired.
- $r \in R$ be the resource on which $s$ has been initially assigned to execute.
- $t = a_r$ be the time at which $r$ is acquired and can be used to execute $s$.
- $r' \in R$ be a different resource that is acquired before $r$, i.e., $a_{r'} < a_r$.
- $T_{x,r}$ be the actual execution time of $s$ on $r$.

We perform resource re-selection whenever $s$ assigned to execute on $r$ can be re-assigned to execute on a different but acquired resource $r'$ when $r$ is unacquired.

We define the **re-selection window** $W_{r',r}$ between $r$ and $r'$ to be $W_{r',r} = T_{x,r'} - T_{x,r}$. When a task can be re-assigned to execute on $r'$ instead of $r$, the task $s$ should execute on $r'$ for a duration of at most $W_{r',r}$ before it commits to executing on $r'$. There are two cases to consider: (1) $W_{r',r} \leq 0$; (2) $W_{r',r} > 0$. If $W_{r',r} \leq 0$, then at $t = a_{r'}$, $s$ should be re-assigned to execute on $r'$ as $T_{x,r'} \leq T_{x,r}$ and $TTC_{r'} \leq TTC_r$. If $W_{r',r} > 0$, then $s$ should execute on $r'$ while waiting for $r$ to be acquired. Assuming that task $s$ has not finished executing on $r'$ before $r$ is acquired, $s$ should be re-assigned back to $r$ if $a_r - a_{r'} < W_{r',r}$, and continue executing on $r'$ otherwise.

The re-selection process can be generalized to allow tasks to be re-assigned to multiple, different, concurrently acquired resources. When multiple, different resources are concurrently acquired before its originally assigned resource, a task is re-assigned to the acquired resource that yields the lowest $TTC$. Any further task re-assignment will be between the task's originally assigned and newly re-assigned resources.

### 3.4.3 Re-selection with Predicted Execution Times

Here, we describe the consequences of performing resource re-selection using task execution time predictions rather than with the actual task execution times. We explore the conditions under which the resource re-selected using execution time predictions differ from the resource re-selected using actual task execution times. We provide bounds on the increase in the task's $TTC$ as a result of executing on a resource re-selected using task execution time predictions rather than using actual task execution times.

Given a task $s$ and a resource $r$, we denote the predicted task execution time and actual task execution time of a task $s$ on a resource $r$ as $\hat{T}_{x,r}$ and $T_{x,r}$, respectively. We define $T_{q,r}$ as the amount of time spent acquiring resource $r$. Since each resource $r$ is unacquired at $t = 0$

and is acquired at $t = a_r$, $T_{q,r} = a_r - 0 = a_r$. We define a task's actual time-to-completion as $TTC_r$, where $TTC_r = T_{q,r} + T_{x,r}$. Note that the definition of $TTC_r$ is calculated by measuring when resource $r$ is acquired. Given two resources $r$ and $r'$, we denote the predicted re-selection window of $r$ and $r'$ as $\hat{W}_{r',r} = \hat{T}_{x,r'} - \hat{T}_{x,r}$.

We define the act of performing resource re-selection using actual task execution times as the function $RESEL$, where:

$$RESEL(T_{x,r}, T_{x,r'}, a_r, a_{r'}) = \begin{cases} r, & (W_{r',r} > 0) \wedge (a_r - a_{r'} < W_{r',r}) \\ r', & (W_{r',r} \leq 0) \vee ((W_{r',r} > 0) \wedge (a_r - a_{r'} \geq W_{r',r})) \end{cases}$$

We define the act of performing resource re-selection using predicted task executions times as the function $P\_RESEL$, where:

$$P\_RESEL(\hat{T}_{x,r}, \hat{T}_{x,r'}, a_{q,r}, a_{q,r'}) = \begin{cases} r, & (\hat{W}_{r',r} > 0) \wedge (a_r - a_{r'} < \hat{W}_{r',r}) \\ r', & (\hat{W}_{r',r} \leq 0) \vee ((\hat{W}_{r',r} > 0) \wedge (a_r - a_{r'} \geq \hat{W}_{r',r})) \end{cases}$$

We say that the use of predicted task execution times incorrectly re-selects resources if $RESEL(T_{x,r}, T_{x,r'}, a_r, a_{r'}) \neq P\_RESEL(\hat{T}_{x,r}, \hat{T}_{x,r'}, a_r, a_{r'})$, where $a_r$, $a_{r'}$ are fixed inputs to functions $RESEL$ and $P\_RESEL$. Given fixed values of $T_{x,r}, T_{x,r'}, \hat{T}_{x,r}, \hat{T}_{x,r'}, a_r, a_{r'}$ for task $s$ and resources $r$ and $r'$, we express the penalty of incorrectly re-selecting resources using predicted task execution times as $penalty = |TTC_{P\_RESEL} - TTC_{RESEL}|$. Going forward, we use $RESEL(\ldots)$ and $P\_RESEL(\ldots)$ as shorthand for the functions $RESEL(T_{x,r}, T_{x,r'}, a_r, a_{r'})$ and $P\_RESEL(\hat{T}_{x,r}, \hat{T}_{x,r'}, a_r, a_{r'})$, respectively.

We consider Case 1, where $W_{r',r} \leq 0$. So $RESEL(\ldots) = r'$. If $\hat{W}_{r',r} \leq 0$, then $P\_RESEL(\ldots) = RESEL(\ldots) = r'$. If $\hat{W}_{r',r} > 0$, then $P\_RESEL(\ldots) = r'$ if $a_r - a_{r'} \geq \hat{W}_{r',r}$ and $P\_RESEL(\ldots) = r$ if $a_r - a_{r'} < \hat{W}_{r',r}$. So, when $W_{r',r} \leq 0$, $\hat{W}_{r',r} > 0$ and $a_r - a_{r'} < \hat{W}_{r',r}$,

$$\begin{aligned} penalty &= |TTC_r - TTC_{r'}| \\ &= |(T_{q,r} - T_{q,r'}) + (T_{x,r} - T_{x,r'})| \\ &< ||\hat{W}_{r,r'}| + |W_{r,r'}|| \\ &= |\hat{W}_{r',r}| + |W_{r',r}| \end{aligned} \tag{3.7}$$

We consider Case 2, where $W_{r',r} > 0$. If $\hat{W}_{r',r} \leq 0$, then $P\_RESEL(\ldots) = r'$. If $a_r - a_{r'} \geq W_{r',r}$, then $RESEL(\ldots) = r'$. However, if $r$ is acquired at $t = a_r$ such that $a_r - a_{r'} < W_{r',r}$, then $RESEL(\ldots) = r$. So, when $W_{r',r} > 0$, $\hat{W}_{r',r} \leq 0$ and $a_r - a_{r'} < W_{r',r}$,

$$penalty = |TTC_{r'} - TTC_r| < W_{r',r} \tag{3.8}$$

If $\hat{W}_{r',r} > 0$, it may still be possible to incorrectly re-select resources if $W_{r',r} \neq \hat{W}_{r',r}$. If $W_{r',r} > \hat{W}_{r',r}$, then it is possible that resource $r$ is acquired at time $a_r$, where $a_{r'} + \hat{W}_{r',r} < a_r < a_{r'} + W_{r',r}$. In this case, $P\_RESEL(\ldots) = r'$ and $RESEL(\ldots) = r$. So,

$$\begin{aligned} penalty &= |TTC_{r'} - TTC_r| \\ &= |T_{q,r'} + T_{x,r'} - (T_{q,r} + T_{x,r})| \\ &= |(T_{x,r'} - T_{x,r}) - (T_{q,r} - T_{q,r'})| \\ &\leq |W_{r',r} - \hat{W}_{r',r}| \end{aligned} \tag{3.9}$$

If instead $W_{r',r'} < \hat{W}_{r',r}$, then it is possible that resource $r$ is acquired at time $a_r$, where $a_{r'} + W_{r',r} < a_r < a_{r'} + \hat{W}_{r',r}$. In this case, $P\_RESEL(\ldots) = r$ and $RESEL(\ldots) = r'$. So,

$$\begin{aligned} penalty &= |TTC_r - TTC_{r'}| \\ &= |T_{q,r} + T_{x,r} - (T_{q,r'} + T_{x,r'})| \\ &= |(T_{q,r} - T_{q,r'}) - (T_{x,r'} - T_{x,r})| \\ &\leq |\hat{W}_{r',r} - W_{r',r}| \end{aligned} \tag{3.10}$$

Based on the penalties incurred for incorrectly re-selecting resources, we see that it is more important that the predictions accurately predict the re-selection windows of resource pairs than it is to accurately predict the task execution times on each resource. Even if inaccurate, if the predicted task execution times correctly rank resources, i.e. $T_{x,r} \leq T_{x,r'} \Leftrightarrow \hat{T}_{x,r} \leq \hat{T}_{x,r'}, \forall r, r' \in R$, then a penalty for incorrectly re-selecting resources is only incurred when $W_{r',r} \neq \hat{W}_{r',r}$. This is because $T_{x,r} \leq T_{x,r'} \Leftrightarrow \hat{T}_{x,r} \leq \hat{T}_{x,r'}, \forall r, r' \in R$ is equivalent to $W_{r',r} \leq \hat{W}_{r',r}$.

In fact, predicted task execution times will always correctly re-select resources if $\forall r \in R$, $\hat{T}_{x,r} = T_{x,r} + c, c \in \mathbb{R}$ as it implies that $W_{r',r} = \hat{W}_{r',r}$. From this, we see that re-selecting resources with actual task execution times is a special case of $\hat{T}_{x,r} = T_{x,r} + c$, where $c = 0$. Thus, accurate execution time predictions are sufficient but not necessary to guarantee correct resource re-selection.

Still, the penalty incurred for incorrectly re-assigning a task can be nontrivial. In our experiments, we performed resource re-selection to re-assign tasks originally assigned to execute on XSEDE supercomputers to OSG using task execution time predictions that correctly rank the supercomputers and OSG. While some tasks were incorrectly re-assigned, we show experimentally that the increase in $TTC$ experienced by some tasks as a result of incorrect re-assignment is acceptable as resource re-selection reduced the $TTC$ of the workload.

# Chapter 4

# Experiments

We performed three sets of experiments to measure the performance of selecting resources based on the information available from XSEDE supercomputers and XSEDE OSG using LIM, introduced Sec. 3.2. The first set of experiment studies the accuracy of the $T_x$ predictions generated by LIM for single-threaded, single-process, compute intensive tasks on XSEDE supercomputers and OSG. The second set of experiments compares the $TTC$ of workloads executed on resources selected using $T_x$ predictions generated by LIM with the $TTC$ of workloads executed on randomly selected resources. The third set of experiments compares the $TTC$ of workloads executed by performing resource re-selection with the $TTC$ of the workloads executed in the second set of experiments.

**Machines:** The machines used in all experiments are the XSEDE supercomputers Bridges [**?**], Comet [**?**] and SuperMIC [**?**], and the XSEDE OSG VirtualCluster [**?**] resource pool (hereafter just OSG). We submitted jobs to the *RM*, *compute*, and *workq* queues of, respectively, Bridges, Comet and SuperMIC. Though OSG is a heterogeneous collection of machines, we use the term 'target machines' to indicate the XSEDE supercomputers and OSG resource pool.

**Application:** For all experiments, we used a task simulating the dynamics of a protein in water, representative of many tasks routinely executed on a diverse type of machines. We used GROMACS 5.0, compiled with single-precision floating-point and SSE4.1 SIMD instructions as it was the default version supported by OSG. Though the XSEDE supercomputers support newer versions of GROMACS, we compiled GROMACS 5.0 to match the GROMACS configuration on OSG because we had little control over the software environment of OSG. Further, since OSG is primarily designed for loosely-coupled, single-threaded jobs, we executed the simulations using a single thread and process on all target machines.

Since the task used for the experiments is 'compute-intensive' with limited I/O load, we

Table 4.1: Clock Speeds, in GHz

| DCR | *base* | *avg* |
|---|---|---|
| Bridges | 2.30 | 2.732 (0.038) |
| Comet | 2.50 | 2.888 (0.001) |
| SuperMIC | 2.80 | 3.589 (0.002) |
| OSG | 2.50 | 2.930 (0.227) |

focused only on its computational requirements. Using LIM, we expressed the task's requirements in terms of the number of cycles to consume using x86 instructions. Similarly, we focused only on the computational capabilities of the resource used, as determined by the clock speed the resource's processor.

## 4.1 Experiment 1: Execution Time Accuracy

We designed a set of experiments to characterize the accuracy of LIM to predict the execution time $T_x$ of a task on the target machines. We used the Amarel cluster [**?**] as the baseline machine from which to collect the information required by LIM. From Amarel, we only used nodes which had Intel Xeon e5-2680 Broadwell cores and 128GB or 256GB memory. While other machines could have been used, Amarel offered rapid access to its resources. Using the information collected from Amarel and from the target machines, we predicted the $T_x$ of the task running on each target machine. We executed the same simulation on the target machines to measure the prediction error of LIM.

### 4.1.1 Setup

The same MD simulation was executed for 1000, 5000, 10000, 25000, 50000, 75000, and 100000 timesteps on the baseline and target machines. We used `perf` [101] to profile the execution of the simulation on Amarel. We ran each simulation between 35–60 times for each number of timesteps. Using the number of cycles used and the instruction rate measured when executing the simulation on Amarel, we predicted the number of cycles required by the task when executed without ILP with Eq. 3.5. We used this prediction along with the base clock speed of the CPUs on the target machines to predict the execution times of simulations when executing on the target machines.

We used XSEDE documentation and processor specifications to identify the base clock speeds, denoted *base*, of the processors of the target machines. Unlike the XSEDE supercomputers, OSG is a pool of heterogeneous resources. For OSG, *base* represents the weighted average of the base clock frequencies of the processors of OSG resources. To calculate the weighted averages, we collected information on the processors available in the OSG resource pool at the beginning of the experimental campaign. Tab. 4.1 shows the values for *base* for the target machines.

We executed the same simulations on each target machine to measure LIM's prediction error. We profiled the execution of the simulation on the target machines with `perf`. Again, we executed the simulation 35–60 times for each number of timesteps. We compared the number of instructions executed on each target machine with that measured from Amarel. We also compared the number of cycles used with the predicted number of cycles used. We measured the average clock speed, denoted $avg$, experienced when executing the simulations on the target machines. Tab. 4.1 shows the average clock speeds measured and their sample standard deviations (given in parenthesis) for each target machine. We used `perf` to measure the simulation's $T_x$ on the XSEDE supercomputers. On OSG, however, we found that only ∼1.2% of the trial runs on OSG resources were able to use both `perf` and GROMACS. Thus, we used the wall-time measurements in the log files of GROMACS simulations to measure the simulation's $T_x$ on OSG.

### 4.1.2 Results

For any number of timesteps, we found that the number of instructions required to execute a GROMACS simulation on resources from the XSEDE supercomputers is within ∼3% of that executed on Amarel. However, the number of instructions executed on OSG resources is on average 22–24% more than that executed on Amarel. This is likely because the GROMACS software configuration on OSG is different than that on Amarel and XSEDE supercomputers, despite the fact that the version, floating-point precision and SIMD instruction set are the same.

Figs. 4.1 – 4.5 give a summary of our findings. All values are shown in the figures as averages, along with their sample standard deviations as error bars. Fig. 4.1 shows the number of cycles required to execute a simulation on Amarel and on the target machines, as well as the predicted number of cycles required. Fig. 4.2 shows the instruction rate measured when executing the simulations on the target machines. It is important to note that the stability of the instruction rate for the XSEDE supercomputers is due to the homogeneity of their resources.

Fig. 4.3 shows that LIM overpredicts the actual number of cycles needed to execute the simulations on XSEDE supercomputers by ∼110–125%. This is because LIM does not take into account the code structure of the GROMACS simulation or the hardware architecture of resources from the target machines. By calculating $p2a\_cy$ for each simulation, we find that the average $\varepsilon$ for the simulations on the XSEDE supercomputers is less than 3%. This means that LIM overpredicts because LIM does not consider information that describes the ILP which the code exploits in the hardware. However, we see that LIM reasonably predicts the number of cycles used to execute the simulations on OSG. This is likely because LIM underpredicted the number of instructions required to execute the simulation on OSG.

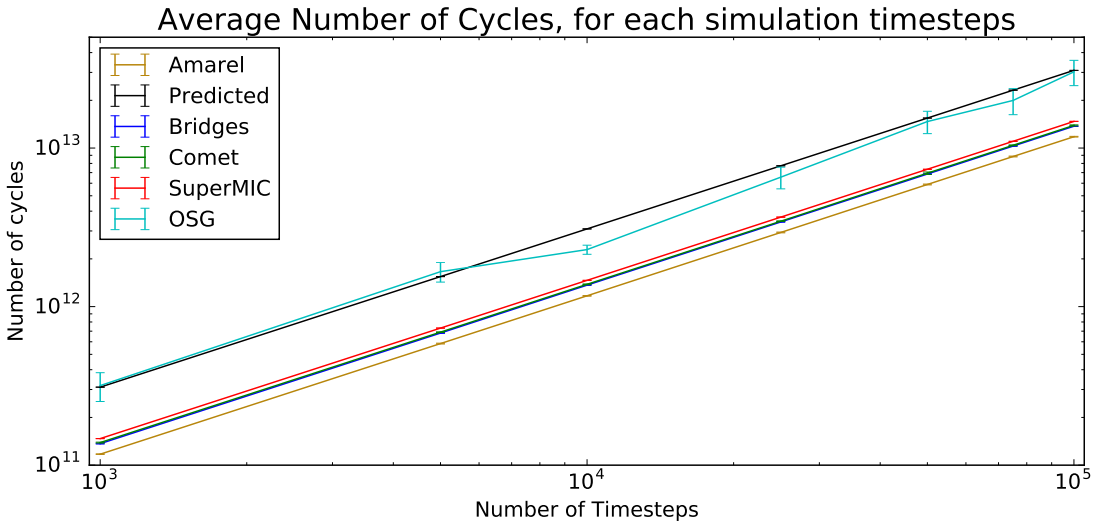



Figure 4.1: Average number of cycles measured on Amarel, Bridges, Comet, SuperMIC, OSG, as well as the average predicted number of cycles

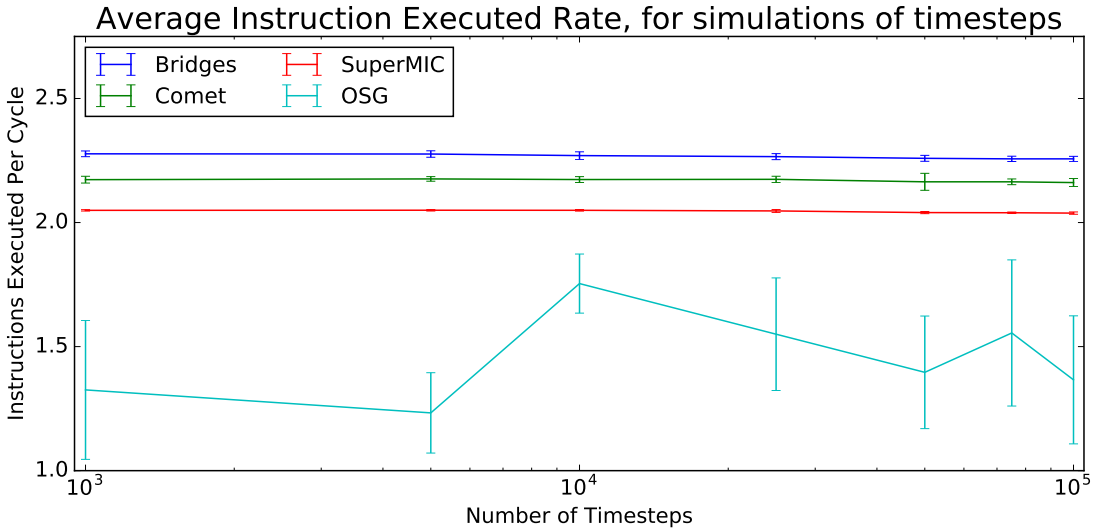



Figure 4.2: Instruction Rate (Instructions executed per cycle) of GROMACS simulations running on Bridges, Comet, SuperMIC and OSG

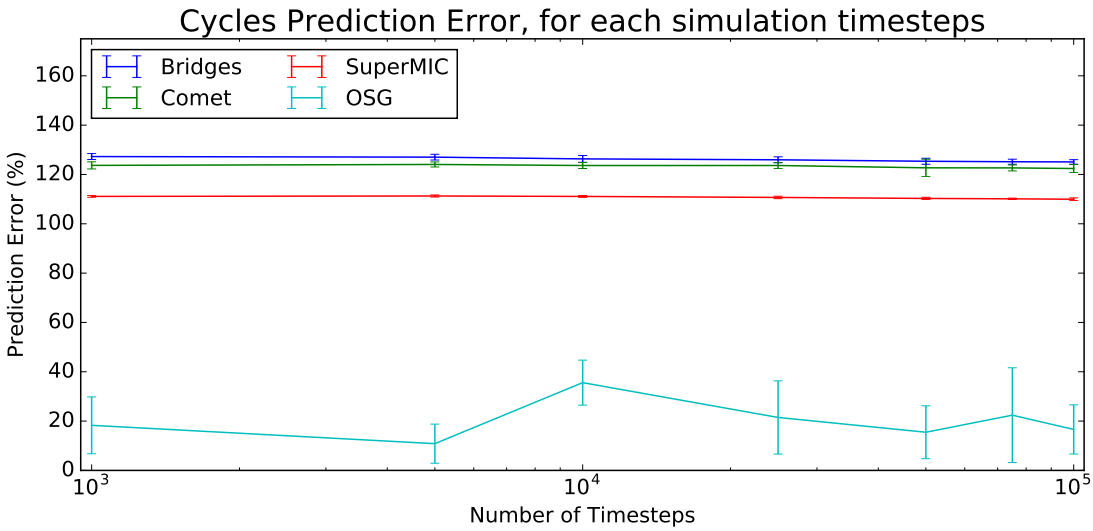



Figure 4.3: Prediction error of the number of cycles required to run GROMACS simulations on Bridges, Comet, SuperMIC and OSG

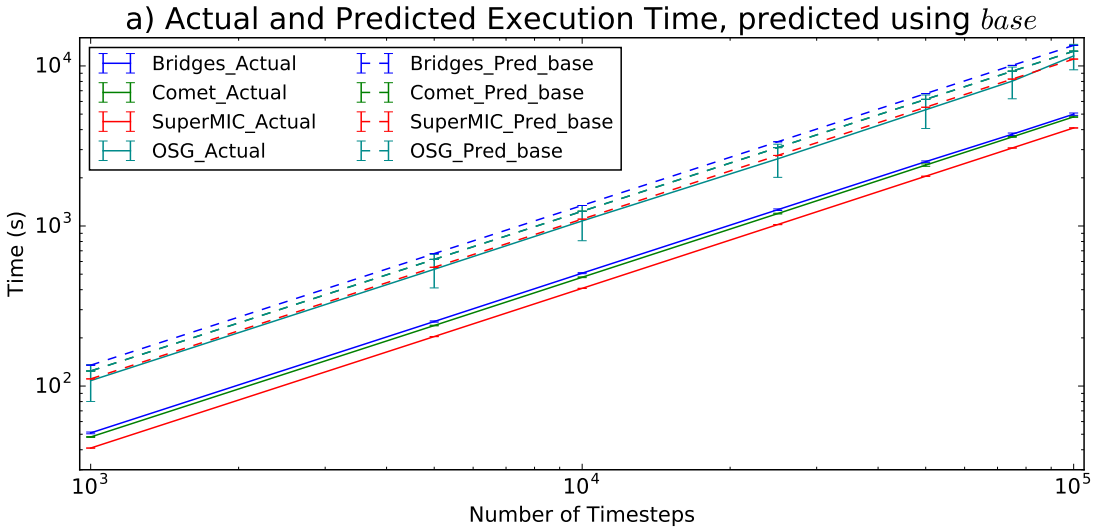



Figure 4.4: Average actual execution times and predicted execution times (using *base* frequencies) of GROMACS simulations running on Bridges, Comet, SuperMIC and OSG.

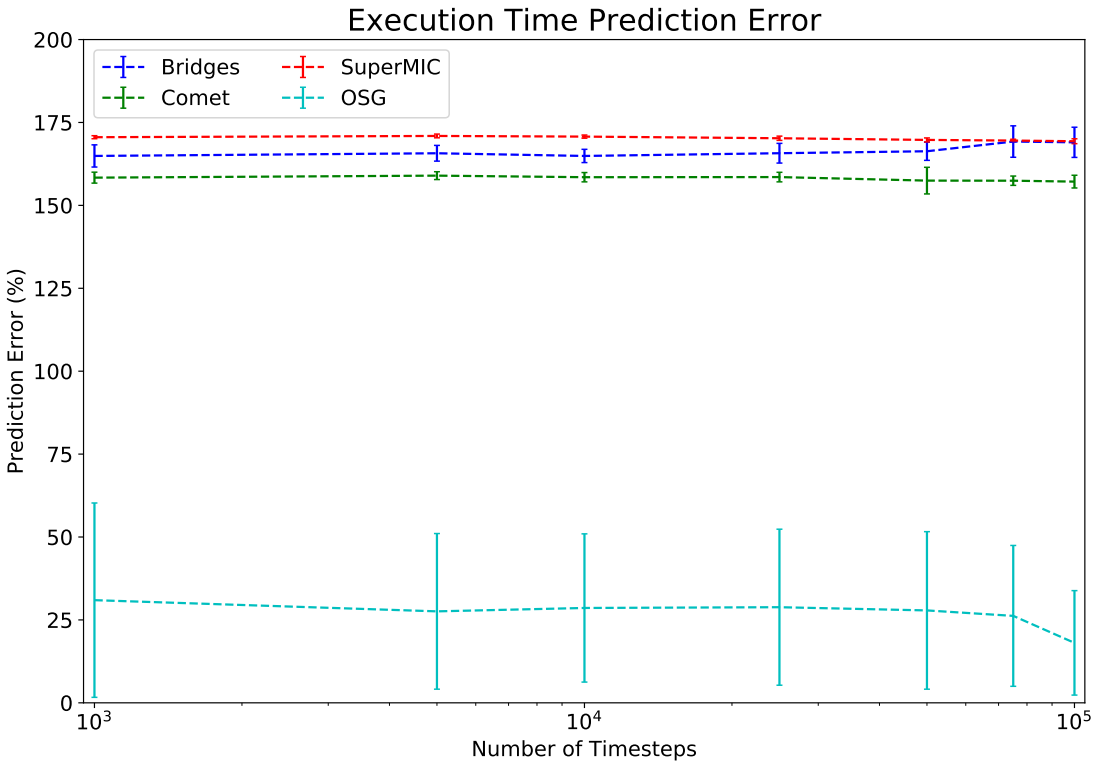



Figure 4.5: Prediction errors of the predicted execution times (using *base* frequencies) of GROMACS simulations running on Bridges, Comet, SuperMIC and OSG

Figs. 4.4 and 4.5 show that LIM overpredicts the task's execution time by 157–171% for the XSEDE supercomputers, and by 18–31% for OSG. While the prediction error in the task execution time is largely due to the prediction error in the number of cycles required, it is also due to the difference in clock frequencies *base* and *avg*. Despite LIM's prediction error, we can select resources from the XSEDE supercomputers using the resource ranking created using the execution time predictions generated from LIM. Fig. 4.4 shows that the rankings of XSEDE supercomputers made using $\hat{T}_x$ and using actual $T_x$ are the same.

However, the ranking of XSEDE supercomputers and OSG made using execution time predictions is not the same as that made using actual execution times. The predicted execution time on OSG is equal to that on Comet, and less than that on Bridges. In reality, the actual execution time on OSG is greater than that on Comet and Bridges. The inversions in the resource rankings created from the predicted execution times likely occurred because the task executes ∼22–24% more instructions on OSG than on the XSEDE supercomputers. To account for the additional instructions executed, we increased the predicted number of cycles required on OSG by 22%. Doing so makes the ranking of XSEDE supercomputers and OSG created using execution time predictions the same as that created using actual execution times. This shows that additional information on the software configurations on OSG resources can improve execution time predictions to correctly rank XSEDE supercomputers and OSG.

## 4.2 Experiment 2: Resource Selection

We performed a set of experiments executing a bag-of-tasks workload across the target machines to compare the performance of selecting resources based on the task's predicted execution time against that of selecting resources randomly from the target machines. The workload is composed of identical MD simulations, each running for $10^5$ timesteps as specified in Sec. 4.1. We measured the performance of each resource selection strategy in terms of the execution time and time-to-completion of the workload, both of which are metrics of particular interest to users. We call the set of runs where the resources were selected randomly or selected using LIM the 'random' and 'model' runs, respectively.

### 4.2.1 Setup

We executed workloads with 64, 128, 256, 512, and 1024 tasks across the target machines, repeatedly over the course of a month. Only one distributed execution of the workload occurred at any given time, preventing self-competition for resource acquisition. We used RADICAL-Pilot (RP) [102] to concurrently acquire resources from the target machines and to distribute the execution of the workload across those resources.

We submitted at most one pilot [103] to the local resource manager of any XSEDE supercomputer to acquire the resources necessary to execute the workload. Concurrently submitting multiple pilots to the same supercomputer would have created self-competition for resources, requiring further investigation of the effects of pilot sizing on the distributed execution of the workload [104]. On OSG, we submitted only single-core pilots to acquire the number of cores required execute the workload. While it is possible to submit multi-core pilots to OSG, the XSEDE OSG documentation recommends against it.

The metric used to select resources is the task's predicted execution time $\hat{T}_{x,task}$. Values of $\hat{T}_{x,task}$ used are LIM's predictions made using *base*. Since we found in Sec. 4.1 that running the task on OSG requires ∼22–24% more instructions than on XSEDE supercomputers, we increased the number of cycles required to execute the task without ILP on OSG by 22%.

We define the time-to-completion of a workload $TTC_{wkd}$ as $T_{q,wkd} + T_{x,wkd}$, where $T_{q,wkd}$ is the time spent only acquiring resources and $T_{x,wkd}$ is the amount of time spent executing at least one task. Since no data movement or pre-processing and post-processing occurs during the workload's execution, $TTC_{wkd}$ only involves $T_{q,wkd}$ and $T_{x,wkd}$. Similarly, we define $TTC_{task}$ as $T_{q,task} + T_{x,task}$, where $T_{q,task}$ is the amount of time a task waits before executing and $T_{x,task}$ as the task's actual execution time.

### 4.2.2 Results

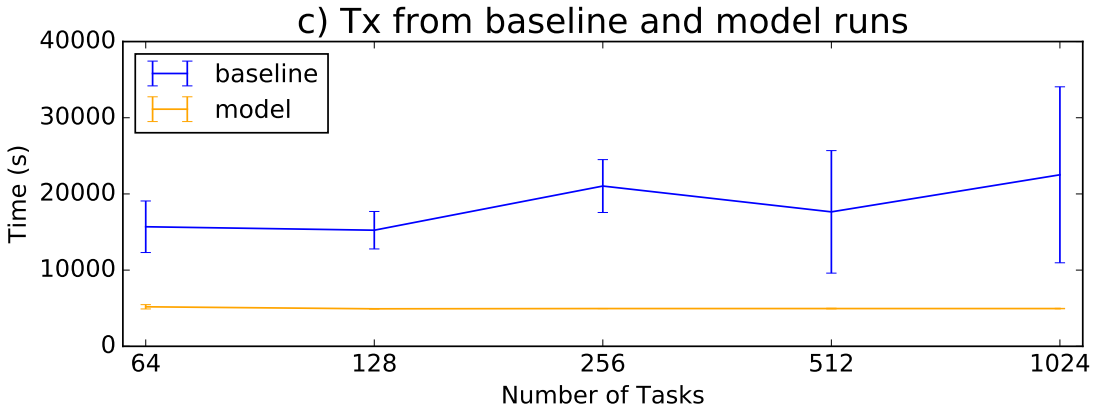
Fig. 4.6 shows the average $T_{x,wkd}$, $T_{q,wkd}$ and $TTC_{wkd}$ and the sample standard deviations of the runs we performed over a month. During the period in which the 'model' runs were performed, LIM selected SuperMIC as it yielded the lowest $\hat{T}_{x,task}$. From Fig. 4.6a, we see that the average $T_{x,wkd}$ of the 'model' runs were 67–78% smaller than that of the 'random' runs. This is because tasks experienced the lowest actual $T_{x,task}$ when executed on SuperMIC. Thus, assigning tasks to execute on SuperMIC minimized $T_{x,wkd}$.

For the 'model' runs, Fig. 4.6a shows that the $T_{x,wkd}$ of the 'model' runs for each workload size are almost identical, and that the sample standard deviations are almost negligible. This is because all tasks in the 'model' runs executed on SuperMIC. The sample standard deviations are larger for 'random' runs because tasks were randomly assigned to execute on the target machines. Moreover, tasks assigned to OSG experienced different $T_{x,task}$ as OSG is a heterogeneous pool of resources.

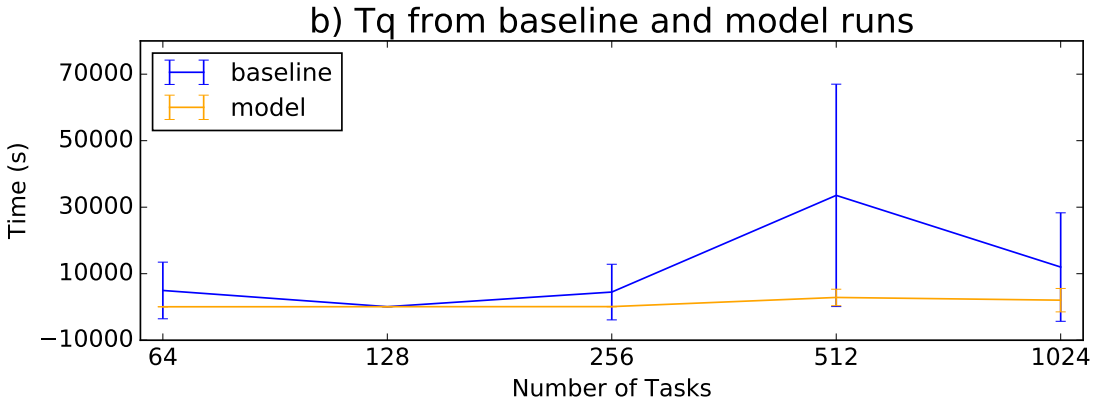
From Fig. 4.6b, we see that the average $T_{q,wkd}$ and the sample standard deviation from the 'model' runs are smaller than those from the 'random' runs. This is due to the different $T_{q,task}$ experienced on different machines. For the 'model' runs, all tasks were assigned to execute on SuperMIC, whose queue waiting times were on much lower and more stable than that of Bridges of Comet. This observation is consistent with the historical data of XDMoD [105].

Since tasks were randomly distributed across multiple machines for the 'random' runs, the queue waiting times experienced by the tasks were largely determined by the queue waiting times experienced by the pilots acquiring the resources. For 'random' runs of workload size 64 or 128, all pilots experienced relatively small queue waiting times. For 'random' runs of larger workload sizes, pilots submitted to Comet or Bridges commonly experienced large queue waiting times. From Fig 4.6b, we see that the average $T_{q,wkd}$ for workload sizes 64 and 128 are smaller than that for larger workload sizes. However, additional experiments are required to determine whether there is a relationship between the sample standard deviation and workload size.

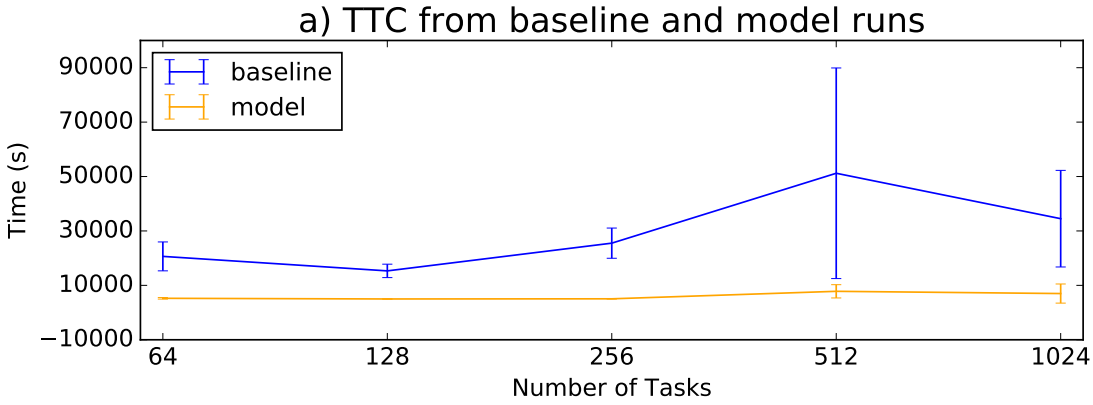
Fig. 4.6c shows that the average $TTC_{wkd}$ measured from the 'model' runs was 67–85% lower than that measured from the 'random' runs. Based on Figs. 4.6a and 4.6c, we see the the reduction in $T_{x,wkd}$ as a result of selecting resources using task execution time predictions contributed to a 29–99% reduction in $TTC_{wkd}$. The large variation in the reduction in $TTC_{wkd}$ as a result of a reduction in $T_{x,wkd}$ is due to the queue waiting times experienced by the pilots. By definition of $TTC_{wkd}$, the reduction in $TTC_{wkd}$ due to the reduction in $T_{x,wkd}$ is inversely related to the reduction in $TTC_{wkd}$ due to the reduction in $T_{q,wkd}$. From Figs. 4.6a and 4.6c, we see larger reductions in $TTC_{wkd}$ as a result of the reduction in $T_{x,wkd}$ when pilots experienced lower queue




(a) $T_{x,wkd}$




(b) $T_{q,wkd}$




(c) $TTC_{wkd}$

Figure 4.6: $T_{x,wkd}$, $T_{q,wkd}$ and $TTC_{wkd}$ measured from the random and model runs

waiting times.

While selecting resources using $\hat{T}_{x,task}$ minimized $T_{x,wkd}$ and reduced $TTC_{wkd}$, $TTC_{wkd}$ is still largely influenced by $T_{q,wkd}$. Thus, the effective selection of resources from XSEDE supercomputers and OSG should account for the queue waiting times experienced when acquiring resources.

## 4.3 Experiment 3: Resource Re-Selection

We study how the flexibility to re-assign tasks at runtime affects $TTC_{wkd}$. We used the log files collected from the 'random' runs in Sec. 4.2 to simulate the process of resource re-selection. We study how resource re-selection affects the $T_{x,task}$, $T_{q,task}$ and $TTC_{task}$ of re-assigned tasks. Furthermore, we compare the $T_{x,wkd}$, $T_{q,wkd}$ and $TTC_{wkd}$ measured from the resource re-selection simulation with those measured from the 'random' and 'model' runs discussed in Sec. 4.2. Though they are simulations, we refer to the set of resource re-selection simulation as 're-selection' runs.

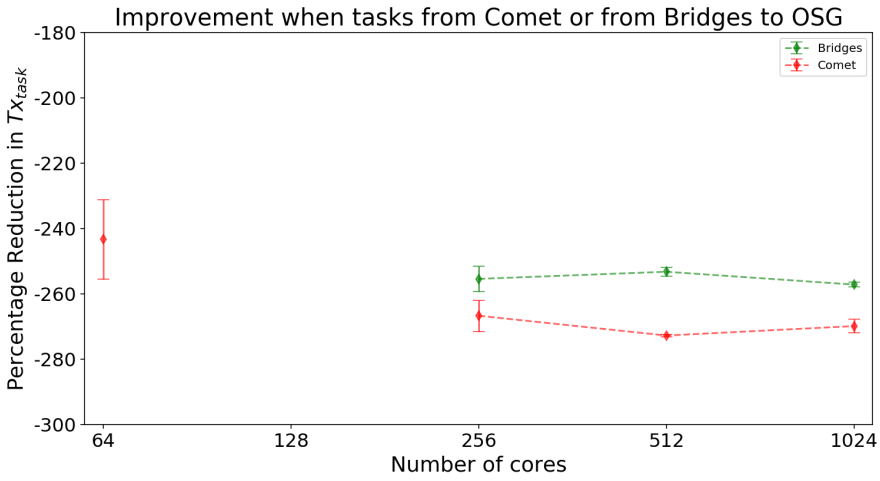
### 4.3.1 Setup

Assumption (6) (given in Sec. 3.4) restricts us to only re-assign tasks originally assigned to an XSEDE supercomputer to execute on XSEDE OSG. Since the queue waiting times of jobs submitted to XSEDE OSG are generally low, tasks re-assigned to OSG can begin executing relatively quickly. This is not the case for jobs submitted to XSEDE supercomputers as the queue waiting times these jobs experience can vary greatly. Strategies to acquire excess resources from supercomputers to allow tasks to be re-assigned to supercomputers is considered future work.

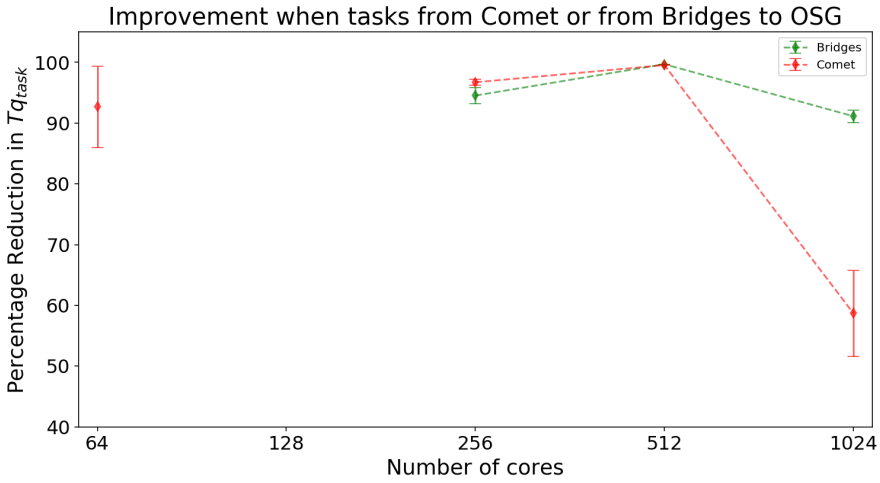
With these constraints, we simulated the process of resource re-selection by identifying tasks originally assigned to an XSEDE supercomputer that should be re-assigned to execute on OSG instead. For each run, we simulated the act of re-assigning tasks to OSG by changing the task's timestamps to the timestamps of the task which had the largest measured $TTC_{task}$ of all tasks that executed on OSG in that run. We used $\hat{T}_x$ predictions generated using LIM for each target machine, with the 22% increase in the predicted number of cycles on OSG, to determine whether a task should be re-assigned.

### 4.3.2 Results

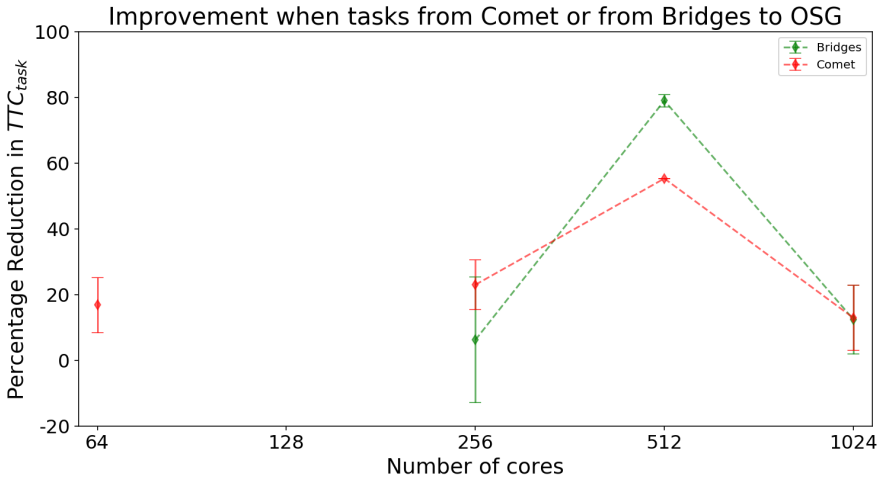
Fig. 4.7 shows the average percentage improvements to the $T_{x,task}$, $T_{q,task}$ and $TTC_{task}$ and the standard deviations of tasks after they have been re-assigned to execute on OSG. However, there are missing data points as not all tasks were candidates for re-assignment. No tasks assigned




(a) Change in $T_{x,task}$

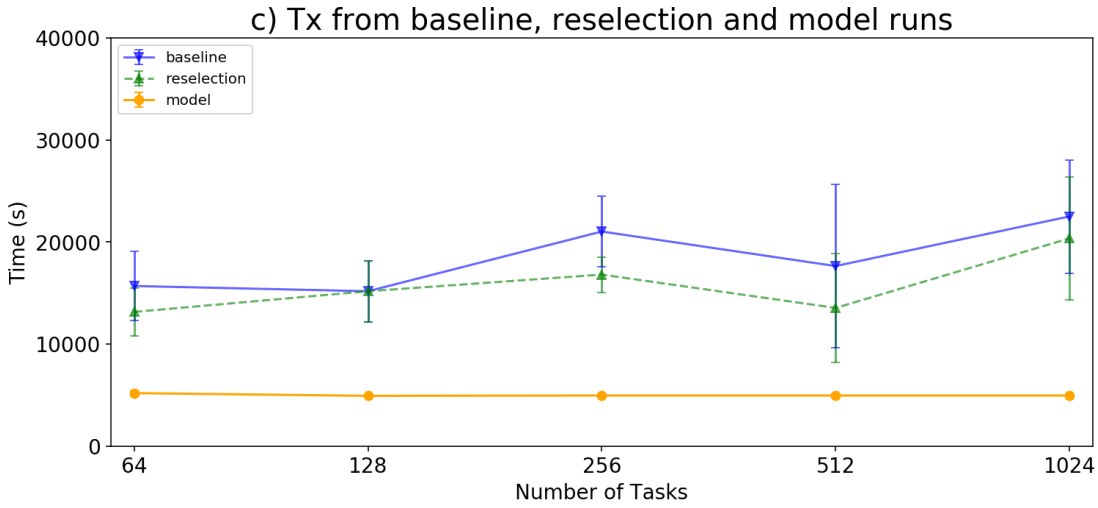



(b) Change in $T_{q,task}$

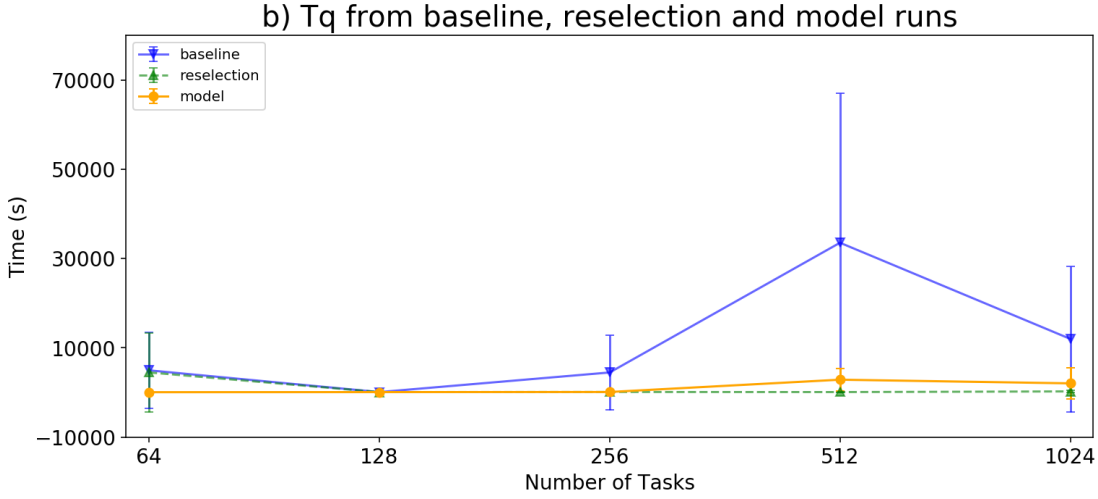



(c) Change in $TTC_{task}$

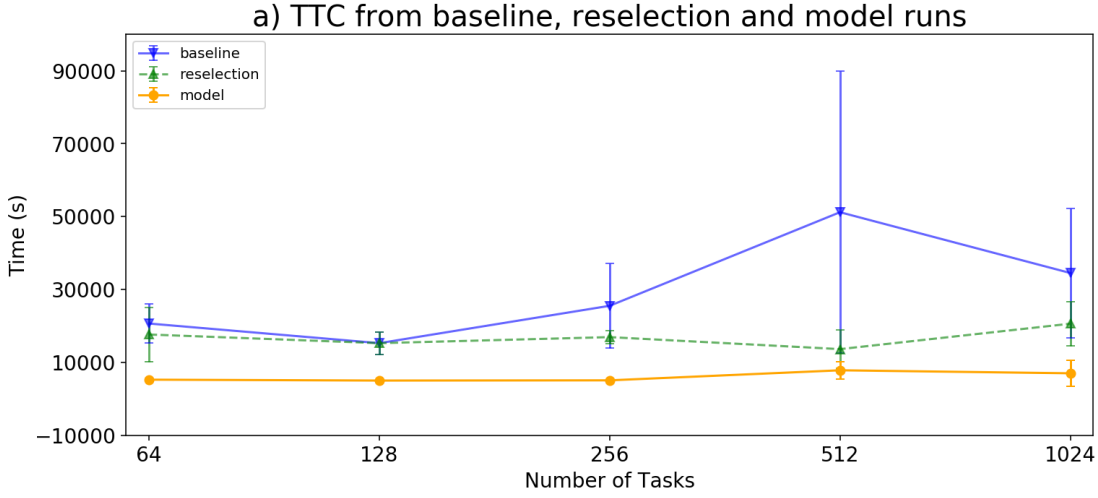
Figure 4.7: Percentage improvements of $T_{x,task}$, $T_{q,task}$ and $TTC_{task}$ when resource re-selection was performed on tasks from random runs. Positive improvement means decreased time; negative improvements means increased time.




(a) $T_{x,wkd}$




(b) $T_{q,wkd}$




(c) $TTC_{wkd}$

Figure 4.8: $T_{x,wkd}$, $T_{q,wkd}$ and $TTC_{x,wkd}$ measured from the random, re-selection and model runs

to SuperMIC were re-assigned to OSG because of the small queue waiting times experienced on SuperMIC. Also, we find that 50% of 'random' runs had tasks originally assigned to Comet or Bridges re-assigned to OSG instead. For 'random' runs of workload size 64, no tasks assigned to Bridges were re-assigned to OSG. Also, for 'random' runs of workload size 128, no task assigned to Bridges or Comet were re-assigned to OSG.

Fig. 4.7a shows that average $T_{x,task}$ of re-assigned tasks were increased by 240–280%. This is because the task's $T_x$ is larger on OSG than on Comet or Bridges. From Fig. 4.7b, we see that the average $T_{q,task}$ of re-assigned tasks were reduced by at least 90%, with the exception of tasks originally assigned to Bridges that belonged to workloads of size 1024. This is because the $T_{q,task}$ experienced by tasks running on Comet or Bridges dominated their $TTC_{task}$. Fig. 4.7c shows how the increase in $T_{x,task}$ and decrease in $T_{q,task}$ affects the $TTC_{task}$ of the re-assigned tasks. From Fig. 4.7c, we see that re-assigning tasks generally reduced their $TTC_{task}$. By re-assigning tasks that were originally assigned to Comet or Bridges to OSG instead, tasks were able to begin their execution immediately and avoided experiencing large $T_{q,task}$ on Comet or Bridges.

Of the tasks that were re-assigned to OSG, ∼30% of tasks originally assigned to Comet and ∼50% of tasks originally assigned to Bridges experienced an increase in $TTC_{task}$. This is because the difference in the queue waiting times experienced on Bridges (or Comet) and OSG was larger than the predicted re-selection window of Bridges (or Comet) and OSG, but smaller than the actual re-selection of Bridges (or Comet) and OSG, i.e. $\hat{W}_{OSG,Bridges} < (T_{q,Bridges} - T_{q,Bridges}) < W_{OSG,Bridges}$ and $\hat{W}_{OSG,Comet} < (T_{q,Comet} - T_{q,Comet}) < W_{OSG,Comet}$. From Sec. 3.4, we see that increases in $TTC_{task}$ caused by this type of incorrect re-assignment is bounded above by $|\hat{W}_{OSG,Bridges} - W_{OSG,Bridges}|$ for tasks originally assigned to Bridges, and $|\hat{W}_{OSG,Comet} - W_{OSG,Comet}|$ for those originally assigned to Comet. Nonetheless, from Fig. 4.7c, we see that the reduction of $TTC_{task}$ due to correct re-assignment was often much larger than the increase in $TTC_{task}$ caused by incorrect re-assignment.

Fig. 4.8 shows the average $T_{x,wkd}$, $T_{q,wkd}$ and $TTC_{wkd}$ and the sample standard deviations of the 're-selection', 'random' and 'model' runs. From Fig. 4.8b, we see that the average $T_{q,wkd}$ from 're-selection' runs is 0–99% lower than of the 'random' runs. The wide range of reduction in $T_{q,wkd}$ is due to the queue waiting times experienced by pilots acquiring resources.

For 'random' runs of workload sizes 64 and 128, all pilots experienced relatively low queue waiting times before acquiring their resources, and tasks generally executed on their originally assigned resources. So, the average $T_{q,wkd}$ of 're-selection' runs of workload sizes 64 and 128 is only 0–9% smaller than that of the 'random' runs of the corresponding workload sizes. For 'random' runs of workload size 256 and higher, pilots submitted to Comet or Bridges commonly experienced large queue waiting times before acquiring resources. Performing resource

re-selection allowed tasks originally assigned to Comet or Bridges to OSG instead to avoid experiencing large $T_{q,task}$. As a result, the average $T_{q,wkd}$ of 're-selection' runs of workload sizes 256 or higher is 97–99% smaller than that of the 'random' runs of the corresponding sizes.

It is important to note that resource re-selection is useful in allowing tasks to avoid experiencing large $T_{q,task}$ on their originally assigned resource by re-assigning them to execute on a different, acquired resource. However, resource re-selection is less useful if the tasks experience small $T_{q,task}$ before they begin executing on their originally assigned resource. Still, for workload sizes of 128 and larger, Fig. 4.8b shows that the average $T_{q,wkd}$ of the 're-selection' runs is similar to that of the 'model' runs.

Fig. 4.8a shows that the average $T_{x,wkd}$ of 're-selection' runs is generally smaller than that of 'random' runs. This is because the tasks that were originally assigned to Comet or Bridges and were re-assigned to OSG were able to execute concurrently with tasks that were originally assigned to OSG. An increase in the number of concurrently running tasks decreases the amount of time spent executing all tasks of the workload. So, for runs of all workload sizes except 128, Fig. 4.8a shows that the average $T_{x,wkd}$ of 're-selection' runs are smaller than that of 'random' runs. The average $T_{x,wkd}$ of 'random' and 're-selection' runs of workload size 128 are the same because no tasks were re-assigned.

While resource re-selection can reduce $T_{x,wkd}$ by increasing the number of concurrently executing tasks, $T_{x,wkd}$ can never be smaller than the largest $T_{x,task}$ experienced by any of the tasks. Since $T_{x,task}$ is highest on OSG, the $T_{x,wkd}$ of a 're-selection' run is no smaller than the largest $T_{x,task}$ experienced by any task that was assigned/re-assigned to execute on OSG. As a result, from Fig. 4.8a, we see that the average $T_{x,wkd}$ of the 'model' runs are consistently smaller than that from the re-selection' runs. This is because every task from the 'model' runs executed on SuperMIC, which has a lower $T_{x,task}$ than on OSG.

From Fig. 4.8c, we see that the average $TTC_{wkd}$ of the 're-selection' runs is 0–73% smaller than that of the 'random' runs. Again, the reduction in $TTC_{wkd}$ is dependent upon the queue waiting times experienced by the pilots. For the 'random runs' of workload sizes 64 and 128, the resources were quickly acquired from the target machines, and only a few tasks were re-assigned. For workload sizes 64 and 128, the average $TTC_{wkd}$ of the 'random' runs is only 0–4% smaller than that of the 're-selection runs'. For workload sizes 256 and larger, the queue waiting times experienced on Comet and Bridges were large, and many tasks were re-assigned. As such, for workload sizes 256 and larger, the average $TTC_{wkd}$ is a ∼33–73% smaller than that of the 'random' runs.

# Chapter 5

# Discussions

From the results of Sec. 4.1, we found that additional information on the GROMACS configuration on OSG should be provided to the user. Despite attempts to replicate the GROMACS configuration on OSG on the other target machines, GROMACS simulations executed 22–24% more instructions on OSG than on the other machines. By adjusting for the additional instructions, we were able to correctly rank all target machines using their predicted task execution times. Using this adjusted ranking, we were able to effectively perform both resource selection and resource re-selection, as shown in Secs. 4.2 and 4.3. We claim that a better understanding of the software environment on OSG resources can improve resource selection across XSEDE supercomputers and grids.

From Secs. 4.2 and 4.3, we see that resources can be effectively selected using inaccurate task execution time predictions if these predictions correctly rank resources. As such, it may not be necessary to collected detailed information about resource capabilities, as was done in [48–51,57], to generate task execution time predictions for resource selection or re-selection. This is reassuring as the information required to accurately predict task execution times can be difficult, if not infeasible, to collected from grid resources [66, 68].

From Sec. 4.3, we see that resource re-selection can reduce the $T_{q,wkd}$ as it allows tasks to execute on other acquired resources. An advantage of resource re-selection is that it does not use queue waiting time predictions, which are difficult to predict accurately. However, the ability to re-assign tasks depends upon which and how many resources are acquired. In Sec. 4.3, tasks were only re-assigned from XSEDE supercomputers to OSG. Resource re-selection can be correctly performed between XSEDE supercomputers with more accurate task execution time predictions by taking advantage of the abundance of information that can be collected about XSEDE supercomputers.

Resource re-selection can be further improved if excess resources from XSEDE supercomput-

ing were acquired. This allows tasks to be re-assigned to execute on different XSEDE supercomputers, further reducing $T_{x,wkd}$ and $T_{q,wkd}$. Deciding the amout of excess resources to acquire from supercomputers for re-assignment touches upon the job sizing problem [104], which studies how many resources should be acquired from batch systems so that the amount of unused resources is minimized. Work in this direction will focus on the trade-offs between the acquisition and utilization of excess resources for resource re-selection and the reduction in $TTC_{wkd}$ as a result of performing resource re-selection.

Since resource re-selection focuses on selecting resources during the execution of a workload, it can be used in conjunction with methods that select resources before the execution of a workload. As an example, the resource re-selection process can be incorporated into the Condor matchmaking algorithm discussed in Sec. 3.3). Rather than returning only one resource with the highest affinity value, $RES\_SELECT$ can return a set of resources with the $k$ highest affinity values. Initially, a task is assigned to execute on one of the $k$ resources. If a resource with a higher or lower affinity value is acquired, the task can decide whether or not to restart its execution on the newly acquired resource based on the affinity values of the two resources and the cost for restarting its execution.

Resource re-selection can also be incorporated into plan-based workload scheduling algorithms to enable them to re-assign tasks based on resource acquisitoins at runtime. Many workload scheduling algorithms do not consider resource acquisition times when performing resource selection [72–84]. While some workload scheduling algorithms are able to decide whether to acquire and use additional resources during the execution of the workload [81, 87–91], they assume that resource acquisition times are small and negligible. As such, they lack the ability to effectively schedule or re-schedule tasks when resources acquisition times are lage. By incorporating resource re-selection, workload scheduling algorithms can adaptively schedule tasks based on resource acqusitions at runtime.

# Chapter 6

# Conclusions and Future Work

In this thesis, we studied how to effectively select resources from XSEDE supercomputers and OSG in the presences of limited information. In Ch. 3, we presented a formalism that provided us with the flexibility to model the cost of task execution based on the information available from XSEDE supercomputers and OSG. We showed that this formalism can be incorporated into the Condor matchmaking algorithm to address the resource selection problem. On the base of this formalism, we created the Limited Information Model (LIM) to predict the execution times of compute-intensive, single-threaded, single-process tasks. To overcome the difficulty of selecting resources using queue waiting times, we developed a process called resource re-selection to re-assign tasks to execute on different, acquired resources at runtime using task execution time predictions and resource acquisition times, but not queue waiting time predictions.

In Ch. 4, we investigated whether task execution time predictions generated using the information from XSEDE supercomputers and OSG can be used to correctly select resources to execute GROMACS simulations. We found that additional information on the GROMACS configuration on OSG is required since GROMACS simulations running on OSG can execute 22–24% more instructions than on XSEDE supercomputers. By accounting for the additional instructions executed on OSG, we were able to generate tasks execution time predictions that correctly ranked the XSEDE supercomputers and OSG.

Also in Ch. 4, we studied how selecting resources based on the predicted task execution times of GROMACS simulations can reduce the time-to-completion of bag-of-task workloads composed of identical GROMACS simulations. We found that selecting resources based on the predicted task execution times resulted in 67–85% lower workload time-to-completion than when resources are selected randomly. However, executing workloads on resources selected using LIM's predictions contributed to ∼29–99% of the reduction in workload time-to-completion. We found that the queue waiting times largely influences workload time-to-completion and should also be

considered to effectively select resources from XSEDE supercomputers and OSG.

Using the log files collected when executing workloads using randomly selected resources, we simulated the act of performing resource re-selection during workload execution. By re-selecting tasks originally assigned to XSEDE supercomputers to OSG instead, tasks were able to avoid experiencing large queue waiting times. Though their execution times increased, most tasks experienced reduced task time-to-completions. As a result, re-selecting resources reduced workload queue waiting times by up to 99% and workload time-to-completion by up to 73%. However, resource re-selection provides little to no reductions in task queue waiting times and workload time-to-completions when the queue waiting times experienced on XSEDE supercomputers are small. Still, results show that resource re-selection can be used to reduce the time-to-completions of workloads executed using supercomputing and grid resources.

One direction in which this work can evolve is by extending the formalism so that it can describe computing tasks which use multithreading or multiprocessing. This extended formalism can be used to create models to predict the execution times of parallel applications using information available from supercomputers and grids. This can inform designers of large computing systems on the type, amount and granularity of information to provide to users to enable them to effectively select resources from multiple infrastructures.

Another direction in which this work can evolve is by investigating how incorporating resource re-selection into workflow scheduling algorithms can affect the execution of workflows using supercomputers. While queue waiting times experienced on supercomputers can be large, many workflow scheduling algorithms do not account for queue waiting times when scheduling workflows. Additional studies can be performed to determine how resource re-selection can be incorporated into existing workflow scheduling algorithms to improve the execution of workloads using supercomputers.

# Bibliography

[1] Hongtao Zhao and Amedeo Caflisch. Molecular dynamics in drug design. *European journal of medicinal chemistry*, 91:4–14, 2015.

[2] Adam Hospital, Josep Ramon Goñi, Modesto Orozco, and Josep L Gelpí. Molecular dynamics simulations: advances and applications. *Advances and applications in bioinformatics and chemistry: AABC*, 8:37, 2015.

[3] Marco De Vivo, Matteo Masetti, Giovanni Bottegoni, and Andrea Cavalli. Role of molecular dynamics and related methods in drug discovery. *Journal of medicinal chemistry*, 59(9):4035–4061, 2016.

[4] Nikolay A Simakov, Joseph P White, Robert L DeLeon, Steven M Gallo, Matthew D Jones, Jeffrey T Palmer, Benjamin Plessinger, and Thomas R Furlani. A workload analysis of nsf's innovative hpc resources using xdmod. *arXiv preprint arXiv:1801.04306*, 2018.

[5] Thomas J Lane, Diwakar Shukla, Kyle A Beauchamp, and Vijay S Pande. To milliseconds and beyond: challenges in the simulation of protein folding. *Current opinion in structural biology*, 23(1):58–65, 2013.

[6] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, et al. The open science grid. In *Journal of Physics: Conference Series*, volume 78, page 012057. IOP Publishing, 2007.

[7] Tsjerk A Wassenaar, Marc Van Dijk, Nuno Loureiro-Ferreira, Gijs Van Der Schot, Sjoerd J De Vries, Christophe Schmitz, Johan Van Der Zwan, Rolf Boelens, Andrea Giachetti, Lucio Ferella, et al. Wenmr: structural biology on the grid. *Journal of Grid Computing*, 10(4):743–767, 2012.

[8] Ian Foster and Carl Kesselman. Computational grids. 1998.

[9] Rajib Mukherjee, Abhinav Thota, Hideki Fujioka, Thomas C Bishop, and Shantenu Jha. Running many molecular dynamics simulations on many supercomputers. In *Proceedings*

*of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, page 2. ACM, 2012.

[10] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146. IEEE, 1998.

[11] OGF JSDL Workgroup. Job submission description language (jsdl) specification, version 1.0.

[12] Marvin Solomon. The classad language reference manual, version 2.1. *Computer Sciences Department, University of Wisconsin, Madison, WI, USA*, 2003.

[13] Tim Banks. Web services resource framework (wsrf)-primer v1. 2. *OASIS committee draft*, pages 02–23, 2006.

[14] Globus resource specification language.

[15] Laurence Field, CERN Gerson Galang, and ARCS Balazs Konya. Glue specification v. 2.0. *Recomendation GFD*, 147, 2009.

[16] Yang-Suk Kee, Dionysios Logothetis, Richard Huang, Henri Casanova, and Andrew A Chien. Efficient resource description and high quality selection for virtual grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 1, pages 598–606. IEEE, 2005.

[17] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 181–194. IEEE, 2001.

[18] Rich Wolski, Neil T Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5):757–768, 1999.

[19] Rajesh Raman and Miron Livny. Matchmaking frameworks for distributed resource management. *University of Wisconsin-Maddison*, 2001.

[20] Paul Z Kolano. Surfer: an extensible pull-based framework for resource selection and ranking. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 563–571. IEEE, 2004.

[21] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *High Performance*

*Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 55–63. IEEE, 2001.

[22] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, volume 1, pages 283–289. IEEE, 2000.

[23] Youn-Seok Kim, Jung-Lok Yu, Jae-Gyoon Hahm, Jin-Soo Kim, and Joon-Won Lee. Design and implementation of an ogsi-compliant grid broker service. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 754–761. IEEE, 2004.

[24] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *International Workshop on Grid Computing*, pages 191–202. Springer, 2000.

[25] Andrei Tchernykh, Juan Manuel Ramírez, Arutyun Avetisyan, Nikolai Kuzjurin, Dmitri Grushin, and Sergey Zhuk. Two level job-scheduling strategies for a computational grid. In *International Conference on Parallel Processing and Applied Mathematics*, pages 774–781. Springer, 2005.

[26] Michael A Iverson, Fusun Ozguner, and Lee C Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 99–111. IEEE, 1999.

[27] Michael A Iverson, Fusun Ozguner, and Gregory J Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on*, pages 263–270. IEEE, 1996.

[28] Rubing Duan, Farrukh Nadeem, Jie Wang, Yun Zhang, Radu Prodan, and Thomas Fahringer. A hybrid intelligent method for performance modeling and prediction of workflow activities in grids. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 339–347. IEEE Computer Society, 2009.

[29] Egin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A Mckee. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing*, pages 196–205. Springer, 2005.

[30] Benjamin C Lee, David M Brooks, Bronis R de Supinski, Martin Schulz, Karan Singh, and Sally A McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 249–258. ACM, 2007.

[31] Artem M Chirkin, Adam SZ Belloum, Sergey V Kovalchuk, Marc X Makkes, Mikhail A Melnik, Alexander A Visheratin, and Denis A Nasonov. Execution time estimation for workflow scheduling. *Future Generation Computer Systems*, 2017.

[32] Tudor Miu and Paolo Missier. Predicting the execution time of workflow blocks based on their input features. *School of Computing Science Technical Report Series*, 2013.

[33] Andréa Matsunaga and José AB Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.

[34] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016, 2004.

[35] Farrukh Nadeem and Thomas Fahringer. Predicting the execution time of grid workflow applications through local learning. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12. IEEE, 2009.

[36] Farrukh Nadeem and Thomas Fahringer. Using templates to predict execution time of scientific workflow applications in the grid. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 316–323. IEEE Computer Society, 2009.

[37] Farrukh Nadeem and Thomas Fahringer. Optimizing execution time predictions of scientific workflow applications in the grid through evolutionary programming. *Future Generation Computer Systems*, 29(4):926–935, 2013.

[38] Hugo Hiden, Simon Woodman, and Paul Watson. Prediction of workflow execution time using provenance traces: practical applications in medical data processing. In *e-Science (e-Science), 2016 IEEE 12th International Conference on*, pages 21–30. IEEE, 2016.

[39] Rafael Ferreira Da Silva, Gideon Juve, Mats Rynge, Ewa Deelman, and Miron Livny. Online task resource consumption prediction for scientific workflows. *Parallel Processing Letters*, 25(03):1541003, 2015.

[40] Rafael Ferreira Da Silva, Gideon Juve, Ewa Deelman, Tristan Glatard, Frédéric Desprez, Douglas Thain, Benjamin Tovar, and Miron Livny. Toward fine-grained online task characteristics estimation in scientific workflows. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, pages 58–67. ACM, 2013.

[41] Erik Elmroth and Johan Tordsson. A grid resource broker supporting advance reservations and benchmark-based resource selection. In *International Workshop on Applied Parallel Computing*, pages 1061–1070. Springer, 2004.

[42] Andrea Clematis, Angelo Corana, Daniele D'Agostino, Antonella Galizia, and Alfonso Quarati. Job–resource matchmaking on grid through two-level benchmarking. *Future Generation Computer Systems*, 26(8):1165–1179, 2010.

[43] Omid Khalili. *Performance prediction and ordering of supercomputers using a linear combination of benchmark measurements.* PhD thesis, University of California, San Diego, 2007.

[44] Tzu-Yi Chen, Omid Khalili, Roy L Campbell, Laura Carrington, Mustafa M Tikir, and Allan Snavely. Performance prediction and ranking of supercomputers. *Advances in Computers*, 72:135–172, 2008.

[45] T Chen, Meghan Gunn, Beth Simon, Laura Carrington, and Allan Snavely. Metrics for ranking the performance of supercomputers. *Cyberinfrastructure Technology Watch Journal: Special Issue on High Productivity Computer Systems*, 2(4):1–8, 2007.

[46] John L Gustafson and Rajat Todi. Conventional benchmarks as a sample of the performance spectrum. *The Journal of Supercomputing*, 13(3):321–342, 1999.

[47] David H Bailey and Allan Snavely. Performance modeling: Understanding the past and predicting the future. In *European Conference on Parallel Processing*, pages 185–195. Springer, 2005.

[48] Darren J Kerbyson, Henry J Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J Wasserman, and Mike Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 37–37. ACM, 2001.

[49] Hongzhang Shan, Erich Strohmaier, Ji Qiang, David H Bailey, and Kathy Yelick. Performance modeling and optimization of a high energy colliding beam simulation code. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 97. ACM, 2006.

[50] Kevin J Barker, Kei Davis, and Darren J Kerbyson. Performance modeling in action: Performance prediction of a cray xt4 system during upgrade. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[51] Jens Simon and Jens-Michael Wierum. Accurate performance prediction for massively parallel systems and its applications. In *European Conference on Parallel Processing*, pages 675–688. Springer, 1996.

[52] Sameer S Shende and Allen D Malony. The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[53] Markus Geimer, Felix Wolf, Brian JN Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[54] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Tools for High Performance Computing 2011*, pages 79–91. Springer, 2012.

[55] Wolfgang E Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. Vampir: Visualization and analysis of mpi resources. 1996.

[56] Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. Periscope: An online-based distributed performance analysis tool. *Tools for High Performance Computing 2009*, pages 1–16, 2010.

[57] Allan Snavely, Nicole Wolter, and Laura Carrington. Modeling application performance by convolving machine signatures with application profiles. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 149–156. IEEE, 2001.

[58] Laura Carrington, Michael A Laurenzano, and Ananta Tiwari. Inferring large-scale computation behavior via trace extrapolation. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1667–1674. IEEE, 2013.

[59] Alfredo Tirado-Ramos, George Tsouloupas, Marios Dikaiakos, and Peter Sloot. Grid resource selection by application benchmarking for computational haemodynamics applications. In *International Conference on Computational Science*, pages 534–543. Springer, 2005.

[60] Elisa Heymann, Alvaro Fernández, Miquel A Senar, and José Salt. The eu-crossgrid approach for grid application scheduling. In *Grid Computing*, pages 17–24. Springer, 2004.

[61] Marios D Dikaiakos. Grid benchmarking: vision, challenges, and current status. *Concurrency and Computation: Practice and Experience*, 19(1):89–105, 2007.

[62] Greg Chun, Holly Dail, Henri Casanova, and Allan Snavely. Benchmark probes for grid assessment. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 276. IEEE, 2004.

[63] Farrukh Nadeem, Radu Prodan, Thomas Fahringer, and Alexandru Iosup. Benchmarking grid applications. In *Grid Middleware and Services*, pages 19–37. Springer, 2008.

[64] Radu Prodan, Farrukh Nadeeem, and Thomas Fahringer. Benchmarking grid applications for performance and scalability predictions. *Handbook of Research on Scalable Computing Technologies*, 1:89, 2009.

[65] George Tsouloupas and Marios D Dikaiakos. Gridbench: A workbench for grid benchmarking. In *European Grid Conference*, pages 211–225. Springer, 2005.

[66] Alexandru Iosup, Dick HJ Epema, Carsten Franke, Alexander Papaspyrou, Lars Schley, Baiyi Song, and Ramin Yahyapour. On grid performance evaluation using synthetic workloads. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 232–255. Springer, 2006.

[67] Alexandru Iosup and Dick Epema. Grenchmark: A framework for analyzing, testing, and comparing grids. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 313–320. IEEE, 2006.

[68] Rob F Van der Wijngaart and Michael A Frumkin. Evaluating the information power grid using the nas grid benchmarks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 275. IEEE, 2004.

[69] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick HJ Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[70] Dror G Feitelson. Parallel workload archive. *http://www. cs. huji. ac. il/labs/parallel/-workload*, 2007.

[71] Dror G Feitelson, Dan Tsafrir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982, 2014.

[72] Henan Zhao and Rizos Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 14–pp. IEEE, 2006.

[73] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D Dikaiakos. Scheduling workflows with budget constraints. In *Integrated research in GRID computing*, pages 189–202. Springer, 2007.

[74] Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 111. IEEE, 2004.

[75] Hamid Mohammadi Fard, Radu Prodan, and Thomas Fahringer. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. *IEEE Transactions on Parallel and Distributed systems*, 24(6):1203–1212, 2013.

[76] Amelie Chi Zhou, Bingsheng He, and Cheng Liu. Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. *IEEE Transactions on Cloud Computing*, 4(1):34–48, 2016.

[77] Maria Alejandra Rodriguez and Rajkumar Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, 2014.

[78] Ilia Pietri, Maciej Malawski, Gideon Juve, Ewa Deelman, Jarek Nabrzyski, and Rizos Sakellariou. Energy-constrained provisioning for scientific workflow ensembles. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 34–41. IEEE, 2013.

[79] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Record*, 34(3):56–62, 2005.

[80] Mustafizur Rahman, Srikumar Venugopal, and Rajkumar Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *e-Science and Grid Computing, IEEE International Conference on*, pages 35–42. IEEE, 2007.

[81] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[82] Zhifeng Yu and Weisong Shi. A planner-guided scheduling strategy for multiple workflow applications. In *Parallel Processing-Workshops, 2008. ICPP-W'08. International Conference on*, pages 1–8. IEEE, 2008.

[83] Georgios L Stavrinides and Helen D Karatza. A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 231–239. IEEE, 2015.

[84] Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Kołodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, 51:61–71, 2015.

[85] Alexey Ilyushkin and Dick Epema. The impact of task runtime estimate accuracy on scheduling workloads of workflows. In *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*.

[86] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 759–767. IEEE, 2005.

[87] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 27–34. IEEE, 2010.

[88] Jia Yu, Rajkumar Buyya, Chen Khong Tham, et al. Qos-based scheduling of workflow applications on service grids. In *Proc. of 1st IEEE International Conference on e-Science and Grid Computing*, 2005.

[89] Jianwu Wang, Prakashan Korambath, Ilkay Altintas, Jim Davis, and Daniel Crawl. Workflow as a service in the cloud: architecture and scheduling algorithms. *Procedia Computer Science*, 29:546–556, 2014.

[90] Maria A Rodriguez and Rajkumar Buyya. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. *Future Generation Computer Systems*, 2017.

[91] Francine Berman, Henri Casanova, Andrew Chien, Keith Cooper, Holly Dail, Anshuman Dasgupta, Wei Deng, Jack Dongarra, Lennart Johnsson, Ken Kennedy, et al. New grid

scheduling and rescheduling methods in the grads project. *International Journal of Parallel Programming*, 33(2-3):209–229, 2005.

[92] Alexey Ilyushkin, Bogdan Ghit, and Dick Epema. Scheduling workloads of workflows with unknown task runtimes. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 606–616. IEEE, 2015.

[93] Warren Smith, Sudhakar Pamidighantam, and John-Paul Navarro. Publishing and consuming glue v2. 0 resource information in xsede. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 25. ACM, 2015.

[94] Daniel Nurmi, John Brevik, and Rich Wolski. Qbets: queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 76–101. Springer, 2007.

[95] Allen B Downey. Using queue time predictions for processor allocation. In *workshop on Job Scheduling Strategies for Parallel Processing*, pages 35–57. Springer, 1997.

[96] Hui Li, Juan Chen, Ying Tao, David Gro, and Lex Wolters. Improving a local learning technique for queuewait time predictions. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 335–342. IEEE, 2006.

[97] Hui Li, David Groep, and Lex Wolters. Mining performance data for metascheduling decision support in the grid. *Future Generation Computer Systems*, 23(1):92–99, 2007.

[98] Ozan Sonmez, Nezih Yigitbasi, Alexandru Iosup, and Dick Epema. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 111–120. ACM, 2009.

[99] Rajath Kumar and Sathish Vadhiyar. Identifying quick starters: towards an integrated framework for efficient predictions of queue waiting times of batch parallel jobs. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 196–215. Springer, 2012.

[100] Prakash Murali and Sathish Vadhiyar. Qespera: an adaptive framework for prediction of queue waiting times in supercomputer systems. *Concurrency and Computation: Practice and Experience*, 2015.

[101] 2015.

[102] André Merzky, Mark Santcroos, Matteo Turilli, and Shantenu Jha. Radical-pilot: Scalable execution of heterogeneous and dynamic workloads on supercomputers. *CoRR*, abs/1512.08194, 2015.

[103] Matteo Turilli, Mark Santcroos, and Shantenu Jha. A comprehensive perspective on pilot-job systems. *arXiv preprint arXiv:1508.04180*, 2015.

[104] B. Tovar, R. Ferreira da Silva, G. Juve, E. Deelman, W. Allcock, D. Thain, and M. Livny. A job sizing strategy for high-throughput scientific workflows. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2017.

[105] 2014.