A PROGRAMMING MODEL AND EXECUTION SYSTEM FOR ADAPTIVE ENSEMBLE APPLICATIONS ON HIGH PERFORMANCE COMPUTING SYSTEMS

by

VIVEKANANDAN BALASUBRAMANIAN

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Dr. Shantenu Jha and Dr. Matteo Turilli

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

June, 2019

ABSTRACT OF THE DISSERTATION

A programming model and execution system for adaptive ensemble applications on high performance computing systems

by Vivekanandan Balasubramanian

Dissertation Director: Dr. Shantenu Jha and Dr. Matteo Turilli

Traditionally, advances in high-performance scientific computing have focused on the scale, performance, and optimization of an application with a large, single task, and less on applications comprised of multiple tasks. However, many scientific problems are expressed as applications that require the collective outcome of one or more ensembles of computational tasks in order to provide insight into the problem being studied. Depending on the scientific problem, a task of an ensemble can be any type of a program: from a molecular simulation, to a data analysis or a machine learning model. With different communication and coordination patterns, both within and across ensembles, the number and type of applications that can be formulated as ensembles is vast and spans many scientific domains, including biophysics, climate science, polar science and earth science.

The performance of ensemble applications can be improved further by using partial results to adapt the application at runtime. Partial results of ongoing executions can be analyzed with multiple methods to adapt the application to focus on relevant portions of the problem space or reduce the time to execution of the application. These benefits are confirmed by the increasing role played by adaptivity in ensemble applications developed

to support several domain sciences, including biophysics and climate science.

Although HPC systems provide the computational power required for ensemble applications, their design and policies tend to privilege the execution of single, very large tasks. On the biggest and busiest systems in the world, queue waiting time for each task can reach days while lack of elastic coordination and communication infrastructure makes distributing the execution of ensemble applications difficult. Further, access, submission and execution methods vary across HPC systems, alongside their policies and performance. HPC systems are increasingly displaying performance dynamism and fluctuations due to aggressive thermal management and throttling. Together, these factors make using HPC systems for adaptive and non-adaptive ensemble applications challenging.

Existing solutions to express and execute ensemble applications on HPC systems range from complex scripts and domain specific workflow systems to general purpose workflow systems. Scripts and domain specific workflow systems serve as point solutions often limited in functionality, usability and performance to the scope of the specific application and the HPC system. General purpose systems, on the other hand, requires retrofitting the ensemble applications using the tools and interfaces provided by the system which can be challenging, when feasible.

The goal of this research is to advance the state-of-the-art by simplifying the programmability of ensemble applications, abstracting complexity of their scalable, efficient and robust execution on HPC systems, and, most importantly, enabling the domain scientists to focus on the computational campaigns and algorithmic innovations that are of importance to their science domains. In this dissertation, we describe several science drivers that employ ensemble applications to address some of the most challenging scientific problems of our time. We address three main challenges of executing ensemble applications at scale on HPC resources: (i), we address application diversity and programmability by developing a generic programming model that treats an 'ensemble' as a first order concern and provides constructs specifically to express ensemble applications; (ii) we develop a software system, called Ensemble Toolkit, to provide scalable and robust execution of ensemble applications while abstracting the

user from the architecture and policies of HPC systems, and resource and execution management; and (iii) we propose and evaluate scheduling strategies to manage the effect of workload heterogeneity and resource dynamism on the time to execute ensemble applications on HPC systems. We discuss several achievements and results obtained in various scientific domains as a consequence of the research and development described in this dissertation.

Acknowledgements

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

High-performance computing (HPC) entails the use of advanced hardware and software capabilities on supercomputers to implement massively parallel computational techniques that solve complex problems through models, simulations and analyses. HPC has become widespread on all aspects of government, academia and industry [1, 2], touching multiple facets of oil and gas exploration, drug discovery, weather prediction and climate modeling, financial modeling, design and optimization of consumer product, advanced business analytics and several other applications [3]. HPC also plays essential roles in national security in communications, cryptography, signals processing, and weapons design and testing [4].

The status-quo process of research and development in these domains encompasses, at a high level, building a physical prototype, performing experiments, analyzing the results to validate, and verify and repeat. By introducing the capability to model and simulate with great accuracy, HPC has transformed the paradigm of research and development. In our research, we focus on using HPC systems for scientific applications in molecular science, climate science and earth science – where complex multivariate physical systems need to be modeled and simulated to understand the behavior of the various components of the system. Our solutions, however, are applicable and extensible to applications in other domains.

Traditionally, advances in scientific HPC have focused on the scale and performance of an application with a large but single task. Depending on the science domain, this task could be an independent simulation as in seismology [5], data analysis processes as in climate science [6], or machine learning models as in molecular dynamics [7, 8] that uses a combination of compute and data resources on HPC systems. "Task", here, is

used to represent a stand-alone process that has well defined executable, inputs, outputs, termination criteria, and dedicated resources.

Moore's Law [9] describes the technology trend that transistor density on processors doubles every two years. Increase in density is accompanied with decrease in the size of the transistors. Dennard scaling behavior describes that the clock frequency of a transistor can be increased with decreasing the size of the transistor [10]. Moore's law coupled with Dennard scaling resulted in exponential performance increases, and along with micro-architecture, architecture, and compiler advances contributed to maintain the focus of scientific applications on the performance of a single task [11].

After close to 20 years of exponential growth, around 2005, this growth in computational power ceased [11]. Two specific factors, that depended on the size of the transistor, limited this growth [12, 13]: (i) the voltage threshold to detect on/off signals could not be reduced; and (ii) the thickness of the gate insulators could not be reduced. Reducing the voltage threshold or insulator thickness would increase the leakage current. Increasing the clock frequencies under such constraints would have led to unsustainable levels of power and heat dissipation.

Computational capacity has since then increased in the number of processors on a single chip and not in the processing power of a single processor [11]. HPC systems, offering large computational capacity, consist of a large number of nodes each consisting of several multi-core processors (CPUs) [14, 15] and specialized accelerators [16, 17]. Consequently, applications employed algorithms that utilized this new dimension of computational growth by executing multiple tasks concurrently and using their collective results to solve their scientific problems faster.

Applications where the collective outcome of a set of tasks is of importance are called ensemble applications, where "ensemble" refers to the set of tasks whose collective outcome is to be evaluated. Individual tasks within the set might be coupled or uncoupled. When coupled, tasks might have global (synchronous) or local (asynchronous) exchanges, and regular or irregular communication. This is in contrast to traditional parameter sweeps, or high-throughput computing (HTC) applications, where tasks are typically identical, uncoupled, idempotent and can be executed in any order.

Many scientific applications in the field of molecular sciences, computational biology [18, 19, 20, 21, 22], seismology [5, 23], weather forecasting [24, 25, 26, 27, 28], bio-informatics [29] are increasingly reliant on the ability to run ensemble-based methods to make scientific progress. This is true for applications that are both net producers of data, as well as aggregate consumers of data. The number and type of applications that can be formulated as ensembles is vast and spans many scientific domains. Some scientific problems that have traditionally been expressed as a single computational task must be reformulated using ensembles so as to overcome limitations of single task execution [30]. For example, in bio-molecular sciences, due to the end of Dennard scaling, and thus limited strong scaling of individual molecular simulations, there has been a shift from running single long running tasks towards multiple shorter running tasks, as evidenced by a proliferation of ensemble-based algorithms [21, 22].

1.1 Challenges

The execution of ensemble applications on HPC systems presents three main challenges: (1) encoding and decomposing scientific problems into algorithms that are amenable to a distributed and coordinated solution; (2) sizing, acquiring, and managing resources for the execution; and (3) managing the execution of the ensembles of the application on the acquired resources. Encoding scientific problems into ensemble applications requires describing tasks with heterogeneous properties, specifying if and how tasks are grouped into ensembles, and specifying any dependencies between tasks and ensembles. It is a challenge to provide an generic, domain-independent interface that enables such a description of ensemble applications.

Sizing resources for ensemble applications depends on calculating the resources needed by each task and those needed by the set of tasks that can be executed concurrently. The challenge of sizing resource increases in complexity when the number of tasks to be executed concurrently is greater than the number of available resources on the HPC system. Acquisition and management of resources requires knowledge of the queuing system used, system policies enforced and software availabilities on the HPC system. The challenge increases manifold as each HPC system differs from another due to

combination of these factors.

Managing the execution of the ensembles requires knowledge of available task launching methods specific to the HPC system, system architecture and policies, and often is bound to the software packages available on the system. Again, the challenge is increased as a consequence of heterogeneity across HPC systems.

Often, there is a friction between the resource requirements of an ensemble and the traditional resource management capabilities of HPC systems. Each task of the ensemble has to be queued onto an HPC system, incurring into a long queue waiting time that adds up to the total time to completion of the ensemble. Alternatively, ensembles of tasks may be executed using multi-processing libraries such as MPI [31]. This results in three drawbacks: (i) users require knowledge of both the deployment and usage of MPI; (ii) application robustness depends on the fault-tolerance of MPI; and (iii) mismatch of objectives as MPI is suitable to execute tasks that are homogeneous and have no interdependencies.

Finally, the distributed execution of an ensemble and the adaptive requirements of the application require tailored coordination and communication infrastructure and protocols, not made readily available to the user via the software and environment provisioned on HPC systems. These factors make using HPC resources for ensemble applications, adaptive and non-adaptive, very challenging, when not unfeasible.

In response to these challenges and requirements, the growing importance of ensemble-based applications in scientific computing on HPC systems, and the absence of middleware providing scalable, extensible and general solutions, we have designed and implemented the Ensemble Toolkit (EnTK). EnTK promotes ensembles to a high-level programming abstraction, providing specific interfaces and execution models for ensemble-based applications. EnTK is engineered for scale and a diversity of computing platforms and runtime systems, and it is agnostic of the size, type and coupling of the tasks comprising the ensemble.

1.2 Research Contributions

The goal of this research is to advance the state-of-the-art by simplifying the programmability of ensemble applications, abstracting complexity of their scalable and robust execution on HPC systems, and, most importantly, enabling the domain scientists to focus on the computational campaigns and algorithmic innovations that are of importance to their science domains.

Firstly, we describe the design, architecture and implementation of EnTK, an ensemble execution system that provides a programming model specific to the encoding of ensemble applications. We characterize its overheads, analyze its weak and strong scaling on a leadership-class HPC systems and compare the performance against requirements from various science drivers. We show that EnTK can support diverse types of ensemble applications, at scale and on several HPC systems, introducing acceptable overheads. We describe the science enabled by this work in various domains.

Secondly, we identify the types of adaptivity common in ensemble applications and enhance EnTK with adaptive capabilities. We characterize the cost of supporting adaptive capabilities in EnTK and show that it introduces acceptable overheads. We implement and execute novel adaptive applications at scale, validate the scientific results obtained from these applications and describe other novel applications developed due to EnTK's adaptive capabilities. As such, we propose EnTK as an important addition to the suite of tools in support of production-grade scientific computing.

Lastly, we discuss the effect of heterogeneity and dynamism in workloads derived from ensemble applications and resources on HPC systems on the total time to execute (TTX) the application. We perform theoretical and empirical evaluations with different levels of heterogeneity and dynamism to study their effects on TTX. We propose a late-binding strategy that can be used to manage their effects and provide up to $0.1x - 100x$ reduction in TTX compared to other strategies.

EnTK exemplifies the building blocks approach for the design, development and integration of middleware [32]. This approach advocates a sustainable ecosystem of software components from which tailored workflow systems can be composed, as

opposed to having to fit workflows to pre-existing frameworks. The building blocks approach overcomes the limited flexibility of monolithic workflow systems by enabling composability and extensibility, and thereby supporting the wide range of workflow requirements. As circumstantial evidence, EnTK has been used to develop several diverse domain-specific workflow systems [32].

1.3 Dissertation Organization

The rest of the dissertation is divided into five chapters. In Chapter 2, we present ensemble applications in greater detail and describe the initial science drivers that motivated this work. The challenges in executing these ensemble applications on HPC systems along with the current general-purpose and special-purpose solutions available to the domain scientists.

In Chapter 3, we describe the requirements, design and implementation of EnTK. We characterize the performance overhead and scalability of EnTK and detail the domain science enabled by EnTK in the various science drivers.

In Chapter 4, we present adaptive ensemble applications and their significance in advancing the status quo in various science domains. We describe the science drivers that employ adaptive ensemble applications and the challenges in executing them on HPC systems. We conclude this chapter with a description of the current solutions available to the domain scientists to address these challenges.

In Chapter 5, we identify the types of adaptivity that is common in adaptive ensemble applications and the challenges in incorporating in execution systems. We discuss the enhancements made to EnTK to support adaptivity. We, then, characterize the overhead of enabling adaptivity in EnTK, validate the implementation of two science drivers using these capabilities. We then discuss the science enabled by these capabilities in various domains.

In Chapter 6, we discuss the heterogeneity and dynamism prevalent in workloads derived from ensemble applications and resources on HPC systems. We perform theoretical and empirical evaluations to derive insight on their effects on TTX and propose

the late-binding strategy to manage the same.

Chapter 2

Ensemble Applications

A large set of scientific applications involve running multiple simulations that try to recreate real-world events. These simulations are numerical computations with constraints and boundaries that realize the physical laws governing the real-world events. The results of these simulations are analyzed to test their feasibility against real-world events.

The goals of these applications depend on the scientific domain in which they are pursued. In molecular science [30, 33, 21, 22], they are used to understand the properties of simple and complex biomolecules. In drug discovery [34], they are used to study the interaction between drugs and maligned proteins to identify feasible drug candidates. As such, these computational approaches complement the expensive nature of wet lab experiments in these science domains. Similar applications can be found in bio-informatics [29], uncertainty quantification [35, 28] and several other scientific domains.

Another type of scientific application consists of large number of data processing tasks that analyze data from sensors and devices that measure real-world events. In climate science [24, 25, 26, 36, 27], they are used to construct accurate models for short-term weather and long-term climatic predictions. In several seismologic applications [23, 37, 38], applications use both simulations and data processing tasks to analyze seismic wave data to understand their propagation around the planet, understand fault lines, and model the crust of the earth.

Such applications where the collective outcome of a set of tasks is of importance are called ensemble applications, where "ensemble" refers to the multiple tasks of the application whose collective outcome is to be evaluated. Task, in this dissertation,

is a term for a stand-alone process that has well defined input, output, termination criteria, and dedicated resources. Specifically, a task is used to represent an independent simulation or data analysis process, running on one or more cores of a HPC system.

Ensemble applications, as exemplified above, provide as novel solutions to scientific problems that could not be solved using existing approaches and improve upon existing approaches reaching unprecedented scales to solve bigger problems faster and better. Understanding the patterns and requirements of ensemble applications is imperative to, both, support the expression and execution of existing ensemble applications and to enable the exploration of novel ensemble applications. In this chapter, we discuss five ensemble applications, some pre-existing and some novel approaches, that serve as science drivers informing us about the requirements and challenges of domain scientists employing these applications.

2.1 Science Drivers

Ensemble applications span multiple science domains as described above and involve two computational layers: at the lower level each task executes a domain specific program; at the higher level, an algorithm codifies the coordination and communication between different ensembles and within an ensemble. The latter specifies set of task dependencies and is referred as a **workflow** or **task-graph (TG)**.

There are five ensemble applications from the molecular science, earth science, climate science and drug discovery domains that serve as the science drivers informing us about the requirements and challenges of domain scientists and motivating the research and development described in this dissertation. We describe the workflow of these drivers, overview of the scientific problem addressed and, wherever possible, the scales at which these ensemble applications are to be executed.

2.1.1 Extensible Tools for Advanced Sampling and analYsis (ExTASY)

Molecular dynamics (MD) simulations with all-atom force-fields allow simulating protein folding and protein kinetics with good accuracy. Reaching biologically relevant scales of

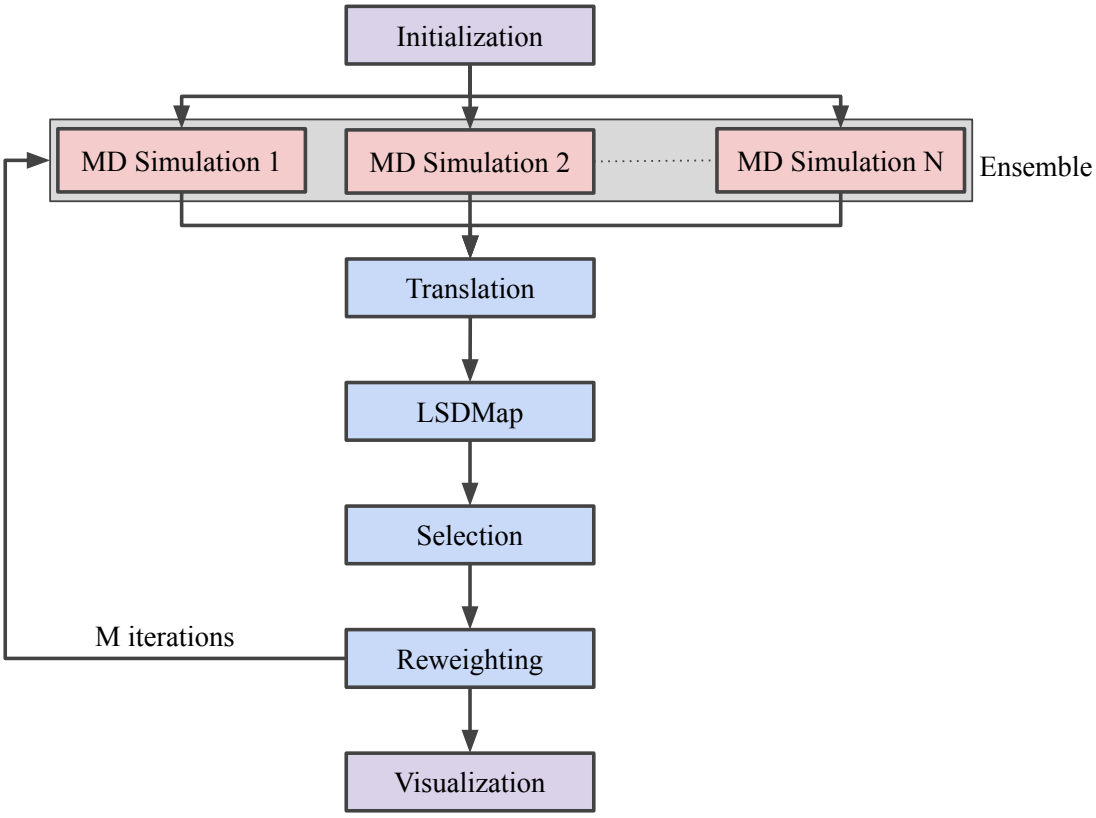

Figure 2.1: ExTASY: Iterative simulation analysis loops for the Diffusion-Map-directed-MD technique

protein folding or drug binding is limited by the required large computational resources and long simulation times. One method of reducing both the computational resources and the simulation times is advanced sampling [39, 20, 40, 19, 41]. Advanced sampling iteratively analyzes the already explored relatively short MD simulations and iteratively adds new short MD simulations from different restart conformations in an informed way to efficiently reach a goal such as crossing rare transition barriers, folding a protein or recover the protein dynamics of a protein. The exact strategy describing where to restart new MD simulations determines the success of this approach, and several different methods have been proposed and investigated [18, 19, 42, 43, 40].

The ExTASY [44] project requires implementation of two advanced sampling algorithms: Diffusion-Map-directed-MD (DM-d-MD) [42] and CoCo-MD [43]. The DM-d-MD technique (Fig. 2.1) improves the computational time spent by choosing which replicas of the protein are used to run the simulations. When replicas are too close to each

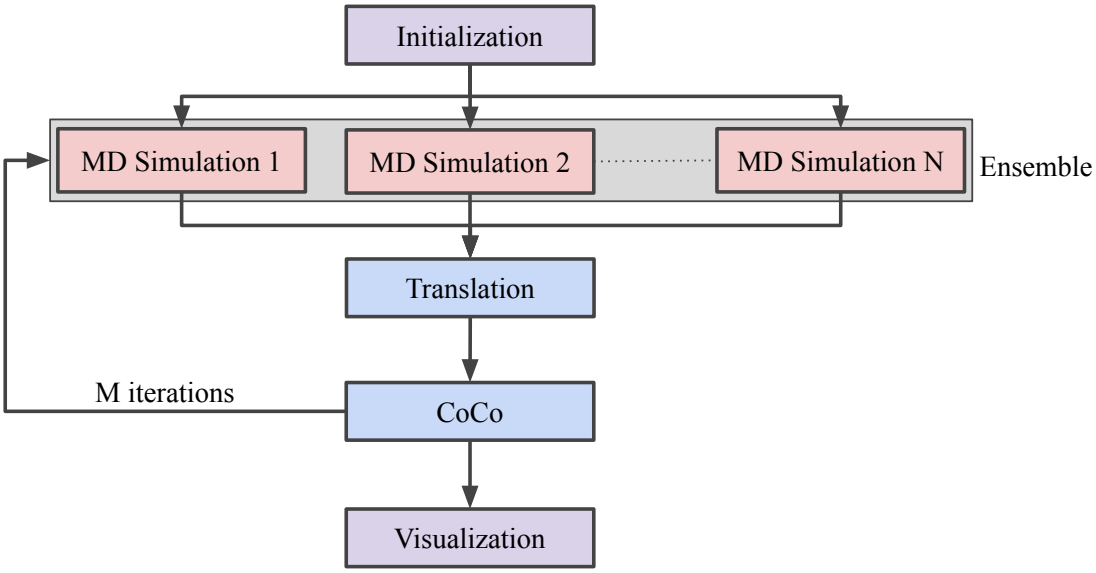

Figure 2.2: ExTASY: Iterative simulation analysis loops for the CoCo-MD technique

other, the MD trajectories will be similar. The information gain from simulating MD with close replicas is small. Part of the replicas which are too close to each other are deleted. To hold the total number of replicas constant, replicas which are too far apart from each other are duplicated. In DM-d-MD, a non-linear dimensionality reduction technique, the Locally Scaled Diffusion Map (LSDMap) [45] is used to calculate the distance between different replicas. The deletion or duplication of replicas would destroy the correct sampling of the protein. By changing the weights of individual replicas, the correct sampling of the protein is obtained.

The CoCo (Complementary Coordinates) technique (Fig. 2.2) was designed as a method to enhance the diversity of ensembles of molecular structures of the type produced by NMR structure determination. The method involves the use of PCA [46, 47, 48] in Cartesian space to map the distribution of the ensembles in a low (typically 2-4) dimensional space, and then the identification of unsampled regions. CoCo generates new conformations for the molecule that would correspond to these regions. These new conformations become the start points for the next iteration of MD simulations. The method is agglomerative - all simulation data generated, current and historic, is used for each analysis.

Both the techniques require O(10)-O(100) iterations of the simulation-analysis loop

with O(100)-O(1000) simulations to study sufficiently the physical events and interactions of medium to large physical systems.

2.1.2 Seismic Tomography

Inversion of full-waveform, wide-bandwidth seismic data [23] is one of the most powerful tomographic technique to study the Earth's interior. Scaling this technique is challenging, mostly because of the amount of computational resources and human labor it needs. These challenges require a more automated approach to the management and execution of the workflow.

Figure 2.3 shows a high level view of the workflow to perform seismic tomography. Seismic data (i.e., seismograms) of physical quantities, like displacement, velocity, acceleration or pressure are recorded as a time series. The goal of the science driver is to iteratively minimize differences between observed and corresponding synthetic data through a pre-defined misfit function. As the adjoint-based optimization procedure is carried on and the data misfit decreases, the model gets closer to reality.

The science driver requires the assimilation of data from about 6,000 earthquakes. Forward and adjoint simulations are the most computationally expensive parts of the workflow, each running on 384 GPUs for a total of 10 million core-hours per iteration. Data processing is relatively computationally inexpensive, utilizing about 48,000 core-hours in each iteration. Post-processing takes about 10,000 core-hours while optimization takes about 1 million core-hours. Currently, each part of the workflow relies on a Python-based proto-workflow management system, Seisflow [49]. However, scaling to higher resolutions and assimilating data from all 6,000 earthquakes requires more automation to ensure reliability, minimize errors at the user level and lower the overall time to solution.

2.1.3 High Resolution Meteorological Probabilistic Forecasts

Analog Ensemble (AnEn) [6] methodology has been used to generate high-resolution, probabilistic forecasts for environmental variables like temperature or cloud cover. Relationships between current and past forecasts from the Weather Research and

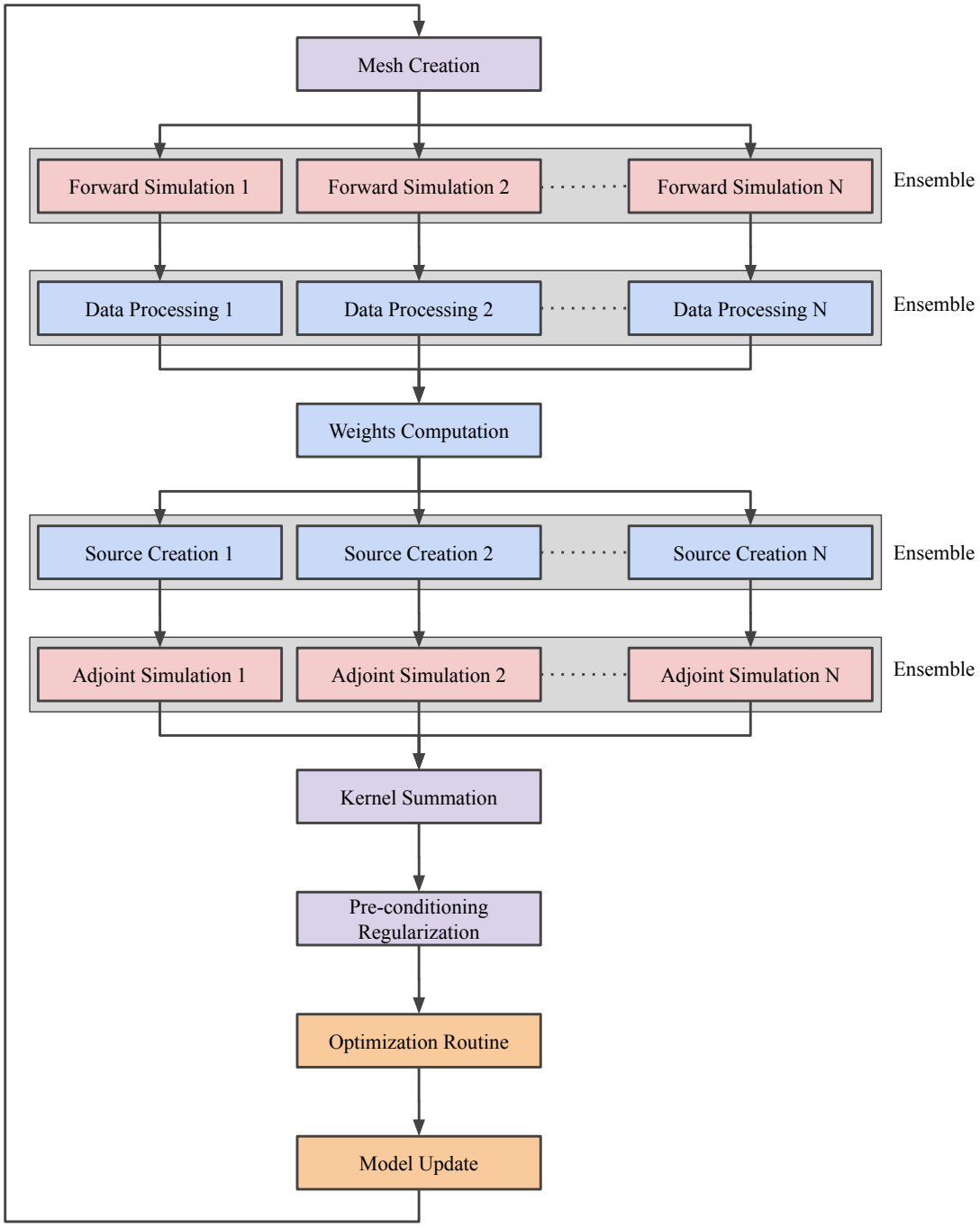

Figure 2.3: Iterative workflow for seismic inversion tomography

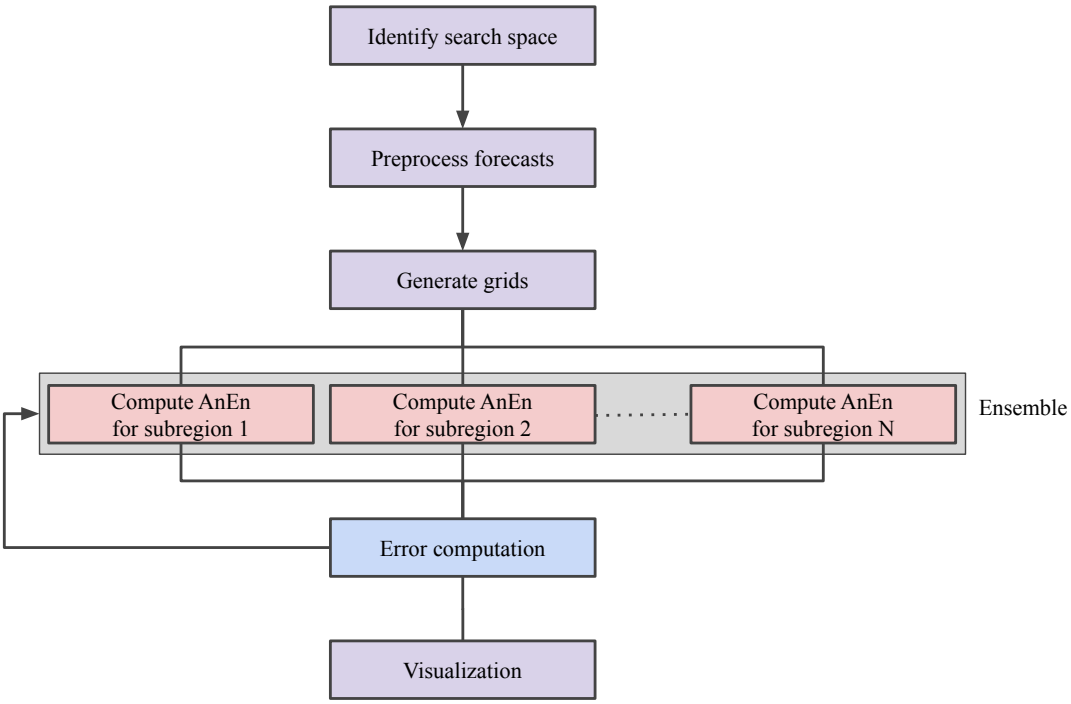

Figure 2.4: Iterative workflow for the adaptive unstructured analog algorithm

Forecast model (WRF) data to generate an analog ensemble for a given time and location. Our implementation finds the most similar historical forecasts, based on a similarity metric. The observations associated with the most similar past forecasts are used as analogs.

The goal of the science driver is to implement a dynamic iterative search process, Adaptive Unstructured Analog (AUA) algorithm, which generates analogs at specific geographical locations, and interpolates the analogs using an unstructured grid. In this way, we avoid computing analogs at every available location, noting that for some output variables, such as temperature, the highest resolution of the analogs is required only at specific regions, where drastic gradient changes occur.

Figure 2.4 shows the workflow of the AUA algorithm. The initialization step specifies the search space and the test space, and sets up starting parameters for the AnEn. The preprocessing step generates preparatory data for the subsequent steps. The largest amount of computation occurs in the iterative computation step where analogs are computed and aggregated multiple times until the available resources are exhausted, or

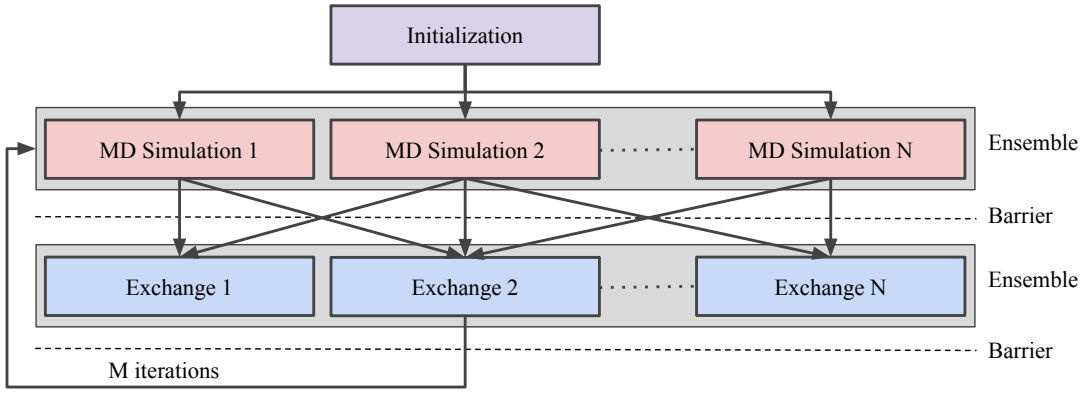

Figure 2.5: Workflow for the synchronous replica exchange algorithm

the prediction error is below a given threshold. The visualization step interpolates the analogs to generate the forecast solution.

2.1.4 Replica Exchange

Replica Exchange is a class of methods [50] that is popularly used to enhance sampling in molecular simulations. Although these methods were introduced for Monte Carlo methods, their use in molecular science has grown rapidly. Replica Exchange based MD is used in a range of scientific disciplines including chemistry, physics, biology and materials science.

Many MD programs [51, 52, 53] offer the capability to perform replica exchange. These implementations demonstrate high performance but have several limitations due to the tight coupling of the replica exchange method and the MD program implementation. These programs are highly optimized code that often require hundreds of human years for development. As such the tight coupling results in duplication of functionality between different MD programs, localization of novel capabilities to specific MD programs and limits the research in new methods as these researchers are not always fully prepared to handle the complexity of the MD programs. Thus, the tight integration introduces a barrier to methodological advances and extensibility.

The goal of this science driver is to separate the MD simulations and exchange programs from the implementation of the replica exchange method. As shown in Figure 2.5 there are phases: one phase is comprised of MD simulations of N different replicas

of the original system, where each replica has different thermodynamic configuration. The second phase involves exchanges of the thermodynamic configurations between replicas using Metropolis-like acceptance criterion. Exchanges of temperature [33], hamiltonians [54] and pH values [55] are to be supported.

2.1.5 High Throughput Binding Affinity Calculation

The US currently spends up to $37 billion on the development of drugs to treat cancer [56, 57]. Targeted kinase inhibitors (TKIs) have become increasingly prominent in the treatment of cancer. TKIs have been developed to selectively inhibit kinases involved in the signaling pathways that control growth and proliferation, which often become dys-regulated in cancers. This targeting of specific cancers reduces the risk of damage to healthy cells and increases treatment success.

Unfortunately, the development of resistance to these drugs limits the amount of time that patients can derive benefits from their treatment. Many cancer centers have begun deep sequencing of patient tumors to identify the genetic alterations driving individual cancers. The ultimate goal is to make individualized therapeutic decisions based upon these data via precision cancer therapy.

The goal of the High Throughput Binding Affinity Calculation (HT-BAC) science driver is to implement the ESMACS [34] (Fig. 2.6) and TIES [58] (Fig. 2.7) protocols at scale on HPC systems. They are two free energy calculation protocols that implement absolute and relative methods, respectively. Absolute free energy methods calculate the binding affinity of a single drug molecule to a protein, while relative methods calculate the difference in binding affinity between two drug molecules. With 100s of drug candidates to analyze, multiple instance of ESMACS and TIES are to be executed concurrently in order to determine the best drug candidate for the patient.

In the ESMACS protocol, there are 25 instances of a 3-step sequence, termed as a replica, that needs to be executed. The minimization and equilibration are performed to bring the starting configuration to a stead-state value after which the production-scale simulation can be performed. Each replica is followed by an analysis to determine the accuracy of the standard free energy calculation made by the simulation.

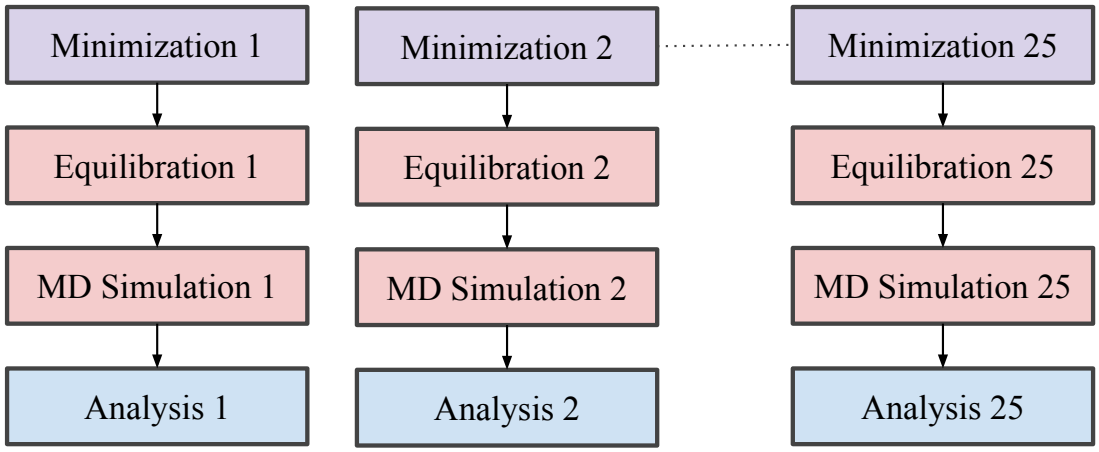

Figure 2.6: Workflow for the ESMACS HT-BAC Protocol

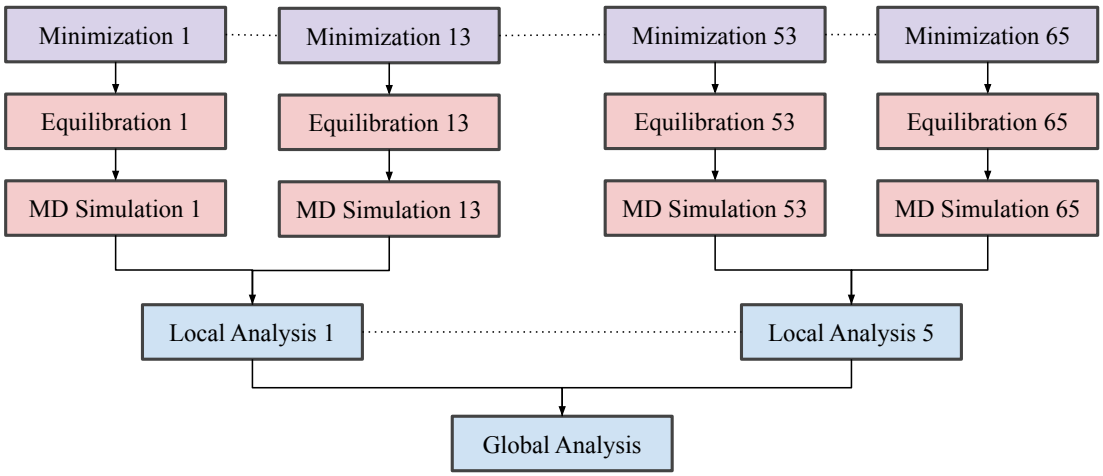

Figure 2.7: Workflow for the TIES HT-BAC Protocol

In the TIES protocol, there are 65 replicas, similar to ESMACS, but are followed by 5 local analyses each operating over 13 replicas. The 5 local analyses are followed by a global analysis to integrate the thermodynamic calculations across all the replicas. As seen in Figures 2.6 and 2.7, in this science driver, we have an ensemble of concurrent replicas in contrast to the previous science drivers.

2.2 Current Solutions: Tools, Systems, and Services

ensemble applications span several scientific domains and vary in their communication and coordination patterns as exemplified in our science drivers above. As described in §1, there are challenges in encoding ensemble applications, sizing, acquiring and managing

resources and managing the coordinated execution of ensembles of the application. Several current solutions that address some, but, not all the challenges.

Executing ensemble applications on HPC systems requires knowledge of resource, data and execution management, specific to the HPC system. Several "middleware" [59] frameworks have been developed to abstract execution details and enable execution of ensemble applications. Software development kits such as gSOAP [60] enable web services for HPC applications. Ninf-G [61] and OmniRPC [62] provide client/server-based frameworks for distributed programming. These solutions provide methods to launch application tasks on remote machines but leave the details of task scheduling, resource and data management, and fault tolerance to the user.

Hadoop and its ecosystem have been ported to HPC systems [63, 64, 65], enabling the use of the MapReduce programming model. In this model, tasks are implemented as functions and the data movement is internally handled. While some ensemble applications are data-flow oriented and thus amenable to be implemented with MapReduce, most of the scientific applications require more flexible and coarse-grained notion of tasks, where a task is a program which may implement different programming models such as MPI, MapReduce, parameter sweep, etc.

Feature-rich workflow systems such as Kepler [66], Swift [67], and Pegasus [68] provide end-to-end capabilities such as resource and execution management, fault tolerance, monitoring and provenance. However, encoding applications using these systems requires acquiring specific knowledge, including learning new languages and paradigms. Using these systems often requires strong collaboration between the domain scientists and the system developers, even in the presence of informative documentation, as adapting these systems to user requirements is non-trivial due to their feature richness and end-to-end design.

Light-weight workflow systems such as Ruffus [69], COSMOS [70], and GXP Make [71] limit the capabilities and prioritize interface simplicity. Whereas, special-purpose workflow systems such as Galaxy [72], Taverna [73], BioPipe [74], and Copernicus [75] implement specific algorithms and focus on providing tailored interfaces to domain scientists. These systems, similar to several others, have been developed from scratch

due to not being able to use existing solutions [76].

One of the limitations observed in existing solutions is that functional and performance enhancements are localized to one framework and cannot be easily ported to other systems. This results in either retrofitting applications in existing solutions or developing new solutions from scratch. In the next chapter, we discuss the building blocks approach which motivates composability and reusability of independent systems to build workflows and workflow systems. We discuss EnTK which enables composition of application using user facing constructs and builds upon and reuses existing execution systems. Similar approaches have been explored by Tigres [77] for application composability but it does not reuse existing systems.

2.3 Summary

In this chapter, we described the different scenarios in which ensemble applications are employed and their significance and impact in their respective science domains. We discussed five science drivers from molecular science, earth science, climate science and drug discovery, that informed us about the requirements and challenges of domain scientists employing these applications. Some of these science drivers are novel applications and the others are improvements to existing applications to scale to bigger physical problems. We reviewed some of the existing solutions to enable the expression and execution of ensemble applications on HPC systems and their limitations. In the next chapter, we derive functional, performance and usability requirements from these science drivers in the next chapter and develop our solutions to address them.

Chapter 3

Ensemble Toolkit

There is a growing significance of ensemble applications in scientific computing on HPC systems, some of which have been discussed in §2.1. Ensemble applications from different science domains differ in their communication and coordination patterns, execution configurations, scale and performance requirements. Due to an absence of generic, portable, and scalable middleware tools to enable the expression and execution of such applications, domain scientists are forced to either develop customized tools or retrofit their application on to general purpose solutions.

In response to these missing capabilities, we developed a new workflow management system, called on Ensemble Toolkit (EnTK), based on requirements derived from the science drivers discussed in §2.1. The design and implementation of EnTK are iterative and driven by science drivers: Users and developers collaborate to elicit requirements, prototype rapidly, and validate the prototype against the requirements. This approach allows the list of requirements to grow based on either new aspects of existing science drivers or due to new science drivers. EnTK is loosely specified in UML, Jenkins and Travis are used for continuous integration and automated testing. Documentation and code are managed and made available via a GitHub repository [78].

EnTK is developed following the principles of the building blocks approach for workflow systems [32] and uses an existing task executing system RADICAL Pilot [79]. In this chapter, we first describe the requirements arising from our science drivers. We describe the building blocks approach, design and implementation of EnTK and how it aligns with the building blocks approach. We characterize the overheads and scalability of EnTK and conclude with the domain science that has been enabled by our work.

3.1 Requirements

As described in §1 and §2, the space of ensemble applications is vast, and thus there is a need for simple and uniform abstractions while avoiding single-point solutions. We elicited requirements about scientific ensemble applications, computing infrastructures (CIs), scale, fault-tolerance, and usability from our science drivers. We classify the requirements into three types: (1) functionality, (2) performance and (3) usability. We describe each of the requirements in detail.

3.1.1 Functional requirements

3.1.1.1 Application Portability

Domain scientists generally have access to multiple heterogeneous CIs. Scientists require that their scientific applications be portable to these different CIs. There can be several reasons for this requirement such as scale of the problem, computational power of a CI, availability of special accelerators, estimated wait times, or funding models which mandate usage of specific CIs. Several current approaches require the user to re-encode their application based on the policies and capacity of the CI, posing has a high overhead for the scientists.

3.1.1.2 Application Diversity

Ensemble applications from diverse scientific domains such as molecular science, climate science, earth science, and polar science may have different communication and coordination patterns, and different number, size and properties of the ensembles, but they share the common need to execute ensembles of tasks. Currently, these applications are encoded either with customized scripts and workflow systems or retrofitted into general purpose systems. There is a need for tools that enable expression of ensembles as first order entities and take advantage of the commonality between different ensemble applications.

3.1.1.3 Fault Tolerance

Three types of faults may occur during the execution of ensemble applications on HPC systems: (1) resource failure on the HPC system; (2) failure of the middleware being used; and (3) runtime failures of the programs being executed. Consequences of such failures include loss in computational time, loss of data obtained during execution, and incorrect computations during execution. Domain scientists require guards from failures in the resource and middleware. In many cases, scientists know how to recover from program failures but do not have the software hooks to invoke them upon failures. The recovery may range from simply re-executing the program or changing configurations or input for the program.

3.1.2 Performance requirements

3.1.2.1 Scalability

The motivating science drivers require execution of O(1000) tasks, with 4096 being the maximum. Depending on the requirements of the scientist, these tasks may be required to run in parallel or may be distributed temporarily into smaller *bags* of tasks. Middlewares need to be capable of supporting both execution modes and, ideally, show linear scaling behavior with the number of tasks.

3.1.2.2 Performance Invariance

Domain specific middleware implement specific algorithms of a particular domain and can neither share their implementations with other solutions nor can they adopt other optimized solutions to improve their performance. In the case of general purpose middleware that require retrofitting of ensemble applications, they result in implementations that are rigid to optimize and introduce overheads due to the retrofitting. Performance of the middleware is required to remain invariant to the domain in which it is being used.

3.1.3 Usability requirements

3.1.3.1 Application Creation

Domain scientists require to be able to quickly encode their algorithms into an application using the constructs provided by the middleware. This requires the constructs to be simple and close to how the domains scientists visualize their ensemble applications.

3.1.3.2 Resource and Execution Management

Domain scientists are interested primarily in their scientific algorithms and the results that they generate. The complexity of acquiring and managing resources on HPC systems, and managing execution of tasks on those acquired resources needs to be abstracted from the domain scientists.

3.2 Design

EnTK is designed to address the above requirements collected from various science drivers. In this section, we discuss a programming model formulated for ensemble applications, constructs provided by EnTK to compose ensemble applications, software architecture of EnTK, resource and execution management performed by EnTK reusing existing systems, and how fault-tolerance is incorporated in EnTK.

3.2.1 Building Blocks approach

As mentioned in §2.2, we suggest the need for a sustainable ecosystem for both existing and new software tools using which tailored workflow systems can be composed. This enables the support of agile development and composition of workflow systems that can be responsive to the wide range of application requirements while leveraging the rich ecosystem of existing software capabilities.

The building blocks approach [32] builds upon component based-software engineering and service-oriented architecture concepts, to suggest modularity at the level of stand-alone software systems and not at the level of modules or routines of a single system alone. The approach suggests that software systems that are self-sufficient, interoperable,

composable, and extensible motivate an integrative approach to develop tailored workflow systems using standardized interfaces. This reduces the expertise required and cost to develop tailored solutions while leveraging and promoting a rich ecosystem of software systems.

EnTK follows the building blocks approach by offering generic constructs for the user to compose end-user applications or domain-specific workflow systems while reusing existing systems for resource and execution management.

3.2.2 Programming Model

The workflows of the science drivers discussed in § 2.1 have varying dimensions and dependencies but what is commonly observed in the presence of ensembles of tasks and sequences of ensembles and tasks. We propose a programming model that promotes ensembles as a first-order concern where ensemble applications are expressed in terms of concurrency and sequentiality of tasks.

We propose three user-facing constructs to enable encoding of applications based on this programming model: Task, Stage, and Pipeline. We called the programming model as the PST model named based on the user constructs. The constructs are defined as the following:

- **Task:** an abstraction of a computational task that contains information regarding an executable, its software environment and its data dependences.
- **Stage:** a set of tasks without mutual dependences and that can be executed concurrently.
- **Pipeline:** a list of stages where any stage i can be executed only after stage $i-1$ has been executed.

Figure 3.1 shows an application described with pipelines, stages, and tasks (PST). The application consists of a set of pipelines, where each pipeline is a sequence of stages, and each stage is a set of tasks. The set of pipelines can execute concurrently, all the

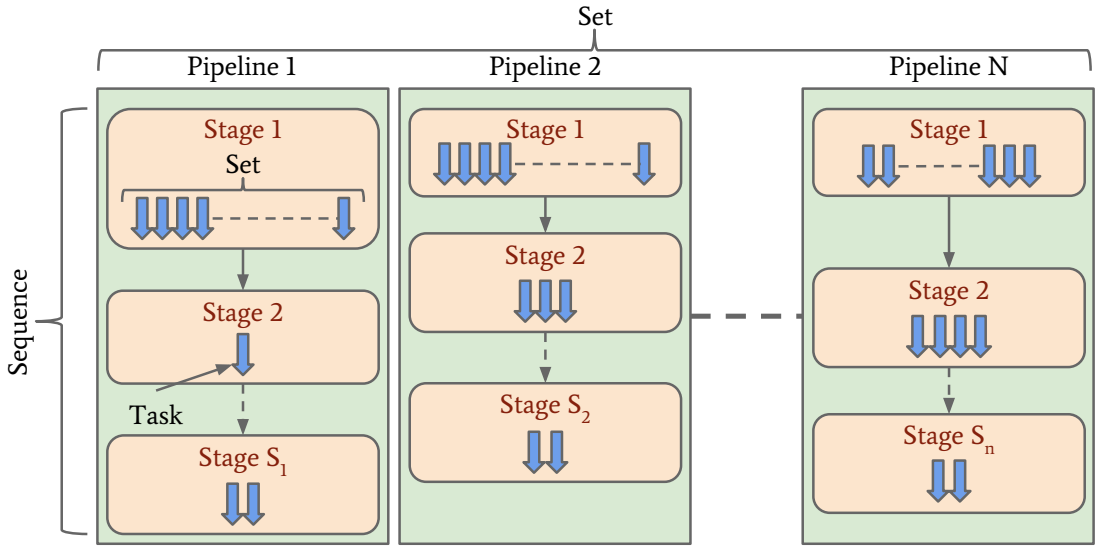

Figure 3.1: Diagrammatic representation of an application consisting of a set of pipelines with varying number of stages and tasks.

stages within each pipeline execute sequentially, and all the tasks of a stage can execute concurrently.

In this way, ensemble applications are described in terms of the concurrency and sequentiality of tasks, without requiring the explicit specification of task dependencies. Note that PST descriptions can be extended to account for dependencies among groups of pipelines in terms of sequences of sets of pipelines.

3.2.3 Architecture

EnTK, as middleware, sits between the user and the HPC system, abstracting resource management and execution management from the user. The user is expected to describe *what*, *when*, and *where* an application needs to be executed, all the complexities of *how* it is executed is abstracted from the user.

Fig. 3.2 shows the components (purple) and sub-components (green) of EnTK, organized in three layers: API, Workflow Management, and Workload Management. EnTK is designed to interface with different runtime systems (RTS) to enable resource acquisition and task scheduling on these resources. The choice of a specific RTS may be driven by application requirements or the configurations and capabilities of the HPC system. EnTK is designed to have software hooks to integrate with different RTSs.

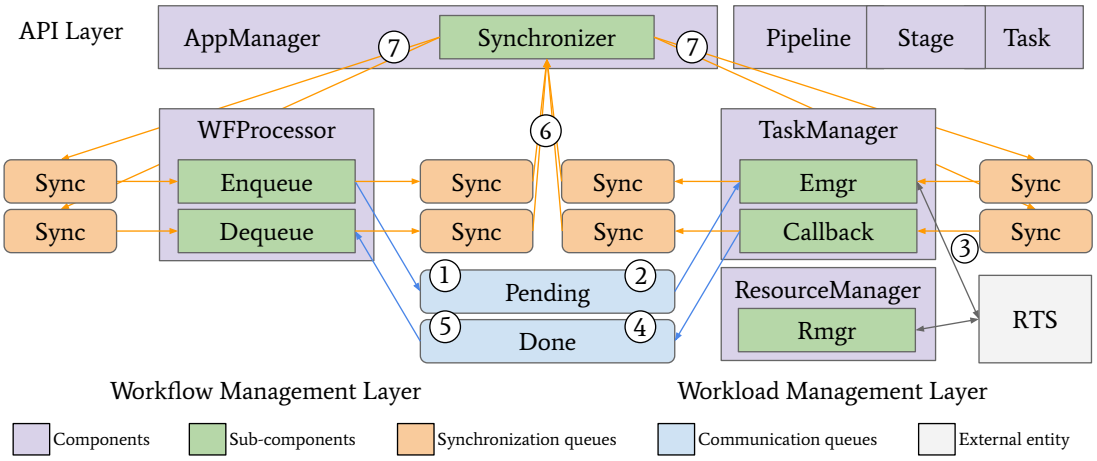

Figure 3.2: EnTK architecture and execution model. Components' (purple) sub-components (green) use queues (blue and orange) to communicate and coordinate the execution of an application via a chosen RTS (gray).

The API layer provides constructs to codify and submit their applications in the form of PST descriptions. The Workflow Management layer retrieves information from the user about available HPC systems and the application, initializes all components of EnTK, identifies executable tasks of the application based on their organization in terms of pipelines and stages, and holds the global state of the application during execution. The Workload Management layer acquires and releases computational resources via a RTS, and executes the provided set of tasks on the acquires resources

The Workflow Management layer has two components: AppManager and WFProcessor. AppManager is the connector between the user and the components of EnTK. It receives the PST description of the application and the description of the resource to use. It passes the PST description to the WFProcessor and that of the resource to the Workload Management layer. It also uses the Synchronizer subcomponent to update the state of the application at runtime and thus keeps the global state of the application. WFProcessor uses the Enqueue subcomponent to enqueue executable tasks to the Workload Management layer and the Dequeue subcomponent to dequeue completed tasks from the Workload Management layer.

The Workload Management layer has two components: ResourceManager and TaskManager. The ResourceManager interfaces with a RTS to acquire and release computational resources from the target HPC system. The TaskManager interfaces

with a RTS via the Emgr subcomponent to execute tasks provided by the Enqueue subcomponent. The Callback subcomponent pushes the completed tasks to the Dequeue subcomponent.

One of the benefits of this architecture is the isolation of the RTS into a stand-alone subsystem. This enables composability of EnTK with diverse RTS and, depending on capabilities, multiple types of HPC systems. Further, EnTK assumes the RTS to be a black box enabling fault-tolerance. When the RTS fails or becomes unresponsive, EnTK can tear it down and bring it back, loosing only those tasks that were in execution at the time of the RTS failure.

3.2.4 Execution Model

The components and sub-components of EnTK communicate and coordinate with each other via control messages for the execution of tasks. Users describe an application via the API, instantiate the AppManager component with information about the target HPC system and then pass the application description to AppManager for execution. AppManager holds these descriptions and, upon initialization, creates all the queues, spawns the Synchronizer, and instantiates the WFProcessor, TaskManager, and ResourceManager. WFProcessor and TaskManager instantiate their own sub-components.

Once the user provides the specification of both, the resource to be requested and the application to be executed, the AppManager first instantiates the ResourceManager and submits a resource request on the target HPC system. Once the resource is allocated for execution, all the other components and sub-components are instantiated.

Once EnTK is fully initialized, WFProcessor initiates the execution by processing a local copy of the application description obtained from the AppManager. Enqueue identifies tasks who either have no dependency or have all the dependencies satisfied and pushes these tasks to the Pending queue (Fig. 3.2, 1). The dependency here refers to the sequential order between stages of a Pipeline or between a sequence of pipelines. Emgr pulls tasks from the Pending queue (Fig. 3.2, 2) and executes them on the acquired resource using the RTS (Fig. 3.2, 3). Once tasks have completed execution, the RTS

notifies the Callback subcomponent which pushes these tasks to the Done queue (Fig. 3.2, 4). Dequeue pulls completed tasks (Fig. 3.2, 5) and tags them as done, failed or canceled, depending on the exit code of the task as reported by the RTS.

Throughout the execution of the application, tasks, stages and pipelines undergo multiple state transitions in both WFProcessor and TaskManager. Each component and subcomponent synchronizes these transitions with AppManager by pushing messages through dedicated queues (Fig. 3.2, 6). AppManager pulls these messages and updates the application states. AppManager then acknowledges the updates via dedicated queues (Fig. 3.2, 7). This synchronization of state updates ensures that AppManager is always up-to-date with any state change, making it the only stateful component of EnTK .

3.2.5 Failure Model

We consider four main sources of failure: EnTK components, RTS, HPC system, and program executed by the tasks. All state updates in EnTK are transactional and are synchronized with the AppManager. Hence any EnTK component that fails can be restarted at runtime. The failed component, upon restart, receives the last successful updates from the AppManager and resumes execution. Tasks that are executing during the time of failure may be lost, but will be resubmitted once all components are active. In case the ResourceManager fails, however, a new resource request would have to be submitted which will face additional wait times on the queues of the HPC system. EnTK is being developed to recover from full failures, where state information is synced to disk before a failure and can be used to resume execution from the last successful update.

Both the RTS and the program are considered black boxes. Partial failures of RTS sub-components at runtime are assumed to be handled locally, not by EnTK . Upon full failure of the RTS, EnTK assumes all the acquired resources and the tasks undergoing execution are lost. EnTK purges any components left over by the failed RTS, starts a new instance of the RTS, acquires a new set of resources, and resumes executing the application. Users can configure the number of times a RTS is restarted upon failure.

The program being executed by a task is assumed to handle all its failures and exceptions locally, including any check-pointing for recovery. EnTK relies on the exit

code raised by the program to determine the success or failure of the task. EnTK does not provide check-pointing capabilities but it can complement programs that perform check-pointing by restarting tasks that failed with the check-pointed data.

Failures in the HPC system may include failure of the computational nodes, filesystems, queueing systems, software environment, etc. Such failures are reported to EnTK indirectly, either as failed resource allocations or failed tasks. Resource allocations can be reacquired and tasks can be restarted, up to a certain number of times as configured by the user. Failures are reported to the user at runtime for live inspected and also logged for postmortem analysis.

3.3 Implementation

EnTK is implemented in Python and uses the RabbitMQ message queuing system [80] as the communication infrastructure and RADICAL-Pilot (RP) [79] as the RTS. All EnTK components and sub-components are implemented as processes and threads respectively. AppManager is the master process spawning all the other processes. Tasks, stages and pipelines are implemented as Python objects, transmitted, via transactions, among processes and threads using queues implemented using RabbitMQ. Synchronization of state updates with the AppManager is achieved by message-passing via these queues.

3.3.1 Communication infrastructure: RabbitMQ

EnTK relies on RabbitMQ to manage the creation of the communication infrastructure to transport the objects and messages among components and sub-components. RabbitMQ provides capabilities to increase durability and backup of messages in transit, persistence of queues, and a handshake policy such that messages lost in transit or during failure are automatically resent. Most importantly, it supports the requirement of managing at least $O(10^4)$ tasks concurrently.

RabbitMQ is a server-based system and requires to be installed before the execution of EnTK . This adds overheads but it also offers the following benefits: (1) producers and consumers do not need to be topology aware because they interact only with the

server; (2) messages are stored in the server and can be recovered upon failure of EnTK components; and (3) messages can be pushed and pulled asynchronously because data can be buffered by the server upon production.

Services such as CloudAMQP [81] and container systems such as Docker [82] dramatically reduce the user overhead in creating and configuring RabbitMQ.

3.3.2 Runtime system: RADICAL Pilot

Currently, EnTK uses RADICAL-Pilot (RP) as the RTS. RP is a runtime system designed to execute sets of tasks via pilots on HPC systems. A pilot is a container job that is submitted to the queueing system of the HPC system. It provides a multi-stage execution mechanism: Resources are acquired via a placeholder job and subsequently used to execute the application's tasks. When a pilot is submitted to a HPC system as a job, it waits in the HPC system's queue until the requested resources become available. Once available, the system's scheduler bootstraps the job on the its compute nodes. RP does not attempt to 'game' the scheduler: Once queued, the pilot is managed according to the system's policies.

RP is a distributed system with four major modules: PilotManager, UnitManager, Agent and DB (Fig. 3.3, purple boxes). PilotManager, UnitManager and Agent have multiple components (Fig. 3.3, yellow boxes), isolated into separate processes. Components are stateless and some of them can be instantiated concurrently to enable RP to manage multiple pilots and tasks at the same time. Concurrent components are coordinated via a dedicated communication mesh, scaling throughput and enabling tolerance to failing components.

Workloads and pilots are described via the RP API and passed to the RP runtime system (Fig. 3.3, 1). The PilotManager submits pilots as jobs (or virtual machines or containers) to one or more CIs via the SAGA API (Fig. 3.3, 2). The SAGA API implements an adapter for each supported type of CI, exposing uniform methods for job and data management. Once a pilot becomes active on a CI, it bootstraps the Agent module (Fig. 3.3, 3). The UnitManager schedules each task to an Agent (Fig. 3.3, 4) via a queue on a MongoDB instance. Each Agent pulls its tasks from the DB module

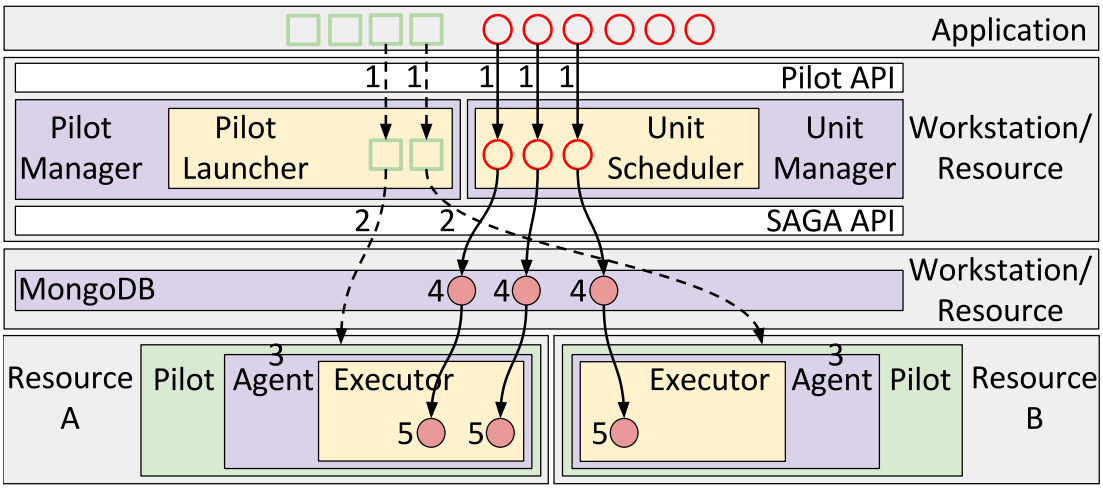

Figure 3.3: RADICAL-Pilot (RP) architecture and execution model. Gray: machines; green: pilot; purple: modules; yellow: components; red: tasks.

(Fig. 3.3, 5), scheduling them on the Executor. The Executor sets up the task's execution environment and then spawns the task for execution.

When required, the input data of a task are either pushed to the Agent or pulled from the Agent, depending on data locality and sharing requirements. Similarly, the output data of the task are staged out by the Agent and UnitManager to a specified destination, e.g., a filesystem accessible by the Agent or the user workstation. Both input and output staging are optional, depending on the requirements of the tasks. The actual file transfers are enacted via SAGA, and currently support `(gsi)-scp`, `(gsi)-sftp`, `Globus Online`, and local and shared filesystem operations via `cp`. Consequently, the size of the data along with network bandwidth and latency or filesystem performance determine the data staging durations and are independent of the performance of the RTS.

With this design and implementation, the functional and usability requirements are satisfied. The separation of application creation, processing the application, resource and execution management, ensures that applications can be composed without specifics of the CI, providing application portability. EnTK API is kept invariant of any specific domain, but simple targeted towards ensemble applications, enabling diverse applications to be supported by the same framework. Fault-tolerance is implemented as re-acquisition when resources fail, restart when EnTK components or the RTS fails, and resubmission

when programs fail.

Usability requirements are also met as the API to create applications is targeted towards ensemble applications, was developed based on discussions with domain scientists, and is continuously updated based on feedback and feature requests obtained from users via the EnTK repository [78]. The resource and execution management complexity is abstracted from the user as the user only describes the *what*, *when*, and *where* aspects of the application; the *how* is managed by EnTK. Performance of EnTK are its validation against the requirements are discussed in the following section.

3.4 Experiments

We perform three sets of experiments using EnTK: characterization of its overheads; characterizing of its weak and strong scaling performance; then implement the seismology and climate science science drivers, described in §2.1 at scale.

3.4.1 Characterization of overheads

We use a prototype of EnTK to benchmark its performance, providing a reference hardware configuration to support execution of up to $O(10^6)$ tasks. We then perform four experiments to characterize the overheads of EnTK.

3.4.1.1 Performance of EnTK Prototype

We prototyped the most computationally expensive functionality of EnTK to instantiate multiple producers and consumers of tasks. Each producer pushes tasks into RabbitMQ queues and each consumer pulls tasks from these queues, passing them to an empty RTS module. We benchmarked configurations with 10^6 tasks and a different number of producers, consumers, and queues, measuring: producers and consumers time; total execution time; base memory consumption when the components are instantiated; and peak memory consumption during the execution.

Fig. 3.4 shows that tuning of the prototype can reduce the processing time linearly, at the cost of increased memory usage. Eight producers and consumers require 107

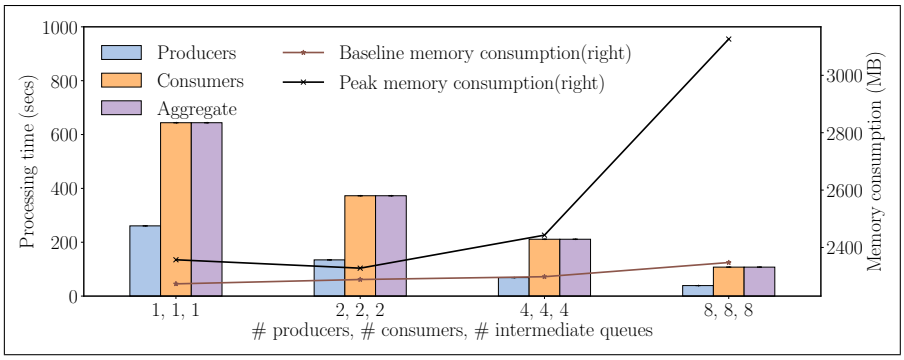

Figure 3.4: Execution time and memory consumed by EnTK prototype with multiple producers and consumers and 10^6 tasks.

seconds to process 10^6 tasks, with a peak memory consumption of 3,126MB. Uneven distributions of producers and consumers resulted in lower efficiencies than when using even distributions.

The execution model of EnTK can be tuned on the basis of this benchmark, workload requirements, and hardware capabilities. This benchmark shows that the performance of the core functionality of EnTK depends on the number of tasks that are processed concurrently. This has relevant implications for the understanding of EnTK overheads and scalability.

3.4.1.2 Overheads, Data Staging and Task Execution Time

We use two programs in our experiments: Sleep and Gromacs [83]. Sleep and Gromacs enable control of the duration of task execution and to compare EnTK overheads across different task programs. We perform our experiments on four CIs: XSEDE SuperMIC, Stampede, Comet and ORNL Titan.

We characterize EnTK overhead against four parameters that are likely to vary among applications: Task executable; Task duration; CI on which the application is executed; and structure of the application, i.e., the way in which tasks are grouped into stages and stages into pipelines. We measured the overheads that dominate EnTK and RTS runtime alongside the total task execution time and, when required, the total data staging time:

- **EnTK Setup Overhead**: Time taken to setup the messaging infrastructure,

Table 3.1: Parameters of the experiments plotted in Figure 5.3.

ID	Computing Infrastructure (CI)	Pipeline, Stage, Task	Executable	Task Duration	Data
1	SuperMIC	(1,1,16)	`mdrun`, `sleep`	300s	TDB
2	SuperMIC	(1,1,16)	`sleep`	1s, 10s, 100s, 1,000s	None
3	SuperMIC, Stampede, Comet, Titan	(1,1,16)	`sleep`	100s	None
4	SuperMIC	(16,1,1), (1,16,1), (1,1,16)	`sleep`	100s	None

instantiate components and subcomponents, and validate application and resource descriptions.

- **EnTK Management Overhead**: Time taken to process the application, translate tasks from and to RTS-specific objects, and communicate pipelines, stages, tasks and control messages.
- **EnTK Tear-Down Overhead**: Time taken to cancel all EnTK components and subcomponents, and shutdown the messaging infrastructure.
- **RTS Overhead**: Time taken by the RTS to submit and manage the execution of the tasks.
- **RTS Tear-Down Overhead**: Time taken by the RTS to cancel its components and to shutdown.
- **Data Staging Time**: Time taken to copy data between tasks using the functionality available on the resource (in this case, the Unix POSIX `cp` command).
- **Task Execution Time**: Time taken by the task executables to run on the CI.

We designed four experiments (Table 5.1) to characterize the overheads added by EnTK and the RP RTS to the time taken to execute an application, excluding the time taken by the resources to become available. These experiments execute applications with different task executable (Experiment 1, Fig. 5.3a); task duration (Experiment 2, Fig. 5.3b); CI (Experiment 3, Fig. 5.3c); and application structure, i.e., the number of pipelines, stages and tasks per application (Experiment 4, Fig. 5.3d).

Fig. 5.3 shows that EnTK Setup Overhead is ≈0.1s across Experiment 1, 2, 4, and ≈0.05s for Titan in Experiment 3. We attribute this difference to the host from which

EnTK was executed. All the experiments on XSEDE machines were performed from the same virtual machine (VM) hosted at TACC, while experiments on Titan had to be performed from an ORNL login node. The ORNL login nodes have faster memory and CPU than the VM.

Fig. 5.3 shows a similar behavior between EnTK Setup Overhead and EnTK Management Overhead. EnTK Management Overhead measures ≈10s on all the runs but those performed on Titan where it measures ≈3s. Also in this case, we attribute this difference to the performance of the VM and login nodes from which EnTK was executed.

In Fig. 5.3, EnTK Tear-Down Overhead and RTS Tear-Down Overhead vary across all four experiments with values between ≈1 and ≈10s for EnTK Tear-Down Overhead and ≈3 and ≈80s for RTS Tear-Down Overhead. We attribute these variations to the time taken by Python to terminate processes and threads. The higher values of RTS Tear-Down Overhead are expected as RP uses significantly more processes and threads than EnTK.

We explain the variations of RTS Overhead in Fig. 5.3 by noticing that, at runtime, RP initiates communications between the CI and a remote database, and reads and writes to the shared file system of the CI to create the execution environment of each task. Further, RP uses third party tools to distribute the execution of tasks across compute nodes. A detailed analysis of the interplay among network latency, I/O performance, and the performance of third party tools and libraries is beyond the scope of this dissertation. This is consistent with EnTK design: the RTS (RP in this case) is assumed to be a black box.

Fig. 5.3 shows that for tasks executing more than 1s, RP overheads have little impact on Task Execution time: As per experiment design, executables of Experiment 1 (Fig. 5.3a) run for ≈300s and those of Experiment 3 (Fig. 5.3c) for ≈100s on all four CIs. In Experiment 2 (Fig. 5.3b), tasks set to run for 1s, run for ≈5s due to RP overhead but tasks set to run for 10s, 100s, and 1,000s run in about that amount of time. In Experiment 4 (Fig. 5.3c), for runs with 16 pipelines and 16 tasks, all the tasks execute concurrently and hence Task Execution Time is ≈100s. However, with 16 stages, tasks execute sequentially, resulting in Task Execution Time of ≈1,600s.

EnTK setup, management, and tear-down overheads vary minimally with the four parameters of task execution we measured. Setup and management overheads depend on the memory and CPU performance of the host on which EnTK is executed, while the tear-down overhead on the Python version utilized. This validates EnTK design and implementation against its requirements: EnTK can be used in various scientific domains, with different task executables, and across heterogeneous CIs.

In absolute terms, EnTK overheads are between ≈10 and 20 seconds but Experiment 3 shows that these overheads can be reduced by running EnTK on a host with better performance. RP RTS shows overheads up to ≈80s, limiting its utilization to applications with at least minutes-long tasks. These limitations are mostly due to the use of Python and its process and thread termination time: EnTK and RP should be coded, at least partially, in a different language to manage the execution of applications of tasks that are O(1) seconds.

3.4.2 Scalability

We perform two experiments to characterize weak and strong scalability of EnTK. As with Experiment 1–4, we measure and compare all overheads, Data Staging Time and Task Execution Time. Weak scaling relates these measures to the amount of concurrency used to execute the application's tasks; Strong scaling to the amount of serialization.

3.4.2.1 Weak scalability

To investigate weak scaling, we run four applications on Titan, each with 1 pipeline, 1 stage per pipeline, and 512, 1,024, 2,048, or 4,096 tasks per stage. Each task executable is Gromacs `mdrun`, configured to use 1 core for ≈600 seconds. The number of acquired cores is equal to the number of the application's tasks. Each task requires 4 input files: 3 soft links of 130B each and 1 file of 550KB.

Fig. 3.6 (right axis) shows that Task Execution Time increases gradually and therefore does not have ideal weak scaling. Analysis of the RTS profiles shows that this behavior is due to delays in the Executor module of the RTS Agent and, specifically, in the current implementation of the Agent scheduler and the ORTE distributed virtual machine of

OpenMPI. Ref. [79] characterizes these delays and their causes.

EnTK Management Overhead remains almost constant till 2,048 tasks as the number of tasks are too small to cause a variation. The overhead, then, increases between 2,048 and 4,096 tasks: With the increase of the number of concurrent tasks, EnTK requires more resources and starts to strain the resources of the host on which it is executed. The other EnTK and RTS overheads appear to be consistent with those already noted in Experiments 1–4.

EnTK neither controls, nor contributes to Data Staging time. Data staging is performed by the RP RTS that, in this experiment, creates 1 directory for each task, writing 3 soft links and copying 1 file within it for a total of ≈1MB. RP uses Unix commands to perform these operations on the OLCF Lustre filesystem. By default, RP is configured with 1 stager and hence files are staged sequentially. Multiple staging workers can be used to parallelize data staging but trade offs with the filesystem performance must be taken into account.

Data Staging time grows linearly with the number of tasks executed: from ≈11s for 512 tasks to ≈88s for 4,096 tasks. As this time mostly depends on the performance of Lustre, a less linear behavior is expected with larger (amount of) files.

3.4.2.2 Strong scalability

To investigate strong scaling, we run four applications on Titan, each with 1 pipeline, 1 stage per pipeline, 8,192 tasks per stage and a total of 1,024, 2,048 or 4,096 cores. Each task executable is Gromacs `mdrun`, configured to use 1 core for ≈600 seconds. In this way, we execute at least 2 generations, each with 4,096 tasks, within the 2 hours walltime imposed by Titan's queuing policies. Data staging is as in the weak scalability experiment.

Fig. 3.7 shows that Task Execution Time reduces linearly with increase in the number of cores. The availability of more resources for the fixed number of tasks explains this linear reduction in the Task Execution Time. EnTK Management Overhead is ≈1s, confirming what already observed in the previous experiments.

All the other overheads and Data Staging Time remain constant across the experiment

runs. This suggests that both EnTK and RP overheads mostly depend on the number of managed tasks, not on the size of the pilot on which they are executed. This is confirmed for RP in Ref. [79].

Fig. 3.4 shows that EnTK can be configured to execute 10^6 tasks in less than 200 seconds and consuming less than 4GB of memory. Extrapolating and accounting for a faster CPU on Titan's login nodes, EnTK should manage enough tasks to fill all of Titan cores with an overhead of less than 20 seconds.

With these experiments, we show that the design and implementation of EnTK also meets the performance requirements posed by the users.

3.5 Domain science enabled

The design and implementation of EnTK meets the requirements derived from the various science drivers. Consequently, this enabled the encoding and execution of the ensemble applications from these science drivers at scale. We discuss the results obtained and science enabled in the various science drivers in this section.

3.5.1 ExTASY

The ExTASY science driver, discussed in §2.1.1, implements advanced sampling by performing short MD simulations and iteratively adding new simulations from different restarting configurations to efficiently sample the configuration space. The simulations are interleaved by analyses which determine what configurations are to be used for the next round of simulations.

In collaboration with scientists from Rice University and University of Nottingham, we developed a domain specific workflow system using the capabilities offered by EnTK. The workflow system, also named ExTASY [84], abstracts the PST model from the users as the users interface only with two config files. The users use one config file to specify the resources to be used and the other config file to hold application-related information such as number of iterations, number of simulations per iteration, length of the simulation, input molecular system, etc. The two config files are parsed to fully specified the EnTK

components. This approach lets the users focus on the algorithm and its parameters without having to manipulate the implementation of resource management and task execution.

In Ref [85], scientists used the ExTASY workflow system to evaluate the CoCo-MD algorithm on two molecular systems, alanine penta-peptide and cyclosporine A, using NCSA Blue Waters [86]. As the algorithm is unsupervised, the user does not need to specify in advance the interesting reaction coordinates, these emerge and adapt automatically as the sampling progresses. The results show that the CoCo-MD algorithm samples space 10 times faster than the conventional MD approach with no adaptations to the MD program. The algorithm, and its implementation in ExTASY, can be used with any MD program and demonstrates the potential of flexible workflow systems in simulation science, and the value of developing tools that maximize the seamless integration of established and new computational methods.

3.5.2 Seismic Tomography

We encoded the seismic tomography workflow described in §2.1.2 and depicted in Fig. 2.3 using EnTK focussing primarily on the reliable execution of the forward simulations. These simulations account for more than 90% of the computation time of the workflow, requiring 384 nodes on ORNL Titan [87]for each earthquake simulation, and 40MB of input data each. When earthquakes are concurrently simulated, they require a sizable portion of Titan and incur a high rate of failures. Without EnTK, these failures result in manual resubmission of computations, adding a significant overhead due to queue wait time on user intervention.

We characterize the scalability of forward simulations with EnTK by running experiments with a varying number of tasks, where each task uses 384 nodes/6,144 cores to forward simulate one earthquake. Understanding this scaling behavior contributes to optimize the execution of the whole workflow, both by limiting failure and enabling fault-tolerance without manual intervention. Ultimately, this will result in an increase of the overall efficiency of resource utilization and in a reduction of the time to completion.

The current implementation of forward simulations causes heavy I/O on a shared

file system. This overloads the file system, inducing crashes or requiring termination of the simulations. EnTK and RP utilize pilots to sequentialize a subset of the simulations, reducing the concurrency of their execution and without having to go through Titan's queue multiple times. This is done by reducing the number of cores and increasing the walltime requested for the pilot.

Fig. 3.8 shows that increasing concurrency leads to a linear reduction of Task Execution Time, with a minimum of ≈180 seconds. Interestingly, reducing concurrency eliminates failures: we encountered no failures in executions with up to 2^4 concurrent tasks and 6,144 nodes. At 2^5 concurrent tasks and 12,288 nodes, 50% of the tasks failed due to runtime issues.

EnTK automatically resubmits failed tasks until they are successfully executed. In the run with 2^5 tasks, EnTK attempts to run a total of 157 tasks. The resulting Task Execution Time was ≈360 seconds, similar to that of a run with with 2^4 concurrent tasks (Fig. 3.8).

EnTK and RP enable reasoning and benchmarking the concurrency of an execution without any change in the executable code. This gives insight on how to tailor a given computational campaign on a specific CI. The insight gained via our experiments can be immediately used in production: On Titan, forward simulations are best executed with 2^4 concurrent tasks. Further, fault-tolerance has an immediate impact on production runs, eliminating one of the most limiting factor of the previous implementation of the workflow.

3.5.3 High Resolution Meteorological Probabilistic Forecasts

We use EnTK to implement the AUA algorithm, described in §2.1.3 to iteratively and dynamically identify locations of the analogs. We also implement the *status quo* method of generating these analogs, i.e., random selection of locations in each iteration. We perform experiments to compare the two implementations and observe the speedup of the proposed algorithm. We repeat the experiment 30 times for statistical accuracy, initializing both implementations using the same initial random locations.

Fig. 3.9 shows the prediction maps and errors obtained from the two implementations.

With 1,800 locations calculated for both prediction maps (Fig. 3.9(b), Fig. 3.9(c)), the AUA algorithm generates a map with certain areas that have a better representation of the analysis than the map generated by a random selection of pixels.

The box plot in Fig. 3.9(d) shows the distribution of the errors for the two implementations. The error converges faster in the AUA algorithm than in the random selection. The total amount of potential locations (pixels) is 262,972; thus both implementations use a small fraction of the available locations but the AUA algorithm is automatically steering the computation at each iteration. EnTK and RP avoid the usual shortcoming of this approach: The evaluation required by the steering can be implemented as a task and iterations do not wait in the HPC queue, even if their number is unknown before execution. These results suggest that the AUA algorithm is well suited for very large domains in comparison to random selection of points.

3.5.4 High Throughput Binding Affinity Calculation

The HT-BAC science driver, discussed in §2.1.5, requires the implementation of the ESMACS and TIES protocols to calculate the binding affinity of drugs to malign proteins. In collaboration with domain scientists from University College, London, we developed a domain specific workflow system, called HTBAC [88], using the capabilities offered by EnTK.

HTBAC abstracts the EnTK API from the user as the users interface with higher level objects such as Protocol and Simulation. Users specify the drug candidate to be analyzed and the corresponding simulation parameters via these objects. Each Protocol corresponded to a drug candidate and several Protocol objects were submitted to the HTBAC system. HTBAC interfaced with EnTK to translate the application description into the PST model and specify the HPC system to be used for the calculation. This approach enables the users to reason at a much higher level that is closer to their science goals and requirements.

In Ref [89], we validate the implementation of the protocols implemented in HT-BAC and present the linear weak scaling behavior of HT-BAC that enables the analysis of multiple drug candidates concurrently on HPC systems.

3.5.5 Other science applications

In the replica exchange science driver, the domain scientists have developed a domain specific workflow system, called Repex [90]. Repex, similar to ExTASY and HT-BAC, abstracts the EnTK API from the user and exposes higher level objects such as Replicas and Exchange methods enabling the users to reason in terms of their domain. Validation and evaluation of performance characteristics of Repex is ongoing.

EnTK is also being used in the polar science domain to process and analyze O(1000) of images to identify seal population and penguin migration. As a relatively recent project, it benefits by the abstractions offered by EnTK to focus on directly on the processing algorithms.

3.6 Summary

In this chapter, we described the functional, performance and usability requirements derived from our science drivers. In response to these requirements, we formulated a programming model specific to ensemble applications, designed EnTK based on the building blocks approach that offers constructs to encode ensemble applications based on this programming model. The design and implementation of EnTK abstracts the complexities of resource and execution management from the user while reusing existing software systems in accordance with the building blocks approach. We characterized the overhead of EnTK based on computing infrastructure, structure of the application, executable program, and task duration and showed that the overhead remains invariant to these factors. We characterized the scalability of EnTK and showed that EnTK shows linear weak and strong scaling behavior for up to O(1000) tasks. Finally, we discussed the science enabled by EnTK in the various science drivers.

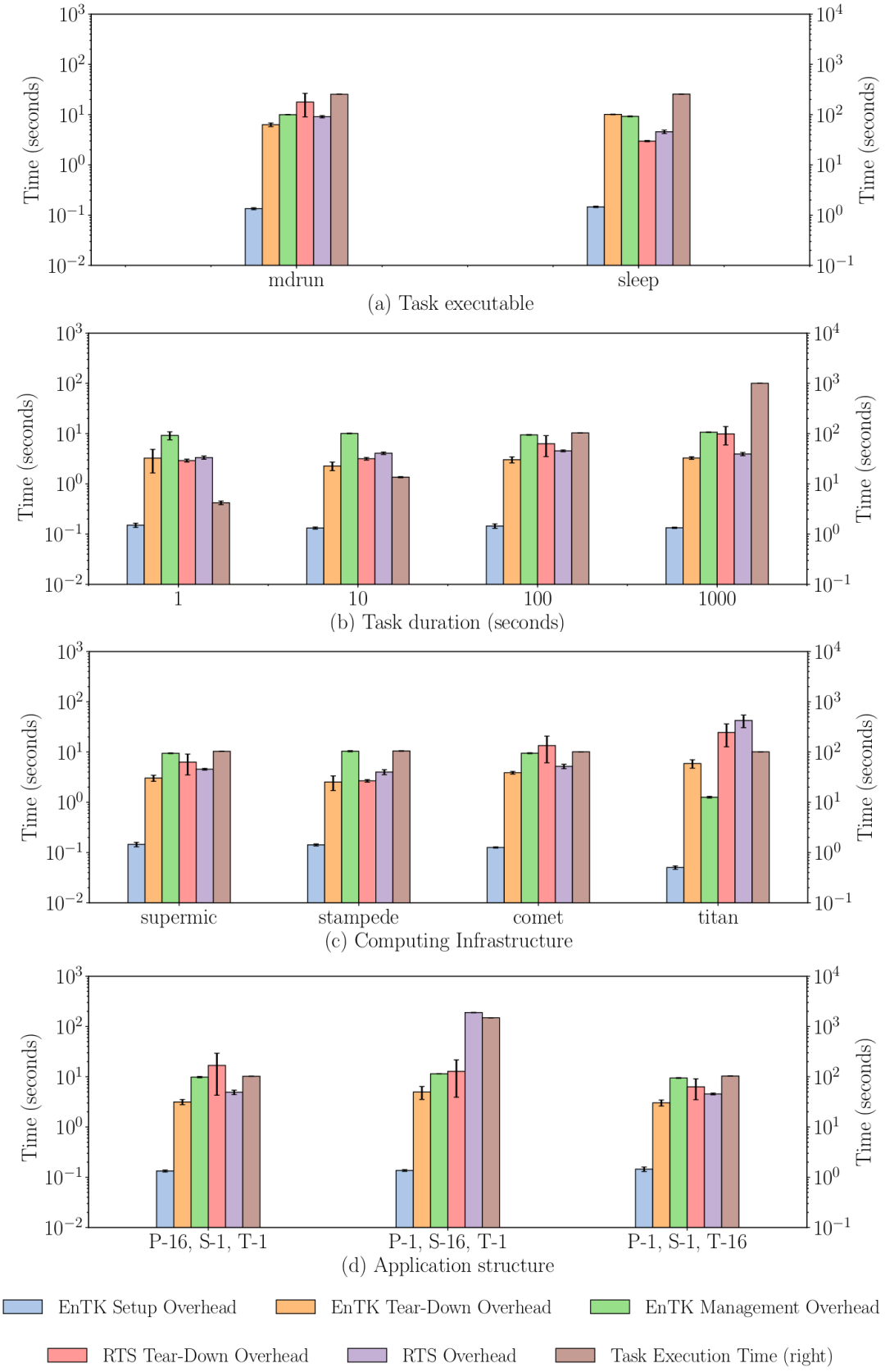

Figure 3.5: Overheads and Task Execution Time as function of (a) Task Executable (Experiment 1), (b) Task Duration (Experiment 2) (c) Computing Infrastructure (Experiment 3) (d) Application Structure (Experiment 4).

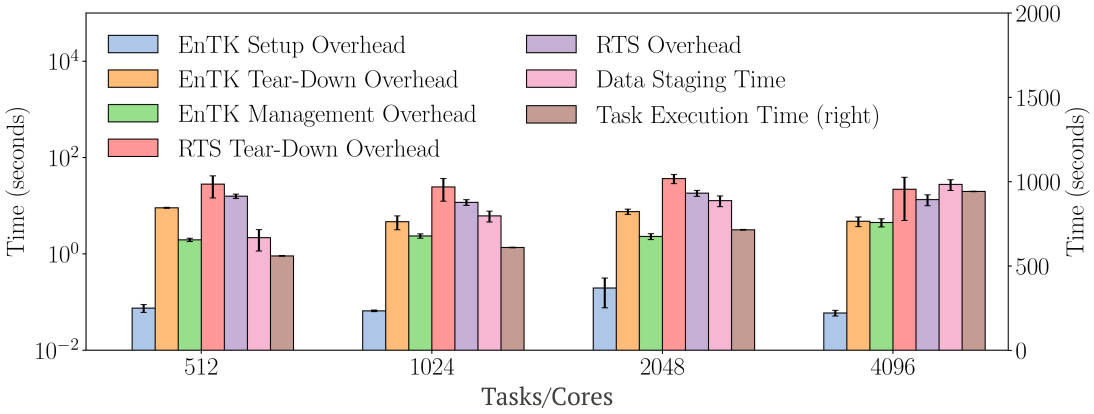

Figure 3.6: Weak scalability on Titan: 512, 1,024, 2,048, and 4,096 1-core tasks executed on the same amount of cores.

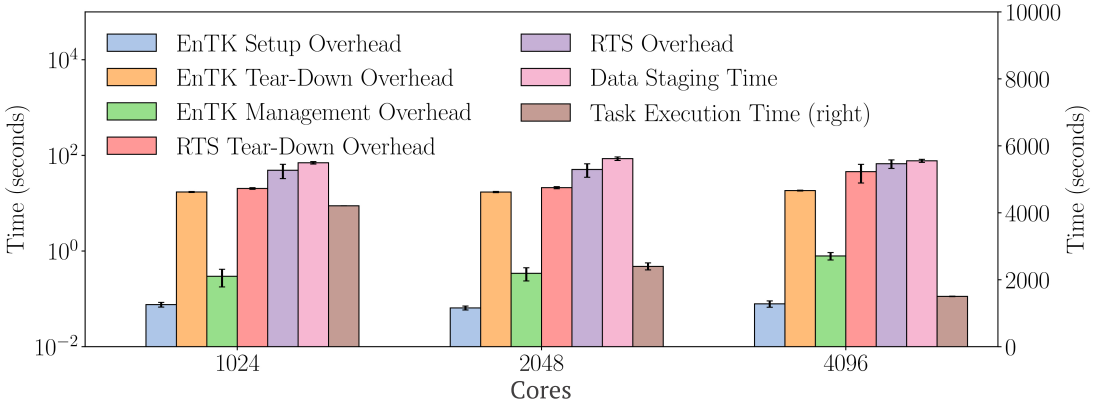

Figure 3.7: Strong scalability on Titan: 8,192 1-core tasks are executed on 1,024, 2,048 and 4,096 cores.

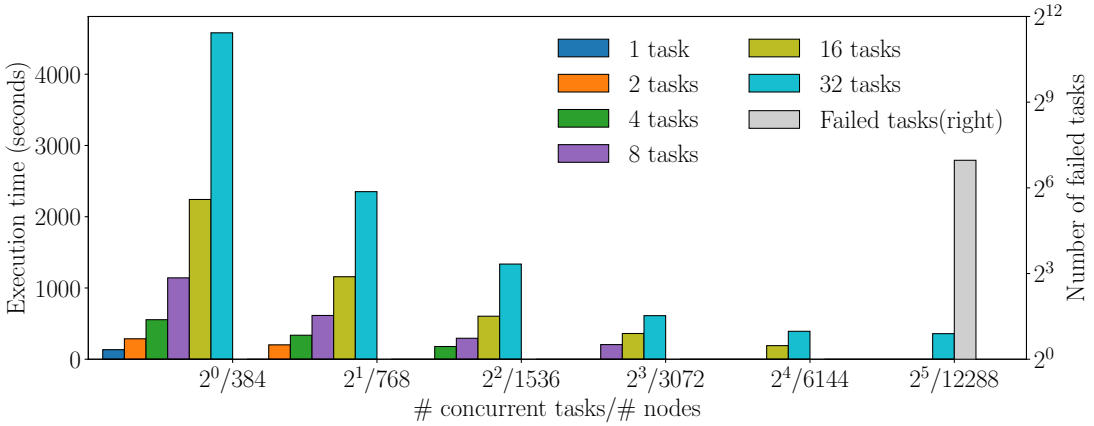

Figure 3.8: Task Execution Time of forward simulations using EnTK at various values of concurrency.

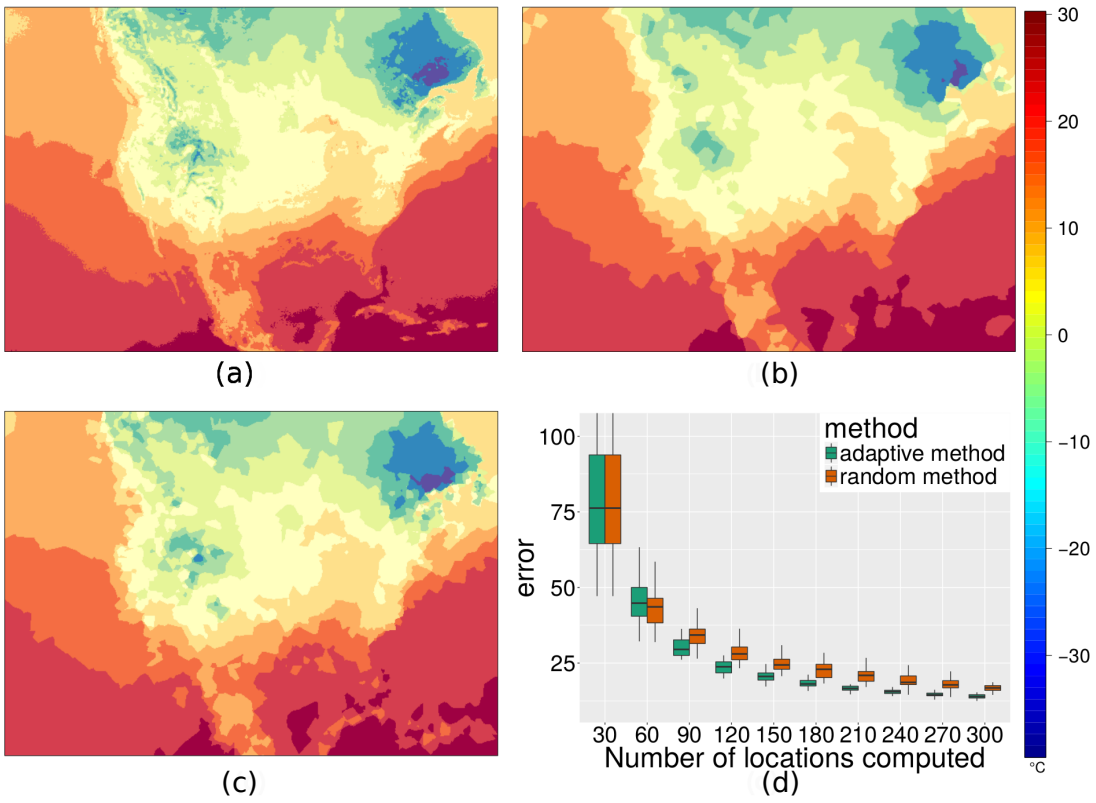

Figure 3.9: Predictions from random and adaptive methods. (a) theoretical true value, (b) the interpolated map from 1,800 randomly picked locations, (c) the interpolated map from 1,800 locations identified using AUA, (d) box plots of the errors for both implementations.

Chapter 4

Adaptive Ensemble Applications

Existing ensemble-based methods have been successful for addressing a number of questions in several science domains. However, researchers are already encountering the limitations of these methods when studying bigger scientific problems; problems that require approaches beyond ensemble applications. In biomolecular modeling [91], studying systems with multiple-timescale behavior extending out to microseconds or milliseconds, or studying even shorter timescales on larger physical systems will not only require tools that can support 100–1000 greater degrees of parallelism but also exploration of adaptive algorithms. In adaptive algorithms, the intermediate results of simulations are used to alter following simulations. Adaptive approaches can increase simulation efficiency by greater than a thousand-fold [92] but require more sophisticated software infrastructure to encode, modularize, and execute complex interactions and execution logic.

In this chapter, we discuss several important adaptive ensemble applications from the biophysics science domain, challenges in encoding and executing these applications and the current set of tools, systems and services available to support adaptive ensemble applications.

4.1 Adaptive Science Drivers

In this chapter, we discuss two representative adaptive ensemble applications from the biophysical domain: Expanded Ensemble and Markov State Modeling. Prior to discussing the implementation of these applications, we describe the underlying algorithms.

4.1.1 Expanded Ensemble

Metadynamics [93] and expanded ensemble (EE) dynamics [94] are a class of adaptive ensemble biomolecular algorithms, where individual simulations jump between simulation conditions. In EE dynamics, the simulation states take one of N discrete states of interest, whereas in metadynamics, the simulation states are described by one or more continuous variables. In both algorithms, each simulation explores the states independently. Additional weights are required to force the simulations to visit desired distributions in the simulation condition space, which usually involves sampling in all the simulation conditions. These weights are learned adaptively using a variety of methods [94].

Since the movement among state spaces is essentially diffusive, the larger the simulation state spaces, the more time the sampling takes. "Multiple walker" approaches can improve sampling performance by using more than one simulation to explore the same state space [95]. Further, the simulation condition range can be partitioned into individual simulations as smaller partitions decrease diffusive behavior [96]. The "best" partitions to spend time sampling may not be known until after simulation. These partitions can be determined adaptively, based on runtime information about partial simulation results.

In this dissertation, we implement two versions of EE consisting of concurrent, iterative ensemble members that analyze data at regular intervals. In the first version, we analyze data local to each ensemble member; in the second version we analyze data global to all the ensemble members by asynchronously exchanging data among members. In our application, each ensemble member consists of two types of task: simulation and analysis. The simulation tasks generate MD trajectories while the analysis tasks use these trajectories to generate simulation condition weights for the next iteration of simulation in its own ensemble member. Every analysis task operates on the current snapshot of the total local or global data. Note that in global analysis, EE uses any and all data available and does not explicitly "wait" for data from other ensemble members. Fig. 4.1 is a representation of these implementations.

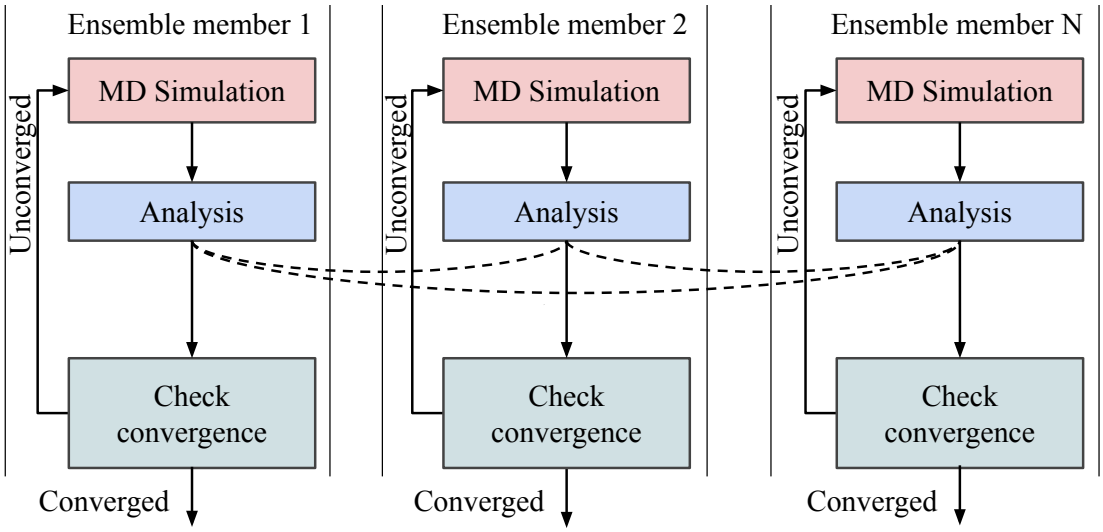

Figure 4.1: Schematic of the expanded ensemble (EE) science driver. Two versions of EE are implemented: (1) local analysis where analysis only data local to its ensemble member; and (2) global analysis where analysis uses data from other ensemble members (represented by dashed lines)

4.1.2 Markov State Modeling

Markov state modeling (MSM) is another important class of biomolecular simulation algorithms for determining kinetics of molecular models. Using an assumption of separation of time scales of molecular motion, the rates of first-order kinetic processes are learned adaptively. In a MSM simulation, a large ensemble of simulations, typically tens or hundreds of thousands, are run from different starting points and similar configurations are clustered as states. MSM building techniques include kinetic information but begin with a traditional clustering method (eg k-means or k-centers) using a structural metric. Configurations of no more than 2Å to 3Å RMSDs are typically clustered into the same "micro-state" [97].

The high degree of structural similarity implies a kinetic similarity, allowing for subsequent kinetic clustering of microstates into larger "macro-states". The rates of transitions among these states are estimated by observing which entire kinetic behavior can be inferred, even though individual simulations perform no more than one state transition. However, the choice of where new simulations are initiated to best refine the definition of the states, improve the statistics of the rate constants, and discover new simulation states requires a range of analyses of previous simulation results, making the

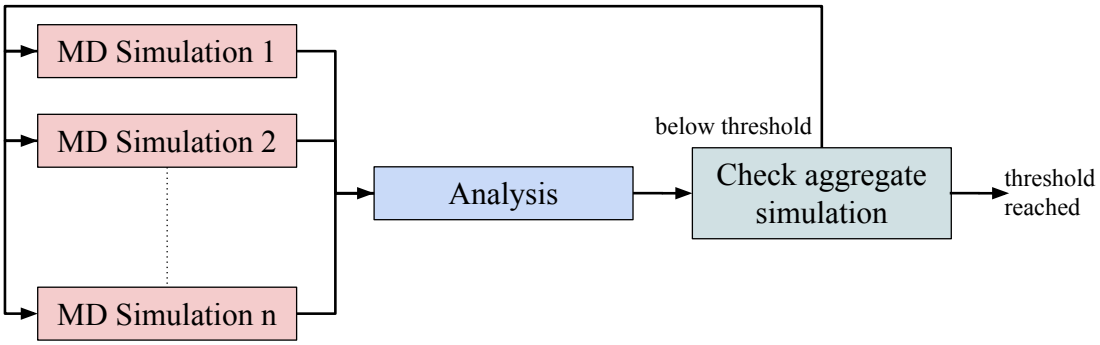

Figure 4.2: Schematic of the Markov State Model science driver.

entire algorithm highly adaptive.

MSM provides a way to encode dynamic processes such as protein folding into a set of metastable states and transitions among them. In computing MSM from simulation trajectories, the metastable state definitions and the transition probabilities have to be inferred. Refs. [98, 99] show that "adaptive sampling" can lead to more efficient MSM construction as follows: provisional models are constructed using intermediate simulation results, and these models are then used to direct the placement of further simulation trajectories. Different from other approaches, in this dissertation we encode this algorithm as an application where the adaptive code is independent from the software packages used to perform the MD simulations and MSM construction.

Fig. 4.2 offers a diagrammatic representation of the adaptive ensemble MSM approach. The application consists of an iterative pipeline with two stages: (i) ensemble of simulations and (ii) MSM construction to determine optimal placement of future simulations. The first stage generates sufficient amount of MD trajectory data for an analysis. The analysis–i.e., the second stage–operates over the cumulative trajectory data to adaptively generate a new set of simulation configurations, used in the next iteration of the simulations. The pipeline is iterated until the resulting MSM converges.

4.1.3 Adaptive versions of previous science drivers

In addition to the science drivers discussed above, several science drivers described in § 2.1 are being investigated with adaptive approaches. In Ref [100], researchers implemented an adaptive version of the the HT-BAC framework. This version makes

two important improvements: (i) the simulations are decomposed into multiple iterations where the starting configuration for each iteration is determined during runtime; and (ii) the number of iterations are determined by the analysis of intermediate results.

Both the ExTASY framework, § 2.1.1, and the analog ensemble algorithm, § 2.1.3 are also being investigated with adaptive approaches. In the former, the configurations to use between the iterations is determined based on the analysis of results obtained during execution. In the latter, the amount of communication between different tasks and the number of iterations to reach thresholds is determined based on results obtained during execution.

4.2 Challenges

Supporting adaptive workflows poses three main challenges. The first challenge is the expressibility of adaptive workflows as their encoding requires APIs that enable the description of the initial state of the workflow and the specification of how the workflow adapts on the base of intermediate signals. The second challenge is determining when and how to instantiate the adaptation. Adaptation is described at the end of the execution of tasks wherein a new TG is generated. Different strategies can be employed for the instantiation of the adaptation [101]. The third challenge is the implementation of the adaptation of the TG at runtime. We divide this challenge into three parts: (i) propagation of adapted TG to all components; (ii) consistency of the state of the TG among different components; and (iii) efficiency of adaptive operations.

4.3 Current Solutions: Tools, Systems, and Services

Adaptive ensemble applications span several science domains including, but not limited to, climate science, seismology, astrophysics, and bio-molecular science. For example, Ref. [102] studies adaptive selection and tuning of dynamic RNNs for hydrological forecasting; Ref. [103] presents adaptive modeling of oceanic and atmospheric circulation; Ref. [104] studies adaptive assessment methods on an ensemble of bridges subjected to earthquake motion; and Ref. [105] discusses parallel adaptive mesh refinement techniques

for astrophysical and cosmological applications. In this dissertation, we focus on biomolecular applications, as examples, employing algorithms to simulate biophysical events.

Algorithms consisting of one or more MD simulations, provide quantitative and qualitative information about the structure and stability of molecular systems, and the interactions among them. Specialized computer architectures enable single MD simulations at the millisecond scale [106] but alternative approaches are motivated by the higher availability of general-purpose machines and the need to investigate biological processes at the scales from milliseconds to minutes. Importantly, although we discuss mostly biological applications, there are many applications of MD in material science, polymer science, and interface science [107, 108].

Statistical estimation of thermodynamic, kinetic, and structural properties of biomolecules requires multiple samples of biophysical events. Algorithms with ensembles of MD simulations have been shown to be more efficient at computing these samples than single, large and long-running MD simulations [95, 109, 110, 111]. Adaptive ensemble algorithms use runtime data to guide the progression of the ensemble, achieving up to a thousand-fold increase in efficiency compared to non-adaptive alternatives [99, 98].

Several adaptive ensemble algorithms have been formulated. For example, replica exchange [112] consists of ensembles of simulations where each simulation operates with a unique value of a sampling parameter, such as temperature, to facilitate escape from local minima. In generalized ensemble simulation methods, different ensemble simulations employ distinct exchange algorithms [113] or specify diverse sampling parameters [114] to explore free-energy surfaces that are less accessible to non-adaptive methods. In metadynamics [93] and expanded ensemble [94], simulations traverse different states based on weights "learned" adaptively. Markov State Model [111] (MSM) approaches adaptively select initial configurations for simulations to reduce uncertainty of the resulting model.

Current solutions to encode and execute adaptive ensemble algorithms fall into two categories: monolithic workflow systems that do not fully support adaptive algorithms and MD software packages where the adaptivity is embedded within the executing

kernels. Several workflow systems [115], including Kepler,Taverna and Pegasus support adaptation capabilities only as a form of fault tolerance and not as a way to enable decision-logic for changing the workflow at runtime.

Well known MD software packages such as Amber, GROMACS and NAMD offer capabilities to execute adaptive ensemble algorithms. However, these capabilities are tightly coupled to the MD package, preventing users from easily adding new adaptive algorithms or reusing the existing ones across packages.

Domain-specific workflow systems such as Copernicus [75] have also been developed to support Markov state modeling algorithms to study kinetics of bio-molecules. Although Copernicus provides an interactive and customized interface to domain scientists, it requires users to manage the acquisition of resources, the deployment of the system and the configuration of the execution environment. This hinders Copernicus uptake, often requiring tailored guidance from its developers.

Encoding the adaptive ensemble algorithm, including its adaptation logic within MD software packages or workflow systems locks the capabilities to those individual tools. In contrast, the capability to encode the algorithm and adaptation logic as an user application promises several benefits: separation between algorithm specification and execution; flexible and quick prototyping of alternative algorithms; and extensibility of algorithmic solutions to multiple software packages, science problems and scientific domains [116, 117]. To realize these promises, we develop the abstractions and capabilities to encode adaptivity at the ensemble application level, and execute adaptive ensemble applications at scale on high performance computing (HPC) systems.

4.4 Summary

In this chapter we discussed the significance of adaptivity and how they enable researchers to go beyond the limitations of traditional ensemble applications. We discussed two science drivers in detail and described adaptive versions of science drivers discussed in § 2. We described the challenges in encoding and executing these applications and the current set of solutions available to researchers and their limitations.

Chapter 5

Ensemble Toolkit for Adaptive Ensemble Applications

Adaptive ensemble applications discussed in § 4 involve two computational layers: at the lower level each simulation or analysis is performed by a program or executable; at the higher level, an algorithm codifies the coordination and communication between different tasks. Different adaptive ensemble applications and adaptive algorithms might have varying coordination and communication patterns, yet are amenable to common adaptations and similar types of adaptations.

In this chapter, we analyze the execution requirements of adaptive ensemble applications, identify the different types of adaptivity required by the science drivers and describe the enhancements made in EnTK to support adaptive ensemble applications. We characterize the overhead of supporting adaptivity in EnTK, validate the implementation of our science drivers and discuss the results obtained by executing adaptive ensemble applications at production scale.

5.1 Understanding workflow adaptivity

Executing adaptive workflows at scale on HPC resources, using ensemble-based methods presents several challenges [117].

For simplicity, in the discussion of adaptivity we represent a workflow as a task-graph (TG) when discussing operations applied to the workflow. Workflows may be represented as a single or multiple disjoint TGs. The following discussion regarding workflow adaptivity apply to other representations of a workflow.

Our analysis of adaptive workflows suggests that The complete TG of is not known prior to execution and may change depending on intermediate runtime results. Execution of adaptive workflows can be decomposed into four operations as represented in Fig. 5.1:

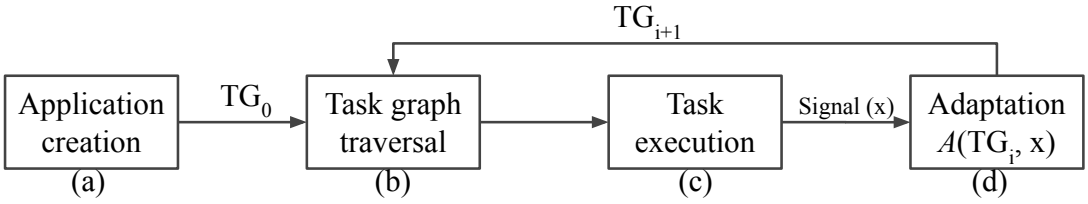

Figure 5.1: **Adaptivity Loop**: Sequence of operations in executing an adaptive workflow

(a) creation of an initial TG, encoding known tasks and dependencies; (b) traversal of the initial TG to identify tasks ready for execution in accordance with their dependencies; (c) execution of those tasks on the compute resource; and (d) notification of completed tasks (control-flow) or generation of intermediate data (data-flow) which invokes adaptations of the TG.

Operations (b)–(d) are repeated till the complete workflow is determined, and all its tasks are executed. This sequence of operations is called an Adaptivity Loop: in an adaptive scenario, the workflow "learns" its future TG based on the execution of its current TG; in a pre-defined scenario, the workflow's TG is fully specified and only operations (a)–(c) are necessary.

Encoding of adaptive workflows requires two sets of abstractions: one to encode the workflow; and the other to encode the adaptation methods (A) that, upon receiving a signal x, operate on the workflow. The former abstractions are required for creating the TG, i.e., operation (a), while the latter are required to adapt the TG, i.e., operation (d).

5.1.1 Types of Adaptations

Adaptivity Loop applies an adaptation method (Fig. 5.1d) to a TG. We represent a TG as $TG = [V, E]$, with the set V of vertices denoting the tasks of the workflow and their properties (such as executable, required resources, and required data), and the set E of directed edges denoting the dependencies among tasks. For a workflow with $TG = [V, E]$, there exist four parameters that may change during execution: (i) set of vertices; (ii) set of edges; (iii) size of the vertex set; and (iv) size of the edge set. We analyzed the 2^4 permutations of these four parameters and identified 3 that are valid and unique. The remaining permutations represent conditions that are either not

possible to achieve or combinations of the 3 valid permutations.

5.1.1.1 Task-count adaptation

We define a method A_{tc} (operator) as a task-count adaptation if, on receiving a signal x, the method performs the following adaptation (operation) on the TG (operand):

$$TG_{i+1} = A_{tc}(TG_i, x) \implies size(V_i) \neq size(V_{i+1}) \wedge size(E_i) \neq size(E_{i+1})$$
$$\text{where } TG_i = [V_i, E_i] \wedge TG_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-count adaptation changes the number of TG's tasks, i.e., the adaptation method operates on a TG_i to produce a new TG_{i+1} such that at least one vertex and one edge is added or removed to/from TG_i.

5.1.1.2 Task-order adaptation

We define a method A_{to} as a task-order adaptation if, on a signal x, the method performs the following adaptation on the TG:

$$TG_{i+1} = A_{to}(TG_i, x) \implies E_i \neq E_{i+1} \wedge V_i = V_{i+1}$$
$$\text{where } TG_i = [V_i, E_i] \wedge TG_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-order adaptation changes the dependency order among tasks, i.e., the adaptation method operates on a TG_i to produce a new TG_{i+1} such that the vertices are unchanged but at least one of the edges between vertices is different between TG_i and TG_{i+1}.

5.1.1.3 Task-property adaptation

We define a method A_{tp} as a task-property adaptation if, on a signal x, the method performs the following adaptation on the TG:

$$TG_{i+1} = A_{tp}(TG_i, x) \implies V_i \neq V_{i+1} \wedge size(V_i) = size(V_{i+1}) \wedge E_i = E_{i+1}$$
$$\text{where } TG_i = [V_i, E_i] \wedge TG_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-property adaptation changes the properties of at least one task, i.e., the adaptation method operates on a TG_i to produce a new TG_{i+1} such that the edges

and the number of vertices are unchanged but the properties of at least one vertex is different between TG_i and TG_{i+1}.

We can represent the workflow of the two science drivers using the notations presented. Expanded ensemble (EE) consists of N ensemble members executing independently for multiple iterations till convergence is reached in any ensemble member. We represent one iteration of each ensemble members as a task graph TG and the convergence criteria with x. An adaptive EE workflow can then be represented as:

parellel_for i in $[1 : N]$:
    while (condition on x):
        $TG_i = A_{tp}(A_{to}(A_{tc}(TG_i)))$

Markov State Modeling (MSM) consists of one ensemble member which iterates between simulation and analysis till sufficient trajectory data is analyzed. We represent one iteration of the ensemble member as a task graph TG and its termination criteria as x. An adaptive MSM workflow can then be represented as:

while (condition on x):
    $TG = A_{to}(A_{tc}(TG))$

5.1.2 Challenges in Encoding Adaptive Workflows

Supporting adaptive workflows poses three main challenges. The first challenge is the expressibility of adaptive workflows as their encoding requires APIs that enable the description of the initial state of the workflow and the specification of how the workflow adapts on the base of intermediate signals. The second challenge is determining when and how to instantiate the adaptation. Adaptation is described at the end of the execution of tasks wherein a new TG is generated. Different strategies can be employed for the instantiation of the adaptation [101]. The third challenge is the implementation of the adaptation of the TG at runtime. We divide this challenge into three parts: (i) propagation of adapted TG to all components; (ii) consistency of the state of the TG among different components; and (iii) efficiency of adaptive operations.

```
from radical.entk import Task, Stage
s = Stage()
t = Task()
<add task properties>
s.add_tasks(t)
s.post_exec = {
                'condition': <function_1 name>,
                'on_true':   <function_2 name>,
                'on_false':  <function_3 name>
              }
```

Figure 5.2: Post execution properties of a Stage consisting of one Task. At the end of the Stage, 'function_1' (boolean condition) is evaluated to return a boolean value. Depending on the value, 'function_2' (true) or 'function_3' (false) is invoked.

5.2 Enhancements in Ensemble Toolkit

In response to these challenges we engineered EnTK with three new capabilities: expressing an adaptation operation, executing the operation, and modifying a TG at runtime.

Adaptations in ensemble workflows follow the Adaptivity Loop described in §5.1. Execution of one or more tasks is followed by some signal `x` that triggers an adaptation operation. In EnTK, this signal is currently implemented as a control signal triggered at the end of a stage or a pipeline. We added the capability to express this adaptation operation as post-execution properties of stages and pipelines. In this way, when all the tasks of a stage or all the stages of a pipeline have completed, the adaptation operation can be invoked to evaluate based on the results of the ongoing computation, whether a change in the TG is required. This is done asynchronously without effecting any other executing tasks.

The adaptation operation is encoded as a Python property of the Stage and Pipeline objects. The encoding requires the specification of three functions: one function to evaluate a boolean condition over `x`, and two functions to describe the adaptation, depending on the result of the boolean evaluation.

Users define the three functions specified as post-execution properties of a Stage or Pipeline, based on the requirements of their application. As such, these functions

can modify the existing TG or extend it as per the three adaptivity types described in §5.1.1.

Ref. [101] specifies multiple strategies to perform adaptation: forward recovery, backward recovery, proceed, and transfer. In EnTK, we implement a non-aggressive adaptation strategy, similar to 'transfer', where a new TG is created by modifying the current TG only after the completion of part of that TG. The choice of this strategy is based on the current science drivers where tasks that have already executed and tasks that are currently executing are not required to be adapted but all forthcoming tasks might be.

Modifying the TG at runtime requires coordination among EnTK components to ensure consistency in the TG representation. AppManager holds the global view of the TG and, upon instantiation, Workflow Processor maintains a local copy of that TG. The dequeue sub-component of Workflow Processor acquires a lock over the local copy of the TG, and invokes the adaptation operation as described by the post-execution property of stages and pipelines. If the local copy of the TG is modified, Workflow Processor transmits those changes to AppManager that modifies the global copy of TG, and releases the lock upon receiving an acknowledgment. This ensures that adaptations to the TG are consistent across all components, while requiring minimal communication.

Pipeline, stage, and task descriptions alongside the specification of an adaptation operation as post-execution for pipelines and stages enable the expression of adaptive workflows. The 'transfer' strategy enacts the adaptivity of the TG, and the implementation in EnTK ensures consistency and minimal communication in executing adaptive workflows. Note how the design and implementation of adaptivity in EnTK does not depend on specific capabilities of the software package executed by each task of the ensemble workflow.

5.3 Experiments

We perform three sets of experiments. The first set characterizes the overhead of EnTK when performing the three types of adaptation described in §5.1.1. The second

set validates our implementation of the two science drivers presented in §4.1 against reference data. The third set compares our implementation of adaptive expanded ensemble algorithm with local and global analysis against results obtained with a single and an ensemble of MD simulations.

We use four application kernels in our experiments: `stress-ng` [118], `GROMACS` [83], `OpenMM` [119] and Python scripts. `stress-ng` allows to control the computational duration of a task for the experiments that characterize the adaptation overhead of EnTK, while `GROMACS` and `OpenMM` are the simulation kernels for the expanded ensemble and Markov state modeling validation experiments.

We executed all experiments from the same host machine but we targeted three HPC systems, depending on the amount and availability of the resources required by the experiments, and the constraints imposed by the queue policy of each machine. NCSA Blue Waters and ORNL Titan were used for characterizing the adaptation overhead of EnTK, while XSEDE SuperMIC was used for the validation and production scale experiments.

5.3.1 Characterization of adaptation overheads

We perform five experiments to characterize the overhead of adapting ensemble workflows encoded using EnTK. Each experiment measures the overhead of a type of adaptation as a function of the number of adaptations. In the case of task-count adaptation, the overhead is measured also as a function of the number of tasks and of their type, single- or multi-node. This is relevant because with the growing of the size of the simulated molecular system and of the duration of that simulation, multi-node tasks may perform better than single-node ones.

Each experiment measures EnTK Adaptation Overhead and Task Execution Time. The former is the time taken by EnTK to adapt the workflow by invoking user-specified algorithms; the latter is the time taken to run the executables of all tasks of the workflow. Consistent with the scope of this dissertation, the comparison between each adaptation overhead and task execution time offers a measure of the efficiency with which EnTK implements adaptive functionalities. Ref. [120] has a detailed analysis of other overheads

Table 5.1: Parameters of the experiments plotted in Fig. 5.3

ID	Adaptation Type	Experiment variable	Fixed parameters
I	Task-count	Number of adaptations	Number of tasks added per adaptation = 16, Type of tasks added = single-node
II	Task-count	Number of tasks added per adaptation	Number of adaptations = 2, Type of tasks added = single-node
III	Task-count	Type of tasks added	Number of adaptations = 2, Number of tasks added per adaptation = $2^{10} * 2^{s}$ (s=stage index)
IV	Task-order	Number of adaptations	Number of re-ordering operations per adaptation = 1, Type of re-ordering = uniform shuffle
V	Task-property	Number of adaptations	Number of property modified per adaptation = 1, Property adapted = Number of cores used per task

of EnTK.

Table 5.1 describes the variables and fixed parameters of the five experiments about adaptivity overheads in EnTK. In these experiments, the algorithm is encoded in EnTK as 1 pipeline consisting of several stages with a set of tasks. In the experiments I–III about task-count adaptation, the pipeline initially consists of a single stage with 16 tasks of a certain type. Each adaptation, at the completion of a stage, adds 1 stage with a certain number of tasks of a certain type, thereby increasing the task-count in the workflow.

In experiments IV–V, the workflow is encoded as 1 pipeline with 17, 65, or 257 stages with 16 tasks per stage. Each adaptation occurs upon the completion of a stage and, in the case of task-order adaption, the remaining stages of a pipeline are shuffled. In the case of task-property adaption, the number of cores used by the tasks of the next stage is set to a random value below 16, keeping the task type to single-node. The last stage of both experiments are non-adaptive, resulting in 16, 64, and 256 total adaptations.

In the experiments I, IV and V, where the number of adaptations varies, each task of the workflow executes the `stress-ng` kernel for 60 seconds. For the experiments II

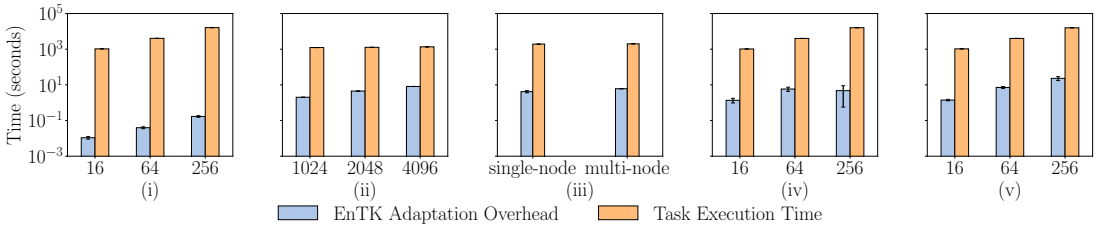

Figure 5.3: EnTK Adaptation Overhead and Task Execution Time for task-count (i, ii, and iii), task-order (iv), and task-property (v) adaptations.

and III with $O(1000)$ tasks, the execution duration is set to 600 seconds so to avoid performance bottlenecks in the underlying runtime system and therefore interferences with the measurement of EnTK adaptation overheads. All experiments have no data movement as the performance of data operations is independent from that of adaptation.

Figs. 5.3(i), 5.3(iv), and 5.3(v) show that EnTK Adaptation Overhead and Task Execution Time increase linearly with the increasing of the number of adaptations. EnTK Adaptation Overhead increases due to the time taken to compute the additional adaptations and its linearity indicates that the computing time of each adaptation is constant. Task Execution Time increases due to the time taken to execute the tasks of the stages that are added to the workflow as a result of the adaptation.

Figs. 5.3(i), 5.3(iv), and 5.3(v) also show that task-property adaptation (v) is the most expensive, followed by task-order adaptation (iv) and task-count (i) adaptation. These differences depend on the computational cost of the Python functions executed during adaptation: in task-property adaptation, the function parses the entire workflow and invokes the Python `random.randint` function 16 times per adaptation; in task-order adaptation, the Python function shuffles a Python list of stages; and in task-count adaption, the Python function creates an additional stage, appending it to a list.

In Fig. 5.3(ii), EnTK Adaptation Overhead increases linearly with an increase in the number of tasks added per task-count adaptation, explained by the cost of creating additional tasks and adding them to the workflow. The Task Execution Time remains constant at $\approx 1200s$, since sufficient resources are acquired to execute all the tasks concurrently.

Fig 5.3(iii) compares EnTK Adaptation Overhead and Task Execution Time when

adding single-node and multi-node tasks to the workflow. The former is greater by $\approx 1s$ when adding multinode tasks, whereas the latter remains constant at $\approx 1200s$ in both scenarios. The difference in the overhead, although negligible when compared to Task Execution Time, is explained by the increased size of a multi-node task description. As in Fig. 5.3(ii), Task Execution Time remains constant due to availability of sufficient resources to execute all tasks concurrently.

Experiments I–V show that EnTK Adaptation Overhead is proportional to the computing required by the adaptation algorithm and is not determined by the design or implementation of EnTK. In absolute terms, EnTK Adaptation Overhead is orders of magnitude smaller than Task Execution Time. Thus, EnTK advances the practical use of adaptive ensemble workflows.

5.3.2 Validation of Science Driver Implementations

We implement the two science drivers of §4.1 using the abstractions developed in EnTK. We validate our implementation of Expanded Ensemble (EE) by calculating the binding of the cucurbit[7]uril 6-ammonio-1-hexanol host-guest system, and our implementation of Markov State Modeling (MSM) by simulating the Alanine dipeptide system and comparing our results with the reference data of the DESRES group [121].

5.3.2.1 Expanded Ensemble

We execute the EE science driver described in §4.1.1 on XSEDE SuperMIC for a total of 2270ns MD simulation time. To validate the process, we carry out a set of simulations of the binding of cucurbit[7]uril (host) to 6-amino-1-hexanol (guest) in explicit solvent for a total of 29.12ns per ensemble member, and compare the final free energy estimate to a reference calculation. Each ensemble member is encoded in EnTK as a pipeline of stages of simulation and analysis tasks, where each pipeline uses 1 node for 72 hours. With 16 ensemble members (i.e., pipelines) for the current physical system, we use $\approx 1K/23K$ node/core-hours of computational resources.

The EE simulates the degree of coupling between the guest and the rest of the system (water and host). As the system explores the coupling using EE dynamics, it binds and

unbinds the guest to and from the host. The free energy of this process is gradually estimated over the course of the simulation, using the Wang-Landau algorithm [122]. However, we hypothesize that we can speed convergence by allowing parallel simulations to share information with each other, and estimate free energies using the potential energy differences among states and the Multistate Bennett Acceptance Ratio (MBAR) algorithm [123].

We consider four variants of the EE method:

- **Method 1:** one continuous simulation, omitting *any* intermediate analysis.
- **Method 2:** multiple parallel simulations without *any* intermediate analysis.
- **Method 3:** multiple parallel simulations with local intermediate analysis, i.e., using current and historical simulation information from only its own ensemble member.
- **Method 4:** multiple parallel simulations with global intermediate analysis, i.e., using current and historical simulation information from all ensemble members.

In each method, the latter 2/3 of the simulation data available at the time of each analysis is used for free energy estimates via the MBAR algorithm. In methods 3 and 4, adverse effects of the Wang-Landau algorithm are eliminated due to the intermediate analyses. These provide a better estimate of the weights that are used to force simulations to visit desired distributions in the simulation condition space (see §4.1.1). Note that in methods 3 and 4, where intermediate analysis is used to update the weights, the intermediate analysis is always applied at 320ps intervals.

The reference calculation consisted of four parallel simulations that ran for 200ns each and with fixed weights, i.e., using a set of estimated weights and not using the Wang-Landau algorithm. MBAR was used to estimate the free energy for each of these simulations.

Fig. 5.4 shows the free energy estimates obtained through each of the four methods with the reference calculation value. Final estimates of each method agree within error to the reference value. Validating that the four methods used to implement adaptive ensembles converge the free energy estimate to the actual value.

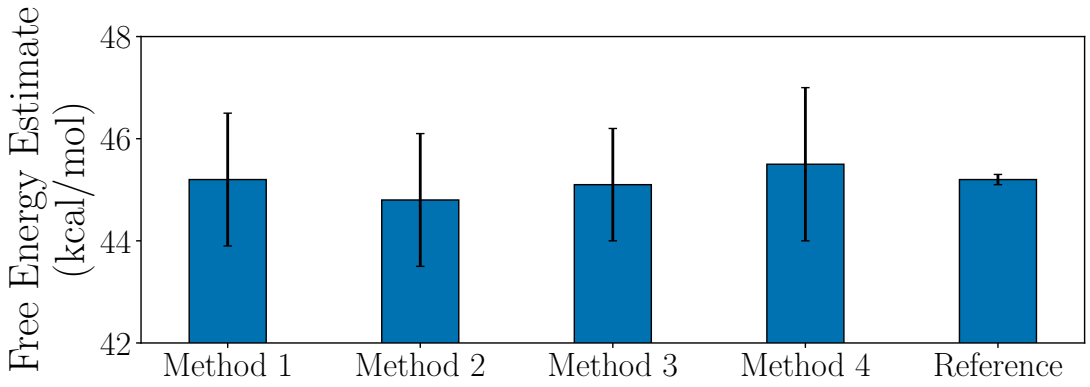

Figure 5.4: Validation of EE implementation: Observed variation of free energy estimate for methods 1–4. Reference is the MBAR estimate and standard deviation of four 200ns fixed weight expanded-ensemble simulations.

5.3.2.2 Markov State Modeling

We execute the MSM science driver described in §4.1.2 on XSEDE SuperMIC for a total of 100ns MD simulation time over multiple iterations. Each iteration of the TG is encoded in EnTK as one pipeline with 2 stages consisting of 10 simulation tasks and 1 analysis task. Each task uses 1 node to simulate 1ns.

We compare the results obtained from execution of the EnTK implementation against reference data by performing the clustering of the reference data and deriving the mean eigenvalues of two levels of the metastable states, i.e., macro- and micro-states. The reference data was generated by a non-adaptive workflow consisting of 10 tasks, each simulating 10ns.

Eigenvalues attained by the macro-states (top) and micro-states (bottom) in the EnTK implementation and reference data are plotted as a function of the state index in Fig. 5.5. Final eigenvalues attained by the implementation agree with the reference data within the error bounds. The validation of the implementation warrants that similar implementations should be investigated for larger molecular systems and longer durations, where the aggregate duration is unknown and termination conditions are evaluated during runtime.

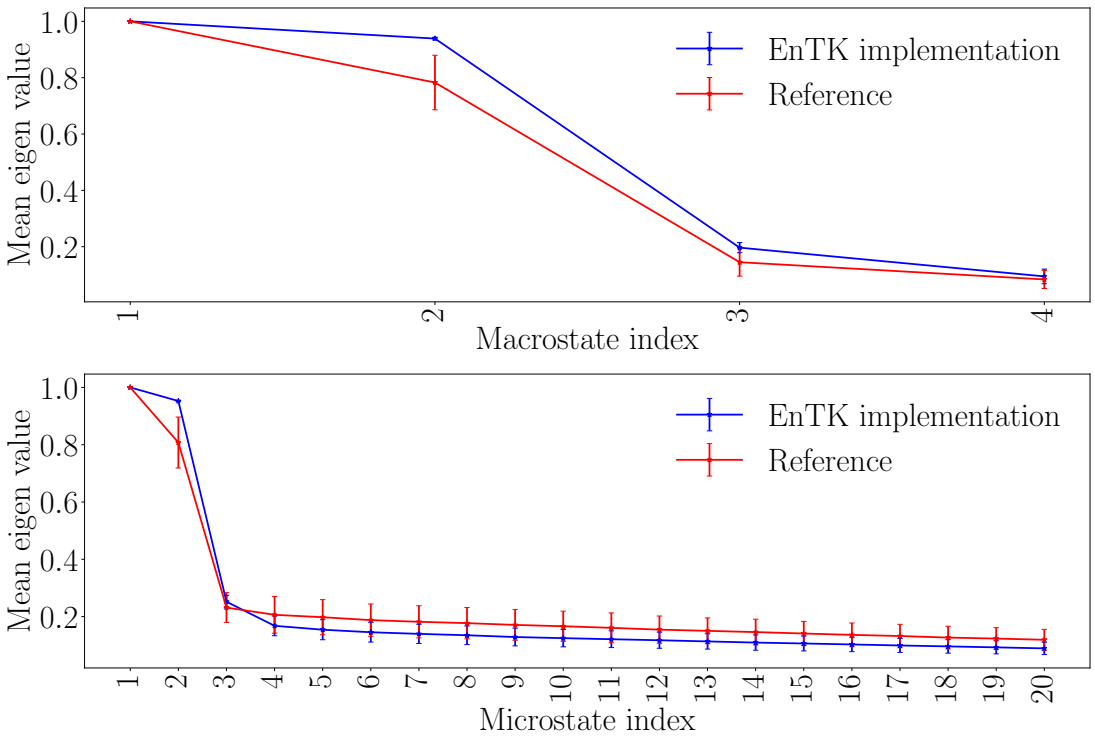

Figure 5.5: Mean eigenvalue attained by the macro-states (top) and micro-states (bottom) by Alanine dipeptide after aggregate simulation duration of 100ns implemented using EnTK compared against reference data.

5.4 Domain science enabled

5.4.1 Evaluation of Methodological Efficiency using Adaptive Capabilities in EnTK

We analyzed the convergence properties of the free energy estimate using the data generated for the validation of EE. The convergence behavior of Method 1 observed in Fig. 5.6 implies that the current method converges faster than ensemble based methods but does not represent the average behavior of the non-ensemble based approach. The average behavior is depicted more clearly by Method 2 because this method averages the free energy estimate of 16 independent single simulations.

The most significant feature of Fig. 5.6 is that all three ensemble based methods converge at similar rates to the reference value. We initially hypothesized that adding adaptive analysis to the estimate of the weights would improve convergence behavior but we see no significant change in these experiments. However, the methodology described here gives researchers the ability to implement additional adaptive elements and test their effects on system properties. Additionally, these adaptive elements can be implemented on relatively short time scales, giving the ability to test many implementations.

Analysis of these simulations revealed a fundamental physical reason that demonstrates a need for additional adaptivity to successfully accelerate these simulations. Although expanded ensemble simulations allowed the ligand to move in and out of the binding pocket rapidly, the slowest motion, occurring on the order of 10s of nanoseconds, was the movement of water out of the binding pocket, allowing the ligand to bind as water backs into a vacant binding pocket. Simulation biases that equilibrate on shorter timescales may stabilize either the waters out or the waters in configurations, preventing the sampling of both configurations. Additional biasing variables are needed to algorithmically accelerate this slow motions, requiring a combination of metadynamics and expanded ensemble simulations, with biases both in the protein interaction variable and the collective variable of water occupancy in the binding pocket. Changes in the PLUMED2 metadynamics code are being coordinated with the developers to make this possible.

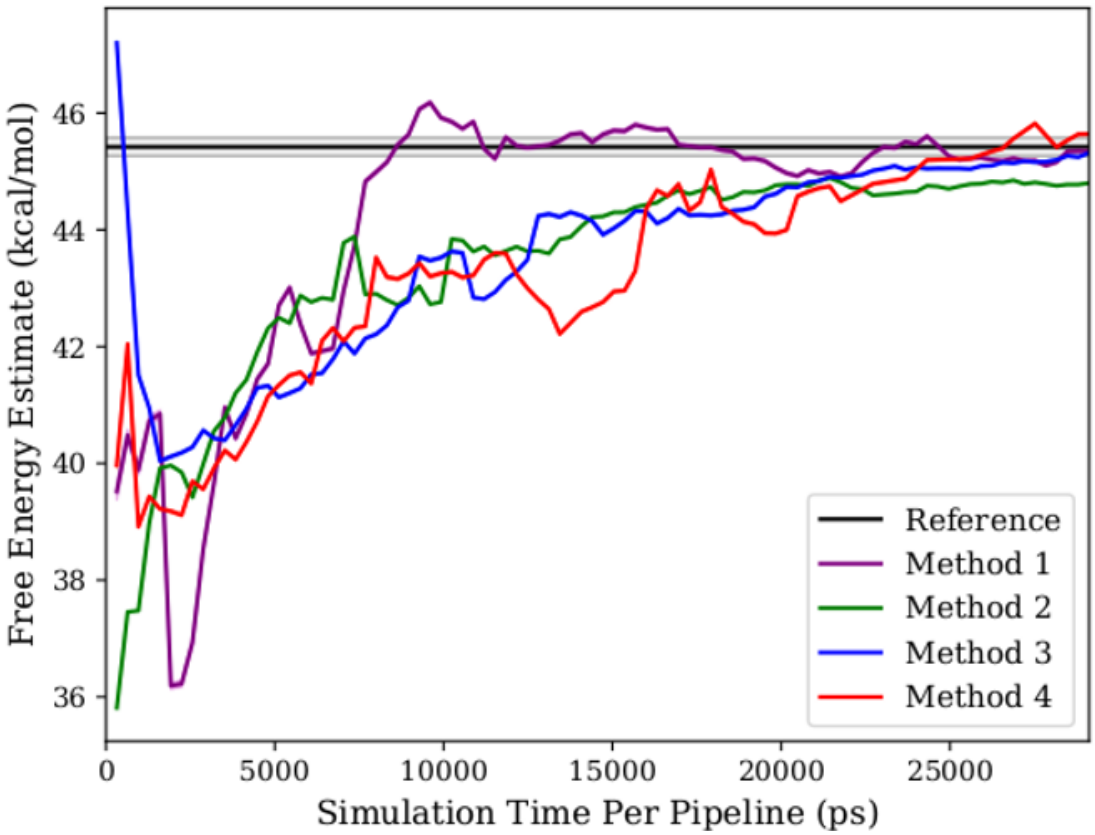

Figure 5.6: Convergence of expanded ensemble implementation: Observed convergence behavior in methods 1–4. Reference is the MBAR estimate of the pooled data and the standard deviation of the non-pooled MBAR estimates of four 200ns fixed weight expanded ensemble simulations.

Analysis of the slow motions of the system suggests the potential power of more complex and general adaptive patterns. Simulations with accelerated dynamics along the hypothesized degrees of freedom can be carried out, and resulting dynamics can be analyzed, automated and monitored for degrees of freedom associated with remaining slow degrees of motion [124]. Accelerated dynamics can be adaptively adjusted as the simulation process continues. Characterization experiments suggest that EnTK can support the execution of this enhanced adaptive workflow with minimal overhead.

5.4.2 Other science enabled

In Ref [100], researchers implemented an adaptive version of the the HT-BAC framework discussed in § 2.1.5. The framework encoded the adapting functions and executed their adaptive quadrature algorithm using the enhancements described in this chapter. As a result, the adaptive HT-BAC framework showed 77% improvement in accuracy of the binding affinity calculation given a fixed amount of computing resources and an improvement of 32% in resource utilization when compared to non-adaptive algorithms for binding affinity calculation.

Adaptive versions of the ExTASY framework discussed in § 2.1.1 have been implemented to investigate the whole energy landscape of 3 proteins and their protein folding times. The adaptive version shows an order of magnitude speed up compared to "brute force" molecular dynamics simulations.

5.5 Summary

In this chapter, we discussed the sequence of operations involved in executing adaptive ensemble applications which informed us about *when*, *where* and *how* adaptivity can be supported. Based on our science drivers, we identified three types of adaptivity that is required to be supported. We described their implementation in EnTK, characterized its overhead and determined that the overhead is orders of magnitude smaller than TTX of ensemble applications. We implemented two adaptive ensemble applications, expanded ensemble and markov state modeling, and validated the results against reference data.

We implemented expanded ensemble at production scale and identified several ways of improving the algorithm and described other science enabled by the adaptive capabilities discussed in this chapter.

Chapter 6

Heterogeneity and Dynamism

In the previous chapters, we discussed the programmability and execution of various adaptive and non-adaptive ensemble applications on HPC systems. Workloads, derived from these applications, and computational resources on HPC systems can be *heterogeneous* and *dynamic*. In our work, so far, we have not considered these aspects of heterogeneity and dynamism when making scheduling decisions. In this chapter, we describe heterogeneity and dynamism in workloads and resources, their implications on the application total time to execute (TTX), and when and how they can be managed by proposing and evaluating a strategy.

We define a group of entities as heterogeneous when the entities differ from each other based on their properties. A heterogeneous workload consists of tasks that differ from each other based on properties like execution time, executed program, required resource, etc. Similarly, a heterogeneous resource consists of computational cores that differ from each other based on properties such as processing speed, memory capacity, filesystem and memory access speed, etc.

We refer to an entity as being dynamic if its properties change over time. A dynamic computational core could have a processing speed, filesystem capacity or memory access speed that fluctuates over time. We refer to the resource, which consists these dynamic computational cores, as also being dynamic as the properties of the resource are composed by the properties of its cores.

EnTK and other systems discussed in §2.2 and §4.3 make the assumption that workloads derived from ensemble applications are homogeneous However, workloads from ensemble applications can be heterogeneous as their tasks can have different properties. Examples include simulations that have different configurations [23], analyses

that process different type and amount of data [6], concurrent pipelines that evaluate different models in climate modeling or concurrent pipelines that are invoked for different drug candidates in drug discovery [89].

Similarly, current systems make the assumption that the performance of the acquired resources are static and uniform. However, this does not reflect the execution performance measured in practice [125, 126, 127]. Resource performance can be highly dynamic and has been shown to fluctuate by up to 100% [125, 127]. Some factors causing such variations include inter-application resource contention [128, 127]; kernel intrusions [127]; thermal control [129]; and process scheduling algorithms [127].

Typically, domain scientists have constraints on the time to solution of their application and a budget on the amount of resources that can be utilized. On HPC systems, time to solution is described as the sum of the time spent in the queue of the HPC system to acquire the resources and the total time to execute the application once the resources are acquired. Queue time depends on queuing policy and system load, and is difficult to predict [130, 131, 132, 133]. Thus we equate the time to solution to the total time to execute, TTX.

Assumptions, described above, on the heterogeneity and dynamism of workloads and resources lead to sub-optimal execution, resulting in a long TTX and low resource utilization. Thus, it is important to develop solutions that consider heterogeneity and dynamism when making runtime decisions such that their TTX and resource utilization are manageable and acceptable. In the following sections, we discuss: (1) current solutions to address this limitation; (2) our strategy to manage heterogeneity and dynamism; (3) design and implementation of an emulator to evaluate our strategy; (4) evaluation of the late-binding strategy; and (5) impact of our findings.

6.1 Current Solutions: Tools, Systems, and Services

Heterogeneity in the workload and resources is typically managed by selecting a subset of entities from the larger set. Quite common in literature, this operation is performed to identify tasks or resources that exhibit optimal performance within required constraints.

Ref [134] uses benchmarks generated by the resource providers against user criteria to select resources. Tools such as Nimrod [135] use "bids" for resources as a weight in their selection. Matchmaker [136] provides language and syntax to match user requests to the most suitable resources. Ref [137] extends this language to select multiple suitable resources. Resource selection is also integrated in frameworks such as Mesos [138] and Pegasus [139] which support different selection algorithms. Similar work can be found for task selection such as delay scheduling in MapReduce frameworks [140], sophisticated cost-time optimization algorithms [141], resiliency [142], and application-driven criteria. Runtime parameters such as TTX have been improved by 2-7% when using historic data regarding the dynamism of the resources [143, 144], by up to 33% when using live probing [145], and parameters such as utilization and energy consumption on clouds can be improved by 28% [146].

As mentioned in our science drivers, the size of the workload is orders of magnitude greater than the resources available and cannot be executed at 100% concurrency and needs to be temporally segmented into different generations. In current solutions, decisions on when and where a task of a workload is to be executed are based either on historical data, data acquired during resource acquisition, or are simply based on algorithms such as round-robin that do not use any information regarding the workload or resource. This decision is also made for tasks in generations that are to be executed later in time, and are not informed about the heterogeneity and dynamism of the workload and resource, leading to sub-optimal execution. We term these approaches as employing an early-binding strategy.

Alternatively, we propose and evaluate a late-binding strategy where scheduling decisions are delayed till the tasks need to be executed, using the latest information regarding the workload and resource. We discuss our strategy further in the next section.

6.2 Late-binding strategy

Ref. [147] investigates the late-binding strategy when the application is non-adaptive and performance of the compute resources are assumed to be uniform and static, but the

availability of compute resources is heterogeneous and dynamic. The paper presents the effectiveness of the strategy as the decision to bind tasks to resources on different HPC systems is made after the resources become available. Compute resources on a HPC system become available for use after a queue wait time, which is difficult to accurately predict and can be significantly high [130, 131, 132, 133]. The paper compared two scenarios: (1) tasks are bound to one large resource request; and (2) tasks are late bound to three smaller resource requests as they become available. The paper showed that the second scenario exhibited lower time to completion than the former, and that the effectiveness of the strategy holds true when the queue times are larger than the execution time of the tasks.

We propose that the late-binding approach can be employed to reduce TTX in the presence of heterogeneity and dynamism in resource and workloads. Prior to proposing our hypothesis, we first define four terms that we will use frequently in the following sections.

Definitions:

- **Core:** Abstraction of a processing unit that can perform a specific number of operations per second. We will refer to the number of operations performed per second by a core as the core performance.
- **Task:** Abstraction of a computational process that performs a specific number of operations during execution. We assume that all tasks use a single-core. We will refer to the number of operations performed by a task as the task length.
- **Resource:** A set of cores where their performance may vary spatially, i.e., performance of different cores of a resource may be different at a given time, and temporally, i.e., the performance of a single core may vary over time.
- **Workload:** A set of tasks where the task length may vary spatially, i.e., number of operations to be performed by different tasks of a workload may be different.

We hypothesize that late-binding, i.e., delaying the scheduling decision of when and on which core a task needs to be executed, can lead to better performance metrics than

early-binding, i.e., scheduling decisions are made as soon as the workload and resource are available.

We make five assumptions that we consider in our hypothesis:

1. Compute resources have already been acquired and are ready to be used for task execution.

2. The size of the acquired resources remains constant. We consider dynamism only in the performance of the acquired compute resources.

3. Entities providing information regarding the performance of the compute resources are external to the system and are assumed to be accurate.

4. We assume that the performance of the cores do not vary during execution but vary only between different task submissions.

5. We assume that there is no cost to make scheduling decisions, i.e., time to make scheduling decisions is 0, and does not influence the scheduling decision.

Combinatorially, heterogeneity and dynamism for workloads and resources gives us 16 possibilities, but not all of them can be realized when focusing on ensemble applications and HPC systems. Workloads derived from ensemble applications consist of tasks that may differ in their properties but the property of a task does not change over time. Consequently, workloads derived from ensemble applications, composed of such tasks, may be heterogeneous but not dynamic. Additionally, HPC systems are composed of several hundreds and thousands of computational cores but all cores have the same architecture. Typically, HPC systems that contain cores with different architectures cannot be acquired at the same time. Thus, computational cores on HPC systems are always homogeneous. Computational cores experience different loads over time and, thus, their performance is dynamic in practice.

Narrowing down the problem space with the above three considerations, we are left with two possible combinations: 2 and 3 in Table 6.1 and Fig. 6.1. For reference, we also consider the case when the workload and resource are both homogeneous and static,

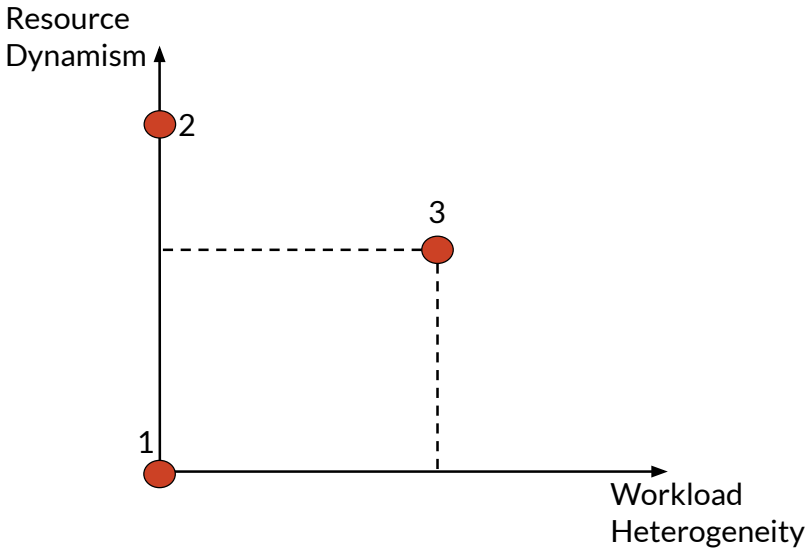

Figure 6.1: Problem space

Table 6.1: Tabular representation of the problem space

Combination	Workload Heterogeneity	Resource Heterogeneity	Workload Dynamism	Resource Dynamism
1	**0**	**0**	**0**	**0**
2	**0**	**0**	**0**	**1**
3	**1**	**0**	**0**	**1**

mentioned as combination 1 in Table 6.1 and Fig. 6.1. These 3 combinations for the problem space in which we will be evaluating early- and late-binding strategies.

In this chapter, we perform a theoretical and empirical evaluation of these three combinations at various levels of heterogeneity and dynamism using early- and late-binding strategies. In order to perform empirical evaluations, we developed an emulator that we describe in the following section.

6.3 Workload Management Emulator

Creating workloads with real-world tasks and resources with cores that exhibit different levels of heterogeneity and dynamism is non-trivial, if not impossible. But studying the effect of heterogeneous workloads and dynamic resources is important to manage the execution of ensemble applications on HPC systems. For this purpose, we developed a

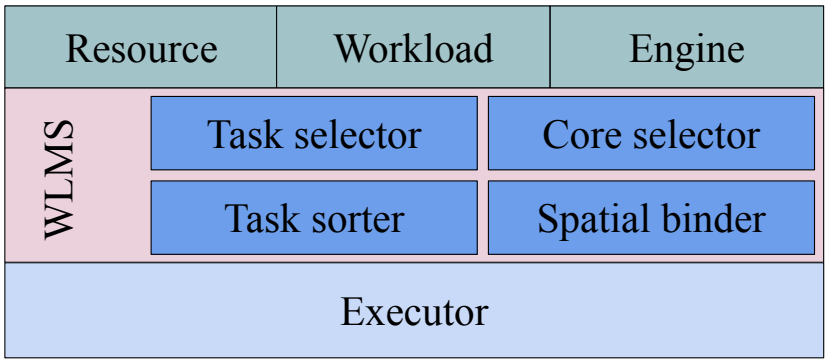

Figure 6.2: Architecture of the WLMS Emulator

workload management emulator that enables (1) creation of workloads with heterogeneous or homogeneous tasks; (2) creation of resource with homogeneous and dynamic cores; and (3) specifying criteria for scheduling decisions; to emulate the execution behavior when using early- and late-binding strategies.

6.3.1 Design

We present the architecture of the emulator in Fig. 6.2. The emulator has three user facing components: Resource, Workload and Engine. The Resource component represents a group of computational cores and can be specified to be heterogeneous or homogeneous and static or dynamic. It can be specified with a distribution from which the performance of its cores are sampled.

Similarly, the Workload component represents a group of tasks and can be specified to be heterogeneous or homogeneous. It can be specified with a distribution from which the length of its tasks are sampled.

The Engine accepts a user specified resource, workload and criteria to be used for scheduling decisions. There are four criteria, corresponding to the four subcomponents of the WLMS component, that can be specified along with the strategy to use: late-binding or early-binding.

Workload Management System or WLMS is the component of the emulator that performs the scheduling decisions to create a schedule of when and on which core a task needs to be executed, which is passed over to the Executor. WLMS consists of

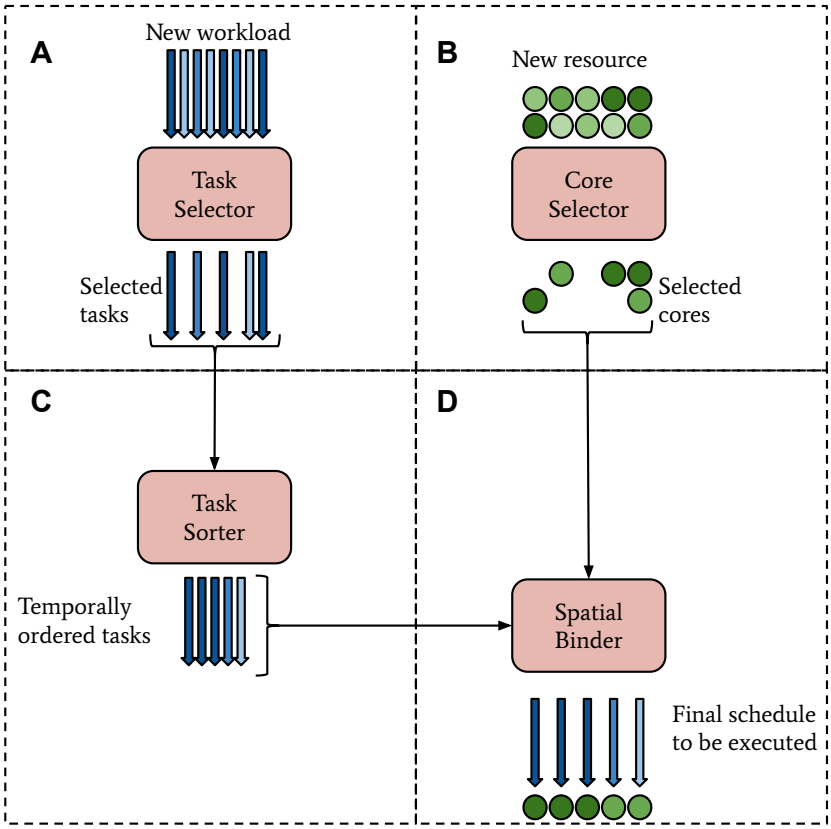

Figure 6.3: Order of the four operations performed by WLMS: A>B>C>D

four subcomponents: Task Selector, Core Selector, Task Sorter, Spatial Binder that correspond to four operations performed by the WLMS. The two selectors are used to identify tasks from workloads or cores from resources that are eligible for execution, for example, cores that offer non-zero computational capability are to be selected. Task Sorter is used to temporally order the tasks of the workload based on the task properties. Spatial Binder is used to place a task on the appropriate core based on the properties of the task and core. The criteria specified by the user determine the rule to use to order the tasks in the Task Sorter and the placement of the tasks in the Spatial Binder. The WLMS, thus, decides both when and where a task should execute. Fig. 6.3 represents the order of operations in the WLMS.

The Engine also informs the WLMS whether early-binding or late-binding strategy is to be used. If early-binding strategy is to be used, scheduling decisions, i.e. task sorting and spatial binding, of all tasks are made based on the initial core performance observed at the time of resource availability and does not take into account the changes

in the core performance if the resource is dynamic. If late-binding strategy is to be used, scheduling decisions are made based on the latest core performance observed, taking into account the performance changes of the cores of a dynamic resource. As described above, we assume that the core performance does not change during task execution but only between different task submissions.

The Executor receives a schedule from the WLMS that describes when and on which core a task needs to be executed. The Executor executes as per the schedule and uses the latest performance of the core as it is aware of resource dynamism. The Executor records the execution and generates profiles that can be analyzed post-mortem.

6.3.2 Implementation

The Emulator is implemented in Python and user facing components are implemented as Python objects using which users can create the scenarios they want to execute.

The Resource and Workload can be specified with two types of distributions: uniform distribution and normal distribution. Currently, the selectors can be configured to either select all entities of a group, entities with properties above a threshold or entities with properties below a threshold. The Task Sorter can be configured to use one of the three criteria: sort in random order, sort in ascending order of task length or sort in descending order of task length. Spatial Binder can be configured with two criteria: random placer which assigns tasks to cores randomly or 'L2FF', which places the longest task on the core where it will finish execution first or earliest.

6.4 Theoretical and empirical evaluation

For each of the three combinations described in Table 6.1, we will estimate the time to execute a workload on a resource where the properties of the workload and resource are described as specified in Table 6.1. In our evaluation, the length of tasks of a workload and performance of cores of a resource are drawn from an uniform distribution where the width of the distribution represents the heterogeneity or dynamism.

6.4.1 Combination 1: Homogeneous and Static Workload, Homogeneous and Static Resource

In combination 1, the workload is homogeneous and static and resource is homogeneous and static. Let the uniform distribution which is sampled for task lengths be between (s_{low}, s_{high}) and the uniform distribution which is sampled for core performance be between (k_{low}, k_{high}). As there is no heterogeneity or dynamism, the uniform distributions which are sampled for task length and core performance have no width, i.e., $s_{high} - s_{low} = 0$ and $k_{high} - k_{low} = 0$. Let the length be of all tasks be S and performance of all cores be K, i.e., $s_{high} = s_{low} = S$ and $k_{high} = k_{low} = K$.

Lets say that there are R cores and S tasks such that $S > R$ and we will have N generations where $N = S/R$ when R tasks are executing concurrently. Let $t^{start}(g)$ be the time at which the first task of generation g starts execution. Let $t^{stop}(g)$ be the time at which the last task of generation g stops execution. Then we can represent the start and stop times at the end of the first generation as,

$$t^{start}(1) = 0,\ 0,\ 0,\ldots.\ \text{R instances} \tag{6.1}$$

$$t^{stop}(1) = \text{T/K},\ \text{T/K},\ \text{T/K},\ \ldots.\ \text{R instances} \tag{6.2}$$

and at the end of the second generation as,

$$t^{start}(2) = \text{T/K},\ \text{T/K},\ \text{T/K},\ldots.\ \text{R instances} \tag{6.3}$$

$$t^{stop}(2) = \text{2T/K},\ \text{2T/K},\ \text{2T/K},\ \ldots.\ \text{R instances} \tag{6.4}$$

Similarly, after N generations we get,

$$t^{stop}(N) = \text{NT/K},\ \text{NT/K},\ \text{NT/K},\ \ldots.\ \text{R instances} \tag{6.5}$$

Thus, we can estimate our TTX, $t^{stop}(N) - t^{start}(1)$, after N generations as,

$$E(TTX) = NT/K \tag{6.6}$$

Equation 6.6 describes that the TTX, in the case of combination 1, is directly proportional to the number of generations and task length and inversely proportional to the core performance.

We use our emulator described in §6.3 to observe the TTX as a function of the number of tasks, core performance, binding strategies and sorting and binding criteria for a homogeneous and static workload and homogeneous and dynamic resource. We specify the workload and resource parameters used for our experiments in Table 6.2.

For each of row of parameters described in Table 6.2, we employ both the early- and late-binding strategies. For each strategy, we employ two criteria: random, which sorts and binds tasks randomly, and L2FF which places the longest task on the core that will finish executing the task first. Thus, each experiment mentioned in Table 6.2 is executed in four scenarios: (i) early binding with random criteria; (ii) late binding with random criteria; (iii) early binding with L2FF criteria; and (iv) late binding with L2FF criteria. The results with random criteria serve as a baseline for our analysis.

Table 6.2: Parameters to investigate combination 1 using the emulator

Exp ID	**Fig ID**	**Task length**	**Core performance**	**# Tasks**	**# Cores**
1	6.4	512	16	128	128
		512	16	256	128
		512	16	512	128
		512	16	1024	128
2	6.5	512	32	128	128
		512	32	256	128
		512	32	512	128
		512	32	1024	128
3	6.6	512	64	128	128
		512	64	256	128
		512	64	512	128
		512	64	1024	128

We present the results obtained in figures 6.4, 6.5 and 6.6 and discuss three insights learned. We see that when, both, workload and resource are homogeneous and static,

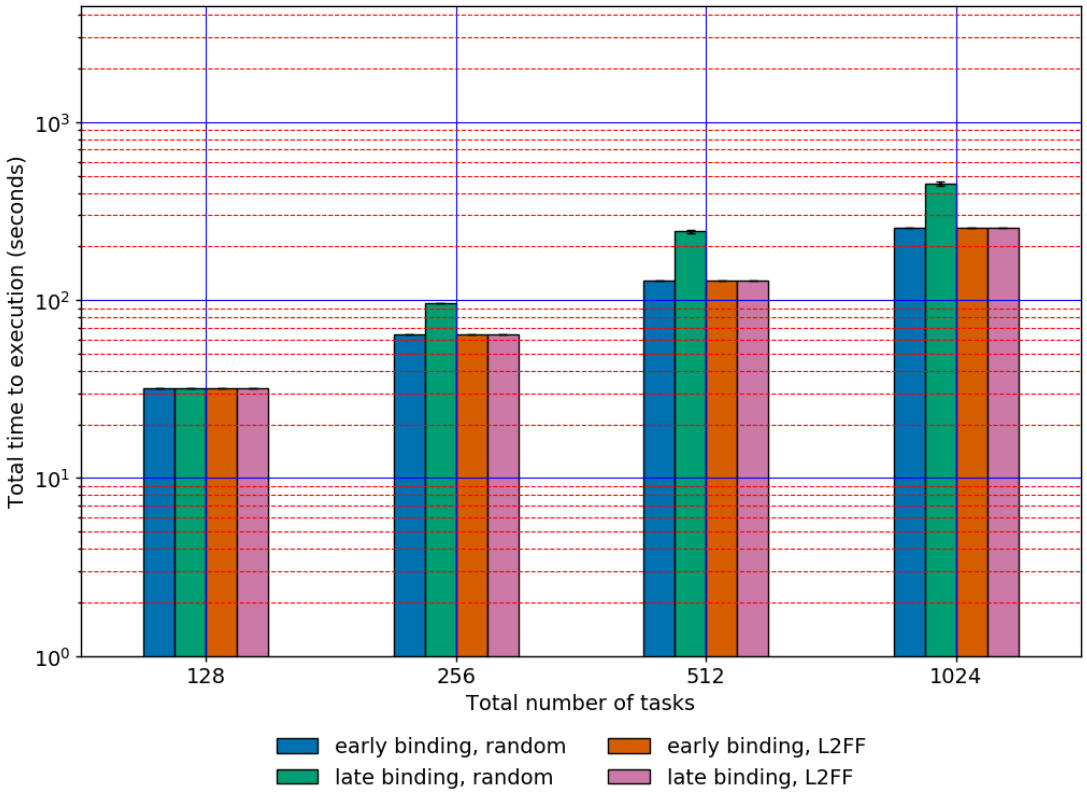

Figure 6.4: Experiment 1: TTX as a function of the total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 512, Core performance (K) = 16, Number of cores (R) = 128

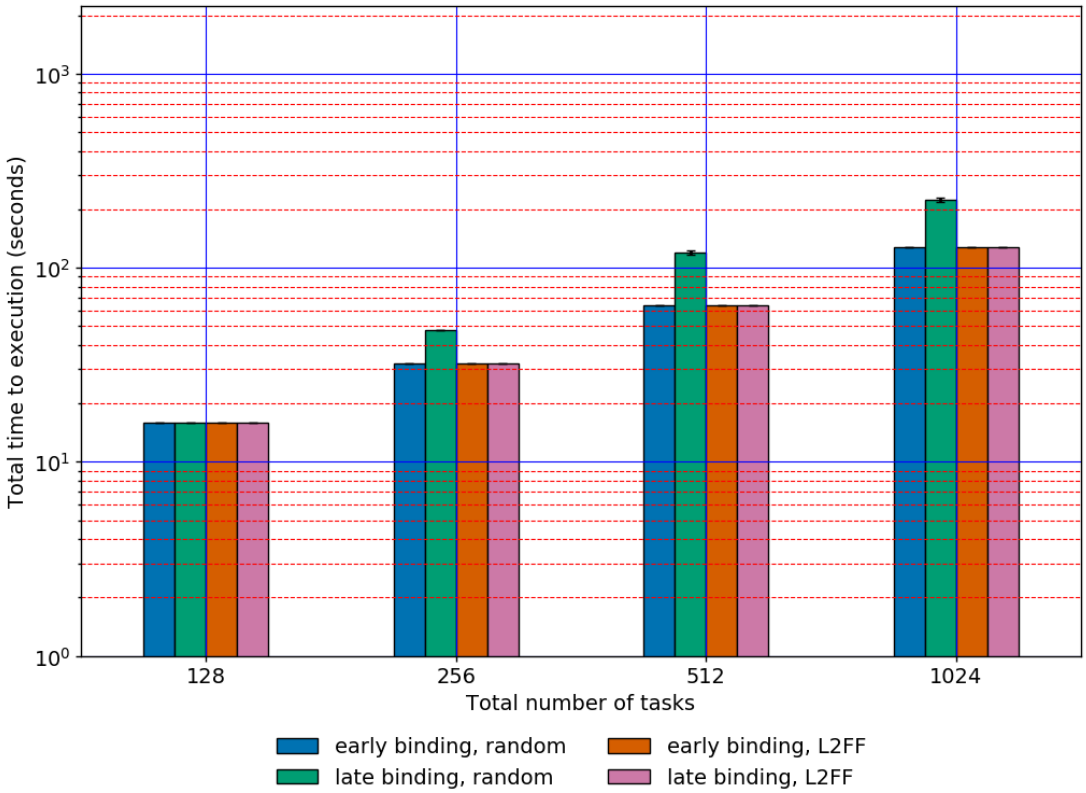

Figure 6.5: Experiment 2: TTX as a function of the total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 512, Core performance (K) = 32, Number of cores (R) = 128

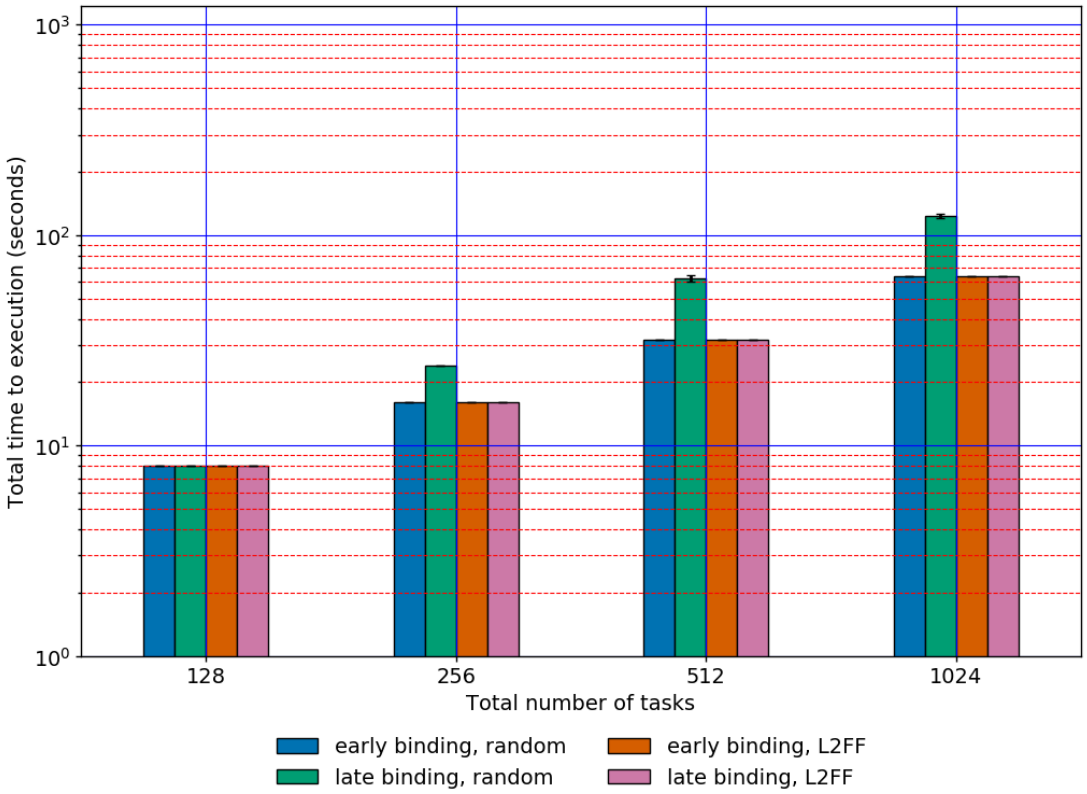

Figure 6.6: Experiment 3: TTX as a function of the total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 512, Core performance (K) = 64, Number of cores (R) = 128

there is no difference in TTX between early-binding with random criteria, early-binding with L2FF criteria and late-binding with L2FF criteria. This is explained by the fact that all tasks and cores have the same properties over time and space and execution time of any task on any core at any time is the same. Note that late-binding with random criteria has higher TTX than the other three as the random sorter and binder has no memory between generations and may place multiple tasks on to the same cores, staggering the resulting TTX.

Secondly, we observe that the TTX increases with increase in the number of tasks which is explained by the increased number of generations as the number of cores is constant. Thirdly, we observe that the TTX reduces with increase in the core performance which is explained by the decrease in the execution time of a task when placed on a core with higher performance.

Although not presented in the experiments above, we can also confirm that, for emulation of combination 1, the TTX varies linearly with the task length explained by the increase (or decrease) in the number of operations to be performed on a core with same performance. All these observations are consistent with equation 6.6 and our emulator matches the expectation for our base case, combination 1.

6.4.2 Combination 2: Homogeneous and Static Workload, Homogeneous and Dynamic Resource

In combination 2, the workload is homogeneous and static and the resource is homogeneous and dynamic. Let the uniform distribution which is sampled for task lengths be between (s_{low}, s_{high}) and the uniform distribution which is sampled at different times during execution for core performance be between (k_{low}, k_{high}). As there is no heterogeneity in the workload, the uniform distribution sampled for task length has no width, i.e., $s_{high} - s_{low} = 0$. Let the length of all tasks be S, i.e., $S = s_{high} = s_{low}$ and the mean performance of all cores be K, i.e., $K = (k_{high} + k_{low})/2$.

Lets say that there are R cores and S tasks such that $S > R$ and we will have N generations where $N = S/R$ when R tasks are executing concurrently. Let $t^{start}(g)$ be the time at which the first task of generation g starts execution. Let $t^{stop}(g)$ be the time

at which the last task of generation g stops execution. Then we can represent the start and stop times at the end of first generation as,

$$t^{start}(1) = 0,\ 0,\ 0,\ldots.\ \text{R instances} \tag{6.7}$$

$$t^{stop}(1) = \text{T/K},\ \text{T/K},\ \text{T/K},\ \ldots.\ \text{R instances} \tag{6.8}$$

In the second generation, the value of the resource performance is sampled from the uniform distribution as the resources are dynamic. Thus, at the end of the second generation we have,

$$t^{start}(2) = \text{T/K},\ \text{T/K},\ \text{T/K},\ldots.\ \text{R instances} \tag{6.9}$$

$$t^{stop}(2) = \text{T/K} + \text{T}/k_1^2,\ \text{T/K} + \text{T}/k_2^2,\ \text{T/K} + \text{T}/k_3^2,\ \ldots.,\ \text{T/K} + \text{T}/k_r^2 \tag{6.10}$$

where k_i^j indicates the performance value of ith core after j generations.

Similarly, after N generations,

$$\begin{aligned} t^{stop}(N) = \text{T/K} + \text{T}/k_1^2 + \ldots. + \text{T}/k_1^N,\ \text{T/K} + \text{T}/k_2^2 + \ldots. + \text{T}/k_2^N,\\ \text{T/K} + \text{T}/k_3^2 + \ldots. + \text{T}/k_3^N,\ \ldots.,\ \text{T/K} + \text{T}/k_r^2 + \ldots. + \text{T}/k_r^N \end{aligned} \tag{6.11}$$

We can estimate our TTX, $t^{stop}(N) - t^{start}(1)$ after N generations as a sum of independent variables drawn of a uniform distribution, which would be a Irwin-Hall distribution [148, 149], a distribution that tends towards a normal distribution based on the number of random variables summed. Its mean and standard deviation can be formulated as,

$$E(TTX) = NT/K \tag{6.12}$$

$$Stddev(TTX) = T * (b - a) * \sqrt{(N-1)/12} \tag{6.13}$$

where $a = 1/r_{high}$ and $b = 1/r_{low}$.

Equation 6.12 describes that the mean TTX, in the case of combination 2, is directly proportional to the number of generations and task length and inversely proportional to the core performance. Equation 6.13 describes that the space, around the mean TTX, in which the actual TTX lies grows linearly with the task length, number of generations and the dynamism of the resource.

We use our emulator described in §6.3 to observe the TTX as a function of the number of tasks, mean core performance, binding strategies and sorting and binding criteria for a homogeneous and static workload and homogeneous and dynamic resource. We specify the workload and resource parameters used for our experiments in Table 6.3

For each of row of parameters described in Table 6.3, we employ both the early- and late-binding strategies. For each strategy, we employ two criteria: random, which sorts and binds tasks randomly, and L2FF which places the longest task on the core that will finish executing the task first. Thus, each experiment mentioned in Table 6.3 is executed in four scenarios: (i) early binding with random criteria; (ii) late binding with random criteria; (iii) early binding with L2FF criteria; and (iv) late binding with L2FF criteria. The results with random criteria serve as a baseline for our analysis.

We present the results obtained in figures 6.7, 6.8 and 6.9 and discuss four insights learned. Firstly, we see that when workload is homogeneous and static and resource is homogeneous and dynamic, there is considerable difference in TTX between early- and late-binding strategies with different criteria. The TTX when using early-binding with random criteria, late-binding with random criteria and early-binding with L2FF, increases with increase in resource dynamism. This behavior is explained as these three do not consider the changes in the core performance and can place tasks on sub-optimal cores. When the resource dynamism is not considered in making scheduling decisions, early-binding with L2FF criteria essentially behaves like early-binding with random criteria after the first generation. This can be seen in the similarity of the TTX of the

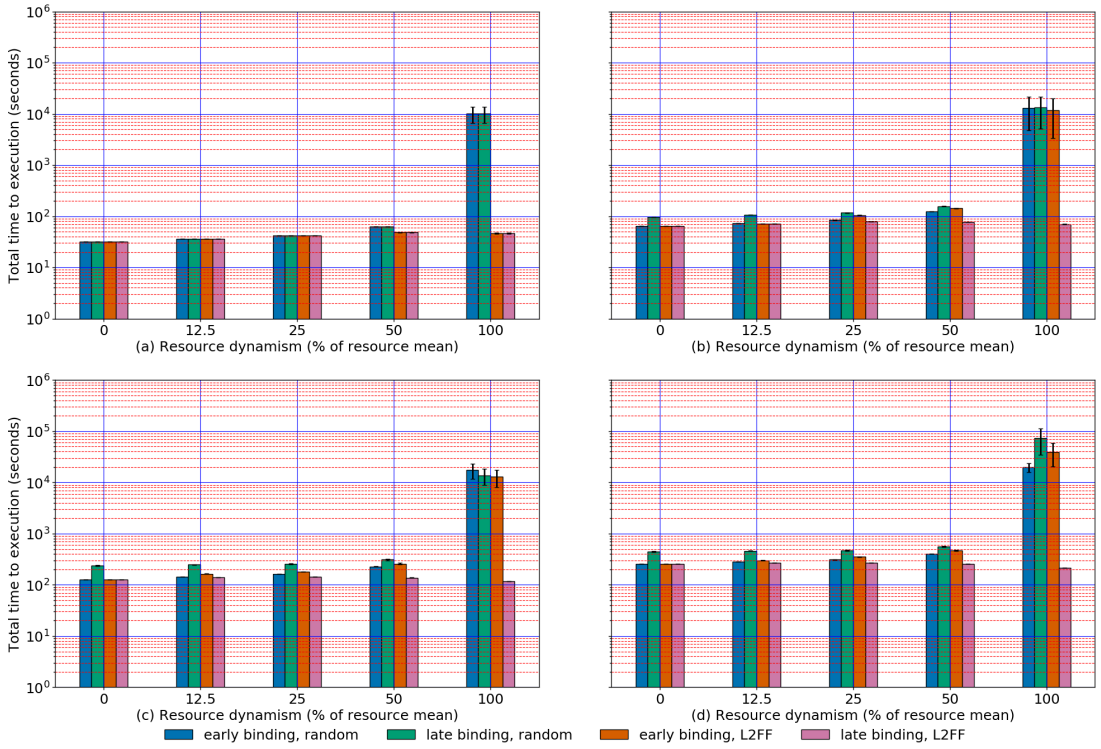

Figure 6.7: Experiment 1: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 512, Mean core performance (K) = 16, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024(d)

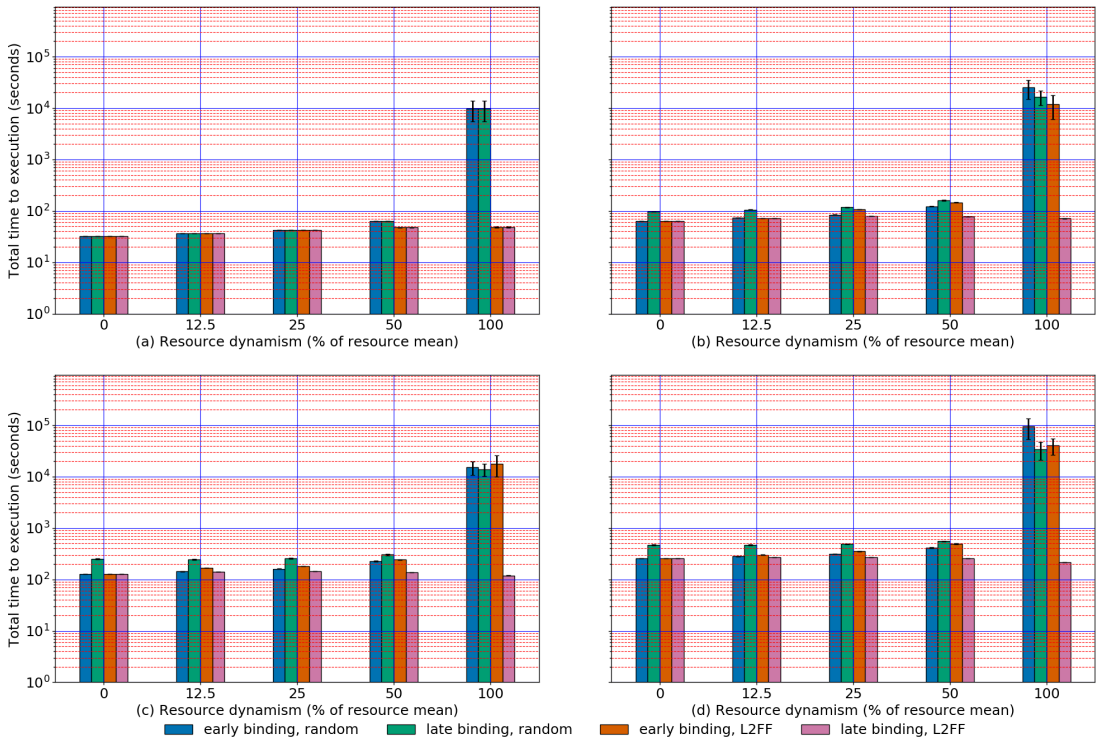

Figure 6.8: Experiment 2: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 1024, Mean core performance (K) = 32, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024(d)

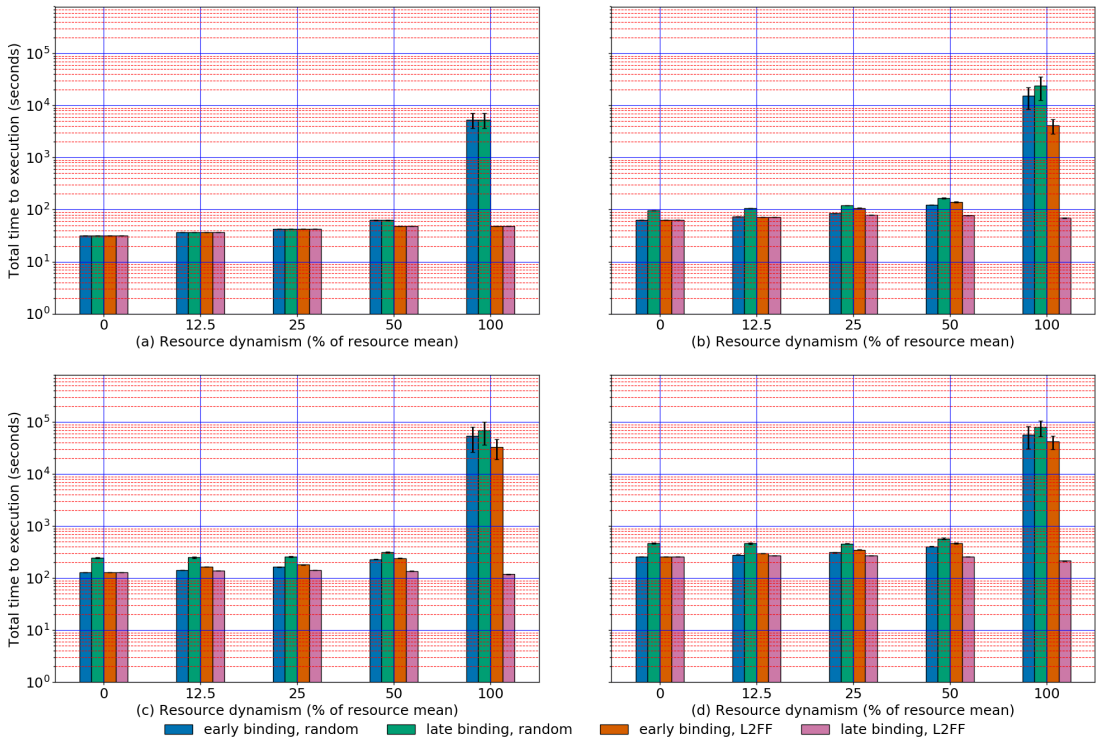

Figure 6.9: Experiment 3: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Number of operations performed by a task (T) = 2048, Mean core performance (K) = 64, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024(d)

Table 6.3: Parameters to investigate combination 2 using the emulator

Exp ID	Fig ID	Task length	Core performance	# Tasks	# Cores
1	6.7 (a)	512	16	128	128
	6.7 (b)	512	16	256	128
	6.7 (c)	512	16	512	128
	6.7 (d)	512	16	1024	128
2	6.8 (a)	1024	32	128	128
	6.8 (b)	1024	32	256	128
	6.8 (c)	1024	32	512	128
	6.8 (d)	1024	32	1024	128
3	6.9 (a)	2048	64	128	128
	6.9 (b)	2048	64	256	128
	6.9 (c)	2048	64	512	128
	6.9 (d)	2048	64	1024	128

two early-binding strategies.

Late-binding with L2FF however shows much lower TTX than the other three approaches. This is explained by the fact the late-binding strategy ensures use of the latest core performance of the dynamic resource and the L2FF criteria places tasks on cores based on its performance. Using late-binding with L2FF ensures that more tasks are executed on faster cores to minimize the overall TTX. It can be seen that this approach provides $0.1x - 100x$ better TTX compared to the other 3 for the parameters in Table 6.3.

Secondly, we observe that the TTX for late-binding with L2FF is consistently lower at 100% than at 50%. This is explained by the fact that with increase in dynamism, there are cores with even greater performance than when dynamism is at 50% and the L2FF ensures that more tasks are executed on these faster cores reducing the overall TTX.

Thirdly, we observe that the TTX increases with increase in the number of tasks which is explained by the increased number of generations as the number of cores is constant. Although not presented in our experiments, we can confirm that the TTX varied proportionally to the task length and inversely with the core performance. These behavior are consistent with our expectations based on equation 6.12.

Finally, across the three experiments we vary values of task length but keep the ratio

of task length to mean core performance constant, which equates to constant mean time to execute a task. We observe that the TTX increases with increase in the task length, although marginally. This is consistent with our expectations based on equation 6.13 as the possibility of a higher TTX increases with the task length. Experiments with greater task lengths can be performed to observe greater fluctuations.

Using the theoretical and empirical evaluation via the emulator, we explain that the execution of homogeneous and static workloads on homogeneous and dynamic resources can benefit from using the late-binding strategy with the L2FF criteria as it leads to a reduction the TTX.

6.4.3 Combination 3: Heterogeneous and Static Workload, Homogeneous and Dynamic Resource

In combination 3, the workload is heterogeneous and static, resource is homogeneous and dynamic. Let the uniform distribution which is sampled for task lengths be between (s_{low}, s_{high}) and the uniform distribution which is sampled at different times during execution for core performance be between (k_{low}, k_{high}). Let the mean length of tasks be S, i.e., $S = (s_{high} + s_{low})/2$ and the mean performance of all cores be K, i.e., $K = (k_{high} + k_{low})/2$.

Lets say that there are R cores and S tasks such that $S > R$and we have N generations where $N = S/R$ when R tasks executing concurrently. Let $t^{start}(g)$ be the time at which the first task of generation g starts execution. Let $t^{stop}(g)$ be the time at which the last task of generation g stops execution.

The estimate of TTX, $t^{stop}(N) - t^{start}(1)$ after N generations can be described by,

$$E(TTX) = NT/K \tag{6.14}$$

$$Stddev(TTX) = T * (d - c) * \sqrt{(N-1)/12} \tag{6.15}$$

where $c = s_{low}/r_{high}$ and $d = s_{high}/r_{low}$.

Similar to case 2, equation 6.14 describes that the mean TTX, in the case of combination 3, is directly proportional to the number of generations and task length and inversely proportional to the core performance. Equation 6.15 describes that the space, around the mean TTX, in which the actual TTX lies grows linearly with the task length, number of generations, heterogeneity of the workload and the dynamism of the resource.

We use our emulator described in § 6.3 to observe the TTX as a function of the number of tasks, mean core performance, binding strategies and sorting and binding criteria for a heterogeneous and static workload and homogeneous and dynamic resource. We specify the workload and resource parameters used for our experiments in Table 6.4.

Table 6.4: Parameters to investigate combination 3 using the emulator

Exp ID	**Fig ID**	**Workload heterogeneity**	**Task length**	**Core perf.**	**# Tasks**	**# Cores**
1	6.10 (a)	12.5%	512	16	128	128
	6.10 (b)		512	16	256	128
	6.10 (c)		512	16	512	128
	6.10 (d)		512	16	1024	128
2	6.11 (a)	25%	512	16	128	128
	6.11 (b)		512	16	256	128
	6.11 (c)		512	16	512	128
	6.11 (d)		512	16	1024	128
3	6.12 (a)	50%	512	16	128	128
	6.12 (b)		512	16	256	128
	6.12 (c)		512	16	512	128
	6.12 (d)		512	16	1024	128
4	6.13 (a)	100%	512	16	128	128
	6.13 (b)		512	16	256	128
	6.13 (c)		512	16	512	128
	6.13 (d)		512	16	1024	128

For each of row of parameters described in Table 6.4, we employ both the early- and late-binding strategies. For each strategy, we employ two criteria: random, which sorts and binds tasks randomly, and L2FF which places the longest task on the core that will finish executing the task first. Thus, each experiment mentioned in Table 6.4 is executed in four scenarios: (i) early binding with random criteria; (ii) late binding with

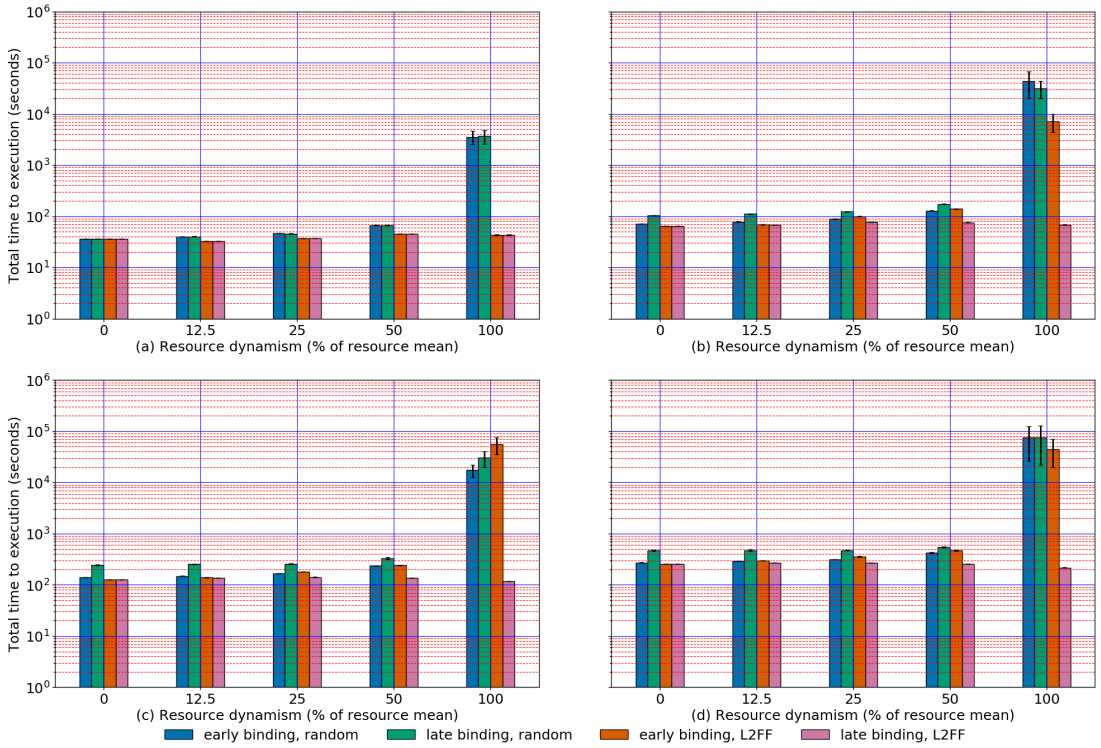

Figure 6.10: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Mean number of operations performed by a task (T) = 512, Workload heterogeneity = 12.5% of T, Mean core performance (K) = 16, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024 (d)

random criteria; (iii) early binding with L2FF criteria; and (iv) late binding with L2FF criteria. The results with random criteria serve as a baseline for our analysis.

We present the results obtained in figures 6.10, 6.11, 6.12 and 6.13 and discuss four insights learned. Firstly, we see that when workload is heterogeneous and static and resource is homogeneous and dynamic, there is considerable difference in TTX between early- and late-binding strategies with the different criteria. Similar to the behavior in combination 2, the TTX when using early-binding with random criteria, late-binding with random criteria and early-binding with L2FF, increases with increase in resource dynamism. This behavior is explained as these three do not consider the changes in the core performance and can place tasks on sub-optimal cores.

Late-binding with L2FF however shows much lower TTX than the other three approaches. This is explained by the fact the late-binding strategy ensures use of the

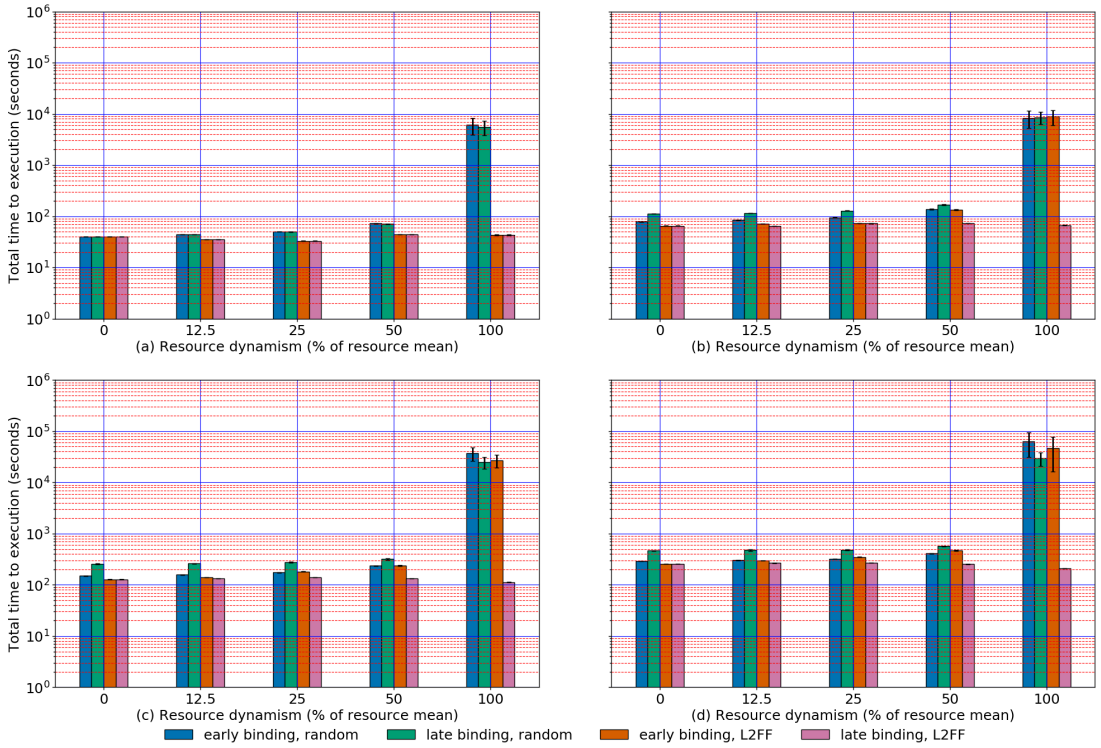

Figure 6.11: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Mean number of operations performed by a task (T) = 512, Workload heterogeneity = 25% of T, Mean core performance (K) = 16, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024 (d)

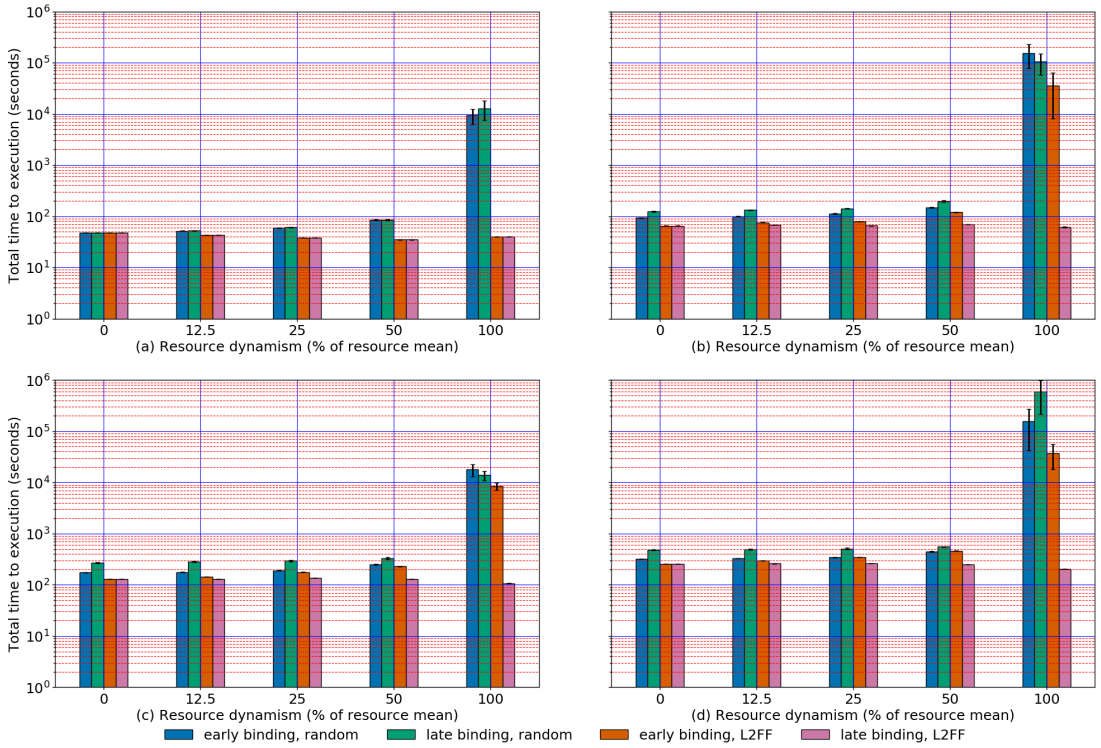

Figure 6.12: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Mean number of operations performed by a task (T) = 512, Workload heterogeneity = 50% of T, Mean core performance (K) = 16, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024 (d)

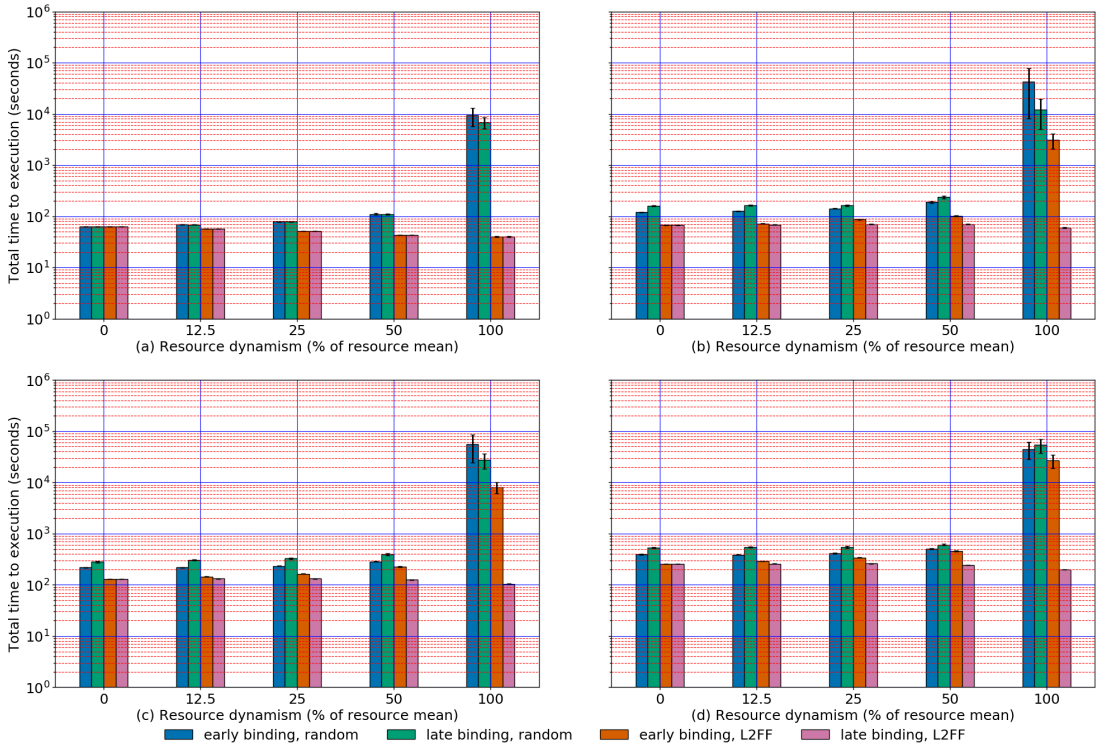

Figure 6.13: TTX as a function of resource dynamism and total number of tasks for early- and late-binding strategies with random and L2FF criteria. Mean number of operations performed by a task (T) = 512, Workload heterogeneity = 100% of T, Mean core performance (K) = 16, Number of cores (R) = 128, Total number of tasks = 128(a), 256(b), 512(c) and 1024 (d)

latest core performance of the dynamic resource and the L2FF criteria places tasks on cores based on its performance. Using late-binding with L2FF ensures that more tasks are executed on faster cores to minimize the overall TTX. It can be seen that this approach provides $0.1x100x$ better TTX compared to the other 3 for the parameters in Table 6.4.

Secondly, we observe that the TTX for late-binding with L2FF is consistently lower at 100% than at 50%. This is explained by the fact that with increase in dynamism, there are cores with even greater performance than when dynamism is at 50% and the L2FF ensures that more tasks are executed on these faster cores reducing the overall TTX. This reduction can be seen also when the workload heterogeneity increases from 50% to 100% and can explained by the presence of shorter tasks when heterogeneity is at 100% when compared to the tasks at 50%.

Thirdly, across the four experiments we vary the workload heterogeneity and, for the same value of resource dynamism, we can see that the TTX increases with increase in the workload heterogeneity when using early-binding with random criteria, late-binding with random criteria and early-binding with L2FF criteria. This is explained by the fact that there are increasing number of long tasks when workload heterogeneity increases and these tasks placed on sub-optimal cores. In the case of late-binding with L2FF criteria, the TTX does not vary with increase in workload heterogeneity as the longer tasks are always optimally placed.

Finally, as in the case of combinations 1 and 2, we observe that the TTX increases with increase in the number of tasks which is explained by the increased number of generations as the number of cores is constant. Although not presented in our experiments, we can confirm that the TTX varied proportionally to the task length and inversely with the core performance. These behavior are consistent with our expectations based on equations 6.14 and 6.15.

Using the theoretical and empirical evaluation via the emulator, we explain that the execution of heterogeneous and static workloads on homogeneous and dynamic resources can benefit from using the late-binding strategy with the L2FF criteria as it leads to a reduction the TTX.

6.5 Impact and challenges

In the experiments above, we have shown that the total TTX can be heavily affected by workload heterogeneity and resource dynamism. Through theoretical and empirical analysis, we showed that our approach of late-binding with the L2FF criteria can be used to manage the effect of heterogeneity and dynamism on TTX. We believe that the results described above, insight derived from the results and the capabilities of the emulator benefit both end users and system developers.

In our experiments, late-binding strategy with the L2FF criteria showed $0.1x - 100x$ better total TTX compared to early-binding strategies when the heterogeneity and dynamism were varied between 12.5%-100%. The improvement was achieved for the same number of cores in all experiments, which, in the context of HPC systems, translates to efficient resource utilization. This is because in HPC systems, the complete resource is held for TTX and is released only after the last task finishes executes, thus, lower the TTX lower the resource utilization. The values used for task length and core performance were representative, but the performance improvements obtained in TTX can translate into hours and days in real-world applications and the gains in resource utilization could enable longer ensemble applications or a campaign of ensemble applications for the same resource budget.

Conversely, using the capabilities of the emulator and results derived above, developers of workload management systems can argue, prior to investing time in developing these systems, whether sophisticated late-binding strategies and different criteria are useful for an application and a HPC system given information regarding the heterogeneity of the workload and dynamism of the resources.

We foresee two main challenges in implementing late-binding strategies and using them in practice: (1) tools and services to measure workload heterogeneity and resource dynamism may not be available, and if available, may not be accurate; and (2) the time to make the scheduling decisions needs to be factored while making the decision. Both of these challenges were assumed to be addressed in our approach in order to simplify the problem space. Our strategy discussed and the emulator developed here provide

important insight and are pre-cursory before developing workload management systems to be used in practice.

6.6 Summary

In this final chapter, we discussed the heterogeneity that exists in workloads derived from ensemble applications and the dynamism in the performance of the resources on HPC systems. We discussed the implications of not accounting for heterogeneity and dynamism while making scheduling decisions. We described the current solutions to this problem, their drawbacks and proposed the late-binding strategy that delays the decision to decide when and where tasks of a workload need to be executed in order to minimize the TTX. We performed a theoretical evaluation of the valid cases and derived formula to depict the space in which the TTX may lie. We developed an emulator to investigate and compare the late-binding strategy against early-binding for different criteria.

Our theoretical and experimental analysis showed that the TTX is proportional to the level of heterogeneity, level of dynamism, total number of tasks to be executed, number of available cores, task length and core performance. Our experiments showed that using the late-binding strategy with L2FF criteria, that places the longest task on the cores on which they finish first, provides $0.1x - 100x$ reduction in total TTX compared to early-binding strategies when the heterogeneity and dynamism are between 12.5%-100%.

Chapter 7

Conclusion

In this dissertation, we presented a body of research in middleware advancing the state-of-the-art to enable the programmability and execution of scientific applications on HPC systems abstracting the complexity of resource and execution management. This enables the domain scientists to focus on algorithmic innovations and computational campaigns of larger scale solving bigger problems without having to manage resource and task execution aspects. Specifically, this research makes the following contributions.

A specific class of scientific applications called ensemble applications is described and their significance in advancing the status-quo in various scientific domains is explained. Requirements and challenges are identified based on five different science drivers and a generic software framework, EnTK, is designed and developed to address missing capabilities in existing solutions. The framework offers a programming model focused on expressing ensemble applications and abstracts resource and execution complexities from the user. The framework provides fault tolerance, minimal overheads, linear strong and weak scaling up to O(1000) tasks on leadership-class HPC systems at NSF and DoE. Complying with the building blocks approach the framework is designed to integrate with different task execution systems based on the requirements of the application. The scientific research activities enabled by EnTK are described signifying the impact of the programming model and execution capabilities offered by EnTK.

Advancement in scientific applications to solve problems that ensemble applications cannot solve have come in the form of adaptive ensemble applications where the science drivers require the capability to modify the application workflow during runtime based on the intermediate results obtained during execution. These adaptive approaches have been shown to improve scientific results by orders of magnitude in theory or

at small scale. Supporting such solutions at production-scale requires capabilities to reliably adapt workflows without interrupting execution and automation to avoid human involvement required when at scale, both of which are not available in current solutions. In response to these missing capabilities, we identified the types of adaptivity common in ensemble applications and enhanced EnTK using three science drivers with adaptive requirements. We characterized the adaptivity overhead to show that it is orders of magnitude lesser than the TTX of these applications, validated the implementation of two adaptive ensemble applications and described the advances enabled by our efforts in multiple scientific domains.

Workloads derived from adaptive and non-adaptive ensemble applications may be homogeneous or heterogeneous and performance of resources on HPC systems can be largely dynamic in nature. The effect of heterogeneity and dynamism in workloads and resources on the TTX and resource utilization is not closely investigated due to lack of necessary tools and services. We develop a workload management emulator to evaluate these scenarios and perform a theoretical and empirical evaluation to show their effects on TTX. We propose a late-binding strategy with 'L2FF' scheduling criteria that can reduce the TTX by $0.1x - 100x$ compared to other common strategies when workload heterogeneity and resource dynamism varies between 12.5%-100%. We discuss the impact of the insight derived through the theoretical and empirical evaluation of results obtained when using the emulator and mention the challenges in implementing such a workload manager and strategies in practice.

Our research advances the state-of-the-art of middleware tools providing the capabilities to solve the scientific problems of today and tomorrow in multiple scientific domains.

7.1 Future Work

Our solutions presented in this dissertation are largely invariant to the scientific domain in which it is utilized and portable to different HPC systems due to, both, the current task execution system used in EnTK as well as the building blocks design approach

which allows EnTK to integrated with other task execution systems. Coupled with our insights derived using our workload management emulator, we foresee four avenues of research and development that can be pursued going forward.

Currently, EnTK coupled with RP enable scalable execution up to O(1000) tasks on several HPC systems. With the advent of HPC systems for pre-exascale and exascale computing, we will soon require scaling up to O(10^4)-O(10^6) tasks. In preparation for this requirement, research efforts need to be made into optimize existing or developing new task execution systems that can reach such scales. Due to the building block approach employed, EnTK can be integrated with these systems through standardized interfaces. This would enable existing applications and domain specific workflow systems developed using EnTK to transparently scale to exascale.

A second avenue of research to reach greater scales of execution is to enable the concurrent use of multiple resources on the same or different HPC systems. This would require enhancing EnTK to acquire and manage multiple resources as well as research into the efficient and reliable scheduling and execution of tasks on these resources.

A third avenue of future research would be in the development of a workload management system that offers late-binding strategies to schedule workloads on to resources. We believe our strategies evaluated and the emulator developed in this research provide important insight and are pre-cursory for developing a workload management system that can be used in practice. As mentioned in § 6.5, such a system can be useful to end users who may benefit from reduced TTX and improved resource utilization.

Finally, a very useful addition to this body of work would be to extend the current capabilities to cloud based systems. Owing to the building blocks approach employed, support cloud systems can be enabled by switch the runtime system used by EnTK. Cloud systems with no queue times have the potential to serve as test beds for applications at small scales before moving to production-scales on HPC systems, complimenting the capabilities of an HPC system.

References

[1] Don Johnston. Hpc matters to our quality of life and prosperity. *Scientific Computing*, 2014. http://www.scientificcomputing.com/article/2014/11/hpcmatters-our-quality-life-and-prosperity.

[2] Stephen J Ezell and Robert D Atkinson. The vital importance of high-performance computing to us competitiveness. *Information Technology and Innovation Foundation, April*, 28, 2016.

[3] IDC. High performance computing in the eu: Progress on the implementation of the european hpc strategy report. *Brussels: European Commission DG Communications Networks, Content and Technology, 2015*, 2015. http://knowledgebase.e-irg.eu/documents/243153/246094/High+Performance+Computing+in+the+EU+-+Progress+on+the+Implementation+of+the+European+HPC+Strategy.pdf/b0adf617-3f50-4a6f-9217-4e0fbb5edd09.

[4] Amber Harmon. Hpc matters: Funding, collaboration, innovation. *ScienceNode*, November 2014. https://sciencenode.org/feature/hpc-matters-funding-collaboration-innovation.php.

[5] Dimitri Komatitsch and Jeroen Tromp. Spectral-element simulations of global seismic wave propagationi. validation. *Geophysical Journal International*, 149(2):390–412, 2002.

[6] Guido Cervone, Laura Clemente-Harding, Stefano Alessandrini, and Luca Delle Monache. Short-term photovoltaic power forecasting using artificial neural networks and an analog ensemble. *Renewable Energy*, 108:274–286, 2017.

[7] Gianni D'Angelo and Salvatore Rampone. Towards a hpc-oriented parallel implementation of a learning algorithm for bioinformatics applications. *BMC bioinformatics*, 15(5):S2, 2014.

[8] Marco Caccin, Zhenwei Li, James R Kermode, and Alessandro De Vita. A framework for machine-learning-augmented multiscale atomistic simulations on parallel supercomputers. *International Journal of Quantum Chemistry*, 115(16):1129–1139, 2015.

[9] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[10] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[11] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual international symposium on computer architecture (ISCA)*, pages 365–376. IEEE, 2011.

[12] David J Frank, Robert H Dennard, Edward Nowak, Paul M Solomon, Yuan Taur, and Hon-Sum Philip Wong. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, 2001.

[13] Thomas N Theis and Paul M Solomon. In quest of the next switch: prospects for greatly reduced power dissipation in a successor to the silicon field-effect transistor. *Proceedings of the IEEE*, 98(12):2005–2014, 2010.

[14] XSEDE. `https://portal.xsede.org/`.

[15] ORNL. `https://www.olcf.ornl.gov/computing-resources`.

[16] Tacc frontera (accessed may 2019). `https://www.tacc.utexas.edu/systems/frontera`.

[17] Ornl summit (accessed may 2019). `https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/`.

[18] Susanna Röblitz and Marcus Weber. Fuzzy spectral clustering by pcca+: application to markov state models and data classification. *Adv. Data. Anal. Classi.*, 7(2):147–179, 2013.

[19] S. Doerr and G. De Fabritiis. On-the-fly learning and sampling of ligand binding by high-throughput molecular simulations. *J. Chem. Theory Comput.*, 10(5):2064–2069, 2014.

[20] Gregory R Bowman, Daniel L Ensign, and Vijay S Pande. Enhanced modeling via network theory: adaptive sampling of markov state models. *J. Chem. Theory Comput.*, 6(3):787–794, 2010.

[21] John D Chodera and Frank Noé. Markov state models of biomolecular conformational dynamics. *Current opinion in structural biology*, 25:135–144, 2014.

[22] Frank Noé, Christof Schütte, Eric Vanden-Eijnden, Lothar Reich, and Thomas R Weikl. Constructing the equilibrium ensemble of folding pathways from short off-equilibrium simulations. *Proceedings of the National Academy of Sciences*, 106(45):19011–19016, 2009.

[23] J. Virieux and S. Operto. An overview of full-waveform inversion in exploration geophysics. *GEOPHYSICS*, 74(6):WCC1–WCC26, November 2009.

[24] Hui Wan, Philip J Rasch, Kai Zhang, Yun Qian, Huiping Yan, and Chun Zhao. Short ensembles: an efficient method for discerning climate-relevant sensitivities in atmospheric general circulation models. *Geoscientific Model Development*, 7(5):1961–1977, 2014.

[25] Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015.

[26] G Cervone, P Franzese, Y Ezber, and Z Boybeyi. Risk assessment of atmospheric emissions using machine learning. *Natural Hazards and Earth System Sciences*, 8(5):991–1000, 2008.

[27] S Alessandrini, L Delle Monache, S Sperati, and G Cervone. An analog ensemble for short-term probabilistic solar power forecast. *Applied energy*, 157:95–110, 2015.

[28] James M Murphy, David MH Sexton, David N Barnett, Gareth S Jones, Mark J Webb, Matthew Collins, and David A Stainforth. Quantification of modelling uncertainties in a large ensemble of climate change simulations. *Nature*, 430(7001):768, 2004.

[29] Jeffrey Martin, Vincent M Bruno, Zhide Fang, Xiandong Meng, Matthew Blow, Tao Zhang, Gavin Sherlock, Michael Snyder, and Zhong Wang. Rnnotator: an automated de novo transcriptome assembly pipeline from stranded RNA-Seq reads. *BMC genomics*, 11(1):663, 2010.

[30] Thomas E Cheatham III and Daniel R Roe. The impact of heterogeneous computing on workflows for biomolecular simulation and analysis. *Computing in Science & Engineering*, 17(2):30–39, 2015.

[31] Marc Snir, William Gropp, Steve Otto, Steven Huss-Lederman, Jack Dongarra, and David Walker. *MPI–the Complete Reference: The MPI core*, volume 1. MIT press, 1998.

[32] Matteo Turilli, Vivek Balasubramanian, Andre Merzky, Ioannis Paraskevakos, and Shantenu Jha. Middleware building blocks for workflow systems. *Computing in Science & Engineering (CiSE) special issue on Incorporating Scientific Workflows in Computing Research Processes*, forthcoming.

[33] Yuji Sugita and Yuko Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical physics letters*, 314(1):141–151, 1999.

[34] Shunzhou Wan, Agastya P Bhati, Stefan J Zasada, Ian Wall, Darren Green, Paul Bamborough, and Peter V Coveney. Rapid and reliable binding affinity prediction of bromodomain inhibitors: a computational study. *Journal of chemical theory and computation*, 13(2):784–795, 2017.

[35] Sethuraman Sankaran and Alison L Marsden. A stochastic collocation method for uncertainty quantification and propagation in cardiovascular simulations. *Journal of biomechanical engineering*, 133(3):031001, 2011.

[36] Thomas M Hamill, Michael J Brennan, Barbara Brown, Mark DeMaria, Edward N Rappaport, and Zoltan Toth. Future of ensemble-based hurricane forecast products. *Bull. Amer. Meteor. Soc*, 2010.

[37] Ebru Bozda, Daniel Peter, Matthieu Lefebvre, Dimitri Komatitsch, Jeroen Tromp, Judith Hill, Norbert Podhorszki, and David Pugmire. Global adjoint tomography: first-generation model. *Geophysical Journal International*, 207(3):1739–1766, 2016.

[38] Nicolas Cherpeau, Guillaume Caumon, and Bruno Lévy. Stochastic simulation of fault networks from 2d seismic lines. In *SEG Technical Program Expanded Abstracts 2010*, pages 2366–2370. Society of Exploration Geophysicists, 2010.

[39] Nina Singhal and Vijay S Pande. Error analysis and efficient sampling in markovian state models for molecular dynamics. *J. Chem. Phys.*, 123(20):204909, 2005.

[40] Jeffrey K Weber and Vijay S Pande. Characterization and rapid sampling of protein folding markov state model topologies. *J. Chem. Theory Comput.*, 7(10):3405–3411, 2011.

[41] Zahra Shamsi, Alexander S. Moffett, and Diwakar Shukla. Enhanced unbiased sampling of protein dynamics using evolutionary coupling information. *Sci. Rep.*, 7(1):12700, 2017.

[42] Jordane Preto and Cecilia Clementi. Fast recovery of free energy landscapes via diffusion-map-directed molecular dynamics. *Physical Chemistry Chemical Physics*, 16(36):19181–19191, 2014.

[43] Charles A Laughton, Modesto Orozco, and Wim Vranken. Coco: a simple tool to enrich the representation of conformational variability in nmr structures. *Proteins: Structure, Function, and Bioinformatics*, 75(1):206–216, 2009.

[44] Extasy. `https://nsf.gov/awardsearch/showAward?AWD_ID=1265929`.

[45] Mary A Rohrdanz, Wenwei Zheng, Mauro Maggioni, and Cecilia Clementi. Determination of reaction coordinates via locally scaled diffusion map. *The Journal of chemical physics*, 134(12):03B624, 2011.

[46] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[47] Edward C Sherer, Sarah A Harris, Robert Soliva, Modesto Orozco, and Charles A Laughton. Molecular dynamics studies of dna a-tract structure and flexibility. *Journal of the American Chemical Society*, 121(25):5981–5991, 1999.

[48] Stanislaw T Wlodek, Terry W Clark, L Ridgway Scott, and J Andrew McCammon. Molecular dynamics of acetylcholinesterase dimer complexed with tacrine. *Journal of the American Chemical Society*, 119(40):9513–9522, 1997.

[49] Ryan T Modrak, Dmitry Borisov, Matthieu Lefebvre, and Jeroen Tromp. Seisflowsflexible waveform inversion software. *Computers & geosciences*, 115:88–95, 2018.

[50] Robert H Swendsen and Jian-Sheng Wang. Replica monte carlo simulation of spin-glasses. *Physical review letters*, 57(21):2607, 1986.

[51] Romelia Salomon-Ferrer, David A Case, and Ross C Walker. An overview of the amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[52] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[53] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. Gromacs: a message-passing parallel molecular dynamics implementation. *Computer physics communications*, 91(1-3):43–56, 1995.

[54] Hiroaki Fukunishi, Osamu Watanabe, and Shoji Takada. On the hamiltonian replica exchange method for efficient sampling of biomolecular systems: Application to protein structure prediction. *The Journal of chemical physics*, 116(20):9058–9067, 2002.

[55] Yilin Meng and Adrian E Roitberg. Constant ph replica exchange molecular dynamics in biomolecules using a discrete protonation model. *Journal of chemical theory and computation*, 6(4):1401–1412, 2010.

[56] U.s. food and drug administration. hematology oncology (cancer) approvals & safety notifications (accessed may 2019. `https://www.fda.gov/drugs/resources-information-approved-drugs/hematologyoncology-cancer-approvals-safety-notifications`.

[57] Zheng Zhao, Hong Wu, Li Wang, Yi Liu, Stefan Knapp, Qingsong Liu, and Nathanael S Gray. Exploration of type ii binding mode: a privileged approach for kinase inhibitor focused drug discovery? *ACS chemical biology*, 9(6):1230–1241, 2014.

[58] Agastya P. Bhati, Shunzhou Wan, David W. Wright, and Peter V. Coveney. Rapid, accurate, precise, and reliable relative free energy prediction using ensemble based thermodynamic integration. *Journal of Chemical Theory and Computation*, 13(1):210–222, 2017. PMID: 27997169.

[59] Michael Stonebraker. Too much middleware. *ACM Sigmod Record*, 31(1):97–106, 2002.

[60] Giovanni Aloisio, Massimo Cafaro, Daniele Lezzi, and Robert van Engelen. Secure web services with globus gsi and gsoap. *Euro-Par 2003 Parallel Processing*, pages 421–426, 2003.

[61] Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka. Ninf-g: A reference implementation of rpc-based programming middleware for grid computing. *Journal of Grid computing*, 1(1):41–51, 2003.

[62] Mitsuhisa Sato, Taisuke Boku, and Daisuke Takahashi. OmniRPC: a Grid RPC system for parallel programming in cluster and grid environment. In *IEEE/ACM CCGrid*, pages 206–213. IEEE, 2003.

[63] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. myhadoop: Hadoop-on-demand on traditional hpc resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego*, 2011.

[64] William Clay Moody, Linh Bao Ngo, Edward Duffy, and Amy Apon. Jummp: Job uninterrupted maneuverable mapreduce platform. In *IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[65] Magpie. `https://github.com/LLNL/magpie`(accessed December 2017).

[66] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[67] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

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

[69] Leo Goodstadt. Ruffus: a lightweight python library for computational pipelines. *Bioinformatics*, 26(21):2778–2779, 2010.

[70] Erik Gafni, Lovelace J Luquette, Alex K Lancaster, Jared B Hawkins, Jae-Yoon Jung, Yassine Souilmi, Dennis P Wall, and Peter J Tonellato. Cosmos: Python library for massively parallel workflows. *Bioinformatics*, 30(20):2956–2958, 2014.

[71] Kenjiro Taura, Takuya Matsuzaki, Makoto Miwa, Yoshikazu Kamoshida, Daisaku Yokoyama, Nan Dun, Takeshi Shibata, Choi Sung Jun, and Junichi Tsujii. Design and implementation of gxp makea workflow system based on make. *Future Generation Computer Systems*, 29(2):662–672, 2013.

[72] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.

[73] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[74] Shawn Hoon, Kiran Kumar Ratnapu, Jer-ming Chia, Balamurugan Kumarasamy, Xiao Juguang, Michele Clamp, Arne Stabenau, Simon Potter, Laura Clarke, and Elia Stupka. Biopipe: a flexible framework for protocol-based bioinformatics analysis. *Genome Research*, 13(8):1904–1915, 2003.

[75] Sander Pronk, Iman Pouya, Magnus Lundborg, Grant Rotskoff, Bjorn Wesen, Peter M Kasson, and Erik Lindahl. Molecular simulation workflows as parallel algorithms: The execution engine of copernicus, a distributed high-performance computing platform. *Journal of chemical theory and computation*, 11(6):2600–2608, 2015.

[76] Mirko Sonntag, Dimka Karastoyanova, and Frank Leymann. The missing features of workflow systems for scientific computations. *Software Engineering 2010–Workshopband (inkl. Doktorandensymposium)*, 2010.

[77] Valerie Hendrix, James Fox, Devarshi Ghoshal, and Lavanya Ramakrishnan. Tigres workflow library: Supporting scientific pipelines on hpc systems. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 146–155. IEEE, 2016.

[78] Entk github (accessed may 2019). https://github.com/radical-cybertools/radical.entk.

[79] Andre Merzky, Matteo Turilli, Manuel Maldonado, Mark Santcroos, and Shantenu Jha. Using pilot systems to execute many task workloads on supercomputers. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 61–82. Springer, 2018.

[80] Pivotal. Rabbitmq. https://www.rabbitmq.com/ (accessed April 2019).

[81] CloudAMQP. Cloudamqp. https://www.cloudamqp.com/ (accessed April 2019).

[82] Docker. Docker. https://www.docker.com/ (accessed April 2019).

[83] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25, 2015.

[84] V. Balasubramanian, I. Bethune, A. Shkurti, E. Breitmoser, E. Hruska, C. Clementi, C. Laughton, and S. Jha. Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 361–370, Oct 2016.

[85] Ardita Shkurti, Ioanna Danai Styliari, Vivek Balasubramanian, Iain Bethune, Conrado Pedebos, Shantenu Jha, and Charles A Laughton. Coco-md: A simple and effective method for the enhanced sampling of conformational space. *Journal of chemical theory and computation*, 2016.

[86] NCSA Blue Waters. https://bluewaters.ncsa.illinois.edu/.

[87] ORNL Titan. https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/.

[88] Htbac github (accessed may 2019). https://github.com/radical-cybertools/htbac.

[89] Jumana Dakka, Kristof Farkas-Pall, Vivek Balasubramanian, Matteo Turilli, Shunzhou Wan, David W Wright, Stefan Zasada, Peter V Coveney, and Shantenu Jha. Enabling trade-offs between accuracy and computational cost: Adaptive algorithms to reduce time to clinical insight, 2018.

[90] Repex, a Replica-Exchange Framework. https://github.com/SrinivasMushnoori/repex/.

[91] Brooke E. Husic and Vijay S. Pande. Markov state models: From an art to a science. *J. Am. Chem. Soc.*, 140(7):2386–2396, 2018.

[92] Gregory R. Bowman, Daniel L. Ensign, and Vijay S. Pande. Enhanced modeling via network theory: Adaptive sampling of markov state models. *Journal of Chemical Theory and Computation*, 6(3):787–794, 2010.

[93] Alessandro Barducci, Massimiliano Bonomi, and Michele Parrinello. Metadynamics. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 1(5):826–843, September 2011.

[94] Riccardo Chelli and Giorgio F. Signorini. Serial Generalized Ensemble Simulations of Biomolecules with Self-Consistent Determination of Weights. *J. Chem. Theory Comput.*, 8(3):830–842, March 2012.

[95] Jeffrey Comer, James C Phillips, Klaus Schulten, and Christophe Chipot. Multiple-replica strategies for free-energy calculations in namd: multiple-walker adaptive biasing force and walker selection rules. *Journal of chemical theory and computation*, 10(12):5276–5285, 2014.

[96] Lorant Janosi and Manolis Doxastakis. Accelerating flat-histogram methods for potential of mean force calculations. *J. Chem. Phys.*, 131(5):054105, August 2009.

[97] Vijay S Pande, Kyle Beauchamp, and Gregory R Bowman. Everything you wanted to know about markov state models but were afraid to ask. *Methods*, 52(1):99–105, 2010.

[98] Nina Singhal and Vijay S Pande. Error analysis and efficient sampling in markovian state models for molecular dynamics. *The Journal of chemical physics*, 123(20):204909, 2005.

[99] Nina Singhal Hinrichs and Vijay S Pande. Calculation of the distribution of eigenvalues and eigenvectors in markovian state models for molecular dynamics. *The Journal of chemical physics*, 126(24):244101, 2007.

[100] Jumana Dakka, Kristof Farkas-Pall, Matteo Turilli, David W Wright, Peter V Coveney, and Shantenu Jha. Concurrent and adaptive extreme scale binding free energy calculations. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 189–200. IEEE, 2018.

[101] Wil MP van der Aalst and Stefan Jablonski. Dealing with workflow change: identification of issues and solutions. *Computer systems science and engineering*, 15(5):267–276, 2000.

[102] Paulin Coulibaly and Connely K Baldwin. Nonstationary hydrological time series forecasting using nonlinear dynamic methods. *Journal of Hydrology*, 307(1-4):164–174, 2005.

[103] Jörn Behrens, Natalja Rakowsky, Wolfgang Hiller, Dörthe Handorf, Matthias Läuter, Jürgen Päpke, et al. amatos: Parallel adaptive mesh generator for atmospheric and oceanic simulation. *Ocean Modelling*, 10(1-2):171–183, 2005.

[104] Chiara Casarotti and Rui Pinho. An adaptive capacity spectrum method for assessment of bridges subjected to earthquake action. *Bulletin of Earthquake Engineering*, 5(3):377–390, 2007.

[105] Zhiling Lan, Valerie E Taylor, and Greg Bryan. Dynamic load balancing for structured adaptive mesh refinement applications. In *Parallel Processing, 2001. International Conference on*, pages 571–579. IEEE, 2001.

[106] David E Shaw, Martin M Deneroff, Ron O Dror, Jeffrey S Kuskin, Richard H Larson, John K Salmon, et al. Anton, a special-purpose machine for molecular dynamics simulation. *Communications of the ACM*, 51(7):91–97, 2008.

[107] Harry A. Atwater and Albert Polman. Plasmonics for improved photovoltaic devices. *Nat. Mater.*, 9:205–213, 2010.

[108] Simone Napolitano, Emmanouil Glynos, and Nicholas B. Tito. Glass transition of polymers in bulk, confined geometries, and near interfaces. *Rep. Prog. Phys.*, 80(3), MAR 2017.

[109] Alessandro Laio and Michele Parrinello. Escaping free-energy minima. *Proc. Natl. Acad. Sci. USA*, 99(20), October 2002.

[110] Luca Maragliano, Benoît Roux, and Eric Vanden-Eijnden. Comparison between mean forces and swarms-of-trajectories string methods. *Journal of chemical theory and computation*, 10(2):524–533, 2014.

[111] John D Chodera, William C Swope, Jed W Pitera, and Ken A Dill. Long-time protein folding dynamics from short-time molecular dynamics simulations. *Multiscale Modeling & Simulation*, 5(4):1214–1226, 2006.

[112] Ayori Mitsutake and Yuko Okamoto. Replica-exchange extensions of simulated tempering method. *The Journal of chemical physics*, 121(6):2491–2504, 2004.

[113] Yuko Okamoto. Generalized-ensemble algorithms: enhanced sampling techniques for monte carlo and molecular dynamics simulations. *Journal of Molecular Graphics and Modelling*, 22(5):425–439, 2004.

[114] Volodymyr Babin, Christopher Roland, and Celeste Sagui. Adaptively biased molecular dynamics for free energy calculations. *The Journal of chemical physics*, 128(13):134101, 2008.

[115] Marta Mattoso, Jonas Dias, Kary ACS Ocaña, Eduardo Ogasawara, Flavio Costa, Felipe Horta, et al. Dynamic steering of hpc scientific workflows: A survey. *Future Generation Computer Systems*, 46:100–113, 2015.

[116] Philip K McKinley, Seyed Masoud Sadjadi, Eric P Kasten, and Betty HC Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[117] Peter M Kasson and Shantenu Jha. Adaptive ensemble simulations of biomolecules. *Current opinion in structural biology*, 52:87–94, 2018.

[118] Stress-ng. `http://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf` (accessed March 2018).

[119] Openmm. `https://github.com/pandegroup/openmm` (accessed March 2018).

[120] Vivek Balasubramanian, Matteo Turilli, Weiming Hu, Matthieu Lefebvre, Wenjie Lei, Ryan T. Modrak, Guido Cervone, Jeroen Tromp, and Shantenu Jha. Harnessing the power of many: Extensible toolkit for scalable ensemble applications. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 536–545, 2018.

[121] Md trajectories of ala2. `https://figshare.com/articles/new_fileset/1026131` (accessed March 2018).

[122] Fugao Wang and D. P. Landau. Efficient, multiple-range random walk algorithm to calculate density of states. *Phys. Rev. Lett.*, 86:2050–2053, 2001.

[123] M. R. Shirts and J. D. Chodera. Statistically optimal analysis of samples from multiple equilibrium states. *J. Chem. Phys.*, 129:124105, 2008.

[124] Pratyush Tiwary and B. J. Berne. Spectral gap optimization of order parameters for sampling complex molecular systems. *Proceedings of the National Academy of Sciences*, 2016.

[125] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 41. ACM, 2013.

[126] Hafiz Fahad Sheikh, Hengxing Tan, Ishfaq Ahmad, Sanjay Ranka, and Phanisekhar Bv. Energy-and performance-aware scheduling of tasks on parallel and distributed systems. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 8(4):32, 2012.

[127] David Skinner and William Kramer. Understanding the causes of performance variability in hpc workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 137–149. IEEE, 2005.

[128] Matthieu Dorier, Gabriel Antoniu, Rob Ross, Dries Kimpe, and Shadi Ibrahim. Calciom: Mitigating i/o interference in hpc systems through cross-application coordination. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 155–164. IEEE, 2014.

[129] Jim Brandt, David DeBonis, Ann Gentile, Jim Lujan, Cindy Martin, Dave Martinez, Stephen Olivier, Kevin Pedretti, Narate Taerat, Ron Velarde, et al. Enabling advanced operational analysis through multi-subsystem data integration on trinity. *Proc. Cray Users Group*, 2015.

[130] Warren Smith, Valerie Taylor, and Ian Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer, 1999.

[131] Aniruddha Marathe, Rachel Harris, David K Lowenthal, Bronis R De Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 239–250. ACM, 2013.

[132] John Brevik, Daniel Nurmi, and Rich Wolski. Predicting bounds on queuing delay in space-shared computing environments. In *Workload Characterization, 2006 IEEE International Symposium on*, pages 213–224. IEEE, 2006.

[133] Daniel Nurmi, Anirban Mandal, John Brevik, Chuck Koelbel, Rich Wolski, and Ken Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 119. ACM, 2006.

[134] 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.

[135] Rajkumar Buyya, David Abramson, and Jon Giddy. Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid. *arXiv preprint cs/0009021*, 2000.

[136] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *hpdc*, page 140. IEEE, 1998.

[137] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and evaluation of a resource selection framework for grid applications. In *null*, page 63. IEEE, 2002.

[138] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[139] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[140] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[141] Rajkumar Buyya and Manzur Murshed. A deadline and budget constrained cost-time optimisation algorithm for scheduling task farming applications on global grids. *arXiv preprint cs/0203020*, 2002.

[142] Omer Subasi, Javier Arias, Osman Unsal, Jesus Labarta, and Adrian Cristal. Programmer-directed partial redundancy for resilient hpc. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, page 47. ACM, 2015.

[143] Lingyun Yang, Jennifer M Schopf, and Ian Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 31. ACM, 2003.

[144] Chris Gregg, Michael Boyer, Kim Hazelwood, and Kevin Skadron. Dynamic heterogeneous scheduling decisions using historical runtime data. In *Workshop on Applications for Multi-and Many-Core Processors (A4MMC)*, 2011.

[145] Ioana Banicescu, Vijay Velusamy, and Johnny Devaprasad. On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. *Cluster Computing*, 6(3):215–226, 2003.

[146] Qi Zhang, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE transactions on cloud computing*, 2(1):14–28, 2014.

[147] Matteo Turilli, Feng Liu, Zhao Zhang, Andre Merzky, Michael Wilde, Jon Weissman, Daniel S Katz, and Shantenu Jha. Integrating abstractions to enhance the execution of distributed applications. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 953–962. IEEE, 2016.

[148] P Nxx. On the frequency distribution of the means of samples from a population having any law of frequency with finite moments, with special reference to pearson's type ii. *Biometrika*, 19(3/4):225–239, 1927.

[149] Philip Hall. The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika*, 19(3/4):240–245, 1927.