7-1-2016

Scheduling Heterogeneous HPC Applications in Next-Generation Exascale Systems

Oscar Hernan Mondragon Martinez

Oscar Hernán Mondragón Martínez
Candidate

Computer Science
Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Patrick G. Bridges, Chairperson

Dorian Arnold

Trilce Estrada

Wei Wennie Shu

Scheduling Heterogeneous HPC Applications in Next-Generation Exascale Systems

by

Oscar Hernán Mondragón Martínez

B.S., Electronics and Telecommunications Engineering, Universidad del Cauca, 2004
M.S., Wireless Systems and Related Technologies, Politecnico di Torino, 2006
M.S., Computer Science, University of New Mexico, 2013

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

July, 2016

Dedication

To Leidy for her unconditional love, tireless support and unlimited patience; and to Emiliano, my inspiration and reason to keep fighting.

Acknowledgments

During the time I was a graduate student in UNM and worked on my dissertation I received valuable feedback and help from numerous people. I would like to express my truly gratitude to all of them.

First, I would like to thank to my advisor, Patrick Bridges for his generosity and extremely useful guidance and support during all my time as a graduate student in UNM. Patrick truly believed in my work and devoted enormous amounts of time and effort sharing with me his knowledge and expertise in computer science and research matters. I am privileged to have worked with him and I feel sincerely grateful for all the help he provided to succesfully complete this work.

I wish to thank to Trilce Estrada, Dorian Arnold, and Wei Wennie Shu, who served as members of this dissertation committee, for their time and valuable suggestions to improve this work. I would like to express my thanks to Kevin Pedretti and Terry Jones who also contributed with feedback and good ideas during different stages of my dissertation. I would like to thank to Amanda Minnich who gave me precious feedback during my dissertation proposal elaboration. I wish to thank to Scott Levy, Kurt Ferreira, and Patrick Widener. I received valuable help and feedback while working with them and learnt a lot on doing quality research. I also want to thank to the members of the Scalable System Lab and the Hobbes project team who always provided good suggestions to improve my work.

I would like to acknowledge to Fulbright Colombia, Colciencias and the Universidad Autonoma de Occidente for the sponsorship they provided during my Ph.D. program, and to the 2013 Exascale Operating and Runtime Systems Program from the DOE Office of Science, Advanced Scientific Computing Research, which supported this work under the award number DE-SC0005050.

Finally, this work would have been imposible without the love and support of my family. I thank my wife, Leidy, and my son, Emiliano. They gave me the necessary encouragement and inspiration during all this process. I thank to my parents Omar and Elcy, for their love and support during this and all my projects; to my siblings Omar Arturo, Diana Karina, and Maria Eugenia; and to my nephews and niece Juan Sebastian, David, Felipe, and Sofia. They were always there for me.

Scheduling Heterogeneous HPC Applications in Next-Generation Exascale Systems

by

Oscar Hernán Mondragón Martínez

B.S., Electronics and Telecommunications Engineering, Universidad del Cauca, 2004

M.S., Wireless Systems and Related Technologies, Politecnico di Torino, 2006

M.S., Computer Science, University of New Mexico, 2013

Ph.D., Computer Science, University of New Mexico, 2016

Abstract

Next generation HPC applications will increasingly time-share system resources with emerging workloads such as in-situ analytics, resilience tasks, runtime adaptation services and power management activities. HPC systems must carefully schedule these co-located codes in order to reduce their impact on application performance. Among the techniques traditionally used to mitigate the performance effects of time-share systems is gang scheduling. This approach, however, leverages global synchronization and time agreement mechanisms that will become hard to support as systems increase in size. Alternative performance interference mitigation approaches must be explored for future HPC systems.

This dissertation evaluates the impacts of workload concurrency in future HPC systems. It uses simulation and modeling techniques to study the performance impacts of existing and emerging interference sources on a selection of HPC benchmarks, mini-applications, and applications. It also quantifies the cost and benefits of different approaches to scheduling co-located workloads, studies performance interference mitigation solutions based on gang scheduling, and examines their synchronization requirements.

To do so, this dissertation presents and leverages a new Extreme Value Theory-based model to characterize interference sources, and investigate their impact on Bulk Synchronous Parallel (BSP) applications. It demonstrates how this model can be used to analyze the interference attenuation effects of alternative fine-grained OS scheduling approaches based on periodic real time schedulers. This analysis can, in turn, guide the design of those mitigation techniques by providing tools to understand the tradeoffs of selecting scheduling parameters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern HPC (High Performance Computing) applications are used to solve advanced scientific problems in areas such as engineering, climate modeling, physics, and molecular dynamics. Those computational science programs generate large amounts of data that have to be prepared and analyzed before being presented to the field expert, who will use the resulting information to understand problems and inform decision making. To efficiently perform this, applications are increasingly coupled with analytics codes. This allows the system to periodically analyze data as it becomes available during application runtime.

These applications run in complex systems with limited resources that have to be shared with other applications and with system management activities. Next generation systems will face new challenges given this growing concurrency of workloads [4, 21, 54]. In addition to the mutual interference that may exist between co-existing codes, applications will have to deal with other interference sources such as background system tasks, resilience routines, runtime adaptation services, and power management policies. The random nature of the disruption events generated by those workloads causes significant variability in the performance of individual

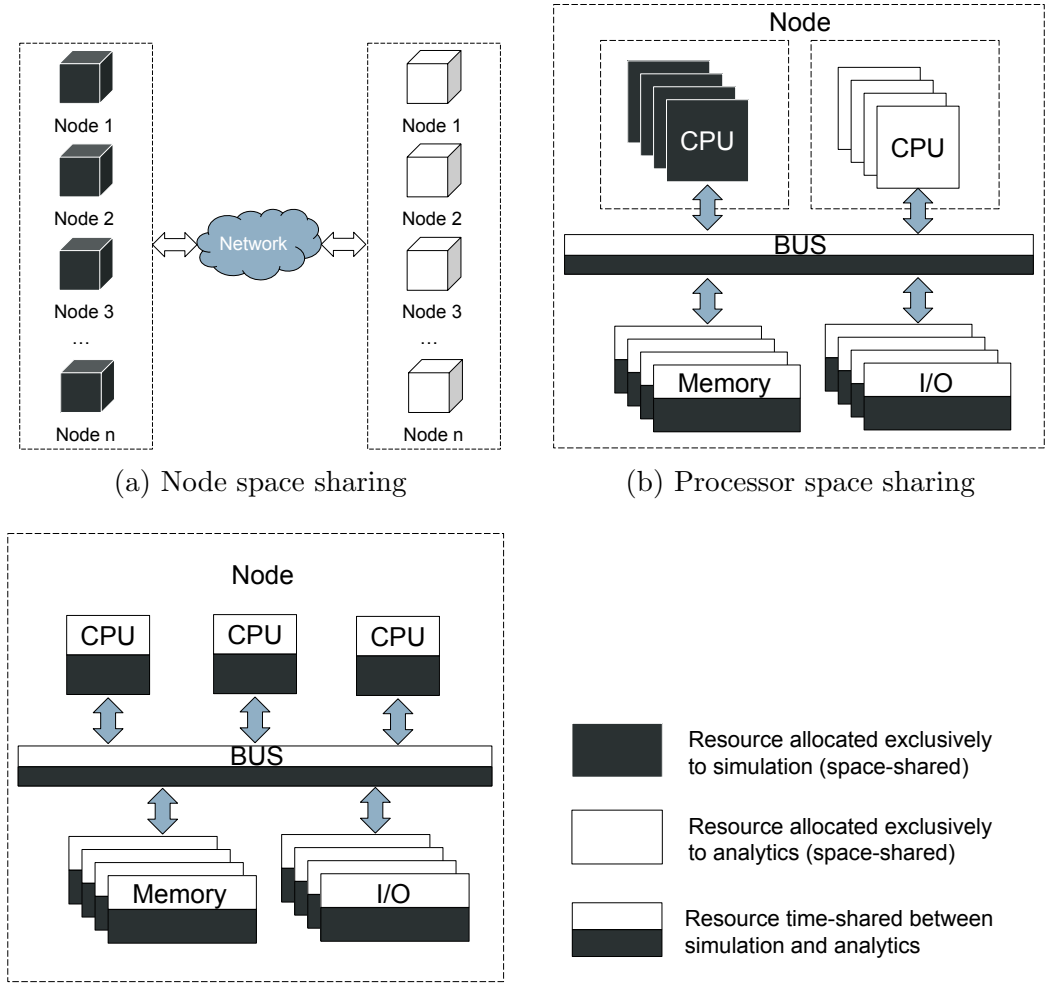
parallel components affecting the overall application performance [21].

In order to optimize the performance of applications, the system software must make effective resource allocation decisions. In current HPC systems, resource allocation is based on coarse-grained *space-shared* policies, where workloads have exclusive access to a set of nodes while they are running. In this scheme, applications frequently use a subset of the compute nodes to run analytics. This requires expensive data movement between simulation and analytics nodes (as shown in Figure 1.1a).

These costs are motivating the use of alternative resource allocation policies where simulation is co-located on a node with analytics. This co-location of workloads can be done using fine grained space-sharing where applications and analytics run on a non-overlapping set of processors inside the node (as shown in Figure 1.1b) or using time-shared allocation policies where processors are shared (as shown in Figure 1.1c) by the coupled workloads. The advantage of time-sharing processors is that analytics can leverage idle processors times to run, using resources that otherwise are wasted [90].

Although moving from node space-sharing to workload co-location policies reduces data movement costs and uses processors more efficiently, the effects of noise and contention for resources can increase significantly [54]. Under the co-location scheme, separate application components contend for system resources such as memory last level caches, DRAM bandwidth, and device drivers. When using time-sharing policies CPU cores are also shared (see Figure 1.1). Additionally, the services required to support a given workload may result in OS activities (sometimes referred to as OS noise) that slow down coupled workloads.

Among the techniques to mitigate performance interference are hybrid space-sharing and time-sharing approaches [2], cooperative time-sharing [90], and gang scheduling [26]. This last approach, for example, concurrently schedules the parallel



(a) Node space sharing

(b) Processor space sharing

(c) Time sharing

Figure 1.1: Resource allocation options

components of a workload across system partitions, facilitating those components making progress in parallel and avoiding the long blocking periods that occur if they are not scheduled at the same time. This technique leverages global synchronization and time agreement mechanisms, however, that are difficult to scale. Given this, alternative performance interference mitigation approaches must be explored for future HPC systems.

This dissertation evaluates the impacts of workload concurrency in future HPC systems. It studies current interference mitigation solutions based on gang scheduling and examines their synchronization requirements. This work also analyzes the interference attenuation effects of alternative fine-grained OS scheduling approaches based on periodic real time schedulers, providing the theoretical foundations to understand their impact on applications performance.

1.1 Modern HPC architectures

Modern HPC applications and systems are undergoing important changes. Large-scale scientific applications are beginning to perform simulation and analysis concurrently instead of sequentially. Similarly, hardware memory and network bandwidth restrictions and system power caps also motivate the need to minimize data movement and power down unneeded hardware components whenever possible.

Under this scheme, system resources have to be shared between simulation and analytics workloads. Additionally, system tasks and other emerging workloads also compete for resources. In addition to the OS short-lived system tasks, which have been traditionally identified as interference sources, next-generation HPC applications will face new sources of contention such as resilience routines, and power management tasks. This section describes a set of workloads and architectures of these systems.

1.1.1 Composite Applications

Modern HPC applications are complex and are composed of coupled individual workloads which are each comprised of several interacting components. Figure 1.2 shows an example of the components of simulation and analytics in an *in situ* analytics

application. Each portion of the applications runs in its own *enclave*, a set of system resources allocated to an application [4]. In Figure 1.2, enclave 1 contains the simulation code and enclave 2 holds the analytics code, and these enclaves are co-located on the same node.

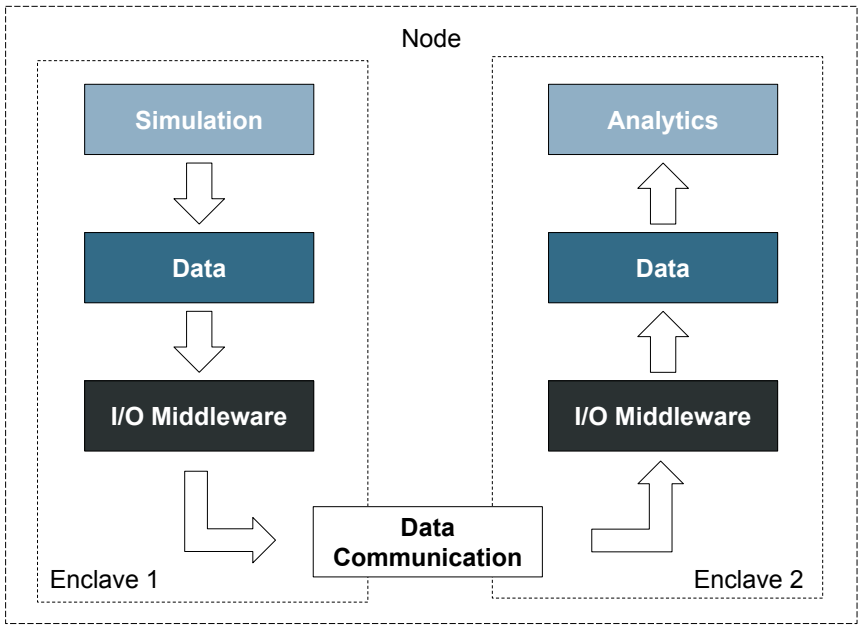


Figure 1.2: *In situ* analytics application

Simulation

In this architecture, a simulation application runs for a number of timesteps previously defined by the user and periodically shares the data that it generates with analytics. This sharing is generally accomplished using a I/O middleware system such as ADIOS [63]. Figure 1.2 shows how these enclave components cooperate within a single node. Note that parallel instances of each workload run on thousands of nodes performing a number of iterations comprised of local computations and barrier syn-

chronization steps. This application model is called *Bulk Synchronous Parallel* [84] because it facilitates the concurrent execution of local computation across processes, and allows parallel progress through the use of synchronization points.

Analytics

As the simulation generates data, it is consumed by analytics which uses the I/O middleware to receive that data from the transport channel and then generate useful information for the application scientist. Often, data flows from simulation to analytics; however data may flow from analytics to simulation to guide the computational operations performed by simulation, such in computational steering codes. Among the tasks performed by analytics codes are visualization and steering [88], cosmology analytics [71], data preparation and histogramming [89], and molecular dynamics analysis and visualization [86].

1.1.2 System Support Activities

In order to support applications, a number of operating system tasks run in the background. Those tasks provide the infrastructure services required by applications, but at the same time are a potential source of interference. This section describes those system activities.

Short-lived Tasks

The interference generated by background system activities has been widely studied [29, 44, 66]. Among the system activities that may affect application performance in greater degree are interrupts, scheduling events, and I/O operations [66]. Although these are generally short duration events, their effects can be significant on

applications. Understanding those effects in future systems is important.

Resilience Tasks

As HPC systems grow in capacity and hardware components, failures will be more common [55, 60]. Those systems will potentially require complex resilience mechanisms to recover from those failures [27,30]. Resilience tasks such as checkpoint/restart will be expensive in future HPC systems and will use a significant percentage of the system available time [67]. This may reduce the time able to be used by applications. Although asynchronous checkpoint/restart, for example, helps to reduce the costs of coordinated checkpoint/restart by relaxing the times in which checkpoints are performed across processes, it is also harmful to applications given the nature of the interference events that it generates.

Power Optimization Tasks

Reducing power consumption has been identified as one of the main challenges that future large-scale systems will face [4]. Advances in hardware infrastructure seek to improve energy efficiency; however, they may not suffice without proper system software support [57]. The implementation of these system power management approaches can cause a wide range of problems to HPC scheduling systems. For example, the use of idle power states to save power can generate performance degradation in applications when the sleep length for a core is overestimated [47].

Likewise, Dynamic Voltage and Frequency Scaling (DVFS) techniques influence both BSP application synchronization steps, and complicate scheduling policy decisions. For example, DVFS may affect the duration of individual bulk synchronous intervals generating an effect similar to that produced by OS noise [57]. Similarly, asymmetric DVFS decisions across cores can potentially have large negative perfor-

mance impacts on coordinated scheduling.

1.2 The Effects of Interference

As described in Section 1.1, bulk synchronous parallel applications are comprised of iterations in which the parallel components of the applications perform local computations and synchronize at the end of each computation step. The duration of each iteration is dominated by the slowest parallel component, since all parallel components must finish the computation step in order to synchronize. Interference is one of the main factors that influence the interval length.

The effects of interference on parallel applications vary depending on the characteristics of the noise events and the applications' granularity (i.e., the length of the computation iteration [82]) [23]. In the absence of any mitigation mechanism, the individual components of parallel applications may not be affected in the same way; instead they will meet interference events of different duration at different times. This significantly affects communication between components and results in significant performance degradation and variability in the application runtime [21].

Noise events perturbing a given process are *amplified* if they affect the expected end time of a parallel operation, for example increasing the wait time on processes during blocking operations. Other noise events disturbing a process may be *absorbed* if they do not increase the parallel operation's total duration. This may occur, for example, if interfering activities occur while the application is already waiting for other reasons.

Previous work [29] has shown that high-duration low-frequency noise patterns (e.g., analytics codes, resilience tasks) happening during blocking collective operations are more likely to be amplified, while low-duration high-frequency noise events

(e.g., short-lived tasks) are more easily absorbed. The application's granularity also influences the noise impact with shorter operations being more sensitive to noise.

1.3 Example

To illustrate these concepts, consider an example based on the Gyrokinetic Toroidal Code (GTC), a code used to study magnetic microturbulence in torus geometries by using a 3D-particle in cell approach to analyze particles interactions [68]. Each particle has a set of features such as position, velocity, and weight [53]. This particle information is shared with PreDatA analytics through an output channel. PreDatA then performs several operations, including global analysis and aggregation of each of the particles' features, range queries to detect particles in a given area, and histogram computations.

As shown in this example, the operations performed both by simulation and analytics codes are complex, as are the interactions between these two workloads. In this example, most of the computational resources are allocated to GTC to perform simulations. Every 5 timesteps, which combined have a duration of approximately 5 seconds, GTC sends data to PreDatA. PreDatA then consumes the simulation data and runs for approximately 100 milliseconds. Both simulation and analytics perform global operations that require communication and synchronization between processes through shared network resources.

In this hypothetical example, both GTC and PreDatA are using one million processes and are time-sharing the same number of CPU cores. While workloads are running, several tasks run in the background in order to facilitate their correct execution. Among these tasks are:

- Short-lived tasks: for example, I/O devices generate interrupts that must be

handled by the corresponding device driver, and the CPU scheduler is executed each time that an interrupt is received.

- Resilience tasks: an asynchronous checkpoint/restart mechanism with checkpointing tasks of one second duration every two minutes like the described in [30] runs in order to protect workloads from system failures.

- Power optimization tasks: a DVFS mechanism attempts to reduce power consumption by independently controlling CPU frequencies.

These sources of interference impact the duration of bulk synchronous intervals and the overall workloads' performance in different ways. The bursty nature of the interference pattern that PreDatA generates in GTC has a potentially high impact on GTC performance. Similarly, the uncoordinated nature of the checkpointing events and their lower-frequency higher-duration pattern may generate even more significant impacts. Since the DVFS mechanism used in this example perform changes in CPU frequencies independently, it will impact the duration of bulk synchronous intervals in different ways, having a potentially high impact on the total workloads' completion time. Finally, short-lived tasks will also contribute to workload performance degradation.

1.4 Mitigating Interference Impacts

As described in Sections 1.2 and 1.3, the effects of interference may be important if applications and noise patterns meet certain characteristics. Given their stochastic nature, interference events impact individual bulk synchronous parallel intervals in a non-uniform way, increasing the length of the total interval. The length of this interval depends on the slowest process reaching the synchronization barrier. One of the approaches to mitigate this phenomena is to synchronize the occurrence of

interference events in order to generate homogeneous interference patterns across the application's processes, facilitating the communication between them.

This can be accomplished by using system-based gang scheduling or gang scheduling using synchronizing collectives in the workloads' code. System-based gang scheduling leverages tight system synchronization services to schedule interference events at the same time. On the other hand, synchronizing collectives do not rely on system synchronization services; however, as the MPI 3.0 standard recommends [32], application developers must be careful on relying on the synchronization capabilities provided by MPI collectives. These current mitigation mechanism will not scale efficiently, however, and new approaches may need to be explored [78] for future large-scale systems. This is especially critical for applications with very short intervals, which require tight synchronization [78], and is even more problematic if they are affected by interference sources with low-frequency high-duration patterns.

Approaches based on periodic real-time scheduling [50, 61] have shown benefits similar to these traditional mitigation approaches. Those alternative techniques require less strict system synchronization services and may help to mitigate the impact of emerging workloads in HPC systems. Limited research has been done on the efficacy of these techniques, however.

1.5 Research Challenges

The new trends on computational science codes and the advent of the new interference sources described in previous sections brings new interesting research challenges. Although several empirical studies have been devoted to study the impact of OS interference on applications, there is a need of further research on the impacts of emerging interference sources. Additionally, there is a need of more rigorous analytical tools to perform that research. This will allow to use theoretical foundations to characterize

those new interference sources, to study their impact on emerging applications, and to study alternative mitigation approaches that can be used in future HPC systems.

The following is a summary of the research questions that I attempt to address in this work:

1. How will emerging interference sources impact applications' performance in next-generation HPC systems, and what is the effect of using different approaches to schedule co-located workloads?
2. What are the synchronization requirements of current performance interference mitigation approaches?
3. How can we effectively characterize emerging HPC interference sources?
4. How can we effectively model the impact of interference sources and mitigation techniques on applications?

1.6 Contributions

My thesis is that fine-grained OS scheduling techniques can be effectively used to support the co-location of HPC applications and emerging workloads in next-generation exascale systems. To evaluate this thesis, I first empirically study current approaches to scheduling co-located workloads. I then develop a model-based approach for studying interference workloads. Finally, I use these techniques to examine the effects of using fine-grained OS scheduling to mitigate performance impacts.

The major contributions of this work are:

- A simulation-based method to evaluating the performance impacts of the co-location of emerging workloads and HPC applications.

- An empirical study of the performance impact of different approaches to scheduling applications and co-located workloads;
- An examination (performed in collaboration with other colleagues) of the synchronization requirements of traditional gang scheduling approaches used to reduce the performance impact on applications;
- An extreme value theory-based approach to modeling traditional and new interference sources, predicting their impact, and informing the design of scheduling techniques to assist in their co-location with HPC applications.
- A validation of the analytical model that I propose, by characterizing several sources of interference and predicting their impacts on a set of applications.
- An evaluation of a fine-grained OS scheduling performance interference mitigation technique using both the empirical and the analytical approaches that I present in this dissertation.

1.7 Dissertation Outline

The remainder of this document is organized as follows: Chapter 2 provides essential background information and discusses related work on application composition, performance interference in HPC applications and extreme value theory concepts. Chapter 3 presents a framework that I developed in collaboration with other colleagues. The chapter then uses the framework to quantify the costs and benefits of different approaches to scheduling applications and workloads on nodes in large-scale applications, including space sharing, uncoordinated time sharing, and gang scheduled time sharing. Chapter 4 presents an extreme value theory-based model for analyzing the performance of bulk-synchronous HPC applications under the presence of co-located workloads. Chapter 5 evaluates fine-grained OS scheduling policies as

a means to support code co-location both empirically and using the extreme value modeling approach that I propose in this dissertation. Finally, Chapter 6 concludes and presents directions for future work.

Chapter 2

Related Work

This chapter describes past work related to this dissertation. Section 2.1 presents an overview of the composition of HPC applications as well as the system mechanisms and scheduling techniques required to support it. Section 2.2 describes previous research on the characterization of sources of interference in HPC systems. Section 2.3 studies works on analytical modeling of applications performance and the impact of interference. Section 2.4 provides a study of current interference impact mitigation strategies. Finally, Section 2.5 contrasts the works described here against this dissertation, and explains the novel contributions of my work.

2.1 Application Composition

A major source of interference for next-generation HPC applications are the workloads used for *in situ* data analysis, steering, data aggregation, and visualization that they feed. Such codes are used, for example, to provide new analysis capabilities to existing simulation codes, optimize I/O performance by reducing system I/O demands, and provide summary information at runtime that scientists can use to

monitor the behavior of the simulation.

2.1.1 Example Composite Codes

There are a large number of analytics codes [71,86,88,89]. In this dissertation, I focus on two modern *in situ* analytics workloads, using them to quantify the potential impact of analytics on applications' performance, the *Bonds* analysis used with the LAMMPS application code and a histogramming analysis for the GTC-P proxy application performed using the PreDatA analytics middleware.

SmartPointer Analysis in LAMMPS: SmartPointer [86] is an analytics and visualization code comprised of configurable analytics services. The *Bonds* capability of SmartPointer facilitates the tracking of cracks generated by the LAMMPS simulation code [73]. Specifically, Bonds directly reads atom bonding information from LAMMPS, and conducts a compute-intensive analysis that determines where in a simulated material adjacent molecules are no longer bonded. It then writes the computed information to a previously configured output channel. Bonds performs no additional communication of its own; it relies on communication by LAMMPS to obtain ghost cell information from other nodes.

PreDatA - Preparatory Data Analytics in GTC-P: PreDatA is a middleware with pluggable components that perform a number of data preparation operations such as data sorting, filtering, and histogram generation. Those operations are predefined according to the users needs [89], and frequently used so that scientists can monitor the progress (and potential correctness) of long-running simulations. A number of applications have used PreDatA to perform *in-situ* analytics, including the Gyrokinetic Toroidal Code (GTC) [51], a computational-science application used for

3D particle-in-cell simulations of plasma micro-turbulence, and Pixie3D [11], a 3D MHD (Magneto Hydro-Dynamics) solver.

2.1.2 System Software Support for Analytics

A range of system software techniques have been developed to support *in situ* analytics including, data movement, consistency management, and scheduling techniques. Figure 2.1 shows an example of the stack that simulation and analytics codes use to support their activities. The Adaptable I/O System (ADIOS) [63] provides an API applications can use to efficiently transport data either to other applications or to the file system. It is a flexible middleware that allows applications to choose between different data transport methods at runtime.

There are several alternatives to data transport. The Transparently Consistent Asynchronous Shared Memory (TCASM) [3] API provides low-overhead interfaces for asynchronous memory sharing between codes, while also providing consistent views of shared data using virtual memory techniques. Flexpath [18] uses a public/subscribe mechanism to facilitate data transport between coupled codes by using either shared-memory, Remote Direct Memory Access (RDMA), or TCP/IP communication. Finally, DataSpaces [22] offers data transport and coordination capabilities using RDMA as communication layer.

New HPC system software architectures, for example the Hobbes exascale operating system [9], seek to provide more systematic support for multi-component applications. The Hobbes XEMEM (Cross Enclave Memory) [52] abstraction, for example, provides a shared-memory system between enclaves which is compatible with XPMEM [87]. The Cross-Enclave Asynchronous Shared Memory (XAMS) [24] approach leverages XEMEM and a copy-on-write technique to facilitate application composition by providing an interface to communicate between application compo-

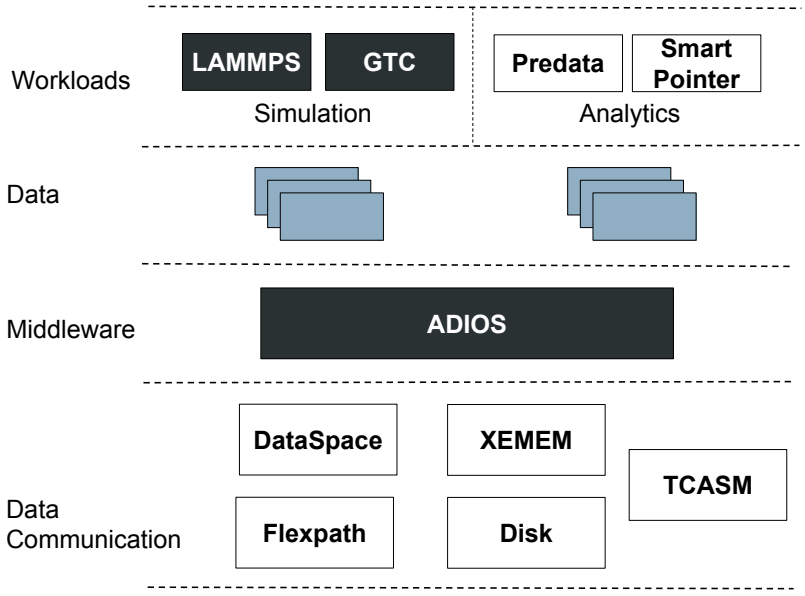


Figure 2.1: System software support for *in situ* analytics

nents using a consumer-producer approach. The coupled codes that I study in this dissertation use ADIOS and XEMEM as middleware and data transport, respectively.

2.1.3 Scheduling Analytics

Analytics can be scheduled in a variety of ways, broadly categorized as either space-shared, where cores are dedicated to analytics, or time-shared, where cores are shared between application and analytics. Space sharing is simpler but requires dedicating resources to analytics, while time-sharing can overlap analytics computation with application computation but directly interfere with application performance.

Time-sharing scheduling approaches include uncoordinated time sharing and gang-scheduled time sharing. In uncoordinated time sharing, each node schedules when

analytics runs using a purely local scheduling policy. In gang-scheduled time sharing, a system-wide mechanism coordinates when analytics runs, for example using synchronized clocks, collective communication in the analytics, or collective communication in the application. Methods that avoid introducing extra global communications are generally preferable because of the cost of those communications in large-scale systems.

New services to cooperatively co-schedule analytics and simulation are also under development. The Goldrush system [90], for example, proposes user-level mechanisms to enable simulation/analytics co-location, particularly on OpenMP applications with significant serial execution sections. This work has demonstrated the viability of time-sharing analytics and simulation but relies on significant changes to the application runtime. It is unclear if similar time-sharing approaches are appropriate for a broader range of applications or more general OS-level support could provide similar capabilities.

Chang et al. in [12] propose an approach in which workloads are classified as I/O-intensive or CPU-intensive. They claim that in high speed networks, where the costs of processing huge amount of packets are elevated, I/O-intensive tasks may not receive enough access to CPU resources, which are usually allocated to CPU-intensive tasks. They propose a solution in which CPU-intensive tasks voluntarily yield CPU cycles to I/O-intensive tasks.

2.2 Performance Interference

A large number of works have empirically studied the impacts of performance interference. Most have focused on the impact of short-lived background system activities, while a few have studied the effects of other interference sources. Different techniques have been used to these studies, ranging from implementation-based to simulation

and emulation-based approaches. This section describes these approaches.

2.2.1 Performance Interference Characterization

OS noise [72] is the most well-known and studied interference source. Some authors [5, 44] have studied the influence of OS noise on BSP applications and the benefits of synchronizing noise events. Pradipta et al. [19] identified the sources of jitter that are potentially more harmful for applications. They found that timer interrupts are the OS noise events that generate the most significant amount of interference, representing approximately a 63% of the total jitter in systems. Further works [29, 66] have investigated the impact of kernel activities on application performance and how system design decisions such as the location of noisy nodes influence performance [28].

Many other sources of interference have also been identified, including asynchronous checkpointing [30, 60], *in situ* analytics systems [90], and system power capping [70]. These sources are particularly interesting because of the increasing importance of application composition and dynamic resource management in next-generation HPC systems and applications [9, 53].

2.2.2 Summary

Despite all these empirical interference studies, however, the HPC community has only a rough understanding of exactly *how* HPC applications are perturbed. For example, OS noise studies have empirically demonstrated that the "shape" of the noise is at least as important as the amount of noise; infrequent, long-duration noise events have been shown to be significantly more disruptive than frequent short-duration noise events [29, 41].

These studies are almost exclusively empirical with little or no theoretical or analytical underpinning to help researchers understand *why* interference interacted with applications in the way it did or provide quantitative predictions about how particular interference sources and applications would interact. In contrast, this dissertation seeks to provide the theoretical foundations to characterize interference sources, predict their impact on applications and understand performance interference mitigation strategies.

2.2.3 Simulating Interference Performance Impact

Several researchers have used simulation and emulation techniques in order to investigate the potential effects of interference on applications running on future large-scale systems. This facilitates study of those impacts using scales and capabilities not present in current systems. A number of works, including that described in this dissertation have used `LogGOPSim`. In the remainder of this section, I describe `LogGOPSim` in detail, as well as information on other simulation approaches.

`LogGOPSim` Extreme-scale Simulator

`LogGOPSim` [42] has been used to investigate the effects of noise and noise-like phenomena on application performance. Hoefler et al. [41] used this simulator to evaluate the effects of OS noise on application performance. Levy et al. [60] demonstrated that `LogGOPSim` could be used to accurately predict the impact of resilience mechanisms on applications performance. Ferriera et al. [30] subsequently used `LogGOPSim` to consider the impact of asynchronous checkpoint/restart on application performance. Widener et al. [85] used `LogGOPSim` to study how non-blocking collectives might be used to reduce the performance penalties associated with asynchronous checkpoint/restart.

`LogGOPSim` uses the LogGOPS model based on LogP [16]. It simulates traces collected from small applications runs, reproducing messages between processes and communication dependencies [30]. It also provides an extrapolation capability which performs simulations at larger scales [42]. `LogGOPSim` simulates the impact of noise by reading a noise trace. This noise trace contains noise events, characterized by their starting time and duration, which are added to the communication or computation events during simulation.

`LogGOPSim` provides the capability to simulate the level of synchronization of interference events by controlling the point in which each process starts reading the noise trace. If the starting points across processes are selected uniformly at random, `LogGOPSim` simulates total uncoordinated scheduling of noise events. If the starting points are selected from a normal distribution, it simulates gang scheduling with different levels of synchronization. The degree of coordination is controlled with the standard deviation of the distribution.

Other Simulation Approaches

Other works have used simulation and emulation approaches. Engelmann et al. [23] use the extreme scale simulator (xSim) [8] to investigate noise impacts on HPC systems with up to 2M nodes and 1 billion cores. They study the effects of noise amplification and noise absorption on MPI collectives and propose this framework as a tool to be used in hardware/software co-design of future HPC systems. Pradipta De et al. [20] propose an emulation-based approach to study the impact of noise and the mitigation effects of jitter synchronization. Unlike `LogGOPSim`, which provides the capability of performing large scale analysis running simulations in a single node, these frameworks require large amounts of systems resources which increase with size of the extrapolation under analysis.

In this dissertation, I leverage `LogGOPSim` to study the impact of new sources of interference on HPC applications. I also use it to collect the baseline application's data that feed my performance interference analytical model.

2.3 Modeling Performance Impact

A few works have used mathematical-based modeling to study application performance interference. A number of works leverage probabilistic and statistics of extremes concepts to provide the theoretical foundations to understand this problem. This section describes these works and introduce the theoretical foundations that they use.

2.3.1 Modeling Extremes

Extreme Value Theory (EVT) and the Expectation of Maximum Values (EMV) concepts have been widely used across a variety of domains to estimate the probability of extreme events. These theories have been recently used in computer science to estimate applications' performance [81] and the impact of jitter [78]. I next discuss these and additional related works.

Extreme Value Theory

Extreme Value theory (EVT) is a sub-field of statistics focused on the behavior of maxima of a set of random variables [43]. It is generally used to analyze or predict the likelihood of extreme events occurring. EVT works with a set of independent, identically distributed (i.i.d.) stochastic events of potentially unknown distribution, and seeks to understand when a new extreme value (a sample larger than those seen

previously) should be expected to happen.

EVT has been used in fields such as hydrology to determine long-term flood plain levels, and for similar actuarial analyses in the insurance and financial industries. It has also been used to estimate the size of demand bursts in internet traffic [37]. Similarly, Uchida et al. [83] present a model that uses extreme value theory to explain the tail of the distribution of throughput and as a tool to predict peaks of throughput.

EVT methods generally focus on the use of the Generalized Extreme Value (GEV) distribution. This distribution combines different related distributions in which the maximum value of a set of independent and identically distributed random variables can be fitted assuming distributions with well-behaved tails.

$$F\left(x \mid \xi, \mu, \sigma\right) = e^{-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{\frac{-1}{\xi}}} \tag{2.1}$$

Where, $-\infty < \xi < \infty$, $-\infty < \mu < \infty$, and $\sigma > 0$

Equation 2.1 shows the cumulative distribution function (CDF) for the GEV distribution, which includes three parameters, shape (ξ), location (μ) and scale (σ). The *location* parameter determines the shift of the distribution to left or right direction. The *scale* parameter defines the dispersion of the distribution; the greater this parameter the greater the dispersion. A property of the scale parameter is that if it is multiplied by a constant, c, the random variable will be multiplied as well. Thus, for a random variable X, $|c| \times \sigma(X) = \sigma(c \times X)$ [76]. The *shape* parameter defines if the distribution of maxima belongs to the lightly-tailed *Gumbel* ($\xi = 0$) distribution, the heavy-tailed *Fréchet* distribution ($\xi > 0$), or the upper bounded-tail *Weibull* distribution ($\xi < 0$).

The *block maxima method* from EVT is a commonly-used technique for fitting the GEV distribution to the maximum of a set of random samples of a random variable. In this method, the observations of a given random variable are divided into blocks of

size n and the maximum value occurrence in each block is determined. A traditional distribution fitting method, commonly maximum likelihood estimation (MLE), is then used to determine the GEV parameters that best fit this set of maxima. The resulting distribution is that of the maxima of blocks of size n. In flood plain analysis, for example, the block maxima method is frequently used with blocks of 365 days to estimate GEV parameters for the distribution of maximum annual water levels.

The regularity of the maximum likelihood estimators and the degree in which they meet the asymptotic property of a GEV distribution can be established by studying the ξ parameter [14, 79]. For ξ parameters greater than -0.5, regular likelihood estimators can be obtained. For ξ parameters between -1 and -0.5 likelihood estimators can be obtained but they do not meet the asymptotic property of a GEV distribution. Finally, for ξ parameters less than -1, is generally not possible to compute the estimators.

Given a concrete GEV, *return level analysis* can then be used to compute the return level, $\bar{R}$. This return level can be interpreted as the value that will, on average, be exceeded once in every N samples of the distribution. Note, that this is once every b blocks when using the block maxima method to fit the GEV distribution. In the common flood plain level, for example, the 100-year flood plain is simply the 100-year return level once the GEV parameters for the distribution of annual maximum waters levels have been determined. Details of this calculation have been omitted for brevity and are available in the literature [14].

Expectation of Maximum Values

In addition to return level analysis, extreme value techniques also sometimes seek to directly calculate the expected value of the maximum of a set of random variables, an area termed *expectation of maximum values* (EMV). A number of approximation

techniques have been proposed for this problem [7, 25, 31, 35, 81].

Modeling Application Runtime Recent work [81] used the expected mean maximum approach (EMMA) to estimate execution times in BSP applications when the probability distribution of process local computation times is known *a priori*. In particular, this technique estimates the expected value of the maximum m instances of a set of i.i.d. random variables X_i with distribution F as:

$$E(max_{i=1}^{m} X_i) \approx F^{-1}(P) \tag{2.2}$$

where $P \approx 0.570376002^{1/m}$.

This dissertation provides a novel approach that uses EVT concepts to characterize interference sources and study their impact on applications. While EMMA [81] used extreme value concepts to derive the expected mean maximum approximation, it did not use extreme value distributions themselves to estimate application performance. This limits EMMA's direct use in application performance estimation. However, the method described in this work for conducting such estimations relies heavily on EMMA for extrapolation. EMMA focuses on runtime estimations for applications running in isolation. Unlike EMMA, our model focuses on the impact of interference in time-shared environments, and does not assume similar computation efforts across BSP iterations.

Modeling Interference Seelam et al. [78] use expectation of maximum values analysis to predict the effects of jitter on applications. To that end they use a simple synthetic Bulk Synchronous Parallel (BSP) benchmark to study the effect of short-lived noise events for BSP computational intervals of different lengths. They compute lower and upper bounds of the expected slowdowns using the approxima-

tions proposed by Bertsimas [6] and Crammer [15], respectively. They then use the data provided by the benchmark's runs to estimate the probabilistic parameters used by those models and estimate slowdowns at larger scales. By using their model, they state that the slowdown experienced by an application is proportional to $\sqrt{n}$, where n is the number of CPUs in the system. They also use the model to study the mitigation effects of co-scheduling.

The Bertsimas and Cramer bounds used by Seelam et al. [78] roughly estimate the impacts of OS noise; however the exact distribution of the noise is not computed. This is reasonable to estimate the modest impacts of OS noise but not sufficient to estimate the impacts of more extreme types of interference. The model that I propose in this dissertation, in contrast, estimates the underlying distribution of BSP computational intervals under the effects of a variety of noise types. Moreover, Seelam et al. leverages the synthetic benchmark for all experiments. This may suffice for well-balanced real applications in co-location with modest interference. However other approaches are needed to study interference sources such as analytics codes or resilience operations.

2.3.2 Other Analytical Models

Tsafrir et. al. [82] used probabilistic methods to demonstrate the linear dependency of applications' performance interference and the size of a system. They propose a simple model that states that given a system of n nodes, if the per-node probability of an application being delayed is p and $p \leq \frac{1}{10n}$, the noise affecting the parallel application on each computational iteration is directly proportional to the system size with a probability of the application being delayed given by $p \times n$. The p probability is measured by benchmarking the application to estimate the probality distribution of computational iterations. This model provides a general sense of the

probability of application delays, but it can not be used to compute the expected application slowdown. The model that I propose in this dissertation seeks to estimate the expected application slowdown.

2.4 Performance Impact Mitigation

Many techniques have been proposed for mitigating the impact of performance interference on HPC applications. A number of works have used gang scheduling-based techniques to minimize the impact of potentially interfering workloads. Other works have propose alternative approaches based on periodic real time scheduling. This section describes those works.

2.4.1 Gang Scheduling

Using gang scheduling has shown performance benefits for BSP applications. When using this technique, the application's processes are co-scheduled. This facilitates the communication between them and reduces the amount of time that the fastest processes have to wait for synchronization.

Feitelson et al. [26] in one of the first works that studied gang scheduling, compared two synchronization techniques: (1) a busy waiting-based gang scheduling technique implemented by the runtime which schedules application processes at the same time; and (2) a blocking, with uncoordinated scheduling, synchronization mechanism. They show that fine-grained parallel applications benefit from using the gang scheduling policy, which otherwise incur overheads due to context switching while blocking to perform synchronization.

More recently, Jones et al. in [44] presented a run-time system to avoid the inter-

ference over large fine-grained parallel applications generated by short-lived system tasks. This run-time coordinates those system tasks in order to run them at the same time. This helps to decrease interference impacts, allowing parallel applications to be co-scheduled at precise times, and improving their performance and scalability. This approach works for both inter-node and intra-node cases. In this dissertation, we use simulation to study the global synchronization requirements of traditional gang scheduling-based mitigation approaches.

2.4.2 Real Time Scheduling Mitigation

Local scheduling policies are generally the purview of the operating and application runtime systems. In best-effort scheduling, typified by Linux's round-robin preemptive priority scheduler, the CPU is shared roughly equally when both analytics and the application need to run. Earliest-deadline-first (EDF) scheduling, in contrast, provides fine-grain control of when each task runs and the share of the processor given to each task, and has been proposed to mitigate performance interference.

Earliest Deadline First (EDF) Scheduling

Earliest Deadline First (EDF) is a periodic real time scheduling algorithm. An EDF scheduler uses a dynamically calculated priority value as the scheduling criterion to choose what task will be scheduled. This priority value is calculated according with the relative deadline of the tasks that are ready to run in a *runqueue*. The tasks with less time before their deadline will receive higher priorities. At each opportunity to make a scheduling decision (e.g., when an interrupt is received), the scheduler executes the task that has the deadline that will expire first.

Each task's scheduling requirements in EDF scheduling are described by a scheduling *period*, T, and the length of the task's *slice*, S, within that period. A task with

a slice S and a period T is guaranteed to execute for S seconds in each T second period. Usually, the deadline for each task is assumed to be the expiration of its current period.

A task uses at least slice (S) seconds of CPU every period (T). The period is defined as the time in which the task may receive a CPU allocation, and the slice is the minimum amount of time received for the task during each period [62]. The portion of the CPU used for a task i (i.e., *task's utilization factor*) is

$$u_i = \frac{S_i}{T_i} \tag{2.3}$$

and the processor's utilization is given by:

$$U = \sum_{i=1}^{n} \frac{S_i}{T_i} \tag{2.4}$$

Where n is the number of EDF tasks running on that processor.

The schedulability condition [62] is given by,

$$\sum_{i=1}^{n} \frac{S_i}{T_i} \leq 1 \tag{2.5}$$

This schedulability condition guarantees that no task will miss deadlines if the CPU utilization is at most 100%.

EDF-Based Mitigation

An alternative performance interference mitigation technique leverages the EDF scheduling policy. In this approach, applications processes are co-scheduled in differ-

ent system partitions and configured with identical EDF parameters. This approach can be used, for example, to reschedule analytics workloads time-sharing CPU cores with simulation, resulting in frequent, limited-duration interruptions of the simulation application.

Bin Lin et al. [61] present an approach based on per-node EDF schedulers. This technique provides mitigation capabilities comparable to those offered by gang scheduling by setting the same slice and period values to all of the application's parallel components. This approach schedules virtual cores to the same host CPU cores and leverages the synchronization and isolation capabilities of EDF to mitigate the effects of contention and keep the virtual machines' computation rate constant. In this dissertation, I use this approach to study the performance interference mitigation effects of using periodic real scheduling to schedule co-located workloads.

Kato et. al. [50] propose a gang EDF scheduler for multithreaded applications on multicore systems. In this approach, they claim that all the threads of a multithread application must be scheduled together. For that, they enhanced the global EDF policy by scheduling the set of threads of the same application at the same time when there where enough physical cores available. When there are not enough cores available, threads must wait for a time slice with sufficient core availability. This approach may cause CPU fragmentation, priority inversion [58], and execution delay. In order to solve these problems, Sukwong et al. in [80] propose a scheduler based built on top of KVM's Completely Fair Scheduler (CFS) in which sibling virtual cores are balanced to different physical cores, and which does not use strict synchronization.

2.5 Summary

In this chapter, I presented two examples of composite applications, and the system support required to assist their interactions. This dissertation focuses on studying the

impacts of time-sharing HPC systems between application's components or any other pair of workloads and the trade-offs of selecting scheduling techniques to support that resource sharing.

`LogGOPSim` [42] has been widely used in many HPC application interference studies because of its ability to handle different interference traces and to simulate interference impact at large scales. This dissertation leverages and extends `LogGOPSim` for quantifying the performance impact of various interference sources, for collecting BSP intervals' start and end times and to validate and evaluate our modeling approach.

In addition, our study of the GEV model extends `LogGOPSim` with additional capabilities, allowing it to be used to examine application performance at scales that were previously computationally intractable because of runtime considerations. For example, there are a number of `LogGOPSim` workloads which can take days or weeks to simulate, and the discrete-event structure of `LogGOPSim` makes it difficult to parallelize. Using the model described in this dissertation, researchers can instead use `LogGOPSim` to simulate the impact of interference on HPC applications with many small-scale parallel runs and then use our model to extrapolate large-scale application performance impact.

I compared the analytical model approach proposed on this dissertation against related work developed using extreme value concepts. The work described in this dissertation is the first of which I am aware that provides an analytical underpinning to studies of emerging HPC application interference other than shor-lived system activities and that can be used to make quantitative predictions in that regard.

Finally, I presented past work focused on performance impact mitigation strategies for HPC applications. The model I describe in this dissertation provides the theoretical foundations to understand *how* those mitigation techniques work on Bulk

Synchronous Parallel HPC applications.

Chapter 3

Characterizing Performance Impact

3.1 Introduction

Running analytics on the same processor as the application minimizes data movement and provides much finer-granularity control over resource allocation, but can potentially interfere with application performance. Unfortunately, there has been little work quantifying the costs of application/analytics time sharing. Some recent work has shown that applications and runtimes can be modified to schedule analytics in ways that minimize interference [90], but the generality of such approaches is unclear.

This chapter presents a framework that I developed in collaboration with Kurt Ferreira, Scott Levy, Patrick Widener, and Patrick Bridges to quantify the costs and benefits of different approaches to time-sharing processors between applications and analytics. Section 3.2 describes this framework. In Section 3.3, we characterize two different analytics codes and quantify the degree to which time-sharing analytics and

simulation perturbs overall performance. We show that uncoordinated time sharing of cores between applications and analytics can have catastrophic performance consequences. Gang-scheduled time sharing, however, can significantly reduce these overheads to that of dedicating cores to analytics, but requires global synchronization.

We also use the framework to study the degree of synchronization provided by frequently used synchronization mechanisms, as well as the synchronization requirements of gang scheduling-based mitigation mechanisms. In Section 3.4 we show that system-based mechanisms generally offer synchronization certainties within tens of microseconds, but in some cases uncertainties of hundreds of milliseconds are observed in HPC systems. On the other hand, synchronizing collectives offer synchronization guarantees in the order of 10 ms.

Because of the importance of gang scheduling, we then evaluate the degree to which analytics needs to be gang scheduled to minimize application performance perturbation. Our results show that coarse synchronization of analytics activities across nodes, within approximately 10-100 ms, is sufficient to eliminate most time-sharing overheads for many applications. Some applications, however, require gang scheduling to an accuracy of less than 1 ms to eliminate these overheads. These results suggest that alternatives to traditional gang scheduling must be explored.

My contributions to the work presented in this chapter are the design of the performance impact experiments, and the approach to study the impact of *in situ* analytics codes by modeling scheduling events as application jitter. I also collected and characterized all the analytics' scheduling traces required for this model while analytics codes were running in co-location with their coupled simulation codes. The experiments to study the synchronization requirements of gang scheduling were performed by Scott Levy, Kurt Ferreira, and Patrick Widener using the `LogGOPSim` version currently under development in Sandia National Labs.

Overall, this chapter makes the following contributions:

- A simulation-based approach that exploits the concept of application jitter as a means to model the costs of time-sharing in situ analytics;
- A comparison between coordinated time-sharing, uncoordinated time-sharing, and a simple analytical model of the performance of space-sharing of analytics; and
- A study of how the degree of synchronization in the gang scheduling of analytics impacts application performance.

3.2 Evaluating Application/Analytics Performance Interactions

Quantifying the costs and benefits of different approaches to scheduling main application and analytics tasks is potentially challenging, particularly for the case of time sharing an application and analytics on a CPU core [65]. In this section, I present an approach which leverages simulation to study time-sharing impacts and a simple model to analyze space-sharing effects. Additionally, I describe the workloads that we use for these analyses.

3.2.1 Quantifying Analytics Scheduling Impacts

We use a modeling and simulation approach to understand the impact of different strategies to scheduling application codes and analytics. This approach allows a level of fidelity and control not always possible in implementation-based approaches. It

also allows us to examine performance at scales not generally available for systems research.

We model the impact of an analytics task on application performance by modeling the effect of the lost CPU cycles used by the analytics. In the space-sharing case, we assume near perfect strong-scaling of the application. The application slowdown is modeled as the time needed to carry out the computation that would otherwise be performed by the cores that are lost to the analytics task. As an example, if one core out of every 32 cores is dedicated to analytics, the application will take $\frac{1}{31} = 3.225$ percent longer time to compute the same problem. For the time-sharing case, we model the impact of fine-grained analytics scheduling using an analogy to OS noise. From the perspective of the application, each analytics scheduling instance is a CPU detour which is characterized by the start time and duration of the event.

While this approach focuses on the first-order cost of analytics interference from the perspective of lost CPU cycles, it does have important limitations. First, second-order effects of how communication within an analytics task may interfere with application performance are not captured by this approach. Second, this analytics-as-OS-noise approach ignores slowdowns due to increased memory and network pressure. Modern HPC applications and analytics are optimized to access memory in strided and blocked patterns that are easily predicted by hardware mechanisms that should mitigate these effects so long as the granularity of time sharing is not extremely low.

3.2.2 Simulating Time-sharing

To understand the interactions and quantify the costs of time-sharing cores between analytics and simulation codes, we use `LogGOPSim`. We determine time sharing performance interference using `LogGOPSim` by treating the co-located analytics tasks as OS noise. To do so, we first measure the computational requirements of unperturbed

analytics when running in-line with simulation using the Linux `ftrace` utility [75]. The `sched_switch` events provide a trace of CPU time slices used by analytics when best-effort scheduled by Linux alongside the application. We provide these events as a noise trace to the simulator. In each of these cases, the average amount of CPU time allocated to analytics remains unchanged; all that changes is the period over which this time is allocated.

As described in Chapter 2, `LogGOPSim` allows the study of the impact of scheduling synchronization. In this section, I study the effects of using completely uncoordinated scheduling and perfectly gang-scheduling of analytics on simulation performance. A more detailed analysis of the effects of synchronization is presented in Section 3.5.

3.2.3 Application Workload Details

For the experiments performed in this chapter we selected a set of representative HPC computational-science codes based on the variety of operations that they perform and on the different communication patterns that they use. In this section we describe those codes.

- LAMMPS: a multi-domain computational-science application used in molecular dynamics research. In this work, we study the LAMMPS *2D Crack* and *Lenard-Jones* potentials [73].
- CTH: a simulator used to study deformations generated by strong shocks [39].
- HPCCG: an application used to solve conjugate gradient problems [38, 77].
- LULESH: a hydrodynamics simulator [49].

3.3 Performance Interference Characterization

To better understand performance interactions between analytics and simulation, we first characterize the performance behavior of the selected representative analytics codes, Bonds and PreDatA. We then use the approach described in Section 3.2 to obtain an initial evaluation of how co-locating these workloads with different applications influences performance. Specifically, we evaluate how the performance characteristics of these analytics workloads perturb the performance of applications using a time-sharing strategy versus a space-sharing policy that dedicates a portion of the system's processors to analytics.

3.3.1 Noise Characterization

We collected scheduling traces of PreDatA and Bonds while they where running co-located with GTC-P and LAMMPS Crack, using the `ftrace` Linux kernel tracer as described in Section 3.2. For comparison, we also include an OS noise trace from the Chester Cray XK7 TDS (Test and Development System) at Oak Ridge National Lab collected using the selfish detour benchmark of the netgauge tool [40].

Figure 3.1 shows the resulting CPU detour traces. In addition to the Chester OS noise trace, we consider OS noise traces collected as part of previous work [85] from Volta, Muzia, and RedSky, which are Cray XC30, Cray XE6 and SunBlade x6275 systems, respectively. Table 3.1 shows the CPU overhead and the mean (μ_d) of the duration of the interference events, as well as the mean (μ_i) of the inter-arrival noise events times.

With the exception of the Volta OS noise, the collected OS noise traces have significantly less CPU overhead than the analytics codes. Moreover, the means of the duration for PreDatA and Bonds are several orders of magnitude above the ones

found in the OS noise traces. Similarly, the analytics codes' mean inter-arrival times are considerably higher (i.e. lower-frequency events). This is critical, because, as shown in a previous work [29], high-duration, low-frequency application noise events generally have dramatically higher impacts on applications performance.

in-situ analytics codes			
	duration		inter-arrival time
Trace	$\mu_d(\mu s)$	$cpu(\%)$	$\mu_i(\mu s)$
PreDatA	460.2	2.44	18904.8
Bonds	722.5	2.80	25810.4
Collected OS noise traces in HPC Systems			
	duration		inter-arrival time
Trace	$\mu_d(\mu s)$	$cpu(\%)$	$\mu_i(\mu s)$
Volta	14.0	4.43	316.2
Chester	1.5	0.07	2282.6
Redsky	2.7	0.63	435.5
Muzia	1.5	0.04	3979.5

Table 3.1: Mean CPU overhead, duration and inter-arrival times for the two *in situ* analytics workloads, along with comparative the OS noise profile statistics.

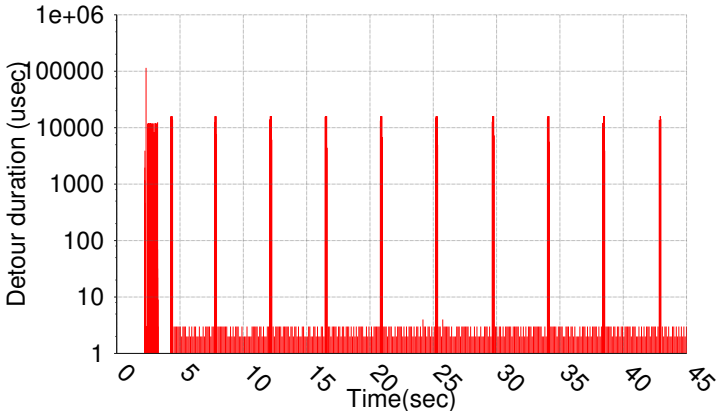
3.3.2 Noise Performance Impact

We next examine the impact of either time-sharing Bonds and PreDatA with simulation or running it space-shared on dedicated cores. To do so, we use the CPU detour traces analyzed in Section 3.3.1 along with the `LogGOPSim` simulator, as discussed in Section 3.2. For the space-sharing case, we allocate either one core out of 16 or one core out of 32 to analytics.

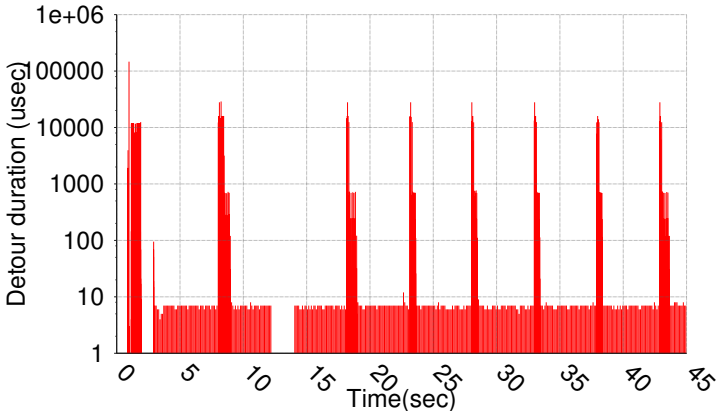
Figure 3.2 shows the effects of time-sharing and space-sharing CPU cores between simulation and analytics codes. For time-sharing, we consider the impact both with and without perfectly coordinated gang scheduling. This figure demonstrates that unsynchronized time-sharing of analytics codes is incredibly disruptive to the per-

formance of all applications, resulting in simulation slowdowns of almost of 1600% in some cases despite the fact that the analytics runs for only 2.5% of the time on average. Bonds and PreDatA result in similar slowdowns. In contrast, the impact of our optimistic model of space-sharing on application performance is minimal, either 3.23% or 6.67%. On real systems, the performance impact would be higher due to data movement and non-uniform memory access penalties.

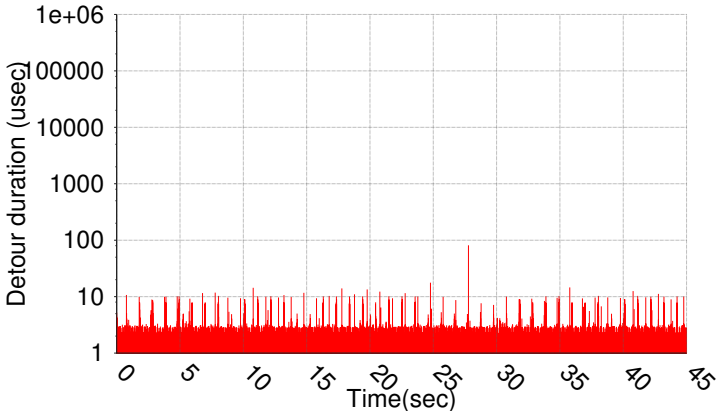
On the other hand, using perfectly coordinated gang scheduling to schedule analytics across the nodes of the system results in minimal slowdowns. In particular, the overhead of analytics drops to near its baseline (2.44% for PreDatA, 2.80% for Bonds) for every application that we considered. This result is consistent with the existing research on OS noise (*cf.* [41]).



(a) Bonds scheduling traces

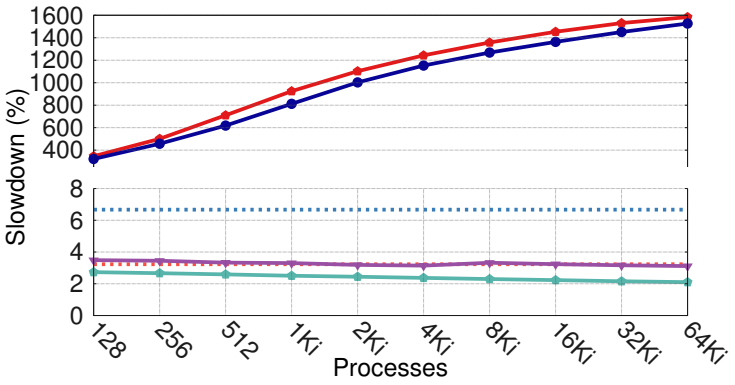


(b) PreDatA scheduling traces

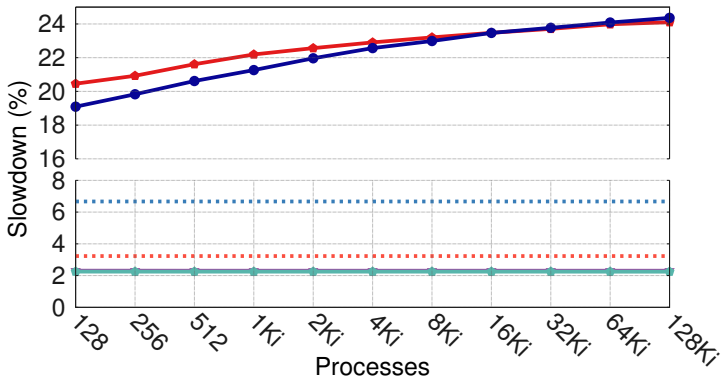


(c) Chester OS noise trace

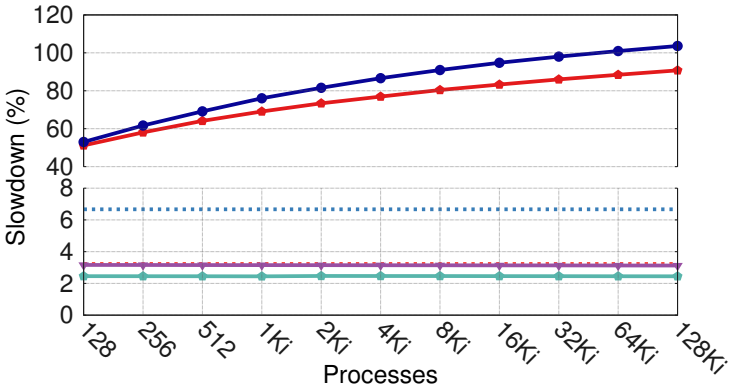
Figure 3.1: Scheduling traces for Bonds and PreDatA *in-situ* analytics codes and noise profile collected for Chester Cray XK7 TDS system.



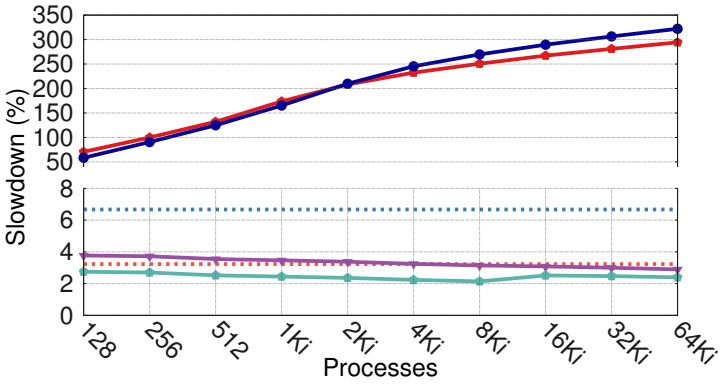
(a) LAMMPS Crack



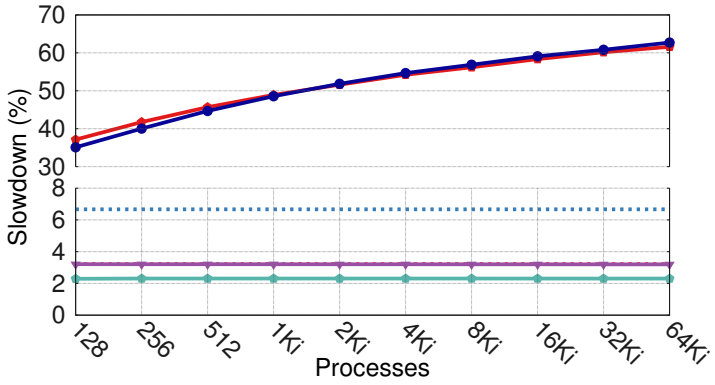
(b) LAMMPS LJ



(c) HPCCG



(d) CTH



(e) LULESH

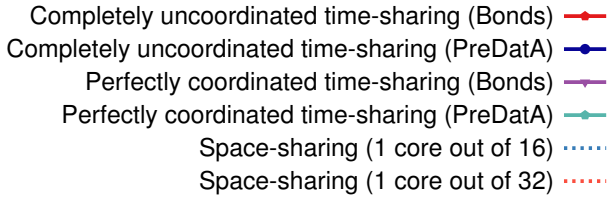


Figure 3.2: Performance impact on applications co-located with Bonds and PreDatA *in-situ* analytics for a completely uncoordinated time-sharing policy, a perfectly coordinated time-sharing policy, and two space-sharing policies.

3.4 Synchronization Mechanisms

Coordination of interference sources across processes has shown benefits on applications' performance. This synchronization can be achieved in several ways. Among the prevalent approaches to coordinate interference activities are leveraging system-based gang scheduling, which requires tight system synchronization services, and using synchronizing collective operations. This section studies the synchronization guarantees provided by those mechanisms.

3.4.1 System-based Synchronization Services

In [45], we examined the degree of synchronization and time-agreement levels offered by two large-scale supercomputers which leverage the Network Time Protocol (NTP) services as synchronization mechanism. Using a simple benchmark we collected clock skew data from two systems: Titan a Cray XK7 system located at Oak Ridge National Laboratory, which use a Gemini interconnet with a 3D torus topology [1]; and Mira system located at Argonne National Laboratory with a 5D torus interconnet [56]. Each of these systems has end-point latencies of less than 3 μs.

The results of these experiments show that NTP can offer synchronization certainties of around 10 μs on Mira. However, surprisingly this level of synchronization is not supported on Titan, where we observed levels of uncertainty of hundreds of milliseconds, with a worst case of 600 ms. This is critical, since applications using system-based gang scheduling often require synchronization guarantees of tens of milliseconds or less to properly mitigate interference, Section 3.5 shows.

The data supporting this study and additional details are publicly available [46] under the DOI 10.13139/1130048.

3.4.2 Synchronizing Collective Operations

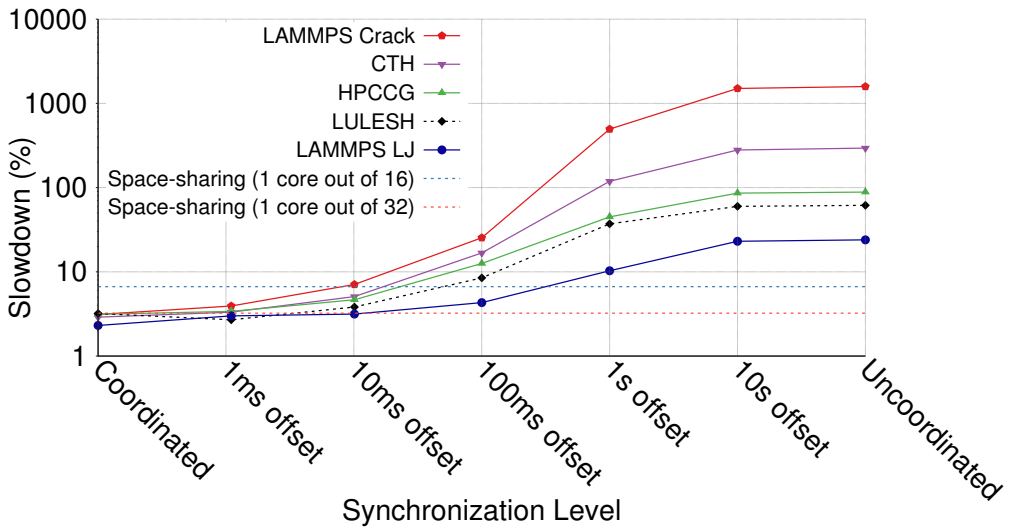
An alternative to provide synchronization to workloads is to incorporate synchronizing collectives in the workload code. As we show in [59], this approach provides synchronization guarantees of 10 ms when using a dissemination algorithm (e.g., MPI_Allreduce). However, MPI standard 3.0 warns about relying on the synchronization capabilities offered by collectives [32]. Additionally, this approach may require modifications of existing workloads code.

3.5 Synchronization Requirements

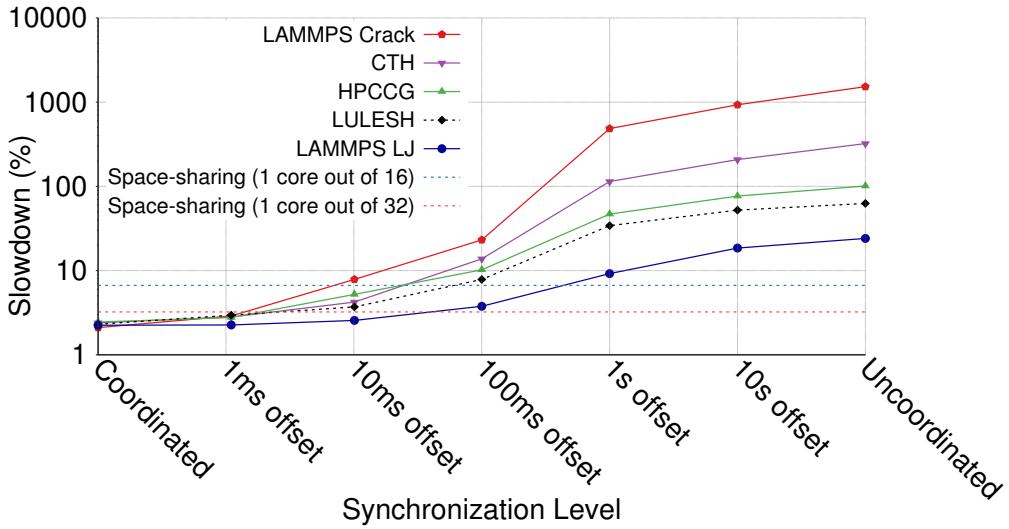
As shown in this chapter, gang scheduling of analytics can reduce the overhead of time-scheduled analytics to as low or lower than space-sharing. However, perfectly synchronizing activities across nodes is impossible in a real distributed system; communication delays and hardware variation limit the degree to which distributed activities can be coordinated. Modern HPC systems, network latency, jitter, and system software concerns limit reliable inter-node synchronization to a few microseconds or possibly tens or even hundreds of milliseconds, depending on the mechanism used and system hardware and software characteristics [45], as described in the previous section.

In this section, we study the level of synchronization required to schedule analytics codes in order to effectively mitigate their impact on applications. To accomplish this, we conduct a series of experiments in which we vary the degree of inter-node synchronization using the capabilities that `LogGOPSim` offers for this, which I described in Chapter 2.

Figure 3.3 shows how the performance of different applications changes as we vary the degree to which the time-shared analytics are synchronized. This figure shows



(a) Applications composed with Bonds



(b) Applications composed with PreDatA

Figure 3.3: Slowdowns for applications co-located with Bonds and PreDatA *in-situ* analytics for analytics synchronization levels varying from the perfectly coordinated case to the completely uncoordinated case for 64 Ki processes. Different levels of synchronization are simulated adding offsets to the time at which noise traces start on different analytics processes.

the performance impact on each application running on 64 Ki processes. Results at other scales are very similar and are elided for clarity of presentation.

In these results, the LAMMPS Crack potential exhibits both the highest performance impact for co-located analytics and the tightest synchronization requirements. For example, if we compose LAMMPS Crack with Bonds analytics, the synchronization variance must be 10 ms or less in order to keep overhead below 10%. The CTH results show similarly high overhead and tight synchronization requirements. In contrast, the overhead of composing LULESH with either Bonds and PreDatA analytics remains below 10% even for an order of magnitude more synchronization variance (100ms). The LAMMPS LJ potential can tolerate much less inter-node synchronization; even just coordinating the execution analytics to within one second is sufficient to significantly reduce the slowdown of the application. Finally, we observe that as the analytics workload is less and less synchronized across processes, the performance impact approaches that of the completely unsynchronized case.

3.6 Analysis and Discussion of Results

3.6.1 Time-sharing vs. Space-sharing

Based on the results of the previous sections, time sharing of analytics can equal or beat the performance of space-shared analytics, given sufficient scheduling support. In particular, both OS-level scheduling techniques and coarse gang scheduling are sufficient to lower the overheads of time sharing of analytics to below that of an *optimistic* estimate of the performance of space-shared analytics. In addition, time sharing can more precisely match the actual performance needs of analytics, as opposed coarse-grained resource allocation via space sharing.

3.6.2 Variations in Application Response

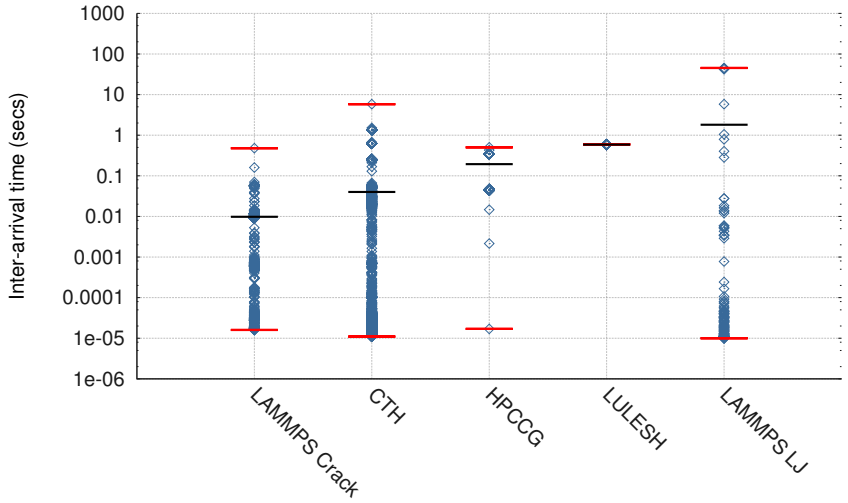
Applications response to time sharing of analytics varied greatly. Some workloads, particularly LAMMPS Crack and CTH, were much more sensitive to interference and required significantly greater efforts to mitigate this interference.

To more broadly understand the source of this variation, we examined the collective communication inter-arrival times of each of the applications we tested. Past research [30] has shown that this characteristic can influence application response to interference from resilience actions. To do so, we analyzed each application communication trace, and plotted the minimum, maximum, and mean collective communication inter-arrival times.

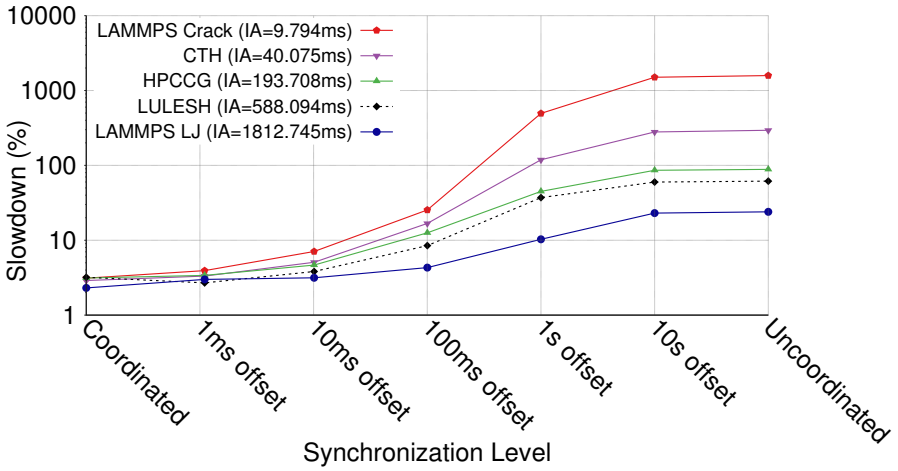
Figure 3.4 shows the mean, minimum, and maximum collective inter-arrival times for collective communications in the five applications studied in this chapter. For comparison, I also reproduce the Bonds subfigure of Figure 3.3, supplementing labels with the mean collective inter-arrival times. This figure shows that collective inter-arrival times correspond well with the degree to which different applications are perturbed by time-sharing the processor with analytics. Specifically, applications with lower collective inter-arrival times (that is, more frequent use of collectives) are both more sensitive to interference from analytics, and require tighter synchronization to mitigate this interference.

3.6.3 Effectiveness of Gang Scheduling

Coarse gang scheduling, for example at the 10 ms granularity, is sufficient to mitigate time-sharing overheads for applications with high collective inter-arrival times. For applications in this regime, existing synchronization techniques are sufficient to reduce the overheads of time sharing analytics on current machines. For applications



(a) Inter-arrival times for a set of applications



(b) Applications' slowdowns for different synchronization levels

Figure 3.4: Measured mean, minimum, and maximum collective inter-arrival times for MPI collective operations for the five studied applications. For reference, the data from Figure 3.3 is reproduced again, with application names supplemented with measured mean collective inter-arrival (IA) time.

with lower collective inter-arrival times, for example CTH and the LAMMPS Crack workload, other methods are needed.

3.7 Summary

In this chapter, I characterized the performance impact of two modern *in-situ* analytics codes on next generation HPC applications. In comparison with other performance interference examined in extreme-scale systems, these workloads have CPU usage patterns of longer duration and lower frequency, which tend to have high impact on applications performance. Although we observed higher requirements in terms of overall CPU overhead, we expect a cap of at most 3% of total CPU reservations for the new analytics codes designs. This allows these workloads to make progress without affecting the applications performance significantly.

I also presented a new approach to simulate next-generation *in-situ* HPC compositions, drawing an analogy with OS interference. Using that approach, we simulated the compositions of analytics with five key applications commonly used for HPC research, and quantified the performance impact of those compositions using different scheduling policies. We showed that time sharing policies provide similar or better performance to applications than space sharing policies, with the benefit of fewer computational resources.

Finally, We also evaluated the synchronization requirements when analytics codes are gang-scheduled using global synchronization mechanisms. We demonstrated that in order to effectively mitigate the impact on applications performance, the systems must provide time agreement within a few tens of milliseconds across nodes. We also showed that applications with shorter inter-collectives times require more stringent levels of scheduling synchronization in order to avoid performance degradation.

Those levels of synchronization certainties are hard to guarantee for some of the current synchronization mechanisms. Alternative approaches with less synchronization requirements may be needed to support performance interference mitigation in next generation large-scale systems. If synchronization within a few milliseconds is

sufficient, current synchronization mechanisms are sufficient to schedule analytics. If tighter synchronization is necessary, on the other hand, then either tighter clock synchronization must be provided, alternative time-sharing performance mitigation strategies must be found, or general time-sharing of analytics should be abandoned in favor of space-sharing.

Chapter 4

Modeling HPC Application Interference

4.1 Introduction

This chapter presents a novel framework for modeling, characterizing, and predicting the impact of interfering activities on bulk-synchronous programming HPC applications. This framework uses *extreme value theory* (EVT) to analyze the impact of interference on bulk synchronous applications; EVT provides tools both for characterizing how the "shape" of BSP periods is impacted by interference and for predicting how changes in the number of application processes will impact the length of these periods. This method allows us to quantify both existing and emerging sources of HPC performance interference and to compare and contrast their behavior.

This chapter describes the following contributions:

- A stochastic model that can be used to characterize and extrapolate the performance of HPC applications in the presence of representative sources of in-

terference in next-generation HPC systems, including predicting the runtime of HPC applications and the effect of interference on BSP computing periods as the application scales;

- A simple and efficient method of characterizing sources of interference and their impact on BSP computing periods of different lengths; and
- An evaluation of the ability of the model to predict the performance impact of different sources of interference on HPC applications.

4.2 Modeling HPC Performance Interference

I propose a stochastic model of HPC application performance interference in order to better explain the impact of different types of interference on application performance, quantify interference sources, and predict how interference will impact application performance at large scales. This section presents an overview of my modeling approach and discusses issues related to the use of this model with real HPC applications. Specifically, this section discusses:

- The general stochastic model I propose for HPC applications and its assumptions about application structure;
- How I estimate parameters for this stochastic model with small-scale runs of applications and interfering workloads;
- How I determine the size and number of application runs needed for estimation; and
- How I extrapolate application runtime performance in the presence of interfering workloads to larger scales.

4.2.1 Modeling Approach

I assume HPC applications can be modeled as BSP applications perturbed by interference. As shown in Figure 4.1, this assumes that an application can be described as a sequence of k intervals each of which comprises *local computation* followed by synchronization. For each interval, I assume the BSP interval lengths at each process are independently and identically distributed according to an unknown (i.e., general) distribution. I similarly assume that perturbation changes the distribution in each interval. Note that I do not assume that the distributions of BSP interval lengths at each process in *different* intervals are identically distributed, as sequences of intervals that correlate to realistic HPC applications contain a number of intervals which may behave differently.

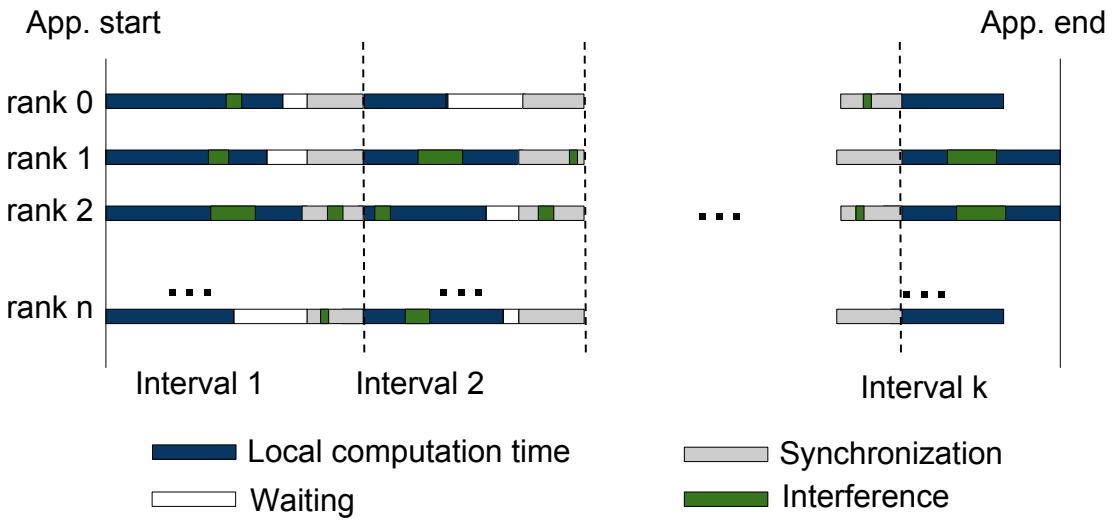


Figure 4.1: BSP Application affected by interference

In terms of the model, the global synchronization at the end of each BSP interval causes that interval to have a length that is the maximum of the generally-distributed i.i.d. random variables that comprise that interval. As such, I model the distribution

of the set of processes in a particular BSP interval using the generalized extreme value distribution. I can then use standard estimation techniques to estimate the GEV parameters for each BSP interval in a perturbed application.

Unlike previous approaches [78, 81], this approach does *not* assume an *a priori* distribution of the compute times in a particular interval on a single process. Positing a distribution for *every* BSP interval in an application would be both error-prone and unwieldy, particularly when those intervals are perturbed by different sources of interference. In contrast, using the GEV distribution requires only comparatively weak independence assumptions; while point-to-point communications potentially violate independence, their impact on the accuracy of this model is modest for real applications, as shown in Section 4.5.

4.2.2 Estimating Model Parameters

For every BSP interval in an application being modeled, I estimate the location (μ), scale (σ), and shape (ξ) parameters of the corresponding GEV distribution. I do so using the block maxima method and maximum likelihood estimation (MLE) previously outlined in Chapter 2. In this method, individual samples are grouped into fixed *block sizes* in which a maximum is taken, and a set of these block maxima are used to estimate GEV parameters. In our cases, the number of processes computing in the BSP interval being sampled is the block size, while the number of instances of that BSP interval (generally from independent application executions) are the number of block samples used for MLE estimation. Determining an appropriate block size and number of block samples needed is somewhat complex and described in Section 4.2.3.

For data collection, I run the application and interfering workload at modest scale either on real hardware or in a simulator such as LogGOPSim. In these runs,

I record the process-relative times (e.g. the times from the end of `MPI_Init()`) at which each process exits a synchronizing collective operation (e.g., `MPI_Barrier`). I then group pairs of times from each process into BSP intervals, providing a sample of the maximum of each block.

4.2.3 Choosing Block Sizes and Number of Blocks

Determining the block size and the number of blocks to use for estimation is a key challenge when using the block maxima method. Using both large blocks and large numbers of blocks is preferable but can be computationally expensive. For example, obtaining 1000 blocks samples with a block size of 1024 can potentially require 1000 application runs each with 1024 processes. On the other hand, choosing block sizes that are too small can lead to significant model bias, and using too few blocks results in high variance in model parameter estimates [14].

I address this problem iteratively. I begin by choosing an initial estimate of the number of blocks to use and block size empirically based on the amount of computation time available for estimation and the desired variance in parameter estimates. I then repeatedly test if model extrapolations pass a statistical test for *all* of the BSP intervals in the application. If all tests pass, I use the current block size; if any test fails, I increase the block size up to a limit again determined by available computational resources. If I reach the maximum feasible block size, I then determine a threshold BSP interval below which to smooth sample maxima, as described below.

Testing for Parameter Bias

I use a method based on normality tests [10] to determine if the block size is sufficiently large to have an unbiased estimate. In particular, I seek to find the block size

at which expected maximum value (EMV) extrapolations are normally distributed. For each tested block size, I use a combination of observed BSP intervals and (unsmoothed) bootstrap samples in order to compute sets of expected maximum values for a given extrapolation. I then perform normality tests on the set of results to determine if the resulting predictions are normally distributed with a 95% confidence interval.

Reducing Required Block Sizes

In some cases, the block sizes required to achieve normally-distributed EMV estimates can be very large. This is particularly true when BSP intervals are very short. Those cases, require large block sizes in order to obtain normally-distributed EMV extrapolations.

To address this, I adapt a technique for smoothing sample extremes [13, 17, 33, 36, 69, 74] developed for when the asymptotic assumption for the normality of the MLE estimators is hard to satisfy for a given block size. Because this smoothing process can modify distributions, however, I seek to minimize its use. In particular, I only smooth after reaching the largest block size at which it is feasible to sample, at which point I choose a threshold BSP interval length θ_s such that all intervals which do not result in normally-distributed extrapolations are below this threshold. I then smooth BSP sample maxima for all intervals with a length below this threshold. I smooth using a simple 5-sample running average low-pass filter; I do not consider more sophisticated smoothing strategies, though this is a potentially important area for future work.

4.2.4 Extrapolating Model Performance

Once model parameters for each BSP interval have been estimated, I then use the EMMA approximation in Equation 2.2 to compute the expected length of each BSP interval at larger scales [81]. In my approach, F corresponds to the estimated GEV distribution for a given interval and m corresponds to the target number of *blocks* at the extrapolated scale. In this case, $m = \frac{p}{n}$, where p is the number of processes at the extrapolated scale and n is the block size used for model estimation as described in Section 4.2.1.

Summing the expected values of all BSP intervals at a given scale estimates the runtime of an application subject to interference at that scale. This assumes that the lengths of successive BSP intervals are statistically independent; this is a reasonable assumption because, by definition, BSP interval $k-1$ ends on all processes before BSP interval k begins.

4.3 Model Validation

To validate this modeling approach, I developed a synthetic BSP application in which the distribution of process runtime between synchronizing collectives can be carefully controlled, a set of interfering workloads, and both empirical measurement and simulation. I first use this framework to validate the approach's ability to estimate and extrapolate runtimes with known BSP interval distributions. Following this, I validate the model's handling of varying interference workloads. Finally, I compare predictions generated by the proposed model with this test against those made using the previously-published EMMA model.

4.3.1 Validation Framework

The framework used for validation consists of three main components: a synthetic BSP test program, a set of interfering workloads, and systems on which to evaluate the ability of the model to handle these workloads.

Synthetic BSP Application

I implemented a synthetic BSP application to be able to carefully control computation intervals and validate the model's ability to capture the impact of variations in BSP interval duration. This synthetic application consists of a loop which repeatedly performs local computation and then executes `MPI_Barrier`. It takes as input the desired number of local computation intervals (e.g., number of iterations), the distribution of those times (e.g., Pareto, exponential), and the parameters associated to each distribution (e.g., Pareto scale and shape, exponential mean, etc.). For each iteration, the benchmark uses the GNU Scientific Library (GSL) [34] and the Scalable Parallel Random Number Generators library (SPRNG) [64] to randomly generate a local computation time that follows the provided distribution. Listing 4.2 shows pseudo-code for the synthetic application.

```
barrier_loop(distribution, distribution_params, iterations) {
  for( i = 0; i < iterations; i++) {
    local_comp_time = gen_random_time(distribution, distribution_params);
    local_computation(local_comp_time);
    MPI_Barrier(MPI_COMM_WORLD);
  }
}
```

Figure 4.2: Synthetic test pseudo-code

In this test, each *iteration* of the loop constitutes a sample of the BSP distribution in question. Note that this differs from the modeling approach described in

Section 4.2 for real applications, where each pair of synchronizing collectives delineates a BSP interval to be modeled independently, with each application comprising many independent intervals. As a result, this synthetic test can be used to gather a large number of samples for a given BSP interval length distribution very quickly.

Interference Sources

I assembled a collection of multiple interference sources representing different sources and distributions of interference against which to validate my modeling approach:

- **OS noise:** I use OS noise traces from two HPC systems at Oak Ridge National Laboratory to validate the model's ability to handle well-known OS-based interference source, as well as one synthetic trace similar to the described by Ferreira et al. [29]. For real-world systems, I use interference traces from Titan, a Cray XK7 system, and Rhea, an Infiniband Linux cluster. I used the netgauge tool [40] to collect these interference profiles. I also use a 3.34% CPU overhead synthetic trace with noise events and inter-arrival times following a Gaussian distribution with a mean event duration of 100 ms, inter-arrival times of 3 seconds, and standard deviations of 10 ms and 300 ms. This is similar to the noise traces used by Ferreira et al.

- **Asynchronous checkpointing:** I used a synthetic interference trace that represents asynchronous checkpointing interference containing noise events with a 1-second duration and a two minute period. This profile matches traces used in recent studies of the performance impact of asynchronous checkpointing [30].

- ***In situ* analytics:** I validate the model against the interference effects of *in situ* analytics in two representative applications, the Gyrokinetic Toroidal Code (GTC) [51] and LAMMPS [73]. For this, I used the traces collected for

the experiments described in Chapter 3; for convenience, I refer to the GTC analytics as "PreDatA" (the framework used to implement them in [89]) and the LAMMPS analytics as "Bonds". These CPU overheads of these noise traces are 2.44% (PreDatA) and 2.8% (Bonds).

Workload Execution

I executed the benchmark and interference workloads on both a real-world system and a simulator. When validating against varying BSP interval lengths and no additional interference, I collected results using Titan; the low baseline level of interference of its Cray Linux Environment OS [48] provides a low-variance environment for validation. Titan does not provide a simple means of controlling co-located workloads, making it difficult to use for validation against varying interference sources. Consequently, I use `LogGOPSim` to validate model predictions with varying interference loads.

4.3.2 Validation Against Varying Local Computation Distributions

To validate the model with different application computation distributions, I ran the synthetic application without interfering workloads on the Oak Ridge Titan HPC system. For these tests, the synthetic application runs 1000 iterations with each process, and I used local compute times that were either exponentially distributed with means of 10 ms, 40 ms, and 160 ms, or Pareto distributed with scales of 10 ms, 20 ms, and 40 ms, along with a Pareto shape parameter of $\alpha = 3$. These tests were chosen to represent both exponentially and heavy-tailed compute distributions.

I conducted a single 512-core, 1000-iteration run on Titan to estimate GEV parameters for each computation distribution, resulting in 1000 BSP interval samples

with a block size of 512. I then used the methodology described in Section 4.2 to estimate the runtime of the test-case up to 16,384 cores and compared both measured and extrapolated runtimes. In all cases, extrapolations were normally distributed with no smoothing.

In addition, I also analytically computed the runtime for exponentially-distributed BSP intervals using the well-known formula for the maximum of n i.i.d. exponential random variables:

$$E(X_n) = \frac{1}{\lambda} H_n \tag{4.1}$$

where n is the number of random variables, $\frac{1}{\lambda}$ is the mean and H_n is the harmonic number. For comparison, I also include predictions made using the EMMA method with *a priori* knowledge of the distribution of local computation times in the test-case.

Exponential mean (ms)	shape (ξ)	location (μ) (ms)	scale (σ) (ms)
10	0.01	62.2576	9.755
40	0.01	248.757	39.019
160	0.01	994.806	156.0348

Table 4.1: Estimated GEV parameters for the exponential distribution experiments

Pareto scale (ms)	shape (ξ)	location (μ) (ms)	scale (σ) (ms)
10	0.37	80.662	27.218
20	0.37	161.131	54.451
40	0.37	322.158	108.891

Table 4.2: Estimated GEV parameters for the pareto distribution experiments

Tables 4.1 and 4.2 show the estimated GEV parameters for each experiment, and Figures 4.3 and 4.4 show the observed, modeled, and analytically predicted runtimes

for the synthetic application exponentially distributed with a mean of 160 ms and when using a Pareto distribution with scale parameter of 40 ms and shape 3, respectively. In each case, my proposed modeling approach correctly predicted the experimentally observed execution times with prediction errors less than 1.3% and 2.8% at 16,384 processes for the exponential and Pareto distributions, respectively. For the heavy-tailed Pareto distribution, both EMMA and my method under-estimate execution times; this is a result of limitations of using the EMMA method with heavy-tailed distributions, an effect previously observed by EMMA's creators [81].

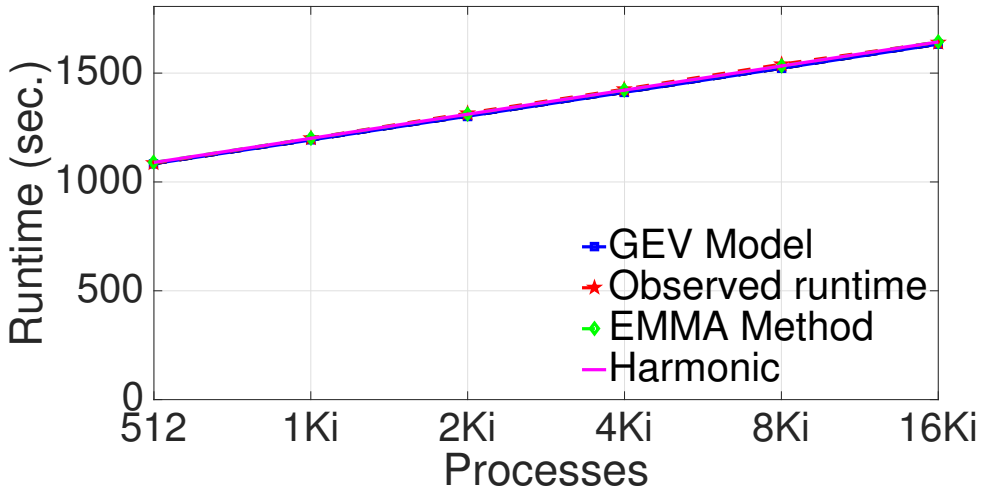


Figure 4.3: Runtime estimates for the synthetic application with local computation times exponentially distributed with mean = 160 ms. The runtimes estimated using the model are compared against the observed runtimes, the runtimes estimated by the model proposed in [81], and the harmonic function of the n-node count.

4.3.3 Validation Against Varying Interference Sources

For validation against varying interference sources, I use the test-case with deterministic BSP intervals and the interfering workloads described in Section 4.3.1. Because of the difficulty in co-locating arbitrary interference workloads with an application

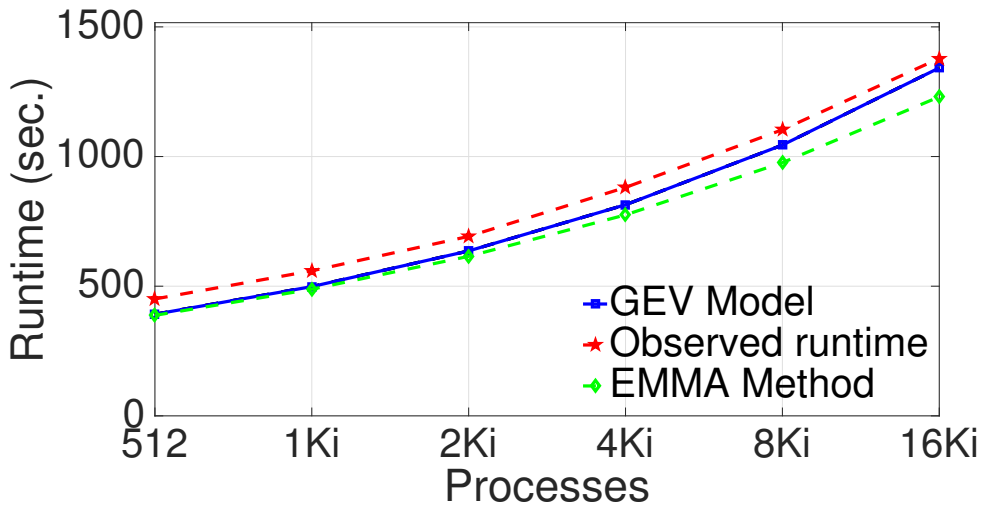


Figure 4.4: Runtime estimates for the synthetic benchmark with Pareto-distributed local computation times (Pareto shape (α) = 3, scale = 40 ms). The runtimes estimated using the model are compared against the observed runtimes and the runtimes estimated by the model proposed in [81].

on production systems, I use the `LogGOPSim` simulator as described in Section 4.3.1. For these experiments I run the synthetic benchmarks with local computation times in the range of 100 μs to 10 seconds. I estimate the GEV model parameters using 1000 blocks of 512 processes and extrapolate the runtime to process counts up to 16,384 nodes. I use a smoothing parameter of θ_s = 600 ms to reduce the required block size to 512 processes.

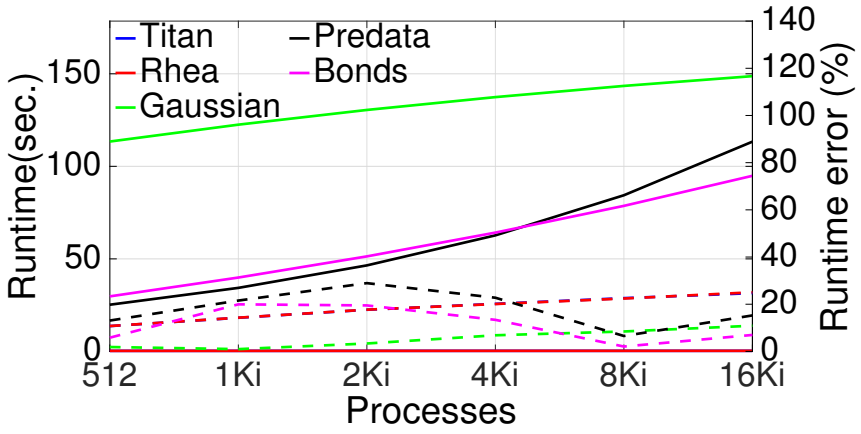
Figure 4.5 shows the GEV model predictions of the performance impact on the synthetic benchmark. The model accuracy improves as the length of local computation times increase. For example, it predicts runtimes up to 16,384 nodes with a prediction error of less than 3.9% for a 100 ms interval. Larger local computation times produce uniformly smaller prediction errors. Smaller local compute times also result in reasonable prediction errors for all interference sources. In particular, prediction errors at 100 μs are approximately 30% or less, sufficient for making performance predictions for most applications (as I discuss in Section 4.5). Predictions

of the impact of the asynchronous checkpointing task are not accurate for small BSP intervals (as discussed in Section 4.3.3) and are not shown for intervals below 100 ms.

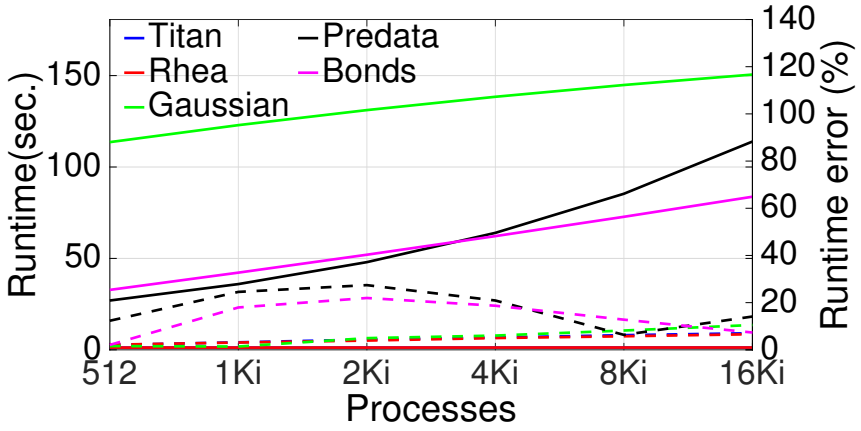
4.3.4 Noise Performance Impact Comparison vs. EMMA Method

I have shown (Section 4.3.2) that the EMMA method can successfully predict execution time when the distribution of local computation times is known *a priori*. To better understand the limitations of this approach compared to our proposed GEV-based approach, I also validate its viability in the presence of interference using `LogGOPSim` simulation, the synthetic test-case with local computation times of 100 ms of duration, and the synthetic trace with Gaussian-distributed noise events. Because the distribution of compute times in the presence of interference is difficult to determine, I evaluate the use of EMMA fitted to multiple candidate distributions.

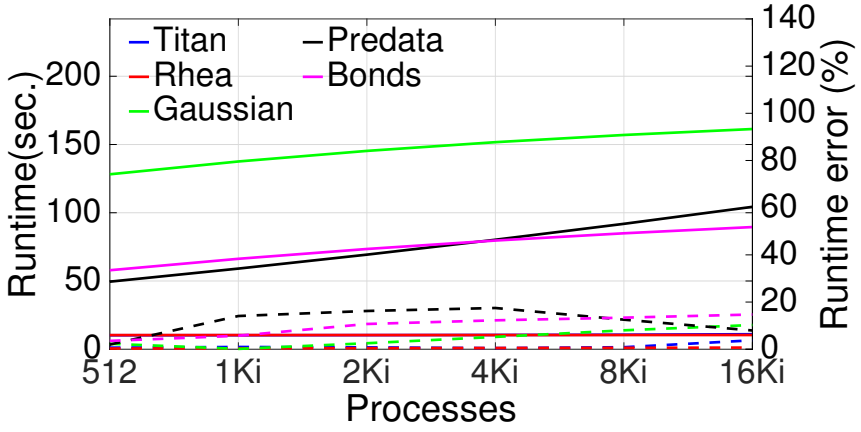
Figure 4.6 shows the results of these experiments. While the EMMA method can predict application runtime if the correct prior distribution is known and fitted (in this case, the Raleigh distribution is most effective), the performance of EMMA is highly dependent on choosing the correct prior distribution, a choice that will vary with both interference source and application BSP interval behavior. This is particularly problematic for real applications which can contain tens or hundreds of separate BSP intervals of varying length and distribution. In contrast, the method described in this dissertation makes much weaker distributional assumptions and so can be applied more easily to a large number of intervals, including those perturbed by interference.



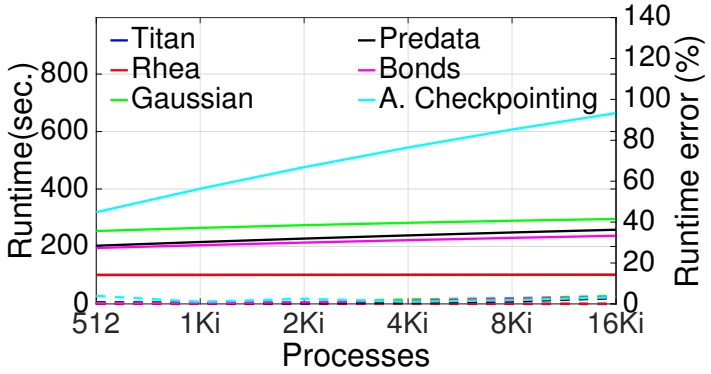
(a) local computation = 100 μs



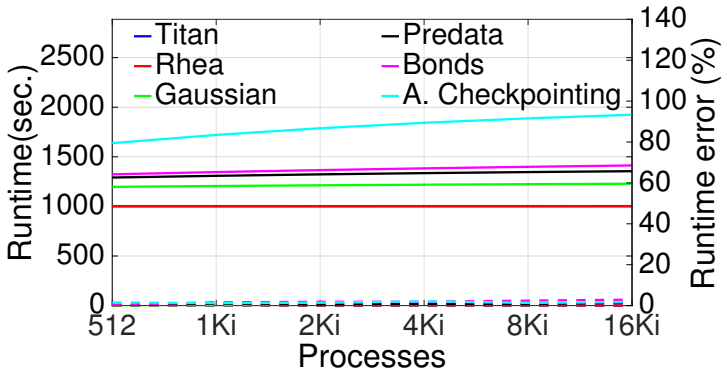
(b) local computation = 1 ms



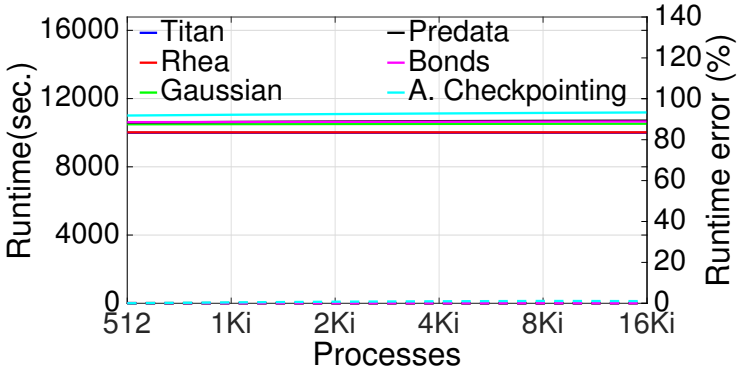
(c) local computation = 10 ms



(d) local computation = 100 ms



(e) local computation = 1 s



(f) local computation = 10 s

Figure 4.5: GEV model performance impact predictions for the synthetic BSP test-case co-located with different interfence sources. Solid lines are the predicted runtimes in seconds, and dashed lines the percent error of the predicted runtimes with respect to the simulated runtimes. Separated figures showing predictions and errors are provided in Appendix A.1.

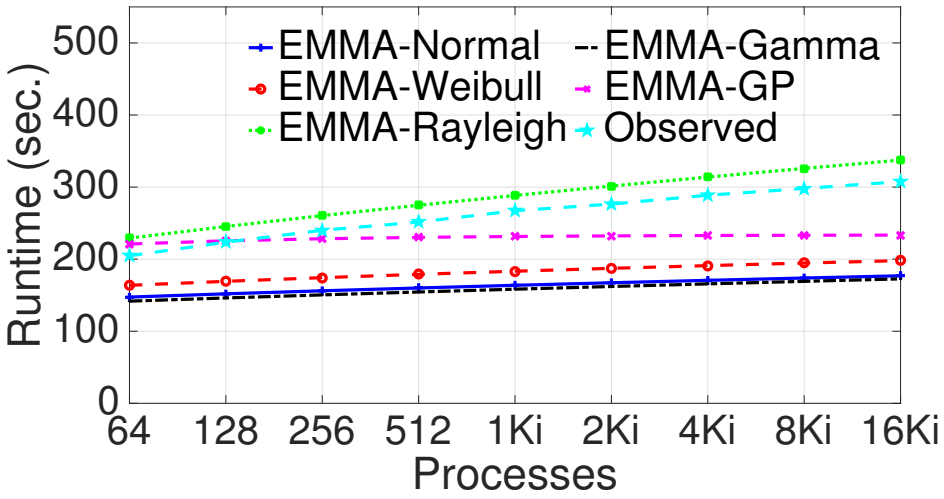


Figure 4.6: Performance impact prediction for the BSP synthetic application in the presence of interference using EMMA EMV method. I use BSP intervals of 100 ms, and an interference source with Gaussian-distributed noise events with a mean duration of 100 ms and an inter-arrival time of 3 seconds. The BSP intervals are fitted to different candidate distributions.

4.4 Interference Workload Characterization

Understanding how a particular source of interference will affect HPC application performance is difficult. As previous studies [29] have shown, interference sources with the same mean interference duration can have radically different effects on application performance. Because of the many potential sources of interference in next-generation systems, including resilience, power management, and analytics, characterizing the potential effects of different interference sources is important in next-generation systems.

The GEV-based model described in Section 4.2 and the synthetic application described in Section 4.3.1 can be used to characterize different interference sources. Specifically, the synthetic application can be used to measure how an interference source perturbs BSP intervals of different lengths, and the proposed modeling approach can be used to project how this perturbance changes at scale. The result is a profile of how an interference source behaves at a particular scale.

In order to demonstrate this capability, I generated a performance impact profile for OS noise, asynchronous checkpointing, and *in situ* analytics (*see* Section 4.3.1) on 16,384 nodes using simulation. For this analysis, I use a block size of 512 nodes and 1000 block maxima samples. I performed this analysis for a range of local computation times between 100 μs and 10 seconds.

Figure 4.7 shows the estimated performance impacts in terms of the projected slowdowns of BSP intervals of different lengths for all validated BSP interval lengths. These results demonstrate the impact of different noise sources on BSP intervals of different lengths, and the importance of being able to characterize and predict the impact of interference on application performance.

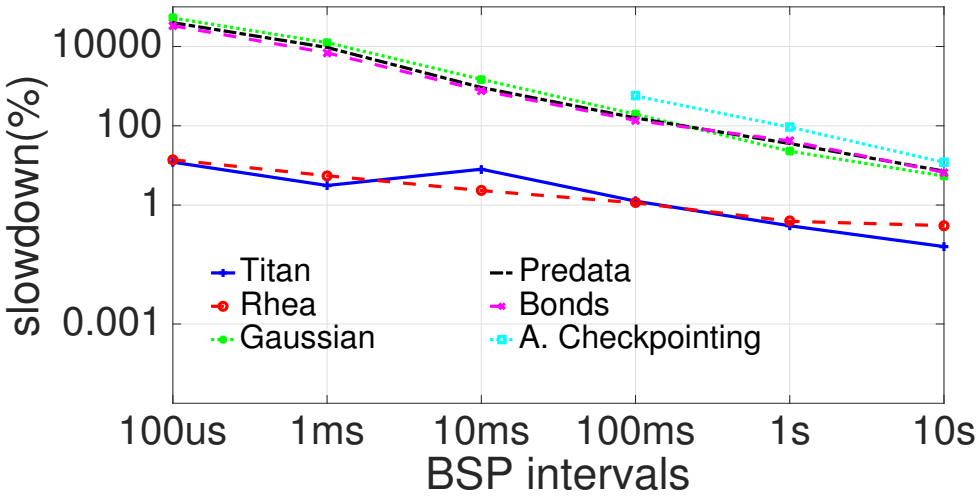


Figure 4.7: Noise profiling of different interference sources using the GEV model with a block size of 512 at 16,384 nodes.

4.5 Predicting Application Performance Impact

In addition to characterizing noise, this modeling approach can also be used to directly predict the scaling performance of applications subject to interference. It does so by applying the model to *every* BSP interval in a complete application. In the remainder of this section, I evaluate the ability of the model to predict the performance impact of three sources of interference on a set of applications and mini-applications.

4.5.1 Evaluation Methodology

I use a set of four workloads, consisting of applications and mini-applications, to evaluate the model's ability to predict application scaling performance. In addition to HPCCG, LAMMPS-LJ, and LULESH, described in Chapter 3, I studied CoMD, which is a molecular dynamics code created as part of Mantevo suite of

mini-applications [77].

Figure 4.8 shows the minimum, mean, and maximum lengths of the intervals between synchronizing collectives for the selected applications. As this figure shows, the statistical distributions of the interval lengths for these applications are diverse, as are the lengths of the intervals.

Because I am evaluating multiple interference workloads, I again use `LogGOPSim`: small-scale runs collect BSP interval length information for estimation, and large-scale runs obtain simulated runtimes against which to compare model predictions. All experiments in this section estimate GEV parameters using simulation runs of 512 processes (i.e. estimation block size is 512), and each 512-process simulation is run 500 times to generate estimation samples for each BSP block. I again use a smoothing threshold of $\theta_s = 600$ ms, the lowest threshold such that all BSP intervals with non-normally distributed extrapolations are smoothed, for all tests.

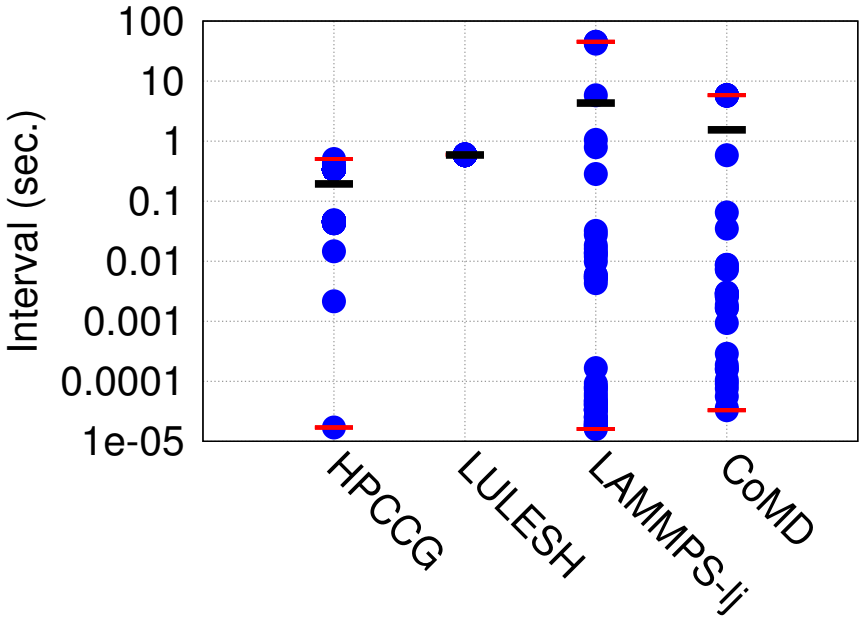


Figure 4.8: Measured intervals between synchronizing MPI collective operations for four applications.

4.5.2 OS Noise and Analytics Interference

Figure 4.9 shows how each of three OS noise profiles affect application slowdown. This figure also shows the error in predicted runtime. Similarly, Figure 4.10 shows how these four applications respond to interference from two analytics workloads. These results demonstrate that our modeling approach accurately captures a wide range of application and interference behaviors, in all cases with prediction error of less than 3.2%.

In the case of OS noise, for example, my approach correctly predicts the performance differences between the low-interference OS noise profiles used in modern supercomputing systems and the high impact of synthetic traces of historical OS noise interference patterns. It similarly correctly predicts the difference in application sensitivities, for example between HPCCG and the other applications.

4.5.3 Asynchronous Checkpointing Interference

In contrast, Figure 4.11 illustrates the current limitations of this model with certain extreme interference workloads. In this figure, I consider one such source of interference: asynchronous checkpointing. As discussed in Section 4.3, my modeling approach accurately predicts the impact of this workload on longer BSP intervals, but is inaccurate for very small BSP intervals. As a result, the approach successfully predicts the performance scaling of LULESH, which has almost exclusively long BSP intervals, but makes very inaccurate predictions for the other three applications, which contain a large number of BSP intervals of 100 ms or less.

Addressing the model's ability to handle low-frequency noise and very small BSP intervals is a key direction for future work.

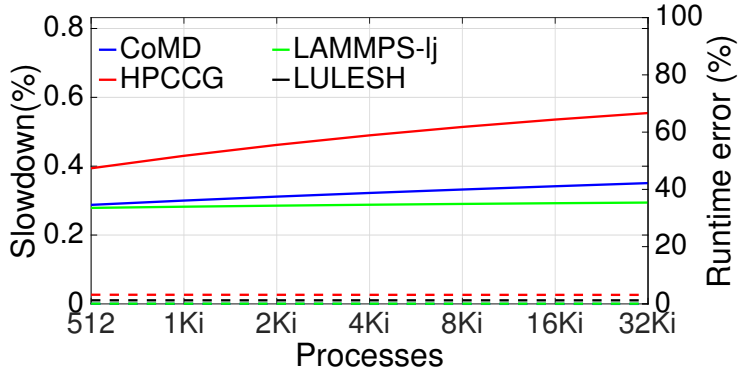
As an initial solution, I propose a *hybrid GEV model* that consists in a com-

bination of the *direct GEV model* explained in Section 4.2 and the noise profiling technique presented in Section 4.4. This technique uses a *hybrid threshold* θ_h to decide the method to model the BSP intervals. I use as reference the application intervals without contention: BSP intervals less than θ_h are modeled using the noise profiling technique, while BSP intervals above θ_h are directly modeled.

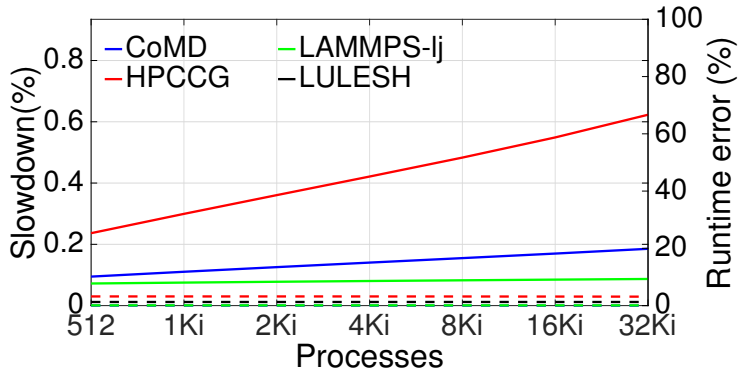
Since the noise profile of a interference source is constructed using a discrete number of BSP intervals lengths, I use interpolation to find the slowdown generated for a specific interval length. As I found that the logarithmic function of the slowdowns is approximately linear with respect to the interval lengths, the interpolated slowdown for the target interval length is based on the linear interpolation of the logarithm of the slowdowns generated by the neighbor lengths. For BSP intervals outside the set of modeled interval lengths, I use nearest neighbor extrapolation.

The use of this hybrid approach is appropiate to study the impact of interference sources with large *minimum block size* requirements, for which accurate approximations are hard to achieve. This is particularly useful when BSP intervals are small and the interference source has a bursty behaviour that makes difficult to collect the required number of samples to obtain regular likelihood estimators even after using the smoothing approach described in Section 4.2.3.

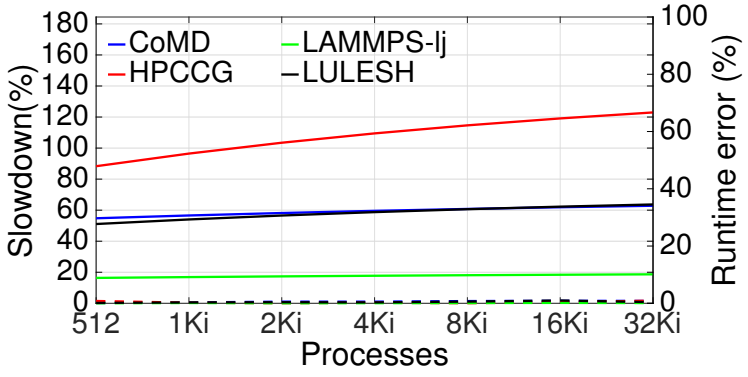
As shown in Figure 4.12, initial results show that this technique may be able to address the estimation and accuracy problems the current techniques encounter when modeling asynchronous checkpointing. Fully exploring and integrating this technique into our overall methodology is a work in progress, however, and completing this research remains a direction for future work. In particular, developing a methodology for determining when to directly estimate BSP interval perturbance and when to interpolate between synthetic measurements is a key outstanding challenge in this direction.



(a) Rhea

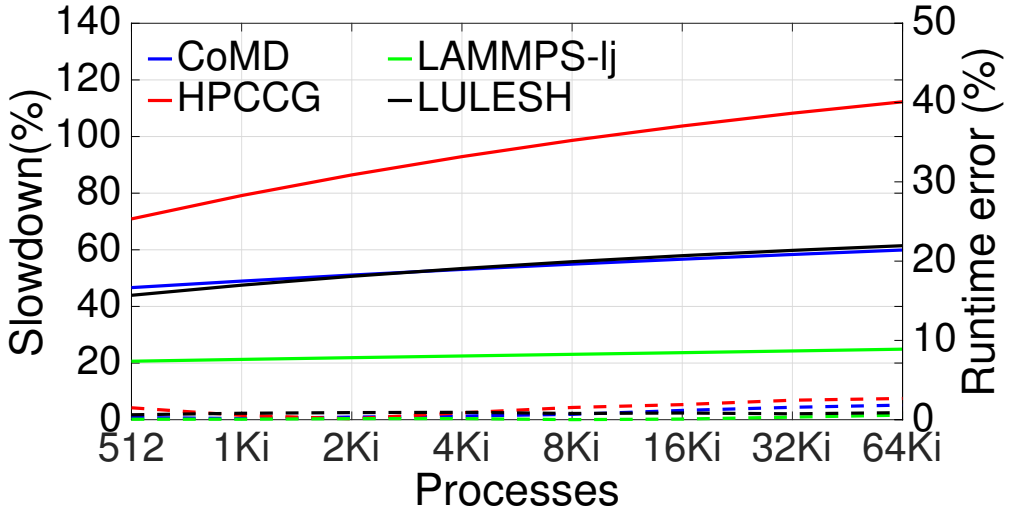


(b) Titan

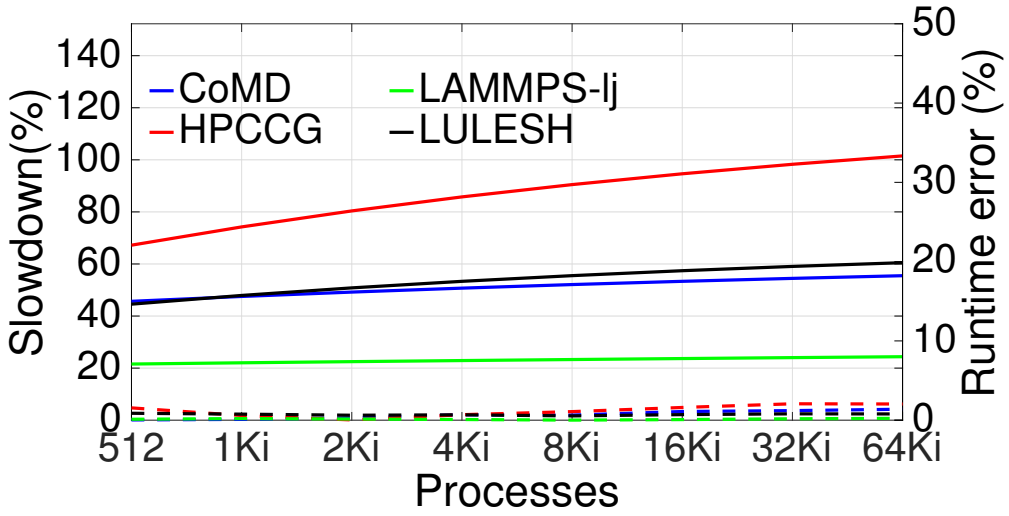


(c) Gaussian

Figure 4.9: Impact estimation using the GEV model of two OS noise interference sources and a synthetic noise trace on a set of applications. Solid lines are the applications' slowdown (percent) and dashed lines are the percent error of the predicted runtimes with respect to the simulated runtimes. Separated figures showing predictions and errors are provided in Appendix A.2.



(a) PreDatA



(b) Bonds

Figure 4.10: Impact estimation using the GEV model of two *in situ* analytics on a set of applications. Solid lines are the applications' slowdown (percent) and dashed lines are the percent error of the predicted runtimes with respect to the simulated runtimes. Separated figures showing predictions and errors are provided in Appendix A.2.

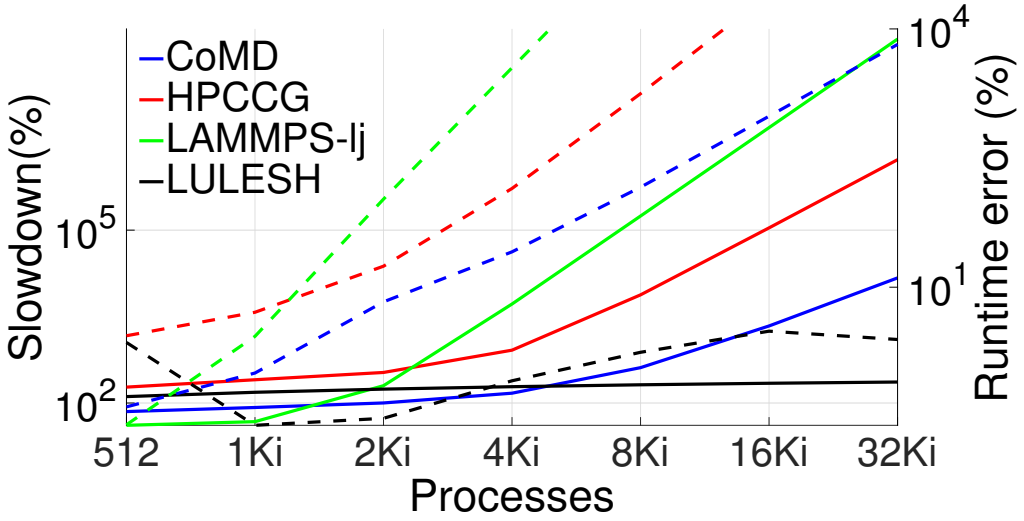


Figure 4.11: Estimation using the GEV model of an asynchronous checkpointing task producing noise events of 1 second duration every two minutes on a set of applications. Solid lines are the applications' slowdown (percent) and dashed lines are the percent error of the predicted runtimes with respect to the simulated runtimes. Separated figures showing predictions and errors are provided in Appendix A.2.

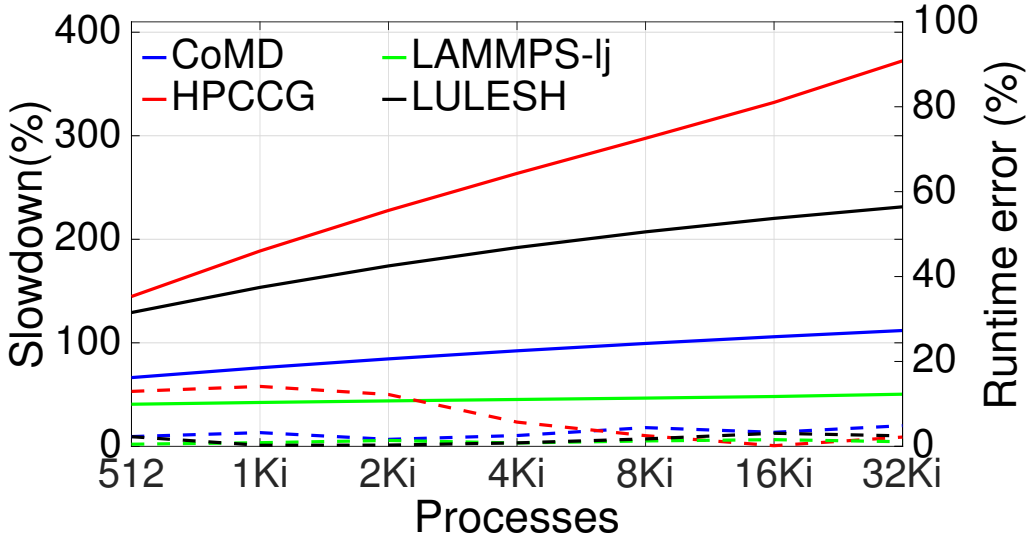


Figure 4.12: Preliminary results estimating the impact of asynchronous checkpointing on application performance using a hybrid combining direct estimation of large BSP interval perturbance and estimation of small BSP interval perturbance using synthetic benchmark results. Separated figures showing predictions and errors are provided in Appendix A.2.

4.6 Summary

This chapter presented a new stochastic modeling approach to understanding, characterizing, and predicting the impact of interference sources on HPC applications. It does so by leveraging properties of the generalized extreme value distribution, making use of prior work on extrapolating the expected value of sets of maximum distributions, and developing a step-by-step methodology to apply these techniques to real HPC applications. The resulting technique has been validated against multiple interference sources, and successfully predicts the performance impact for these sources for multiple HPC applications and mini-applications.

The main direction for future work in this area is improving the methodology's ability to handle very low-frequency interference sources for applications with small BSP intervals. This case presents a fundamental challenge due to the disparity between scope of the interference being sampled and the granularity at which small BSP intervals sample that interference. As a result, directly characterizing such interference in these cases is very challenging.

I propose two different approaches for addressing this challenge. One approach is to examine more sophisticated smoothing schemes than the simple running average low-pass filter technique considered in this dissertation. Other technique is to use a hybrid approach where instead of directly estimating the distribution of very small application BSP intervals when perturbed by noise, large numbers of samples are used with the synthetic application as a proxy for these intervals, as they can be sampled much more cheaply, and different BSP length slowdowns interpolated between.

Chapter 5

Understanding Performance Impact Mitigation Strategies

5.1 Introduction

This chapter examines the use of fine-grained OS scheduling techniques to mitigate time-sharing overheads. This approach is motivated both by past results showing that high-frequency, low-duration interference has little impact on application performance [29,30], as well as research using EDF scheduling [62] to gang schedule virtual machines [61]. My results demonstrate that careful earliest-deadline first scheduling of analytics workloads can virtually eliminate the overheads of time-sharing analytics with applications in most cases, and reduce overheads for the most sensitive applications to 10% or less. I also use my GEV model to understanding *how* this mitigation technique works.

This chapter makes the following contributions:

- An evaluation of the viability of using a fine-grained OS scheduling approach

based on periodic real-time scheduling to mitigate performance interference; and

- A demonstration of the use of GEV modeling techniques for understanding the potential impact of such interference mitigation technique.

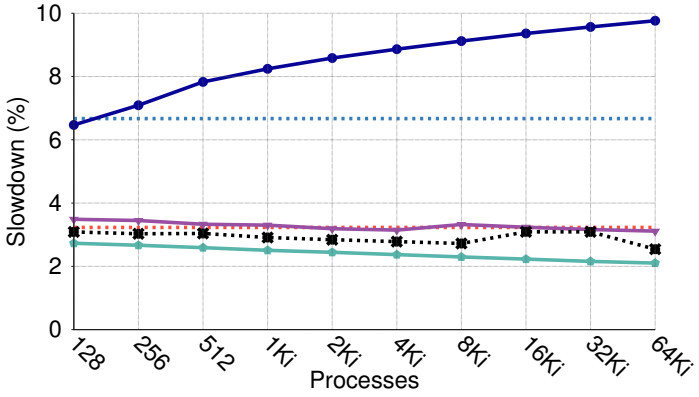
5.2 EDF-based Mitigation of Analytics Interference

I investigate the viability of using EDF scheduling to mitigate time-sharing performance interference. For a given EDF period, I allocate sufficient time to analytics to guarantee its average CPU utilization is at least as high as that measured in Section 3.3.1. Thus, I generally simulate allocating a 3% utilization factor to the analytics code. In this section, I examine how well EDF scheduling mitigates analytics interference at a single period and utilization. Then, I examine how integrating the gang scheduling techniques studied in the previous section with EDF scheduling lowers time sharing overheads.

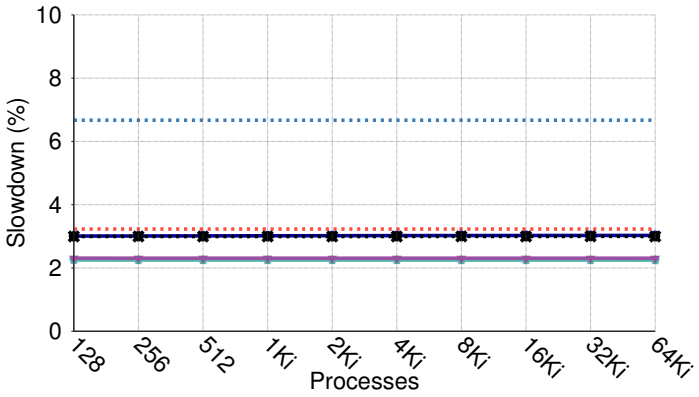
5.2.1 EDF vs. Best-effort Scheduling

I first compare the impact of using fine-grained EDF-scheduled time sharing of processors. I simulate an EDF scheduler with a period of $T_i = 10$ ms and analytics utilization factor of $u_i = 3\%$. To simulate EDF scheduling, I process the noise trace with a utility that transforms it using an earliest-deadline schedule with a given period and slice. Note that this assumes that analytics can be delayed significantly without perturbing application performance. Existing systems such as ADIOS [63] and TCASM [3] use actual or virtual copy techniques to do so. In addition, I simu-

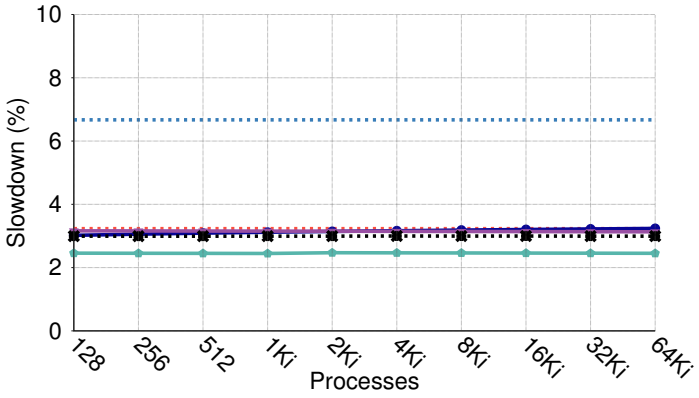
late schedulers with both completely uncoordinated scheduling periods and periods that are perfectly synchronized across nodes.



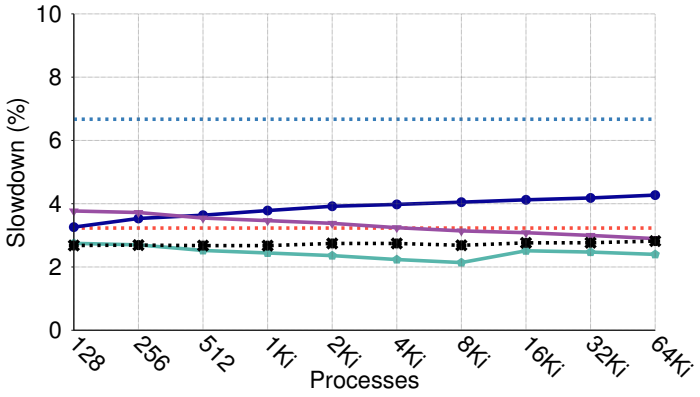
(a) LAMMPS Crack



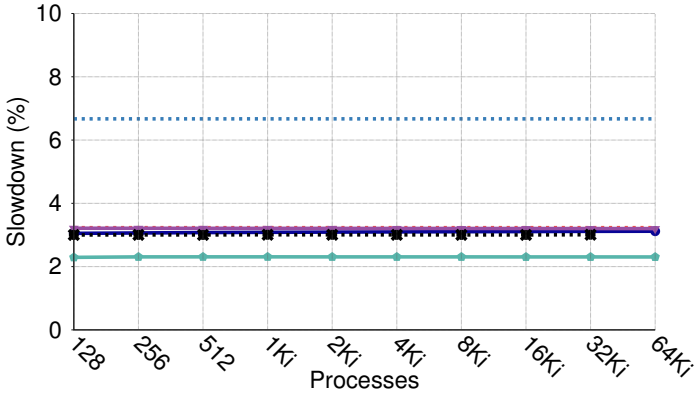
(b) LAMMPS LJ



(c) HPCCG



(d) CTH



(e) LULESH

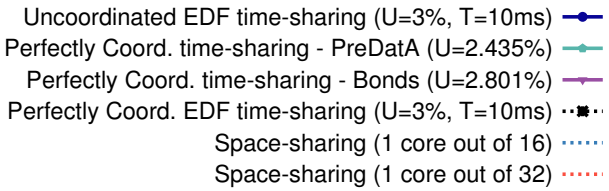


Figure 5.1: Performance impact of various applications time-sharing CPU cores with Bonds and PreDatA under different scheduling policies. For EDF policy, I use a period of 10 ms and an utilization factor of 3%

Figure 5.1 shows the results for those experiments. Most importantly, uncoordinated EDF scheduling of analytics dramatically reduces the performance impact of time-shared analytics across the range of node counts. For example, the performance slowdown of EDF-scheduled LAMMPS Crack on 64 Ki nodes is approximately 10%, compared with the almost 1600% observed in Section 3.3.2. For every other workload, uncoordinated EDF scheduling reduces the performance impact of time-shared analytics to less than 4%, compared to the 50-300% slowdown observed when using uncoordinated time-shared execution previously shown in Figure 3.2. Synchronizing the scheduling periods of the EDF schedulers across nodes further reduces interference, particularly for LAMMPS Crack.

5.2.2 EDF Scheduler Synchronization Impact

Based on the results of Section 5.2.1, I next examine how different levels of synchronization between different EDF schedulers impact simulation performance. For these experiments, I used the same offsetting mechanism used in Section 3.5. I again used a period of $T_i = 10$ ms and a utilization factor of $u_i = 3\%$, and used synchronization offsets with five different standard deviation values: 0s (perfectly coordinated), 300 μs, 1500 μs, 3000 μs, and ∞ (completely uncoordinated).

Figure 5.2 shows that most applications benefit little from EDF scheduler synchronization. While the most sensitive applications, specifically LAMMPS Crack and CTH, do obtain some additional benefits, the general amount of overhead is low and the necessary synchronization needed to further reduce this overhead is comparatively high.

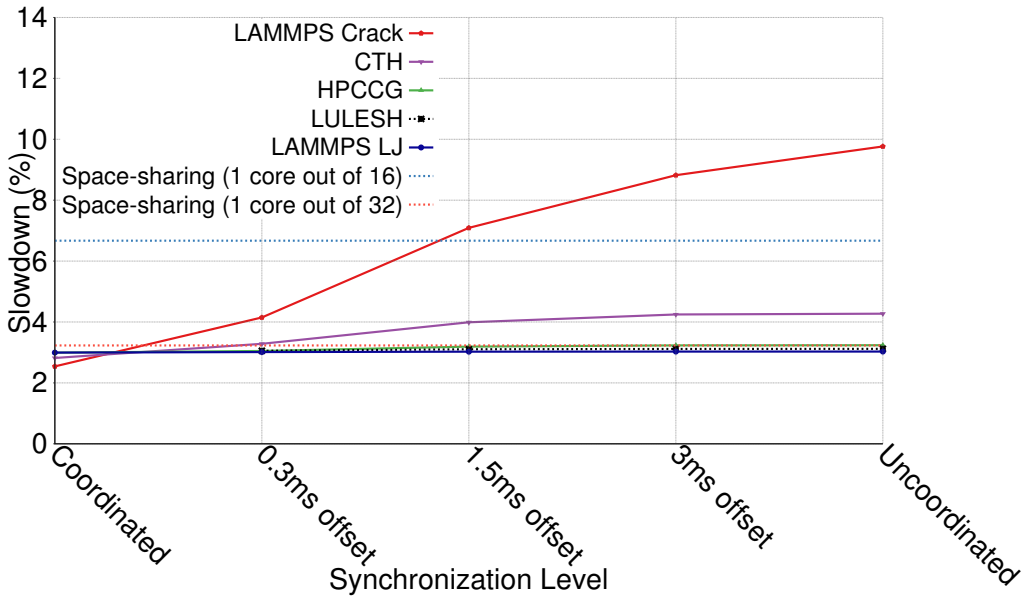


Figure 5.2: Slowdowns for applications co-located with an EDF-scheduled workload with a utilization factor of 3% and 10 ms period for a 64ki nodes count. I use different levels of synchronization for the co-located workload by adding offsets to the time at which noise traces start on different processes.

5.2.3 Gang Scheduling vs. EDF Scheduling

Gang scheduling and EDF scheduling both successfully mitigate noise, though with effectiveness depending on the application characteristics. Based on these results, the most appropriate technique depends heavily on the expected application load. As described in Chapter 3, current synchronization techniques can effectively mitigate performance interference for applications with high inter-collective times.

Applications with lower inter-collective times require other methods. EDF or another similar fine-grained fair share scheduling algorithm are possible, as is adding collective communications to the analytics to tightly coordinate their activities. Various fair share schedulers are available as optional features in modern commodity operating systems; many HPC operating systems do not necessarily include them,

however. Similarly, current HPC job launch systems do not currently support time-sharing of cores between different executables; some applications implement this sharing themselves.

Finally, while each technique is valuable, combining the two appears to provide only marginal additional benefits. System software and application designers should consider the costs and benefits of each, and choose the appropriate one for their application and system based on the application's communication requirements, the ease with which sufficient gang scheduling can be provided, and the availability of fine-grained scheduling disciplines in the underlying operating system.

5.3 Guiding Interference Mitigation

In addition to predicting application performance impact, extreme value modeling of HPC application interference can also provide guidance on approaches to mitigate the performance impact of interference. In this section, I demonstrate the use of the modeling approach described in Chapter 4 to predict the mitigation impact of EDF and to explain its effect on application performance. Previous work has hypothesized that EDF scheduling can decentralize the gang scheduling of interference events [61]. Using the extreme value model that I propose in this dissertation, I profile the impact of EDF scheduling on interference sources and examine how it impacts application performance to better understand its potential effectiveness.

5.3.1 Using EVT to Characterize EDF-scheduling Mitigation Impacts

I first model and simulate the performance impact of the PreDatA *in situ* analytics interference source on different BSP interval lengths, simulating scheduling of the

interference with both best-effort scheduling and EDF scheduling with a scheduling period of 10 ms. Because the average CPU utilization of PreDatA is 2.44%, I simulate allocating a 3% utilization factor to the EDF-scheduled workload, enough to meet PreDatA's CPU demands. I use the synthetic application described in Section 4.3.1 with local computation times ranging from 100 μsec to 10 sec and 1000 loop iterations, and I gather estimation data using 16,384 process runs.

Figure 5.3 shows the impact of EDF scheduling on BSP interval slowdown. Using EDF scheduling reduces the slowdowns for these intervals by several orders of magnitude, and works particularly well for BSP intervals longer than the EDF scheduling period. The predicted slowdown for these intervals falls to the level of the amount of CPU allocated to the analytics workload, 3%.

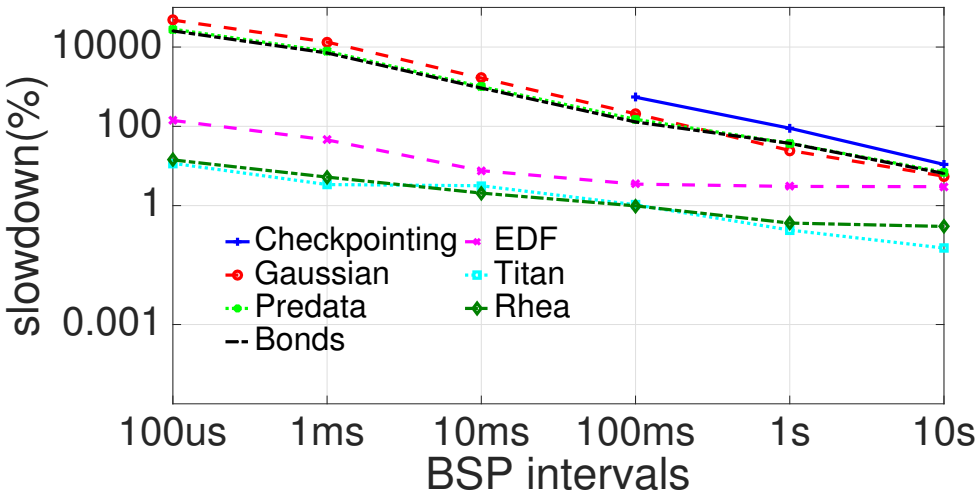
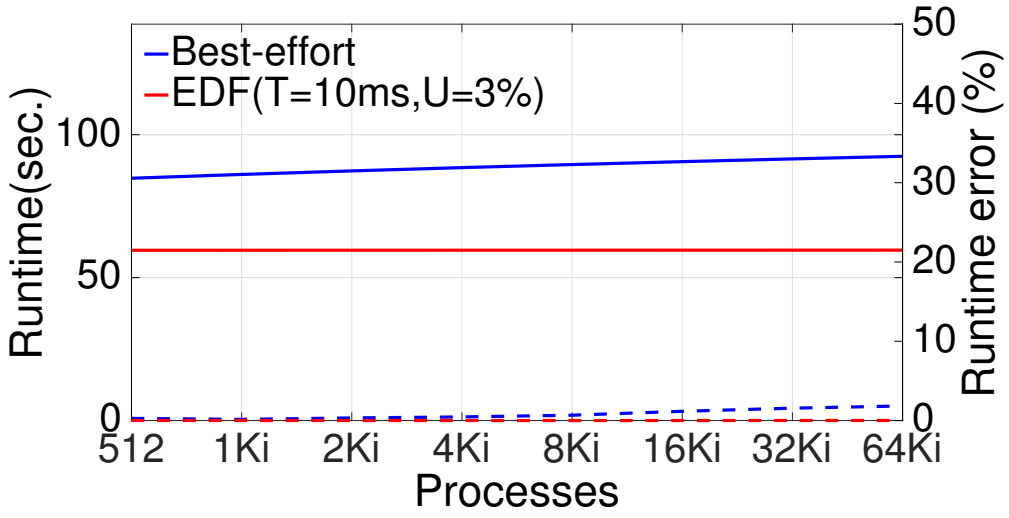


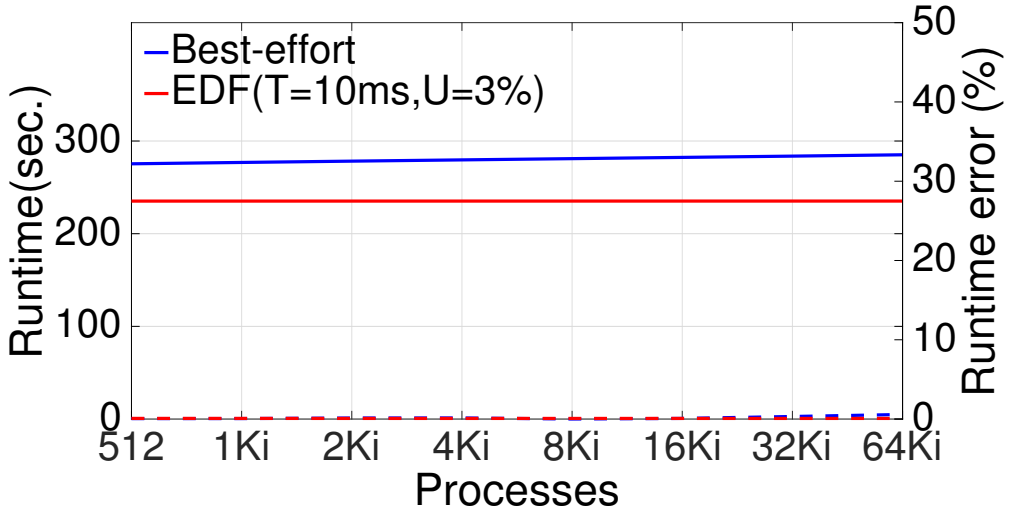
Figure 5.3: EDF ($T_i = 10$ ms, $u_i = 3\%$) scheduling mitigation effect on the synthetic BSP test-case for a range of interval lengths at 16,384 processes

This performance impact also carries through to applications. Figure 5.4 shows how EDF scheduling of the PreDatA analytics interference source impacts a set of applications both in terms of runtime and prediction error. As shown in the figure, the GEV modeling approach accurately predicts the performance reduction when

interference is EDF scheduled, and has a runtime prediction error of less than 3.2% at 65,536 nodes.

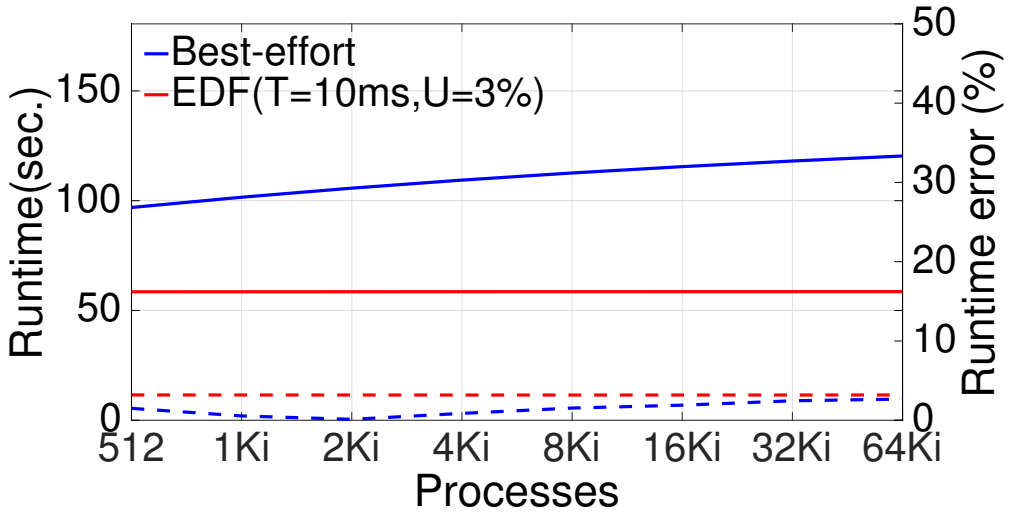


(a) CoMD

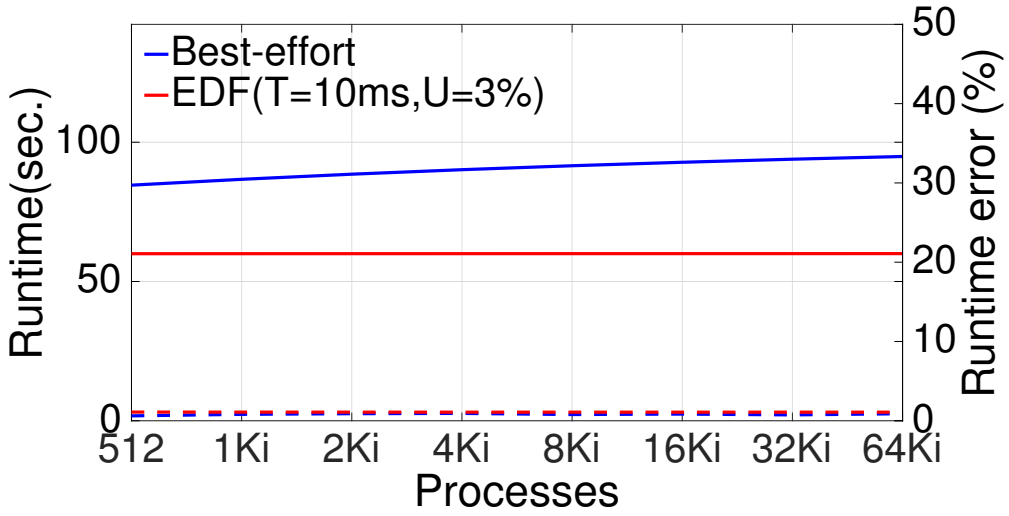


(b) LAMMPS lj

Illustrating the source of this performance improvement, Figure 5.5 shows the cumulative distribution functions of the different BSP intervals perturbed by PreDatA for a 512 node count (the number of nodes that I used for the base case in order to compute GEV parameters). EDF-scheduled interference results in BSP intervals of



(c) HPCCG



(d) LULESH

Figure 5.4: Impact estimation using the GEV model of PreDatA on a set of applications using EDF and best-effort scheduling policies. Solid lines are applications' runtimes in seconds and dashed lines are the percent error of the predicted runtimes with respect to the simulated runtimes. Separated figures showing predictions and errors are provided in Appendix A.3.

essentially fixed length, while the best-effort scheduled interference has a significant tail.

Table 5.1 shows the estimated GEV parameters. The values of the estimated parameters help to explain the mitigation effects of EDF. As described in Chapter 2, the value of the scale parameter defines the dispersion of a probability distribution. In this case small values are preferable. As can be seen, the scale parameter is significantly smaller when analytics is scheduled using EDF for any size of the BSP interval, meaning that one of the effects of EDF is that the application's BSP intervals lengths become more concentrated around a central value.

The shape parameter which defines the tail of the probability distribution provide more information on the effects of mitigation. For both cases and for all interval lengths the shape parameter is greater than -0.5, which means that the likelihood estimators are regular and meet the asymptotic condition of the GEV distribution, as described in Chapter 2. This parameter has a maximum value of 0.11 for PreDatA EDF-scheduled and a maximum value of 0.42 when a best effort policy is used. The greater this parameter, the more heavy-tailed is the distribution. The impact of PreDatA is particularly high on very small intervals (e.g., less than 10 ms) where the resulting distribution is Fréchet.

The location parameter shows that the effect of EDF is to reduce significantly the right shift of the distribution with respect to best-effort. This is especially true for very small intervals. For example, for the 100 μs interval length the shift to the right is reduced by 30 times. In general, the combination of small location and small value of shape is desirable, and define well-behaved BSP interval distributions with potentially low impact on applications.

5.3.2 Trade-offs with Selecting EDF Scheduling Parameters

In this section, I demonstrate the capability of the GEV-based model as a tool to inform the design of EDF schedules. For this analysis, I simulate the synthetic

	Best effort			EDF		
Interval	shape (ξ)	location (μ)	scale (σ)	shape (ξ)	location (μ)	scale (σ)
0.1	0.42	19.45	8.9	-0.11	0.6	0.05
1	0.42	21.25	8.81	-0.34	1.66	0.03
10	0.09	42.05	12.66	-0.11	10.81	0.04
100	-0.11	191.06	20.78	0.11	103.49	0.04
1000	-0.22	1237.03	124.63	0.11	1030.42	0.2
10000	-0.25	10508.77	218.19	0.09	10300.44	0.18

Table 5.1: Estimated GEV parameters for PreDatA using best-effort and EDF ($T_i =$ 10 ms, $u_i = 3\%$) scheduling for the synthetic benchmark with BSP intervals between 100 μs and 10 s for a 512 nodes count (base case).

benchmark time-sharing CPU cores with an EDF-scheduled workload with different EDF parameters configurations.

First, I study the effects of varying EDF slices and periods while keeping a constant utilization factor of 3%. I use different EDF periods in the range on 10 ms to 10 s. For this analysis I applied the GEV modeling approach described in Chapter 4 with a block size of 512 nodes, and a block number of 1000. Figure 5.6a shows the impacts of those EDF workloads at 16,384 processes for different BSP intervals and Figure 5.6b shows the impacts across different process counts for a BSP interval of 100 ms.

Next, I compute the cumulative density functions of the resulting distributions for a BSP interval of 100 ms. Table 5.2 and Figure 5.6c show the computed GEV parameters and the resulting CDFs for the 100 ms BSP interval. The resulting GEV distribution parameters confirm that the granularity of the implicit synchronization provided by EDF can be controlled by the length of the period. Smaller periods provide better synchronization.

The scale parameter helps to explain this observation. For example, the scale parameter for a 10 ms EDF period is 8 and 130 times smaller than the scale parameter

for a 100 ms and a 1000 ms EDF period, respectively. This means that the distribution of BSP intervals is less dispersed in the former case and better synchronization can be achieved, Figures 5.6a and 5.6b confirm this with lower EDF periods having a significantly greater mitigation effect despite that all the EDF workloads have the same utilization factor (e.g., $u_i = 3\%$).

EDF period (ms)	shape (ξ)	location (μ)	scale (σ)
10	0.13	103.5	0.04
100	-0.03	106.11	0.32
1000	-0.1	146.41	5.22
10000	-0.22	447.49	40.14

Table 5.2: Estimated GEV parameters for EDF scheduled analytics for different EDF periods and $u_i = 3\%$ for the synthetic benchmark with BSP intervals of 100 ms.

For all the cases, the shape parameters have small values determining distributions tending to be lightly tailed. However, an interesting observation is that EDF-scheduling periods equal or greater than the length of the BSP interval (i.e., 100 ms in this case) have slightly better behaved tails (i.e., smaller shape values) but larger location and scales parameter (as shown in Table 5.2 and Figure 5.6c) which have a high impact on application performance (see Figure 5.6a).

Next, I performed a similar experiment but this time using a constant EDF-period of 10 ms and varying the utilization factor. Table 5.3 shows how the scale parameter increases as the utilization factor increases; the same is true for the location parameter. The shape parameter values are small for all cases. These resulting GEV parameters are consistent with the slowdowns shown in Figures 5.7a and 5.7b.

Figure 5.8 summarizes all the tradeoffs that I presented above. Short periods improve synchronization of interference events across processes and therefore reduce the performance impact on applications. However, short periods increase scheduling overheads, so the cost of short scheduling periods is potentially relevant to understand

utilization factor (%)	shape (ξ)	location (μ)	scale (σ)
3	0.13	103.5	0.04
5	0	105.83	0.11
10	-0.09	111.95	0.23

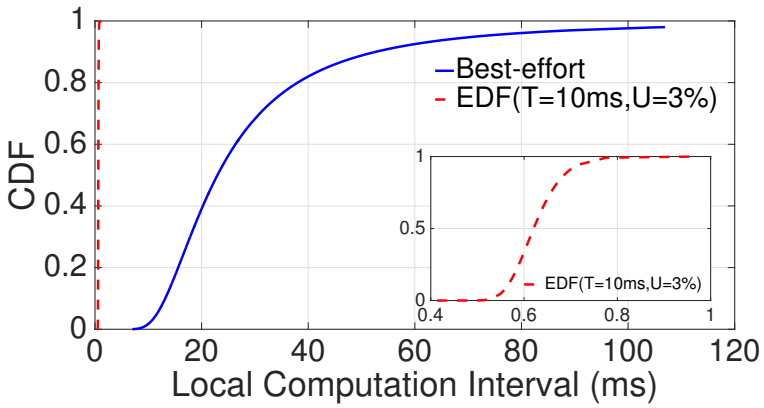
Table 5.3: Estimated GEV parameters for EDF scheduled analytics for different EDF utilization factors and an EDF period of 10 ms for the synthetic benchmark with BSP intervals of 100 ms

if using EDF scheduling is actually viable in real systems or requires short scheduling periods that may not be desirable.

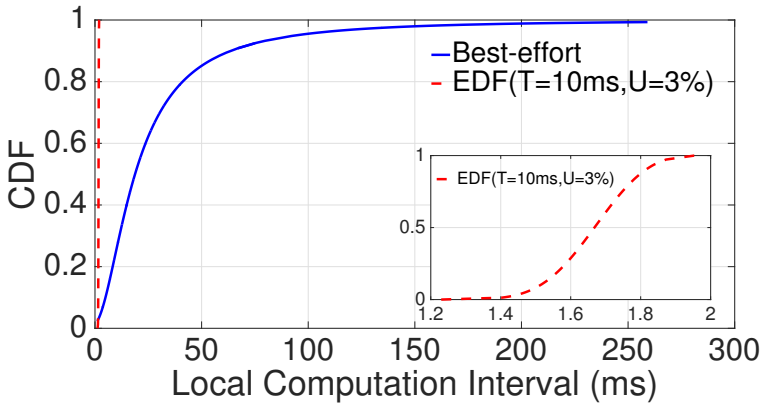
As expected, scheduling EDF workloads with lower utilization factors helps to improve the application performance. However, there is a tradeoff between the performance impact generated on applications as a result of the utilization factor allocated to analytics and the associated analytic's processes *scheduling delay* and *scheduling latency*.

I next estimated the scheduling delay and scheduling latency of Bonds analytics when scheduled using EDF with different values of period and utilization. In this context, scheduling latency is the difference between a process wakeup time and the process switch time. Scheduling delay is the difference between the observed process's end time and the computed end time if the process was running continuously starting at process switch time. From the collected trace of best-effort scheduling events for Bonds, I reschedule the tasks in EDF schedules with different values of slice and period. The scheduling delay and scheduling latency are then computed according with the definition I provided above. Figure 5.9 shows the result of this estimation. The tradeoff between analytics utilization factor and scheduling latency is the most important. Although lower values of analytics' utilization factors have less impact on simulation, they have a significant impact on scheduling latency, which could be critical or not depending on the analytic's degree of communication. In general,

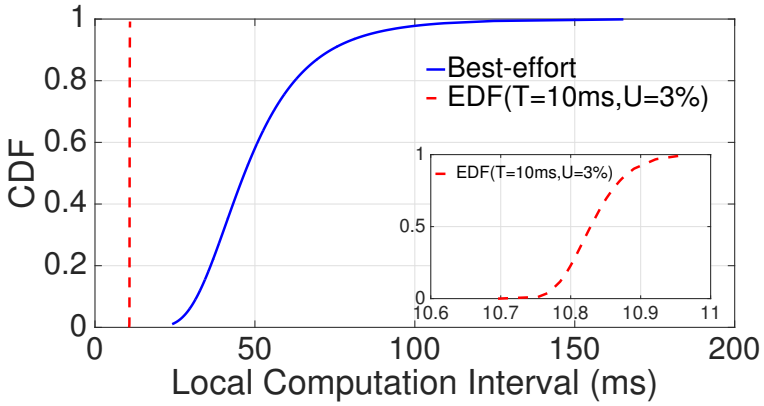
if communication is low in analytics and simulation does not depend on partial analytics results, the impact of scheduling delay is potentially low.



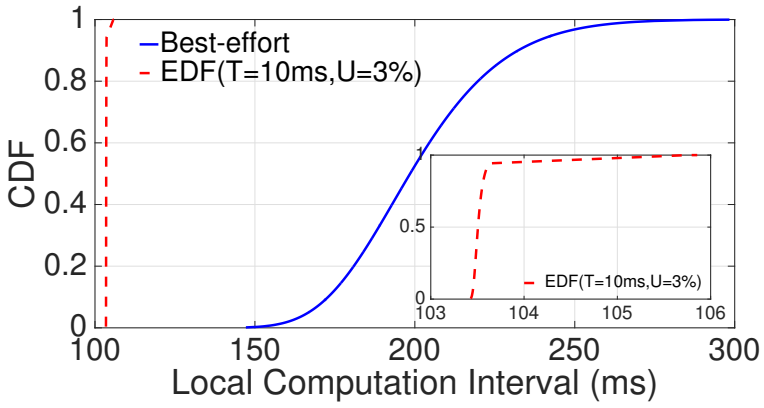
(a) local computation = 100us



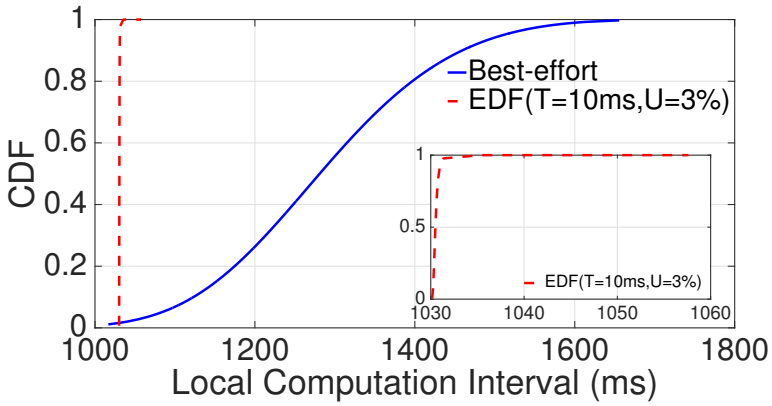
(b) local computation = 1 ms



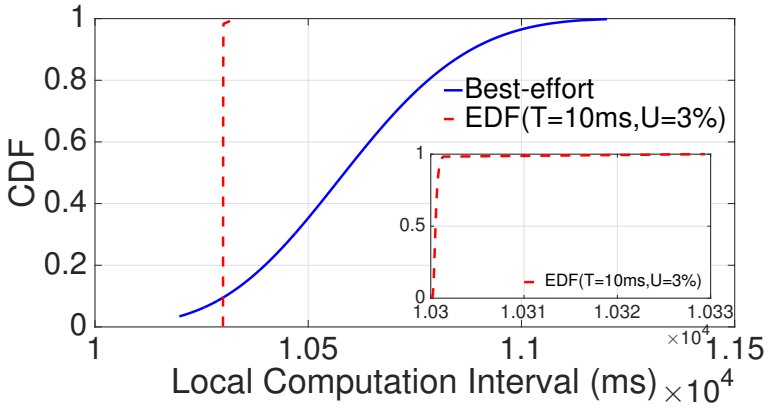
(c) local computation = 10 ms



(d) local computation = 100 ms

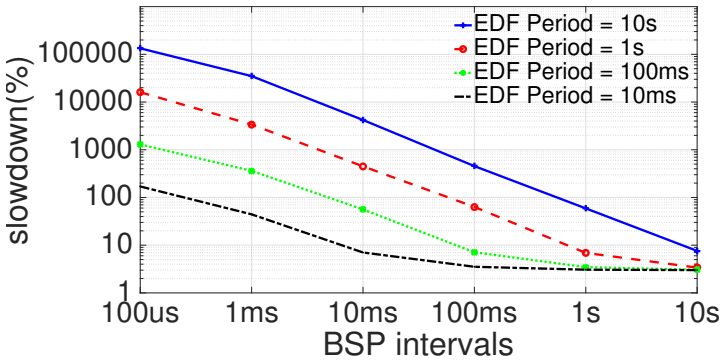


(e) local computation = 1s

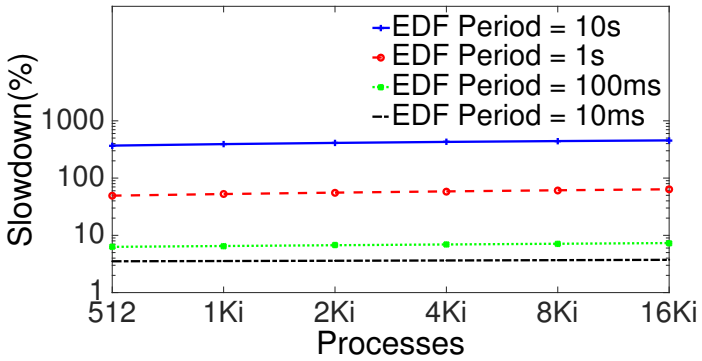


(f) local computation = 10s

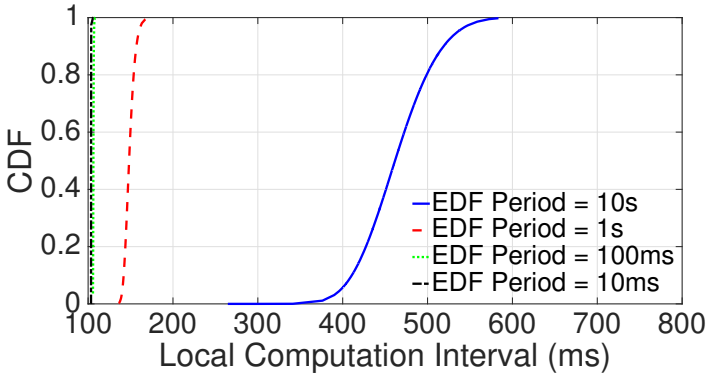
Figure 5.5: Cumulative density functions for different values of BSP intervals under PreDatA interference source using best-effort and EDF scheduling policies for a 512 nodes count. EDF period is 10 ms and utilization 3% for all the experiments.



(a) EDF $u_i = 3\%$, different BSP Intervals

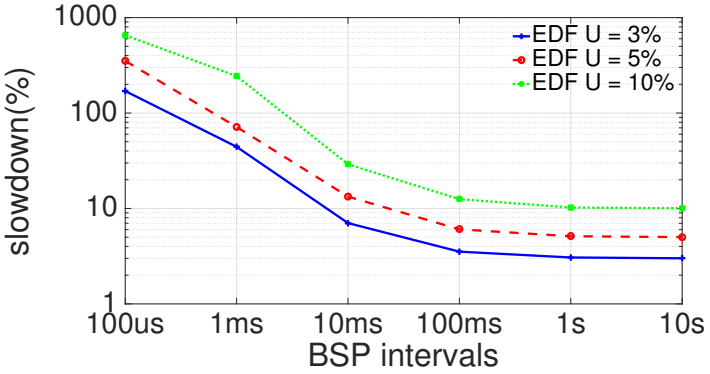


(b) EDF $u_i = 3\%$, different process counts (BSP Interval = 100 ms)

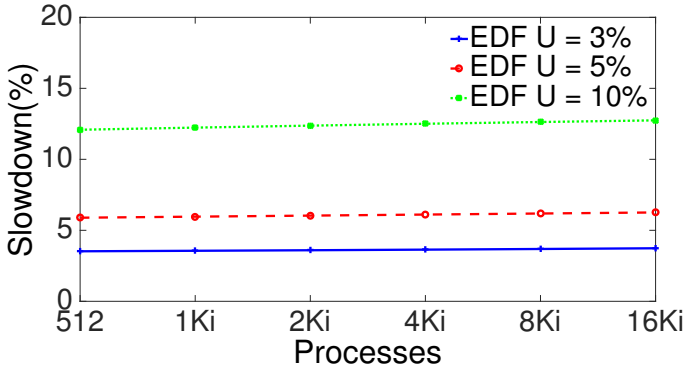


(c) CDFs for the BPS interval = 100 ms case

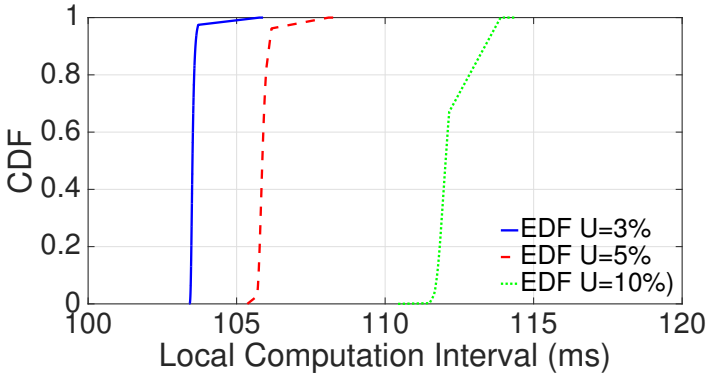
Figure 5.6: Slowdowns of the synthetic benchmark time-sharing CPU cores with EDF-scheduled workloads with an utilization factor of 3% and various values of EDF period. Figure 5.6c shows the cumulative density functions for the BSP interval = 100 ms case.



(a) EDF Period = 10 ms, different BSP Intervals

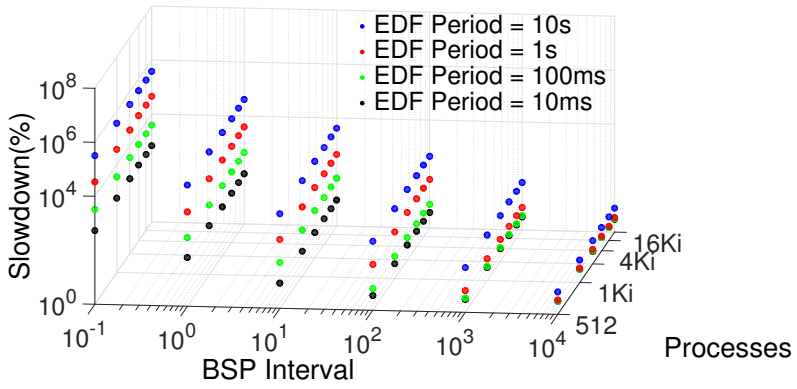


(b) EDF Period = 10 ms, different process counts (BSP Interval = 100 ms)

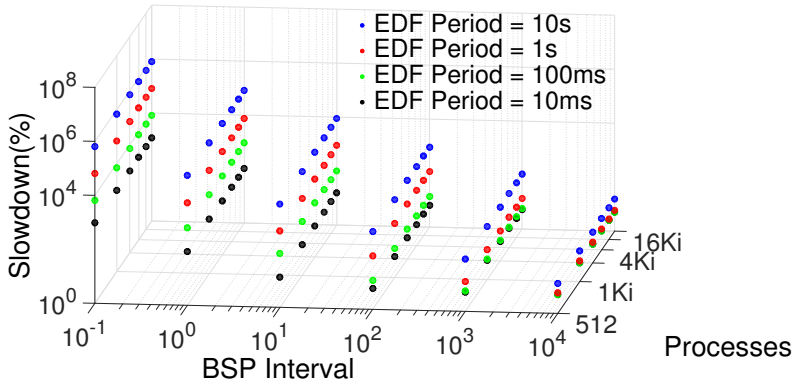


(c) CDFs for the BPS interval = 100 ms case

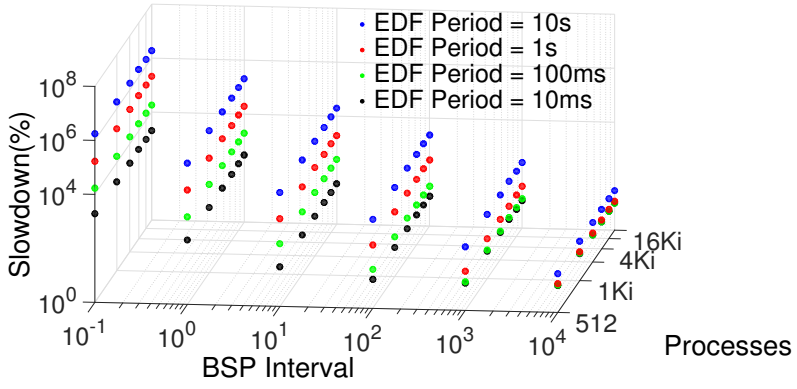
Figure 5.7: Slowdowns of the synthetic benchmark time-sharing CPU cores with EDF-scheduled workloads with EDF Period = 10 ms and different utilization factors. Figure 5.7c shows the cumulative density functions for the BSP interval = 100 ms case.



(a) EDF $u_i = 3\%$



(b) EDF $u_i = 5\%$



(c) EDF $u_i = 10\%$

Figure 5.8: EDF parameters tradeoffs for different utilization factors

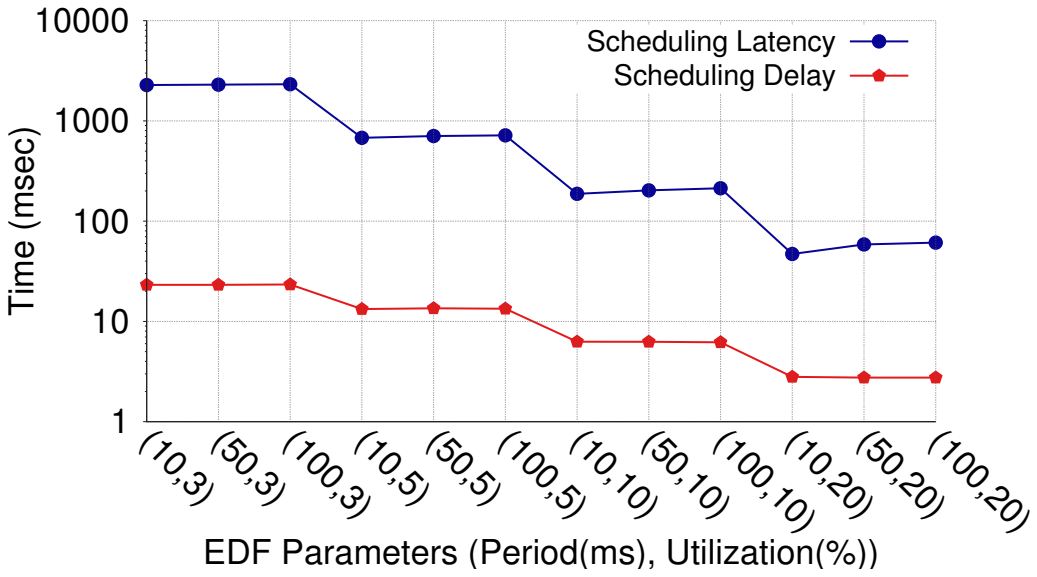


Figure 5.9: Scheduling delay and scheduling latency for different EDF Parameters

5.4 Summary

In this chapter, I evaluated EDF-based mitigation strategies and used the model described in Chapter 4 to better understand their impact on application performance. These results demonstrate that, for applications with frequent collective communication operations, fine-grained EDF scheduling of analytics and application activities can be used to mitigate overheads.

The GEV model explains that mitigation effect; EDF scheduling actually mitigates performance interference differently from gang scheduling — instead of reducing the effective number of processes computing in a BSP interval, it instead shapes the distribution of the interference. In other words, EDF scheduling reduces the impact of interference by changing its tail behavior, thereby changing how this interference scales.

Chapter 6

Conclusion & Future Work

6.1 Summary

In this dissertation, I investigated the impact of new sources of interference in next-generation large-scale systems, studied traditional mitigation techniques, investigated alternative mitigation approaches based on fine-grained OS scheduling, and examined the synchronization requirements of those techniques. To that end I used both empirical and analytical methods.

Chapter 2 described past work on application composition, and on characterization, modeling and mitigation of performance interference. None of these works provide a comprehensive approach to study the impacts of emerging sources of interference on HPC applications nor the effects of mitigation strategies. The more closely related works to this dissertation are [78, 81], which leverage extreme value theory concepts to study applications' performance and the impacts of jitter but unlike this dissertation, those works rely on assumptions about the distribution of individual BSP intervals, and were not used to predict the performance impact of interference on complete applications.

Chapter 3 quantified the costs and benefits of different approaches to scheduling applications and analytics on nodes in large-scale systems, including space sharing, uncoordinated time sharing, and gang-scheduled time sharing. The results demonstrated that coarse gang scheduling of analytics can reduce the overheads of time sharing applications and analytics to as low or lower than space sharing. Additionally, this chapter examined the synchronization mechanisms used in HPC systems and the synchronization requirements of co-located codes in order to mitigate performance impacts. The results of this analysis showed that alternative mitigation strategies with reduced synchronization requirements are needed in next-generation systems.

In Chapter 4, I presented a new model for analyzing the performance effects of time-sharing on bulk-synchronous HPC applications based on the use of *Extreme Value Theory*. After validating this model against both synthetic and real applications, I used both simulation and modeling techniques to profile next-generation interference sources and characterize their behavior and performance impact on a selection of HPC benchmarks, mini-applications, and applications.

Finally, Chapter 5 showed how the Extreme Value Theory-based model can be used to understand *how* an alternative interference mitigation technique based on periodic real time scheduling works. By using the model, I examined how EDF-scheduling workloads time sharing CPUs with applications affect the distribution of BSP intervals and the application's performance. I also demonstrated how my proposed model can be used to guide the design of mitigation alternatives based on EDF scheduling, providing the needed tools to study the tradeoffs of selecting EDF parameters.

6.2 Future Work

In this dissertation I proposed a modeling and simulation based framework useful to study a wide range of interference sources and mitigation approaches. A number of additional sources of interference can be investigated using this approach. The following are potentially interesting directions of future work related to those studies that can leverage the work that I presented in this dissertation.

6.2.1 Characterizing Dynamic Hardware Impacts

The model that I propose in this dissertation is flexible enough to study interference sources related to power management and other dynamic hardware activities such as dynamic clock speeds and varying system temperature. These activities can be modeled as interference events therefore our model should accurately predict their impact in a similar fashion to the sources investigated in this work.

6.2.2 Investigating Emerging Programming Models

Emerging programming models seek to reduce global synchronization, for example by allowing applications to overlap computation with collective operations whenever possible or by using local collectives. This increases the length of the intervals between global synchronization, but does not eliminate it entirely however. It also reduces the independence of distributed calculations. I expect my approach will remain feasible but addressing independence assumptions of the length of per-process intervals will be important for such applications.

6.2.3 Studying Interference Sources with Large Block Size Requirements

Finally, as mentioned in Chapter 4, one of the model's limitations is to accurately predict the impact of interference sources with low-frequency noise events on applications with very small BSP intervals (e.g., less than 1 ms). For those cases, a hybrid approach that combines the noise characterization method that I described in Section 4.4 and the direct GEV model that I presented in Section 4.2 may allow to effectively investigate those impacts. That hybrid model would use the noise characterization technique to examine the impact of short BSP intervals, which require large block sizes, and the direct GEV modeling for the rest of BSP intervals. Figure 4.12 shows some preliminary results of using that approach. Further research is needed to understand the tradeoffs of selecting BSP interval lengths' thresholds to switching between these methods.

6.2.4 Investigating Communicating Analytics workloads

A number of new analytics codes are becoming to use a non-trivial amount of collective operations to communicate. The effects of that communication on applications' performance have not been studied for previous works and is a potentially interesting opportunity for future work.

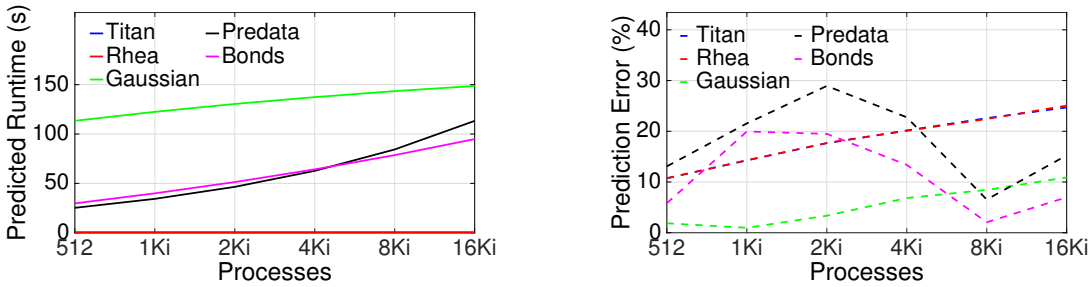
Appendices

Appendix A

Predicting Interference Performance Impact

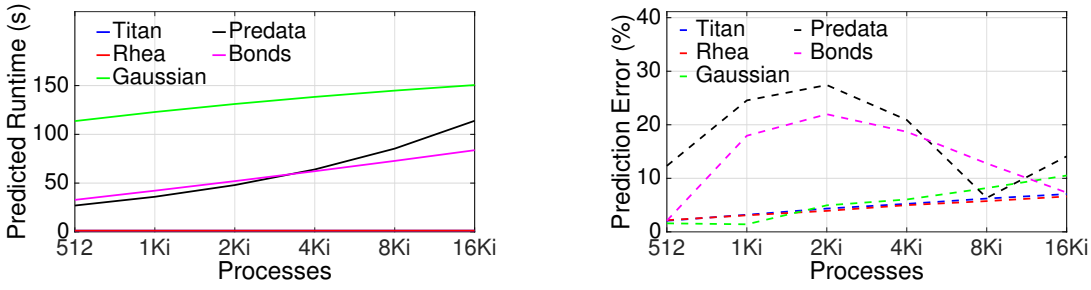
The following sections reproduce the performance interference impact figures presented in Chapters 4 and 5, with the GEV model prediction results and prediction errors in separated figures in order to show the prediction errors with more detail. Figures in Section A.1 show the performance impact of different interference sources on the synthetic BSP benchmark for different local computation times. Figures in Section A.2 show the performance impact of different interference sources on applications. Finally, figures in Section A.3 show the performance impact of PreDatA *in situ* analytics code scheduled using both best effort and EDF scheduling policies on different applications.

A.1 Interference Performance Impact Prediction on the Synthetic BSP Benchmark



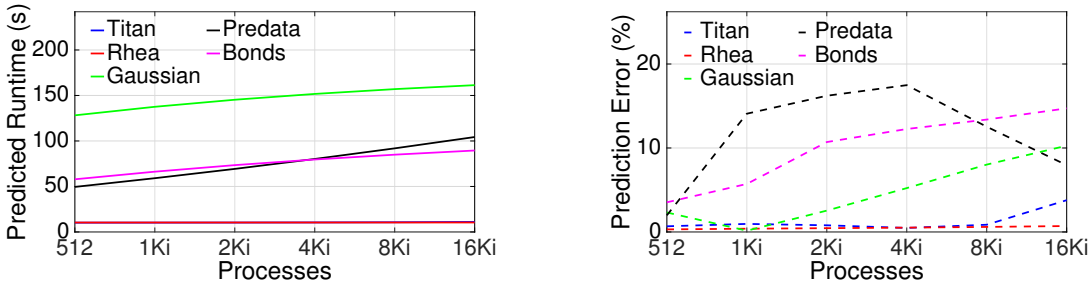
(a) Runtime Prediction (b) Prediction Error

Figure A.1: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 100 μs.



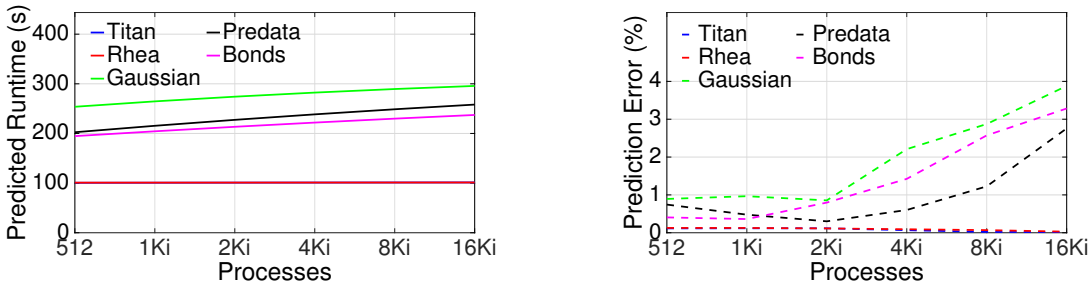
(a) Runtime Prediction (b) Prediction Error

Figure A.2: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 1 ms.



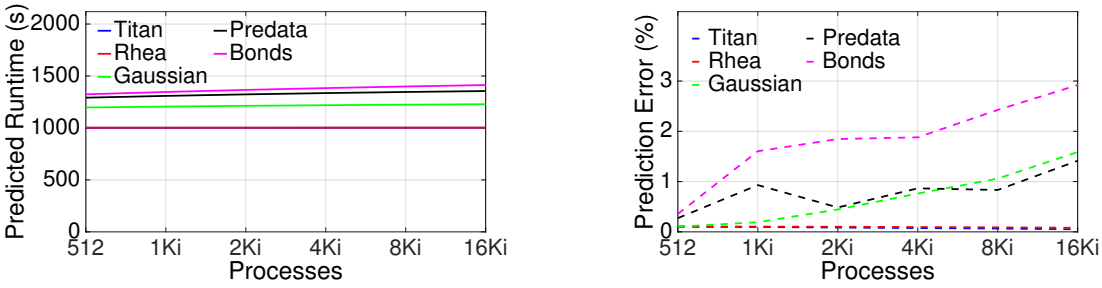
(a) Runtime Prediction (b) Prediction Error

Figure A.3: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 10 ms.



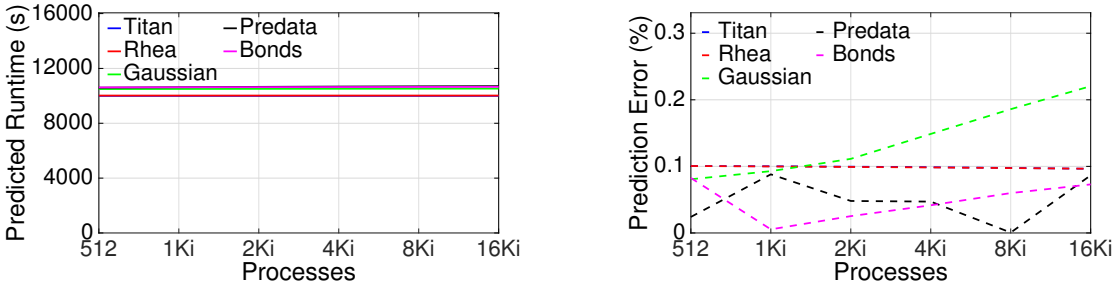
(a) Runtime Prediction (b) Prediction Error

Figure A.4: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 100 ms.



(a) Runtime Prediction (b) Prediction Error

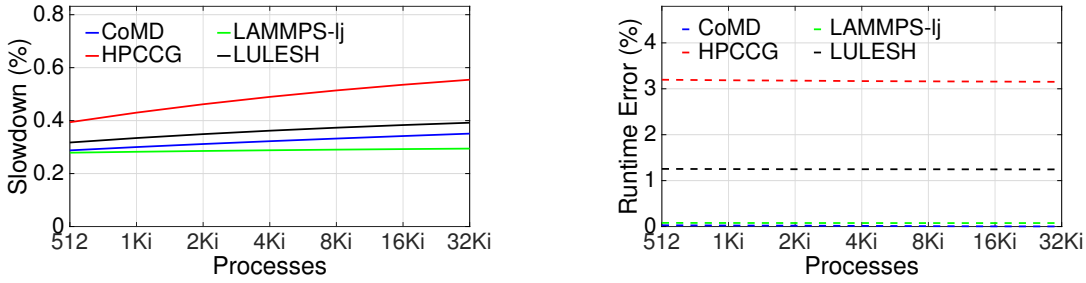
Figure A.5: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 1 s.



(a) Runtime Prediction (b) Prediction Error

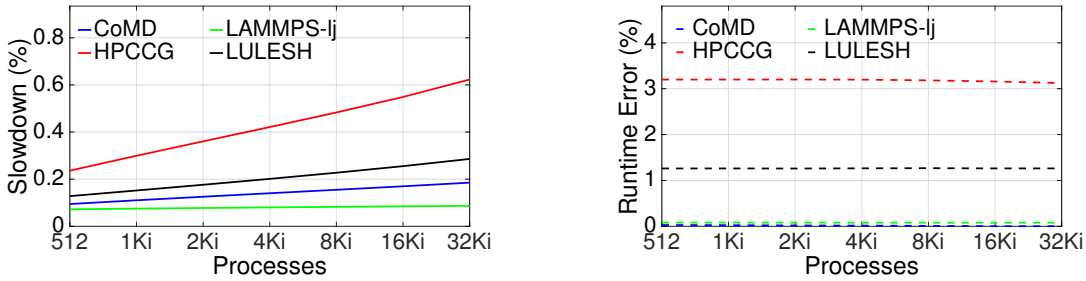
Figure A.6: GEV model prediction of the performance impact of interference sources on the BSP synthetic benchmark with local computation of 10 s.

A.2 Interference Performance Impact Prediction on Applications



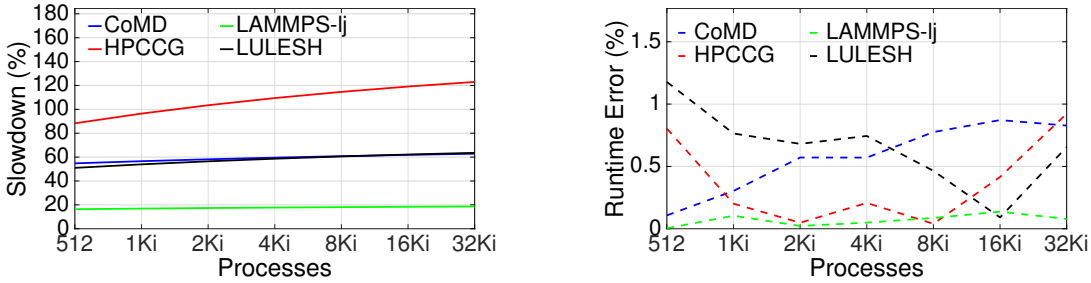
(a) Runtime Prediction (b) Prediction Error

Figure A.7: Rhea OS noise performance impact on applications predictions using the GEV model.



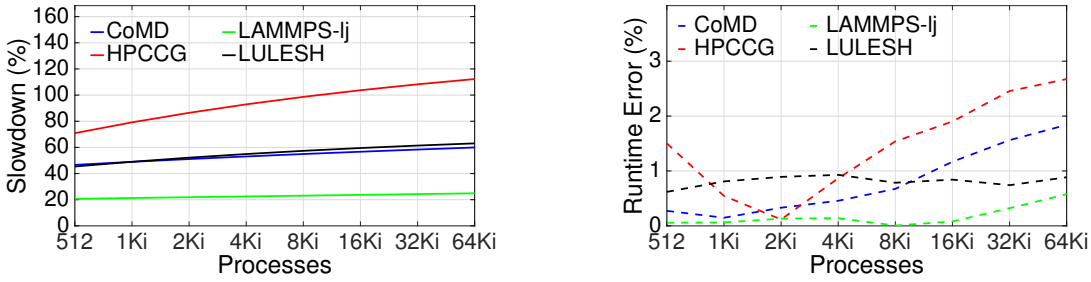
(a) Runtime Prediction (b) Prediction Error

Figure A.8: Titan OS noise performance impact on applications predictions using the GEV model.



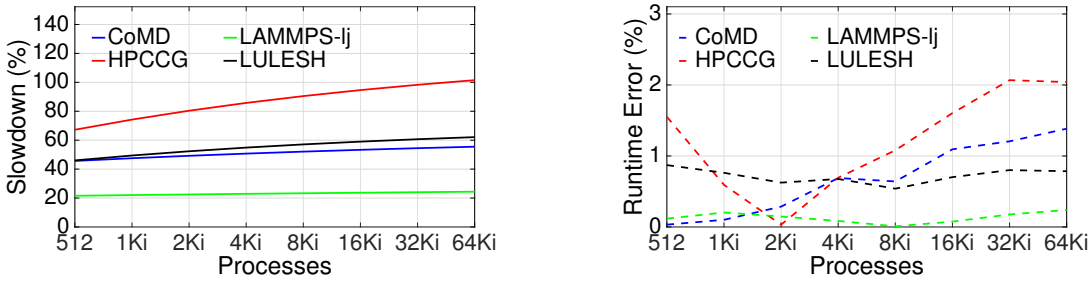
(a) Runtime Prediction (b) Prediction Error

Figure A.9: Synthetic gaussian noise performance impact on applications predictions using the GEV model.



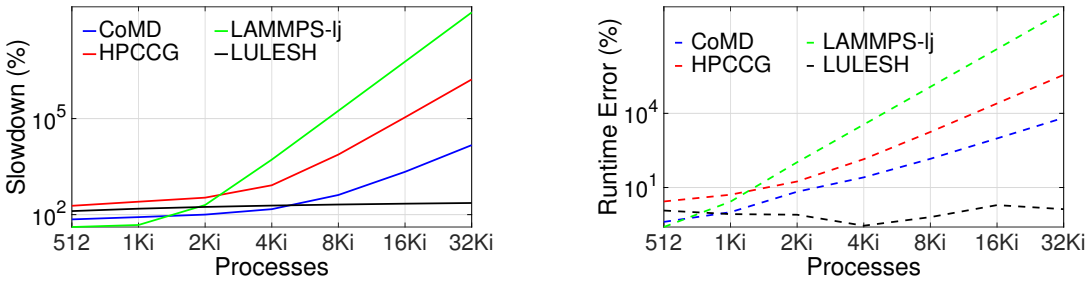
(a) Runtime Prediction (b) Prediction Error

Figure A.10: PreDatA performance impact on applications predictions using the GEV model.



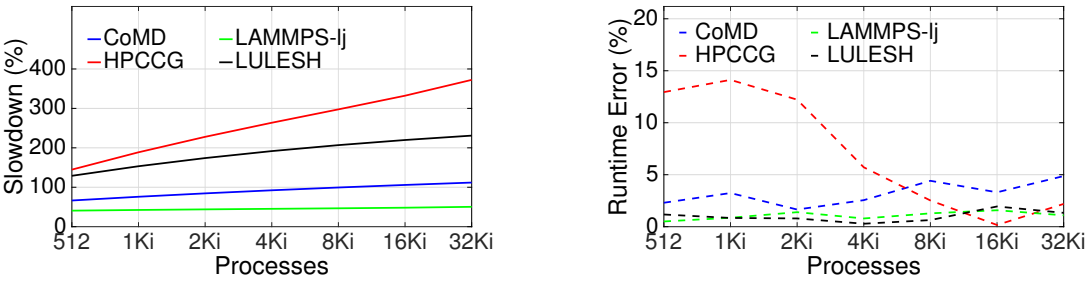
(a) Runtime Prediction (b) Prediction Error

Figure A.11: Bonds performance impact on applications predictions using the GEV model.



(a) Runtime Prediction (b) Prediction Error

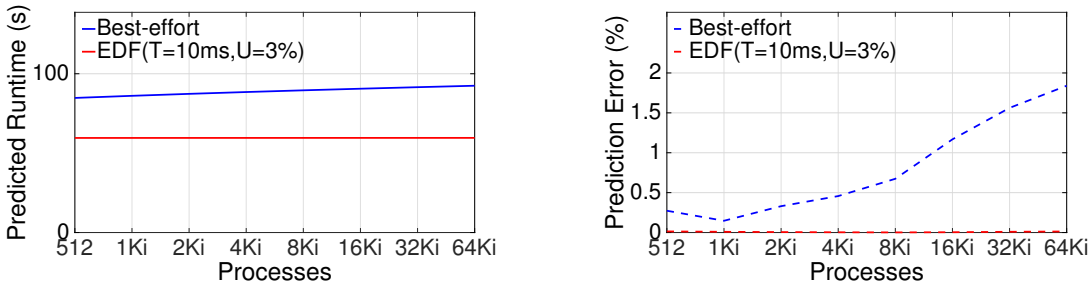
Figure A.12: Asynchronous checkpointing performance impact on applications predictions using the direct GEV model.



(a) Runtime Prediction (b) Prediction Error

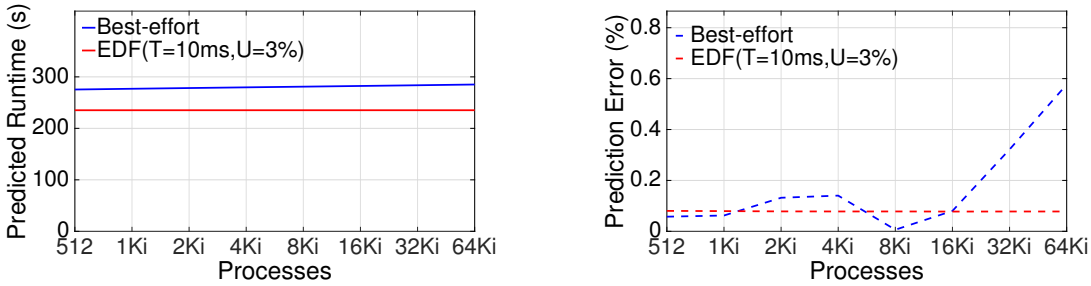
Figure A.13: Asynchronous checkpointing performance impact on applications predictions using the hybrid GEV model.

A.3 EDF-Scheduled Workloads Performance Impact Prediction



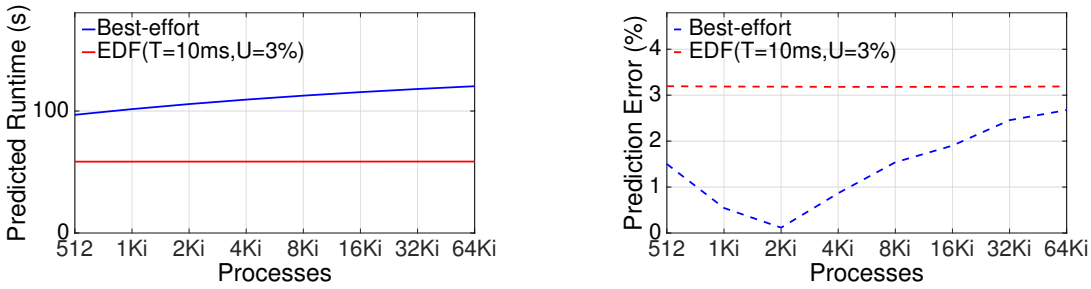
(a) Runtime Prediction (b) Prediction Error

Figure A.14: PreDatA performance impact on CoMD prediction using the GEV model. Best effort and EDF scheduling are compared.



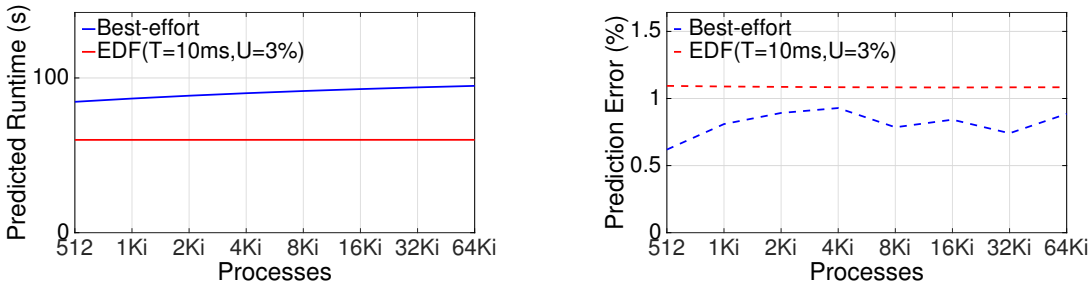
(a) Runtime Prediction (b) Prediction Error

Figure A.15: PreDatA performance impact on LAMMPS-lj prediction using the GEV model. Best effort and EDF scheduling are compared.



(a) Runtime Prediction (b) Prediction Error

Figure A.16: PreDatA performance impact on HPCCG prediction using the GEV model. Best effort and EDF scheduling are compared.



(a) Runtime Prediction (b) Prediction Error

Figure A.17: PreDatA performance impact on LULESH prediction using the GEV model. Best effort and EDF scheduling are compared.

References

[1] OLCF. The Oak Ridge Leadership Computing Facility. `https://www.olcf.ornl.gov`, Accessed: 06-2016.

[2] Jemal H. Abawajy and Sivarama P. Dandamudi. Scheduling parallel jobs with CPU and I/O resource requirements in cluster computing systems. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 336–343. IEEE, 2003.

[3] Hakan Akkan, Latchesar Ionkov, and Michael Lang. Transparently consistent asynchronous shared memory. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, page 6. ACM, 2013.

[4] P. Beckman, R. Brightwell, B.R. de Supinski, M. Gokhale, S. Hofmeyr, S. Krishnamoorthy, M. Lang, B. Maccabe, J. Shalf, and M. Snir. Exascale operating systems and runtime software report. *US Department of Energy, Technical Report, December*, 2012.

[5] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. The influence of operating systems on the performance of collective operations at extreme scale. In *IEEE Conference on Cluster Computing*, September 2006.

[6] Dimitris Bertsimas, Karthik Natarajan, and Chung-Piaw Teo. Tight bounds on expected order statistics. *Probability in the Engineering and Informational Sciences*, 20(04):667–686, 2006.

[7] Gunnar Blom. Statistical estimates and transformed beta-variables. 1958.

[8] Swen Böhm and Christian Engelmann. xSim: The extreme-scale simulator. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 280–286. IEEE, 2011.

[9] Ron Brightwell, Ron Oldfield, Arthur B. Maccabe, and David E. Bernholdt. Hobbes: Composition and virtualization as the foundations of an extreme-scale OS/R. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, page 2. ACM, 2013.

[10] Yuzhi Cai and Dominic Hames. Minimum sample size determination for generalized extreme value distribution. *Communications in Statistics Simulation and Computation®*, 40(1):87–98, 2010.

[11] Luis Chacón. A non-staggered, conservative, finite-volume scheme for 3d implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications*, 163(3):143–171, 2004.

[12] Zhibo Chang, Jian Li, Ruhui Ma, Zhiqiang Huang, and Haibing Guan. Adjustable credit scheduling for high performance network virtualization. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 337–345. IEEE, 2012.

[13] V. Chavez-Demoulin. *Two problems in environmental statistics: Capture-recapture analysis and smooth extremal models*. PhD thesis, Ph. D. thesis. Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, 1999.

[14] Stuart Coles, Joanna Bawa, Lesley Trenner, and Pat Dorazio. *An introduction to statistical modeling of extreme values*, volume 208. Springer, 2001.

[15] Harald Cramér. *Mathematical methods of statistics*, volume 9. Princeton university press, 1945.

[16] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.

[17] A.C. Davison and N.I. Ramesh. Local likelihood smoothing of sample extremes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(1):191–208, 2000.

[18] Jai Dayal, Dick Bratcher, Greg Eisenhauer, Karsten Schwan, Michael Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 246–255. IEEE, 2014.

[19] Pradipta De, Ravi Kothari, and Vijay Mann. Identifying sources of operating system jitter through fine-grained kernel instrumentation. In *Cluster Computing, 2007 IEEE International Conference on*, pages 331–340. IEEE, 2007.

[20] Pradipta De, Ravi Kothari, and Vijay Mann. A trace-driven emulation framework to predict scalability of large clusters in presence of OS jitter. In *Cluster Computing, 2008 IEEE International Conference on*, pages 232–241. IEEE, 2008.

[21] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[22] Ciprian Docan, Manish Parashar, and Scott Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[23] Christian Engelmann. Investigating operating system noise in extreme-scale high-performance computing systems using simulation. In *Proceedings of the 11th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2013*, pages 11–13, 2013.

[24] Noah Evans, Kevin Pedretti, Brian Kocoloski, John Lange, Michael Lang, and Patrick G. Bridges. A cross-enclave composition mechanism for exascale system software. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, page 3. ACM, 2016.

[25] Mir Nabi Pirouzi Fard and Björn Holmquist. First moment approximations for order statistics from the extreme value distribution. *Statistical Methodology*, 4(2):196–203, 2007.

[26] Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306–318, 1992.

[27] Kurt Ferreira, Jon Stearley, James H. Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G. Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2011.

[28] Kurt B. Ferreira, Patrick G. Bridges, Ron Brightwell, and Kevin Pedretti. Impact of system design parameters on application noise sensitivity. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing (Cluster 2010)*, September 2010.

[29] Kurt B. Ferreira, Ron Brightwell, and Patrick G. Bridges. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08)*, November 2008.

[30] Kurt B. Ferreira, Patrick Widener, Scott Levy, Dorian Arnold, and Torsten Hoefler. Understanding the effects of communication and coordination on checkpointing at scale. In *Proceedings of the 2014 International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing)*, 2014.

[31] James J Filliben. The probability plot correlation coefficient test for normality. *Technometrics*, 17(1):111–117, 1975.

[32] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.0, 09 2012.

[33] Carlo Gaetan and Matteo Grigoletto. Smoothing sample extremes with dynamic models. *Extremes*, 7(3):221–236, 2004.

[34] Brian Gough. *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[35] Irving I Gringorten. A plotting rule for extreme probability paper. *Journal of Geophysical Research*, 68(3):813–814, 1963.

[36] Peter Hall and Nader Tajvidi. Nonparametric analysis of temporal trend when fitting parametric models to extreme-value data. *Statistical Science*, pages 153–167, 2000.

[37] Felix Hernandez-Campos, J.S. Marron, Gennady Samorodnitsky, and F. Donelson Smith. Variable heavy tails in internet traffic. *Performance Evaluation*, 58(2):261–284, 2004.

[38] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep*, 2009.

[39] E.S. Jr. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylot, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th Intl. Symp. on Shock Waves*, pages 377–382, July 1993.

[40] Torsten Hoefler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm. Netgauge: A network performance measurement framework. In *HPCC*, volume 7, pages 659–671. Springer, 2007.

[41] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[42] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. LogGOPSim: simulating large-scale applications in the LogGOPS model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604. ACM, 2010.

[43] Arthur F Jenkinson. The frequency distribution of the annual maximum (or minimum) values of meteorological elements. *Quarterly Journal of the Royal Meteorological Society*, 81(348):158–171, 1955.

[44] Terry Jones, Shawn Dawson, Rob Neely, William Tuel, Larry Brenner, Jeffrey Fier, Robert Blackmore, Patrick Caffrey, Brian Maskell, Paul Tomlinson, et al. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *Supercomputing, 2003 ACM/IEEE Conference*, pages 10–10. IEEE, 2003.

[45] Terry R. Jones, , George Ostrouchov, Gregory A. Koenig, Oscar H. Mondragon, and Patrick G. Bridges. Time gremlins: The unsatisfying state of time agreement on extreme-scale computers (in preparation).

[46] Terry R Jones and Gregory A Koenig. Clock agreement among parallel supercomputer nodes. Technical report, ORNL-OLCF (Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN), 2014.

[47] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. Tradeoffs between power management and tail latency in warehouse-scale applications. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 31–40. IEEE, 2014.

[48] Larry Kaplan. Cray CNL. In *FastOS PI Meeting and Workshop*, 2007.

[49] Ian Karlin, Abhinav Bhatele, Bradford L. Chamberlain, Jonathan Cohen, Zachary Devito, Maya Gokhale, Riyaz Haque, Rich Hornung, Jeff Keasler, Dan Laney, Edward Luke, Scott Lloyd, Jim McGraw, Rob Neely, David Richards,

Martin Schulz, Charle H. Still, Felix Wang, and Daniel Wong. LULESH programming model and performance ports overview. Technical Report LLNL-TR-608824, December 2012.

[50] Shinpei Kato and Yutaka Ishikawa. Gang EDF scheduling of parallel task systems. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 459–468. IEEE, 2009.

[51] Scott Klasky, Stephane Ethier, Zhihong Lin, Kevin Martins, Douglas McCune, and Ravi Samtaney. Grid-based parallel data streaming implemented for the gyrokinetic toroidal code. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 24. ACM, 2003.

[52] Brian Kocoloski and John Lange. XEMEM: Efficient shared memory for composed applications on multi-OS/R Exascale systems. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 89–100. ACM, 2015.

[53] Brian Kocoloski, John Lange, Hasan Abbasi, David E Bernholdt, Terry R Jones, Jai Dayal, Noah Evans, Michael Lang, Jay Lofstead, Kevin Pedretti, et al. System-level support for composition of applications. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers*, page 7. ACM, 2015.

[54] Brian Kocoloski, Yuyu Zhou, Bruce Childers, and John Lange. Implications of memory interference for composed HPC applications. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 95–97. ACM, 2015.

[55] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, W Carson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[56] Kalyan Kumaran. Introduction to Mira. In *Code for Q Workshop*.

[57] James H. Laros III, Kevin Pedretti, Suzanne M. Kelly, Wei Shu, Kurt Ferreira, John Van Dyke, and Courtenay Vaughan. *Energy-Efficient High Performance Computing: Measurement and Tuning*. Springer Science & Business Media, 2012.

[58] Walter Lee, Matthew Frank, Victor Lee, Kenneth Mackenzie, and Larry Rudolph. Implications of I/O for gang scheduled workloads. In *Job Scheduling Strategies for Parallel Processing*, pages 215–237. Springer, 1997.

[59] Scott Levy, Kurt Ferreira B., Patrick Widener, Patrick G. Bridges, and Oscar H. Mondragon. How I Learned to Stop Worrying and Love Insitu Analytics. In *Proceedings of the Message Passing Interface (MPI) Users and Developers Conference (EuroMPI 2016)*, 2016.

[60] Scott Levy, Bryan Topp, Kurt B Ferreira, Dorian Arnold, Torsten Hoefler, and Patrick Widener. Using simulation to evaluate the performance of resilience strategies at scale. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation*, pages 91–114. Springer, 2014.

[61] Bin Lin and Peter A. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, page 8. IEEE Computer Society, 2005.

[62] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[63] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[64] Michael Mascagni and Ashok Srinivasan. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software (TOMS)*, 26(3):436–461, 2000.

[65] Oscar H. Mondragon, Patrick G. Bridges, and Terry Jones. Quantifying scheduling challenges for exascale system software. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers*, page 8. ACM, 2015.

[66] Aroon Nataraj, Alan Morris, Allen D. Malony, Matthew Sottile, and Pete Beckman. The ghost in the machine: Observing the effects of kernel operation on parallel application performance. In *Proceedings of SC'07*, 2007.

[67] Ron A. Oldfield, Sarala Arunagiri, Patricia J. Teller, Seetharami Seelam, Maria Ruiz Varela, Rolf Riesen, and Philip C. Roth. Modeling the impact of checkpoints on next-generation systems. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*, pages 30–46. IEEE, 2007.

[68] Leonid Oliker, Jonathan Carter, Michael Wehner, Andrew Canning, Stephane Ethier, Art Mirin, David Parks, Patrick Worley, Shigemune Kitawaki, and Yoshinori Tsuda. Leading computational methods on scalar and vector hec platforms. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 62. IEEE Computer Society, 2005.

[69] Francesco Pauli and Stuart Coles. Penalized likelihood inference in extreme value analyses. *Journal of Applied Statistics*, 28(5):547–560, 2001.

[70] Kevin Pedretti, Stephen L. Olivier, Kurt B. Ferreira, Galen Shipman, and Wei Shu. Early experiences with node-level power capping on the Cray XC40 platform. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, E2SC '15, pages 1:1–1:10, New York, NY, USA, 2015. ACM.

[71] T. Peterka, J. Kwan, A. Pope, H. Finkel, K. Heitmann, S. Habib, J. Wang, and G. Zagaris. Meshing the universe: integrating analysis in cosmological simulations. In *Proc. SC12 Ultrascale Visualization Workshop*, Salt Lake City, UT, November 2012.

[72] Fabrizio Petrini, Darren Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of SC'03*, Phoenix, AZ, 2003.

[73] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[74] Ori Rosen and Ayala Cohen. Extreme percentile regression. In *Statistical Theory and Computational Aspects of Smoothing*, pages 200–214. Springer, 1996.

[75] Steven Rostedt. Debugging the kernel using ftrace. `http://lwn.net/Articles/365835/`, 2009.

[76] David Ruppert. *Statistics and data analysis for financial engineering*. Springer, 2011.

[77] Sandia National Laboratory. Mantevo project home page. `http://mantevo.org`, January 2014.

[78] Seetharami Seelam, Liana Fong, Asser Tantawi, John Lewars, John Divirgilio, and Kevin Gildea. Extreme scale computing: Modeling the impact of system noise in multi-core clustered systems. *Journal of Parallel and Distributed Computing*, 73(7):898–910, 2013.

[79] Richard L. Smith. Maximum likelihood estimation in a class of nonregular cases. *Biometrika*, 72(1):67–90, 1985.

[80] Orathai Sukwong and Hyong S. Kim. Is co-scheduling too expensive for SMP VMs? In *Proceedings of the sixth conference on Computer systems*, pages 257–272. ACM, 2011.

[81] Junqing Sun and Gregory D. Peterson. An effective execution time approximation method for parallel computing. *Parallel and Distributed Systems, IEEE Transactions on*, 23(11):2024–2032, 2012.

[82] Dan Tsafrir, Yoav Etsion, Dror G Feitelson, and Scott Kirkpatrick. System noise, OS clock ticks, and fine-grained parallel applications. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 303–312. ACM, 2005.

[83] Masato Uchida. Traffic data analysis based on extreme value theory and its applications to predicting unknown serious deterioration. *IEICE transactions on information and systems*, 87(12):2654–2664, 2004.

[84] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[85] Patrick Widener, Kurt B. Ferreira, Scott Levy, and Torsten Hoefler. Exploring the effect of noise on the performance benefit of nonblocking allreduce. In *Proceedings of the 21st European MPI Users' Group Meeting*, page 77. ACM, 2014.

[86] Matthew Wolf, Zhongtang Cai, Weiyun Huang, and Karsten Schwan. Smartpointers: personalized scientific data portals in your hand. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 20–20. IEEE, 2002.

[87] Michael Woodacre, Derek Robb, Dean Roe, and Karl Feind. The SGI Altix TM 3000 global shared-memory architecture. technical whitepaper, silicon graphics, 2003.

[88] Hongfeng Yu, Chaoli Wang, Ray W. Grout, Jacqueline H. Chen, and Kwan-Liu Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, May/June 2010.

[89] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. Predata–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[90] Fang Zheng, Hongfeng Yu, Can Hantas, Matthew Wolf, Greg Eisenhauer, Karsten Schwan, Hasan Abbasi, and Scott Klasky. GoldRush: resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2013.