SAND2015-10198C

Scheduling In-Situ Analytics in Next-generation Applications

Oscar H. Mondragon, Patrick G. Bridges, and Scott Levy
Department of Computer Science
University of New Mexico
omondrag, bridges, slevy@cs.unm.edu

Kurt B. Ferreira and Patrick Widener
Center for Computing Research
Sandia National Laboratories
kbferre,pwidene@sandia.gov

***Abstract*—Next-generation applications increasingly rely on *in situ* analytics to guide computation, reduce the amount of I/O performed, and perform other important tasks. Scheduling where and when to run analytics is challenging, however. This paper quantifies the costs and benefits of different approaches to scheduling applications and analytics on nodes in large-scale applications, including space sharing, uncoordinated time sharing, gang scheduled time sharing, and EDF-scheduled time sharing. Our results demonstrate that coarse gang scheduling of analytics can reduce the overheads of time sharing applications and analytics to as low or lower than space sharing. In addition, fine-grain EDF scheduling of analytics and applications can make time sharing as fast or faster than space sharing for even the most interference-sensitive applications.**

***Keywords*-Exascale applications, composed applications, in-situ, resource sharing, performance interference.**

I. Introduction

Next-generation HPC applications increasing rely on *in situ* analytics to guide computation, reduce the amount of I/O performed, and perform other important tasks. Running analytics on the same node as a simulation reduces data movement between nodes, potentially saving both time and power in next-generation systems. A number of recent systems [1], [5], [6], [20] provide mechanisms to manage data movement between simulation and on-node analytics tasks.

Scheduling where and when to run analytics is challenging. Dedicating cores to analytics better isolates application and analytics performance, but at the cost of sacrificing cores that could otherwise be used for computation. In addition, dedicating cores to analytics is very coarse grained, making it difficult to reallocate resources when analytics needs less than the full set of cores allocated to it.

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 [33], but the generality of such approaches is unclear.

In this paper, we quantify the costs and benefits of different approaches to time-sharing processors between applications and analytics using a simulation-based approach described in Section III. We then characterize two different analytics codes and quantify the degree to which time-sharing these codes with application codes could perturb application analytics. These results, presented in Section IV, show that uncoordinated time sharing of cores between applications and analytics can have catastrophic performance consequences, but that gang scheduled time sharing of analytics can reduce these overheads to as low or lower than dedicating cores to analytics.

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 in Section V. Our results show that coarse synchronization of analytics activities across nodes, within approximately 10-100ms, is sufficient to eliminate most time-sharing overheads for many applications. Some applications, however, require gang scheduling to an accuracy of 1ms or less to eliminate these overheads.

In Section VI, we then examine the use of fine-grained OS scheduling techniques to mitigate application/analytics time-sharing overheads. This approach is motivated both by past results showing that high-frequency, low-duration interference has little impact on application performance [9], [10], as well as research using EDF-scheduling [23] to gang schedule virtual machines [22]. Our 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.

Overall, this paper makes the following contributions:

- A simulation-based approach to evaluating the performance impact of time-sharing cores between analytics and applications;
- An analysis of the performance impact of different approaches to scheduling applications and analytics;
- A study of how the degree of synchronization in the gang scheduling of analytics impacts application performance;

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

- An evaluation of the viability of using Earliest Deadline First (EDF) scheduling to eliminate application/analytics time-sharing overheads; and
- An analysis demonstrating how the differences in application inter-collective times closely correspond to differences in application performance.

II. Background

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. In this section, we provide background on two such production analytics workloads, system software to support these workloads, and mechanisms for scheduling analytics.

A. Example Analytics Codes

In this paper, we focus on two modern *in situ* analytics workloads, using them to quantify the potential impact of analytics on simulation 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.

Bonds Analysis in LAMMPS: Bonds is an analytics program that enhances the LAMMPS simulation code [26] with crack tracking capabilities. Specifically, Bonds directly reads atom bonding information from LAMMPS, and conducts a compute-intensive analysis that determine 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 [32], 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) [19], a computational-science application used for 3D particle-in-cell simulations of plasma micro-turbulence; and Pixie3D [3], a 3D MHD (Magneto Hydro-Dynamics) solver.

In this work, we focus on the PreDatA 1D and 2D histograms generation operations to process data from GTC-P, a proxy for GTC used for optimization and testing. These analyses perform significant local calculations and a number of small collective communication operations in order to compute global minimums, maximums, and moments. This paper focuses on the first-order costs of how GTC-P's analytics computation interferes with application performance; the second-order effects of how its communication potentially interferes with application performance are also interesting, but beyond the scope of this paper.

B. 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. For example, the Adaptable I/O System (ADIOS) [24] provides an API applications can use to efficiently transport data either to other applications or to the file system. Similarly, the Transparently Consistent Asynchronous Shared Memory (TCASM) [1] API provides low-overhead interfaces for asynchronous memory sharing between codes, while also providing consistent views of shared data using virtual memory techniques.

Services to cooperatively co-schedule analytics and simulation are also under development. The Goldrush system [33], 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 timesharing 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.

New HPC system software architectures, for example the Hobbes exascale operating system [2], seek to provide more systematic support for multi-component applications. The Hobbes XEMEM (Cross Enclave Memory) [20] abstraction, for example, provides a shared-memory system between enclaves (i.e., a subset of system resources with separated OS and runtimes), which is compatible with XPMEM [31]. Similarly, Hobbes is also examining if OS-level scheduling techniques can more broadly support time-sharing of processors between simulation and analytics for a broader set of applications [25]; this research question motivates the research described in this paper.

C. 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 can be uncoordinated time sharing or gang scheduled time sharing. In uncoor-

dinated 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 their cost in large-scale systems.

Local scheduling policies are generally the purview of the operating and application runtime systems. In this paper, we examine both best-effort and earliest-deadline-first (EDF) scheduling. In best-effort scheduling, typified by Linux's round-robin preemptive priority scheduler, the application and share the CPU roughly equally when both analytics and the application need to run. Earliest-deadline-first (EDF) scheduling, in contrast, provides fine-grain control when each task runs and the share of the processor given to each task. Each task's scheduling requirements in EDF scheduling are described by a scheduling *period*, T, and the length of each 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. The ratio of time slice and period is the task's *utilization factor*.

Each of these scheduling approaches impact the application workload differently. Best-effort uncoordinated scheduling results in large interruptions of the application at unpredictable times. Best-effort gang scheduling results in large interruptions of the application at coordinated points each time the analytics workload runs. Uncoordinated EDF scheduling reschedules analytics workloads to result in frequent, limited-duration interruptions of the application workload. Gang-scheduled EDF adds inter-node coordination to synchronize when these small interruptions occur across nodes.

III. 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 [25]. Specifically, time sharing cores between the applications and analytics can impede critical application communication activities. The impact of even modest perturbances of application communication are complex, particularly at large scale, as has been demonstrated by a number of recent OS noise studies [9], [14].

A. Modeling 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 work focuses on the first-order cost of analytics interference from the perspective of lost CPU cycles, this approach 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. Analysis of our two analytics workloads suggests that analytics communication is infrequent, and so we believe that this not a significant limitation. Second, this analytics-as-OS-noise approach ignores slowdowns due to increased memory and network pressure. Again, our analysis suggests that this is not a significant contributor to application slowdown.

B. Evaluation Infrastructure: `LogGOPSim` extreme-scale simulator

To understand the interactions and quantify the costs of time-sharing cores between analytics and simulation codes, we use simulation. Our simulator framework is based on `LogGOPSim` [15]. `LogGOPSim` uses the LogGOPS model, an extension of the well known LogP model [4], to simulate application traces that contain all exchanged messages and group operations. In this way, `LogGOPSim` reproduces all communication dependencies and the transitive closures of all delay chains of the application execution. Therefore, processes can be delayed by perturbations even if they do not directly communicate; the simulator accurately determines these delay chains. `LogGOPSim` can also extrapolate traces from small application runs with p processes to application runs with $k \cdot p$ processes. The extrapolation produces exact communication patterns for all collective communications and approximates point-to-point communications [15]. `LogGOPSim` and its trace extrapolation features have been validated [14], [15]. To simulate the impact of noise on application performance, `LogGOPSim` reads a *noise trace*: an ordered list of the relative start time and duration of CPU detours that represent noise events. Each time `LogGOPSim` simulates a communication or computation event, it deter-

mines whether the event overlaps a CPU detour. If so, the simulated completion of the event is delayed by the duration of the detour.

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 inline with simulation using the Linux `ftrace` utility [28]. 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. To simulate EDF scheduling, we process the noise trace with a utility that transforms it using an earliest-deadline schedule with a given period and slice.

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. Note that this assumes that analytics can be delayed significantly without perturbing application performance. Existing systems such as ADIOS [24] and TCASM [1] use actual or virtual copy techniques to do so.

In addition, `LogGOPSim` allows us to simulate the impact of uncoordinated scheduling and gang-scheduling on simulation performance. To do so, we vary the starting point in the noise trace for each simulated process. If we choose the starting point for each simulated process uniformly at random, `LogGOPSim` will simulate uncoordinated local scheduling of analytics. If we choose the starting point from a normal distribution with a given mean, `LogGOPSim` will simulate gang scheduling with different degrees of synchronization. The standard deviation of the distribution controls the degree of synchronization. For example, if we choose the starting point from a distribution with a standard deviation of zero `LogGOPSim` will simulate perfectly coordinated gang scheduling of the analytics workload. Increasing the standard deviation corresponds to more imperfect inter-node synchronization, possibly due to network and other delays.

Overall, this simulation approach captures the first-order sources of application/analytics performance interference. However, it fails to capture some potentially important second-order effects, including application perturbation of analytics communication and architecture-level performance interference (e.g., cache pollution). 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.

C. Application Workload Details

The following sections present results from simulation experiments based on the behavior of a common set of workloads. These workloads were chosen to be representative of scientific applications that are currently in use and computational kernels thought to be important for future extreme-scale computational science. They include:

- LAMMPS: A scientific application developed in Sandia National Laboratories to perform molecular dynamics simulations. We used the LAMMPS *2D crack* and *Lenard-Jones* potentials [27].
- CTH: A code developed at Sandia National Laboratories for modeling complex problems that are characterized by large deformations or strong shocks [7].
- HPCCG: A simple conjugate gradient solver from the Mantevo suite of mini-applications [12], [29].
- LULESH: An application that represents the behavior of a typical hydrocode [18].

CTH, HPCCG, and LAMMPS are important U.S. DOE applications which run for long periods of time in production modes and exhibit a range of different communication structures. LULESH is an exascale application proxy from the DOE ExMatEx co-design center [8]. We note that, excluding LAMMPS, these applications were designed for other purposes and the impact of analytics protocols was not necessarily considered in that design. Therefore, the results we present using these applications are not intended as feedback for their designers, but rather as indications that the scheduling of analytics must be considered during application design for future systems.

IV. Performance Interference Characterization

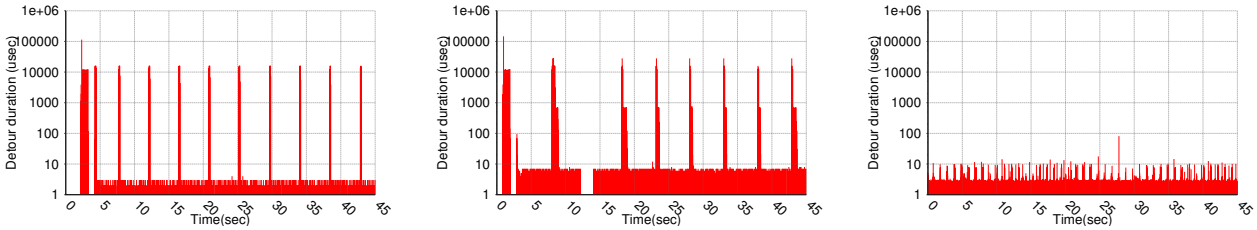
To better understand performance interactions between analytics and simulation, we first characterize the performance behavior of our representative analytics codes, Bonds and PreDatA. We then use the simulation approach described in Section III to obtain an initial evaluation of how co-locating these 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.

A. 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 III. 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 [13].

Figure 1 shows the resulting CPU detour traces. In addition to the *Chester* OS noise trace, we consider OS noise traces that we collected as part of our previous work [30] from *Volta*, *Muzia*, and *RedSky*, which are Cray XC30, Cray XE6 and SunBlade x6275 systems, respectively.

Table I shows the CPU overhead, the mean (μ_d) and the variance (σ_d) of the duration of the interference events, as



(a) Bonds scheduling traces (b) PreDatA scheduling traces (c) Chester OS noise trace

Figure 1. Scheduling traces for Bonds and PreDatA *in-situ* analytics codes and noise profile collected for *Chester* Cray XK7 TDS system.

well as the mean (μ_i) and the variance (σ_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 and variance 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 our previous work [9], high-duration, low-frequency application noise events generally have dramatically higher impacts on applications performance.

in-situ analytics codes					
	duration			inter-arrival time	
Trace	$\mu_d(\mu s)$	$\sigma_d(\mu s)$	$cpu(\%)$	$\mu_i(\mu s)$	$\sigma_i(\mu s)$
PreDatA	460.2	3232.4	2.44	18904.8	47184.2
Bonds	722.5	3991.1	2.80	25810.4	48384.5
Collected OS noise traces in HPC Systems					
	duration			inter-arrival time	
Trace	$\mu_d(\mu s)$	$\sigma_d(\mu s)$	$cpu(\%)$	$\mu_i(\mu s)$	$\sigma_i(\mu s)$
Volta	14.0	1.9	4.43	316.2	592.6
Chester	1.5	1.5	0.07	2282.6	1975.3
Redsky	2.7	0.2	0.63	435.5	383.0
Muzia	1.5	0.3	0.04	3979.5	217.0

Table I
MEAN CPU OVERHEAD, DURATION AND INTER-ARRIVAL TIMES FOR THE TWO *in situ* ANALYTICS WORKLOADS, ALONG WITH COMPARATIVE THE OS NOISE PROFILE STATISTICS.

B. 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 IV-A along with the `LogGOPSim` simulator, as discussed in Section III. For the space-sharing case, we allocate either one core out of 16 or one core out of 32 to analytics.

Figure 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 performance 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.* [14]).

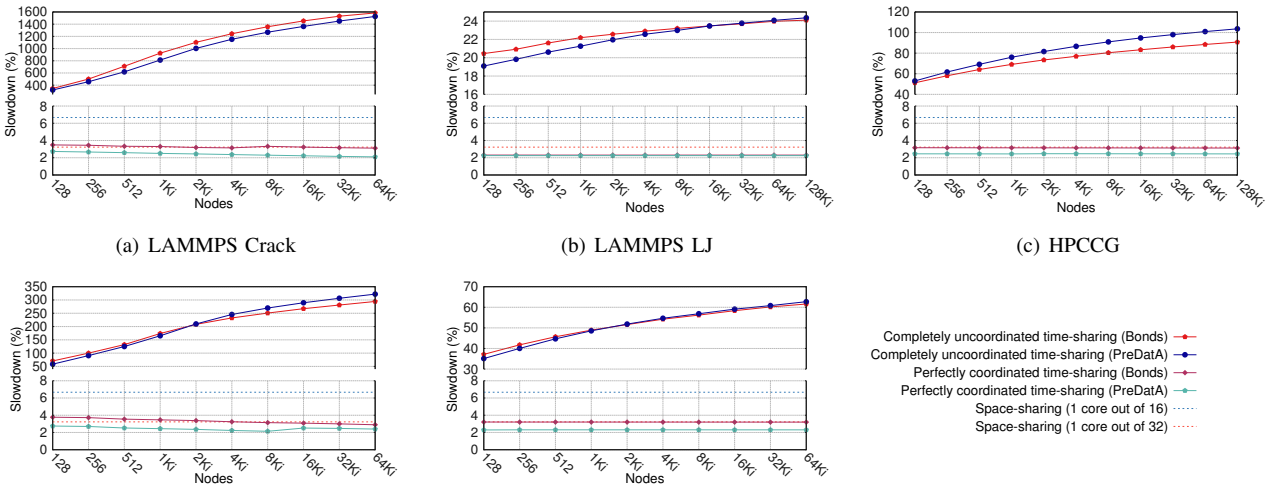
V. SYNCHRONIZATION REQUIREMENTS

As shown in the previous section, gang scheduling of analytics can reduce the overhead of time-scheduling 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 a few tens of milliseconds, depending on the mechanism used and system hardware and software characteristics [17].

In this section, we study *how closely synchronized* time-shared analytics tasks must be to mitigate the overhead of uncoordinated analytics. To accomplish this, we conduct a series of experiments in which we vary the degree of inter-node synchronization by using the techniques described in Section III.

Figure 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 the performance impact on each application running on 64 Ki nodes.[1] Results at other scales are very similar and are elided

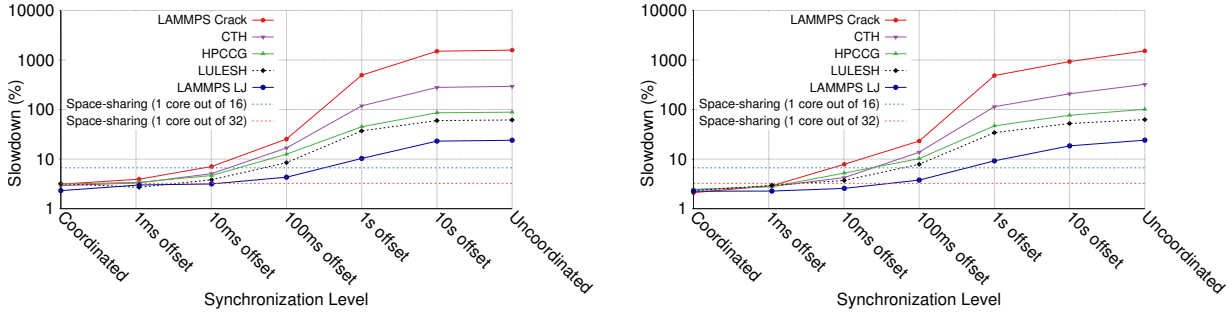
[1]Throughout this paper, we use the binary prefixes defined by the International Electrotechnical Commission (IEC). For example, 1 Ki nodes is equivalent to 1024 nodes.



(a) LAMMPS Crack (b) LAMMPS LJ (c) HPCCG

(d) CTH (e) LULESH

Figure 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.



(a) Applications composed with Bonds (b) Applications composed with PreDatA

Figure 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 nodes. Different levels of synchronization are simulated adding offsets to the time at which noise traces start on different analytics processes.

for space and 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 below 10ms in order to keep overhead below 10%. The CTH results show similarly hign 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 signficantly reduce the slowdown of the application. Finally, we observe that as the analytics workload is less and less synchronized across nodes, the performance impact approaches that of the completely unsynchronized case.

VI. EDF-based Mitigation of Analytics Interference

We next evaluate the viability of using EDF scheduling to mitigate time-sharing performance interference. For a given EDF period, we allocate sufficient time to analytics to guarantee it's average CPU utilization is at least as high as that measured in Section IV-A. Thus, we generally simulate allocatinga a 3% utilization factor to the analytics code.

We evaluate the viability of using EDF scheduling to mitigate analytics-based interference in three steps. First, we examine how well EDF scheduling mitigates analytics interference at a single period and utilization. We then examine the sensitivity of this technique to the choice of scheduling period. Finally, we examine how integrating the gang scheduling techniques studied in the previous section with EDF scheduling lowers time sharing overheads.

A. EDF vs. Best-effort scheduling

We first compare the impact of using fine-grained EDF scheduled time sharing of processors for both analytics and applications code. We simulate an EDF scheduler with a period of $T = 10ms$ and analytics utilization factor of $U = 3\%$. In addition, we simulate schedulers with both completely uncoordinated scheduling periods and periods that are perfectly synchronized across nodes.

Figure 4 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 cost of LAMMPS Crack on 64 Ki nodes is approximately 10%, compared with the almost 1600% observed in Section IV-B. 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 2. Synchronizing the scheduling periods of the EDF schedulers across nodes further reduces interference, particularly for LAMMPS Crack.

B. Trade-offs with selecting EDF scheduling parameters

We next evaluated the degree to which increasing the scheduling period and slice impacts application performance. Short periods increase scheduling overheads, so understanding the importance of short scheduling periods is potentially important to understand if using EDF scheduling is actually viable in real systems or requires short scheduling periods that may not be desirable.

For these experiments, we simulated LAMMPS Crack co-located with Bonds, the most sensitive of the workloads studied. We then varied the scheduler period between 10ms, 50ms, and 100ms while maintaining a constant 3% CPU utilization factor for analytics. As shown in Figure 5, changing the period has essentially no impact on application performance even though the length of the slice allocated to analytics does increase as a result. Three percent of a 100ms scheduling period is sufficiently short to avoid interfering with application performance, while still remaining easily achievable in modern commodity operating systems.

C. EDF Scheduler Synchronization Impact

Based on the results of Section VI-A, we finally examined how different levels of synchronization between different EDF schedulers impacted simulation performance. For these experiments, we used the same offsetting mechanism used in Section V. We again used a period of $T = 10ms$ and a utilization of $U = 3\%$, and used synchronization offsets with three different standard deviation values: 300 μs, 1500 μs, and 3000 μs.

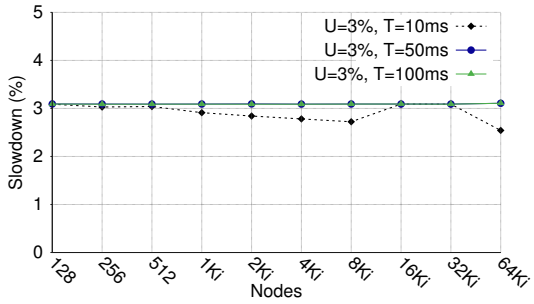


Figure 5. Slowdowns for LAMMPS Crack co-located with an EDF-scheduled workload with a utilization factor of 3% and three different periods.

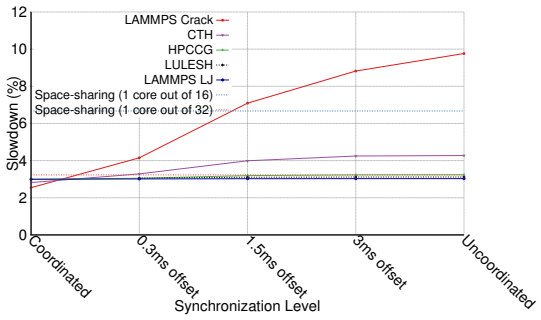


Figure 6. Slowdowns for applications co-located with an EDF-scheduled workload with a utilization factor of 3% and 10ms period for a 64ki nodes count. We use different levels of synchronization for the co-located workload by adding offsets to the time at which noise traces start on different processes.

Figure 6 shows 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.

VII. Analysis and Discussion of Results

A. 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 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.

B. 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

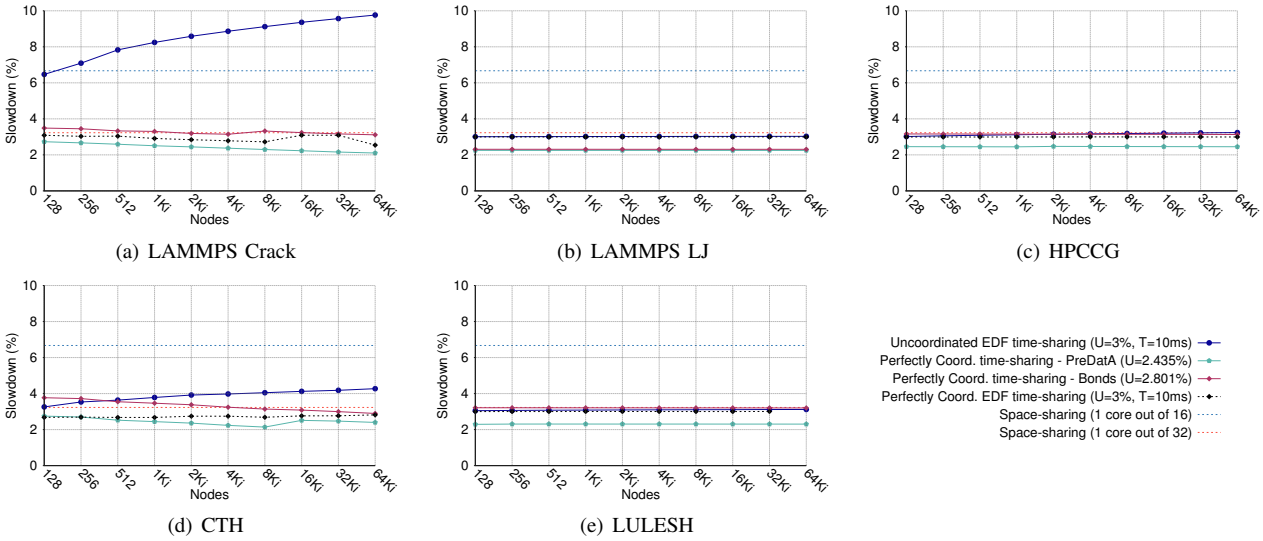


Figure 4. Comparison of the performance impacts on applications of their co-location with Bonds and PreDatA *in-situ* analytics codes and with an EDF-scheduled workload with similar CPU reservation requirements (utilization factor of 3%) and 10ms of period. The figure represents the perfectly coordinated scheduling policies for both approaches and the completely uncoordinated scheduling policy for the EDF-based approach. Additionally, we show the 1-out-of-16 and 1-out-of-32 *space-sharing* policies for reference.

significantly greater efforts to mitigate this interference, either via gang scheduling or EDF scheduling.

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 [10] 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 7 shows the mean, minimum, and maximum collective inter-arrival times for collective communications in the five applications studied in this paper. For comparison, we also reproduce the Bonds subfigure of Figure 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.

C. 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. Coarse gang scheduling, for example at the 10ms 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 with lower collective inter-arrival times, for example CTH and the LAMMPS Crack workload, other methods are needed. EDF or another similar fine-grained fair share scheduling algorithm is one such option, 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 space-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.

VIII. Related Work

Sources of performance interference for HPC applications have been widely studied. Most research has focused on the performance impact of OS noise on large-scale systems [9], [14], but power management [11] and resilience [10], [21], [30] have also been studied. LogGOPSim [15] has widely used in many of these studies.

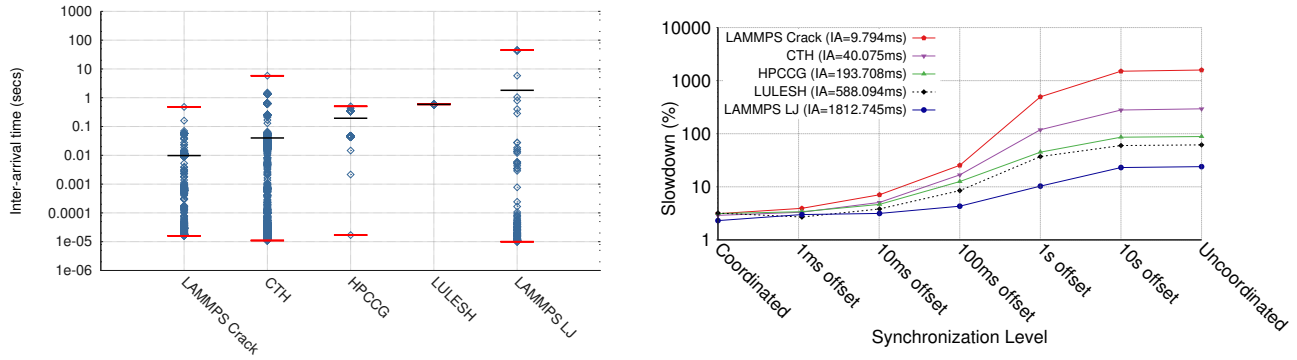


Figure 7. Measured mean, minimum, and maximum collective inter-arrival times for MPI collective operations for the five studied applications. For reference, the data from Figure 3 is reproduced again, with application names supplemented with measured mean collective inter-arrival (IA) time.

Many related techniques have been proposed for mitigating the impact of performance interference on HPC applications. In particular, VSched [22] included a user-level EDF scheduling policy to offer gang-scheduling and compute rate control to parallel applications, and Jones et al. [16] use global gang scheduling to minimize the impact of potentially interfering workloads.

Finally, recent work has shown that it is possible to time-share resources between related applications while minimizing performance interference [33]. This system provides user-level mechanisms to schedule co-located workloads, but requires the runtime to explicitly schedule various application components. In contrast, we propose a kernel-based approach to support a broader range of applications beyond the OpenMP applications Goldrush supports.

IX. Conclusions

Our results demonstrate that, with appropriate system support, scientific applications can time-share cores with analytics codes to minimize data movement and more accurately meet the processor needs of analytics. For many applications, coarse grained gang scheduling of analytics to within a few tens of milliseconds is sufficient to mitigate most overheads. For applications with frequent collective communication operations, fine-grained EDF scheduling of analytics and application activities can be used to further mitigate these overheads.

Acknowledgments

The authors would like to thank Jai Dayal and Terry Jones for their input on different approaches to running and scheduling *in situ* analytics and gang scheduling in HPC systems. This work is supported in part by the 2013 Exascale Operating and Runtime Systems Program from the DOE Office of Science, Advanced Scientific Computing Research, under award number DE-SC0005050, program manager Sonia Sachs, and by the Colciencias-Fulbright Colombia and The Universidad Autonoma de Occidente through the Caldas scholarships program.

References

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

[2] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. 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.

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

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.

[5] J. Dayal, D. Bratcher, G. Eisenhauer, K. Schwan, M. Wolf, X. Zhang, H. Abbasi, S. Klasky, and N. 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.

[6] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[7] J. E. S. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. PetneY, S. A. Silling, P. A. Taylor, 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.

[8] Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx). http://exmatex.lanl.gov/. Retrieved 16 Jan 2014.

[9] K. B. Ferreira, R. Brightwell, and P. 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.

[10] K. B. Ferreira, P. Widener, S. Levy, D. Arnold, and T. 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.

[11] V. W. Freeh, D. K. Lowenthal, R. Springer, F. Pan, and N. Kappiah. Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In *Proceedings of IPDPS*, Denver, CO, 2005.

[12] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep*, 2009.

[13] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm. Netgauge: A network performance measurement framework. In *HPCC*, volume 7, pages 659–671. Springer, 2007.

[14] T. Hoefler, T. Schneider, and A. 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.

[15] T. Hoefler, T. Schneider, and A. 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.

[16] T. Jones. Linux kernel co-scheduling for bulk synchronous parallel applications. In *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers*, pages 57–64. ACM, 2011.

[17] T. R. Jones and G. 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.

[18] I. Karlin, A. Bhatele, B. L. Chamberlain, J. Cohen, Z. Devito, M. Gokhale, R. Haque, R. Hornung, J. Keasler, D. Laney, E. Luke, S. Lloyd, J. McGraw, R. Neely, D. Richards, M. Schulz, C. H. Still, F. Wang, and D. Wong. Lulesh programming model and performance ports overview. Technical Report LLNL-TR-608824, December 2012.

[19] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. 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.

[20] B. Kocoloski and J. 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.

[21] S. Levy, B. Topp, K. B. Ferreira, D. Arnold, T. Hoefler, and P. 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.

[22] B. Lin and P. 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.

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

[24] J. Lofstead, F. Zheng, S. Klasky, and K. 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.

[25] O. H. Mondragon, P. G. Bridges, and T. 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.

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

[27] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics*, 117:1–19, 1995.

[28] S. Rostedt. Debugging the kernel using ftrace. http://lwn.net/Articles/365835/, 2009.

[29] Sandia National Laboratory. Mantevo project home page. http://mantevo.org, Jan. 2014.

[30] P. Widener, K. B. Ferreira, S. Levy, and T. 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.

[31] M. Woodacre, D. Robb, D. Roe, and K. Feind. The sgi altix tm 3000 global shared-memory architecture. technical whitepaper, silicon graphics, 2003.

[32] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Predata–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[33] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. 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.