hal-00983789, version 2 - 29 Apr 2014

Scheduling the I/O of HPC applications under congestion

Ana Gainaru, Guillaume Aupy, Anne Benoit, Franck Cappelo, Yves Robert, Marc Snir

RESEARCH REPORT
N° 8519
April 2014
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8519--FR+ENG

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

Scheduling the I/O of HPC applications under congestion

Ana Gainaru[*†], Guillaume Aupy[‡§†], Anne Benoit[¶‖], Franck Cappelo[**§], Yves Robert[‡§¶‖], Marc Snir[***]

Project-Team ROMA

Research Report n° 8519 — April 2014 — 25 pages

Abstract: A significant percentage of the computing capacity of large-scale platforms is wasted due to interferences incurred by multiple applications that access a shared parallel file system concurrently. One solution to handling I/O bursts in large-scale HPC systems is to absorb them at an intermediate storage layer consisting of burst buffers. However, our analysis of the Argonne's Mira system shows that burst buffers cannot prevent congestion at all times. As a consequence, I/O performance is dramatically degraded, showing in some cases a decrease in I/O throughput of 67%. In this paper, we analyze the effects of interference on application I/O bandwidth, and propose several scheduling techniques to mitigate congestion. We show through extensive experiments that our global I/O scheduler is able to reduce the effects of congestion, even on systems where burst buffers are used, and can increase the overall system throughput up to 56%. We also show that it outperforms current Mira I/O schedulers.

Key-words: IO, HPC, experiment, bandwidth, congestion, scheduling, online

[*] University of Illinois at Urbana Champaign, USA
[†] These authors contributed equally to this work
[‡] LIP, École Normale Supérieure de Lyon, France
[§] INRIA
[¶] Institut Universitaire de France
[‖] University of Tennessee Knoxville, USA
[**] Argonne National Laboratory, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

hal-00983789, version 2 - 29 Apr 2014

Ordonnancement d'I/O d'applications HPC sous contrainte de congestion

Résumé : Dans ce travail, nous proposons des algorithmes efficaces pour pallier aux problèmes de congestion lors des transferts des données de type I/O. Nous les évaluons sur des machines haute performance.

Mots-clés : IO,HPC,bandwidth,congestion

hal-00983789, version 2 - 29 Apr 2014

1 Introduction

With the advent of computationally demanding applications, parallel computers have continued to evolve towards post-petascale computing. At the same time, storage systems struggle to match the data generated by the computation of all running applications. According to Biswas et al. [1], when systems grow 10 times, memory bandwidth needs to grow by at least 20 times so that applications can run efficiently. The challenge is particularly obvious when many applications are executed concurrently. Indeed, while many I/O optimizations are available within each application (application-side collective I/O, software such as MPI-IO, and other network and disk-level optimizations [2, 3]), the interferences produced by multiple applications accessing a shared parallel file system in a concurrent manner frequently break these single-application optimizations.

The current server-side scheduling policies used by HPC systems at the file system level range between simple “first-come first-served” strategies for each storage server to more elaborated strategies. Recently, non-work-conserving disk schedulers, like anticipatory scheduling [4] and the CFQ scheduler [5], were designed to save the spatial locality with concurrent servicing of interleaved requests issued by multiple processes. This strategy keeps the disk head idle after serving a request of a process until either the next request from the same process arrives or the wait threshold expires. All policies, ranging from simplest to more advanced ones, deal with low-level requests, without any information from the applications; they cannot take advantage of particular properties or behaviors of each application. As a consequence, current I/O schedulers are not able to address the global efficiency of the system. As system size continues to increase, schedulers need to have a global view of the I/O requirements of all applications in order to make appropriate decisions.

In this paper, we focus on scheduling applications under I/O bandwidth constraints. Congestion due to I/O interference depends on many factors, namely each individual application size and computation-to-I/O ratio, but also when they start performing I/O with regard to one another. An analysis of the Intrepid system at Argonne shows that congestion can cause up to a 70% decrease in the I/O efficiency seen by an application (Figure 1). We propose a global high-level scheduler that is aware of application I/O past behaviors, and that dynamically coordinates I/O accesses to the parallel file system. Our contributions can be summarized as follows: (1) We design a global scheduler that minimizes congestion caused by I/O interference by considering application past behaviors and system characteristics when scheduling I/O requests. We show that this scheduler reduces I/O delays incurred by each application, and increases overall system throughput. (2) We build a simulator in order to test our scheduler in a large variety of scenarios, and to assess its performance and limitations. We simulate the Intrepid and Mira systems and show that our heuristics obtain better system throughput and application dilation compared to what happens when congestion occurs. Notably, we report that a simulation of our scheduler without burst buffers achieves a better system throughput than the one observed on Intrepid in congested moments. (3) We implement

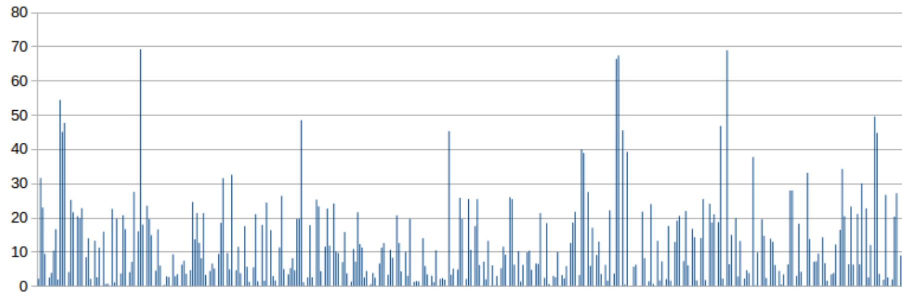


Figure 1: I/O throughput decrease (percentage per application, over 400 applications).

the global scheduler on Argonne's Vesta computer and test its results in the IOR benchmark. We validate our simulation model and show that, besides a small increase in the execution time of applications when congestion does not occur, the results are much better when using our implementation than current Vesta schedulers. (4) A striking result obtained on Vesta is the confirmation of the simulations: in most scenarios, our scheduler outperforms the use of burst buffers without having the incurred cost.

The rest of the paper is organized as follows. We introduce the application model and optimization problems in Section 2. We derive online scheduling heuristics in Section 3. Through a full set of simulations in Section 4, we thoroughly evaluate and compare these heuristics, before reporting actual execution times on Vesta in Section 5. We give some background and related work in Section 6. We provide concluding remarks and hints for future research directions in Section 7.

2 Framework

In this section, we provide a formal description of the application and platform model, and we state scheduling objectives. We target a parallel platform composed of N identical unit-speed processors, each equipped with an I/O card of bandwidth b (expressed in bytes per second). This corresponds to the I/O network from the compute nodes to I/O servers on a typical cluster. We further assume a centralized I/O system with a total bandwidth B (also expressed in bytes per second) from these I/O servers to the disks. Figure 2 shows the model projected over Argonne's Intrepid architecture.

2.1 Application and platform model

We assume that K applications are running concurrently, each of them being assigned to independent and dedicated computational resources, but competing for I/O. For simplicity, we assume the I/O and communication network are

hal-00983789, version 2 - 29 Apr 2014

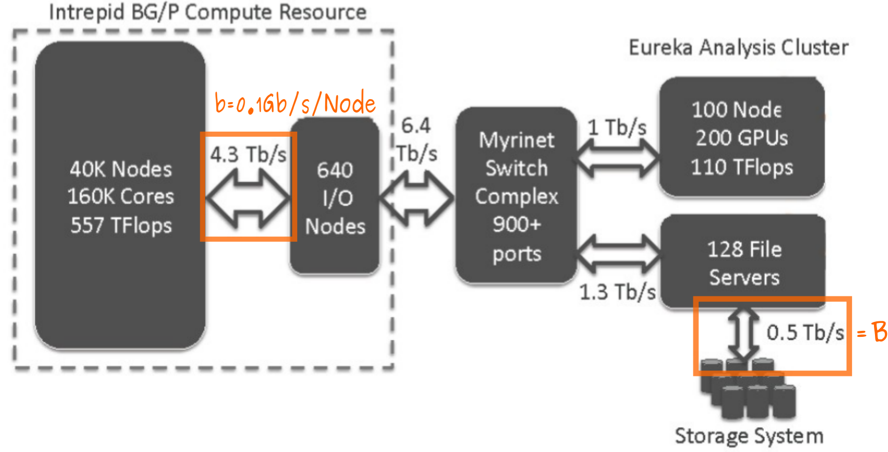


Figure 2: Model instantiation for the Intrepid platform.

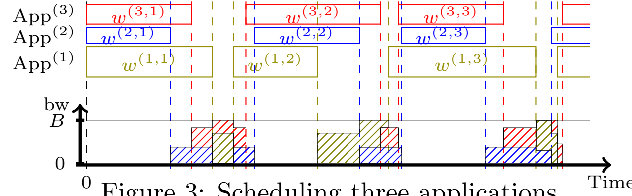


Figure 3: Scheduling three applications.

separated, so that network congestion caused by inter-node communications does not interfere with I/O transfers.

Each application $\text{App}^{(k)}$ is released on the platform at time r_k, executes on $\beta^{(k)}$ dedicated processors, and consists of $n_{\text{tot}}^{(k)}$ *instances* that repeat over time until the last instance is executed. An instance is composed of some chunk of computations followed by some I/O transfer. More precisely, the i-th instance $\mathcal{I}_i^{(k)}$ of $\text{App}^{(k)}$ consists of $w^{(k,i)}$ units of computation (at unit-speed), followed by the transfer of a volume $\text{vol}_{\text{io}}^{(k,i)}$ of bytes to or from the I/O system. Finally, let d_k be the time when the last instance of $\text{App}^{(k)}$ is completed.

Because computational resources are dedicated, we can always assume w.l.o.g. that the next computation chunk starts right after completion of the current I/O transfers, and is executed at full (unit) speed. On the contrary, all applications compete for I/O, and congestion will likely occur. The simplest case is that of an application $\text{App}^{(k)}$ using the I/O system in dedicated mode during a time-interval of duration D. Assume that $\text{App}^{(k)}$ needs to transfer $\text{vol}_{\text{io}}^{(k,i)}$. In that case, let γ be the I/O bandwidth used by each processor of $\text{App}^{(k)}$ during this time-interval. We derive the condition $\beta^{(k)}\gamma D = \text{vol}_{\text{io}}^{(k,i)}$ to express that the entire I/O data volume is transferred. We must also enforce the constraints that: (i) $\gamma \leq b$ (output capacity of each processor); and (ii) $\beta^{(k)}\gamma \leq B$ (total

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

capacity of I/O system). Therefore, the minimum time to perform the I/O transfers for the current instance of $\text{App}^{(k)}$ is

$$\text{time}_{\text{io}}^{(k,i)} = \frac{\text{vol}_{\text{io}}^{(k,i)}}{\min(\beta^{(k)}b, B)}.$$

However, in general, many applications will use the I/O system simultaneously, and the bandwidth capacity B will be shared among all these applications. The I/O of some applications may well take place during several non-consecutive time-intervals, and use different bandwidths. In Figure 3, we show an example of three applications competing for I/O bandwidth. On the top part of Figure 3, we can see the applications doing computations without any constraint. However at the end of their computations, all applications need to transfer some volume of I/O and share the I/O total bandwidth B (bottom part of the figure). When these three applications want to execute some I/O at the same time, congestion occurs and the scheduler needs to choose which bandwidth fraction to assign to each application. The model is very flexible, and only assumes that at any instant, all processors assigned to a given application are assigned the same bandwidth. This assumption is transparent for the I/O system and simplifies the problem statement without being restrictive. Again, in the end, the total volume of I/O transfers for each instance $\mathcal{I}_i^{(k)}$ of $\text{App}^{(k)}$ must be $\text{vol}_{\text{io}}^{(k,i)}$, and the rules of the game are simple: never exceed the individual bandwidth b of each processor, and never exceed the total bandwidth B of the I/O system. Formally, if instance $\mathcal{I}_i^{(k)}$ of application $\text{App}^{(k)}$ does its computation from t_1 to $t_2 = t_1 + w^{(k,i)}$, and the computation of the next instance starts in t_3, then the volume of I/O transferred for $\text{App}^{(k)}$ during the interval $[t_2, t_3]$ should be equal to $\text{vol}_{\text{io}}^{(k,i)}$.

The richness of the model comes from its flexibility for scheduling all the I/O transfers. It corresponds to a practical framework where the central scheduler would assign different I/O bandwidths per time-interval to each application. Depending on how many applications are trying to perform I/O, the scheduler might also decide to prevent some applications from accessing the disk during some time-intervals. This way, the scheduler controls the wait time for all applications and can make sure that they do not exceeding the time-out existing in the I/O system.

2.2 Objectives

We first define $\tilde{\rho}^{(k)}(t)$, the *application efficiency* achieved for each application $\text{App}^{(k)}$ at time t, as

$$\tilde{\rho}^{(k)}(t) = \frac{\sum_{i \le n^{(k)}(t)} w^{(k,i)}}{t - r_k},$$

where $n^{(k)}(t) \le n_{\text{tot}}^{(k)}$ is the number of instances of application $\text{App}^{(k)}$ that have been executed at time t, since the release of $\text{App}^{(k)}$ at time r_k. Because we exe-

cute $w^{(k,i)}$ units of computation followed by $\text{vol}_{\text{io}}^{(k,i)}$ units of I/O operations on instance $\mathcal{I}_i^{(k)}$ of $\text{App}^{(k)}$, we have $t - r_k \geq \sum_{i \leq n^{(k)}(t)} \left(w^{(k,i)} + \text{time}_{\text{io}}^{(k,i)} \right)$. Without congestion, the schedule would achieve $t - r_k = \sum_{i \leq n^{(k)}(t)} \left(w^{(k,i)} + \text{time}_{\text{io}}^{(k,i)} \right)$, and the optimal application efficiency, where all I/O resources are available in dedicated mode for $\text{App}^{(k)}$, is

$$\rho^{(k)}(t) = \frac{\sum_{i \leq n^{(k)}(t)} w^{(k,i)}}{\sum_{i \leq n^{(k)}(t)} \left(w^{(k,i)} + \text{time}_{\text{io}}^{(k,i)} \right)}.$$

Due to I/O congestion, $\tilde{\rho}^{(k)}(t)$ never exceeds $\rho^{(k)}(t)$. We are ready to present the two optimization objectives, together with a rationale for each of them.

• SysEfficiency: Here we aim to maximize the performance of the platform, i.e., the amount of CPU operations per time unit. This objective writes:

$$\text{maximize } \frac{1}{N} \sum_{k=1}^{K} \beta^{(k)} \tilde{\rho}^{(k)}(d_k).$$

Recall that $N = \sum_{k=1}^{K} \beta^{(k)}$ is the total number of processors, and that d_k is the time-step where $\text{App}^{(k)}$ terminates its execution. An upper limit of the system efficiency is $\frac{1}{N} \sum_{k=1}^{K} \beta^{(k)} \rho^{(k)}(d_k)$. The rationale is to squeeze the most flops out of the platform's aggregated computational power. This objective is CPU-oriented, as the schedule will give priority to compute-intensive applications with large $w^{(k,i)}$ and small $\text{vol}_{\text{io}}^{(k,i)}$ values.

• Dilation: We aim to minimize the largest slowdown imposed to each application. This objective writes:

$$\text{minimize } \max_{k=1..K} \frac{\rho^{(k)}(d_k)}{\tilde{\rho}^{(k)}(d_k)}.$$

The rationale is to provide fairness across applications, and it corresponds to the stretch in classical scheduling: each application incurs a slowdown factor due to I/O congestion, and we want the largest slowdown factor to be minimal. This objective is user-oriented, as it gives each application a guarantee on its relative progress rate.

3 Schedules

The scheduler monitors the stream of I/O calls and decides on the fly (as I/O calls appear in the system) which applications are allowed to perform I/O. We define an event as the start or the end of an I/O transfer by some application. At each event, the scheduler looks at the current state of the system, which is represented by the application efficiency and the amount of I/O already performed by each application. Then, based on a given strategy, it chooses a subset of applications and allows them to start or continue their I/O. This scheduler

hal-00983789, version 2 - 29 Apr 2014

does not require any knowledge of the applications running in the system. Applications pay a supplementary cost due to the need to call the scheduler each time they need to perform their I/O. We show in Section 5 that this overhead is well paid off by the benefits of minimizing congestion.

Depending on the strategy used by the online scheduler to select applications at each event, the results might benefit either objective described in Section 2.2. For each strategy, *favoring* application $\text{App}^{(k)}$ means that $\text{App}^{(k)}$ is executed as fast as possible, with bandwidth $\min\left(b\beta^{(k)}, \text{bw}_{\text{avail}}\right)$, where bw_{avail} is the available bandwidth at the moment the decision is taken. Here are the strategies that we experiment with.

• The RoundRobin scheduler favors available applications in a round-robin fashion similar to what the I/O scheduler is doing in HPC systems [6]. This heuristic is useful for comparison. The general idea of scheduling applications is "first-come first-served" (FCFS) with an additional constraint to ensure fairness. More precisely, each time an application needs to transfer some I/O, if there is no congestion, then this application is favored. Otherwise, the application that finished the I/O transfer of its last instance the longest time ago is favored.

• The MinDilation scheduler favors applications with low values of $\frac{\tilde{\rho}^{(k)}(t)}{\rho^{(k)}(t)}$.

• The MaxSysEff scheduler favors applications with low values of $\beta^{(k)}\tilde{\rho}^{(k)}(t)$.

• The MinMax scheduler favors applications with low values of $\beta^{(k)}\tilde{\rho}^{(k)}(t)$, unless there exists an application with a value $\frac{\tilde{\rho}^{(k)}(t)}{\rho^{(k)}(t)}$ below a certain threshold, γ, in which case it favors the application with the lower $\frac{\tilde{\rho}^{(k)}(t)}{\rho^{(k)}(t)}$. This threshold should be defined by the system administrator and depends on the Dilation targeted for the platform.

Note that since $0 \leq \frac{\tilde{\rho}^{(k)}(t)}{\rho^{(k)}(t)} \leq 1$, the MinMax heuristic is exactly MinDilation if $\gamma = 1$, and MaxSysEff if $\gamma = 0$. For all these heuristics, we have also implemented a Priority variant. In this version, the scheduler always chooses applications that already started performing their I/O before favoring any other application. The rationale behind this is that there may be an additional cost incurred by restarting the I/O of an application after an interruption, due to breaking disk locality. Breaking disk locality by alternating multiple applications accessing the device affects their performance and decreases the overall efficiency of the system [6]. Solid-state drives do not present the problem described above since they do not contain any moving mechanical components. This means that future clusters that use only SSD can use the original heuristics without paying the extra cost of not being able to choose the best possible applications that avoid congestion.

4 Simulations

In this section, we report extensive simulations to assess the performance of the heuristics presented in Section 3. In the first set of simulations, we thoroughly

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

study the impact of each heuristic on different scenarios and use multiple applications with similar properties to real applications that ran on the Intrepid system. In the second set, we compare the heuristics to the I/O scheduler of Intrepid and Mira, on traces of applications that run on these platforms when congestion occurs.

4.1 Applications

Intrepid is a BlueGene/P supercomputer used by the Argonne National Laboratory between 2008 and 2014 and was ranked number 3 on the June 2008 Top 500 list. Consisting of 48 racks, 786,432 processors, and 768 terabytes of memory, Mira is a 10-petaflops IBM BlueGene/Q system, 20 times faster than Intrepid and currently ranked number 5 on the November 2013 Top 500 list. A wide range of science and engineering applications have run on BlueGene systems, including those used by the computational science community for cutting-edge research in chemistry, combustion, astrophysics, genetics, materials science, and turbulence. The typical behavior of scientific simulations is defined by alternating computation phases and I/O phases. The I/O phases are in general used either for writing out intermediary results for visualization purposes and/or for checkpointing. Intrepid uses separate networks for communication and I/O, which makes it the perfect system to study the effects of congestion on application and system efficiency.

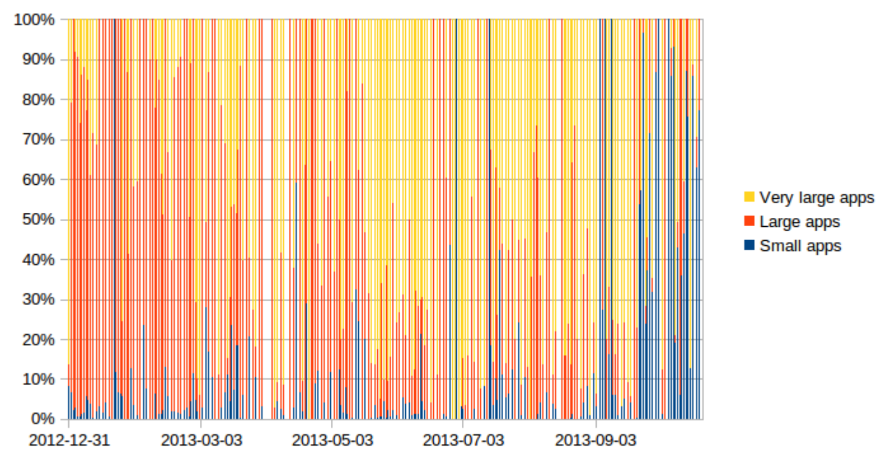
We use Darshan [7], an application level I/O characterization tool developed at Argonne, to capture the behavior of applications running on Intrepid. It intercepts I/O function calls in user space and records access pattern information before the I/O operations are interpreted by the operating system or file system. We analyzed the traces provided by this tool and divided the applications into the following categories [8]:

- *small applications* are applications that run on less than 1,284 nodes, that is less than 20,544 FP cores, or, less than 41,088 integer cores;
- *large applications* are applications that run on more than 1,285 nodes, that is more than 20,560 FP cores, or, more than 41,120 integer cores;
- *very large applications* are applications that run on more than 4,584 nodes, that is more than 123,344 FP cores, or, more than 146,688 integer cores.

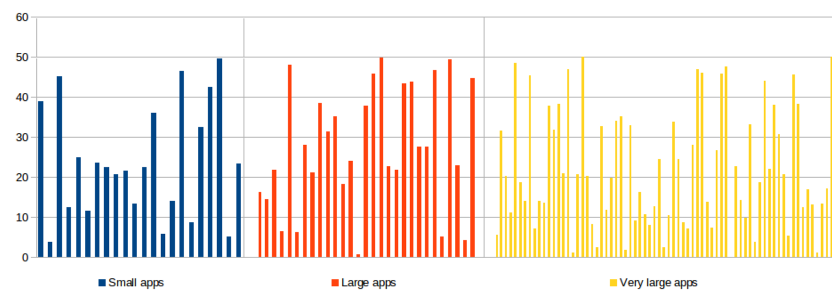
Figure 4 shows how many applications from each type ran on Intrepid during one year from December 2012 to December 2013, how much time each spent doing I/O, and their utilization of the platform. We use this information for generating the simulation scenarios.

In this section, we mainly focus on scheduling periodic applications under I/O bandwidth constraints. Periodic applications follow a pattern which is repeated over time: for all instances of $\mathcal{I}_i^{(k)}$, we have $w^{(k,i)} = w^{(k)}$ and $\mathrm{vol}_{\mathrm{io}}^{(k,i)} = \mathrm{vol}_{\mathrm{io}}^{(k)}$. There are many examples of periodic applications in the HPC community. A simple example would be an application that does not perform any I/O calls, but implements a periodic checkpoint for reliability constraints [9]. Carns et al. [7] use the Darshan I/O characterization tool to capture an accu-

hal-00983789, version 2 - 29 Apr 2014



(a) System usage per day for each application type



(b) Percentage spent doing I/O per application type

Figure 4: Characteristics of application running on Intrepid in 2013.

hal-00983789, version 2 - 29 Apr 2014

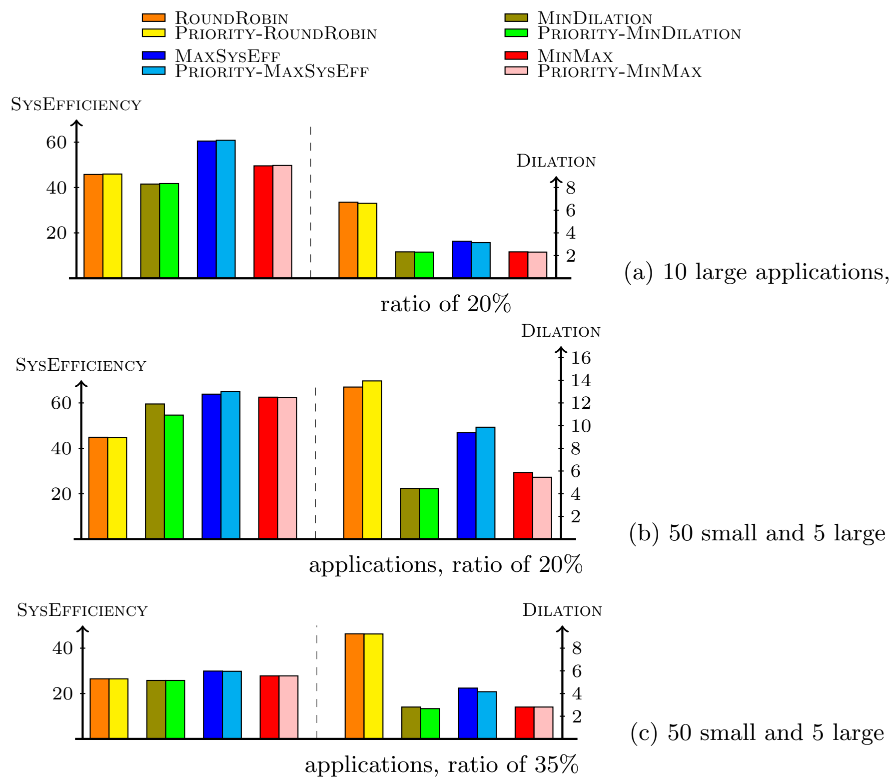
rate picture of I/O patterns in Petascale workloads. In particular, they show that both the S3D application [10] (an application to simulate turbulent combustion using direct numerical simulation of a compressible Navier-Stokes flow) and the HOMME application [11] (an application to model atmosphere physics using spectral element techniques), periodically write out restart files through MPI-IO. Many other applications are periodic. For instance, we were able to verify that the following applications that run on Intrepid, are in fact periodic: the gyrokinetic toroidal code (GTC) [12], Enzo [13], HACC application [14] and CM1 [15]. In Section 4.3 we discuss the impact of application periodicity and show that results are the same for non-periodic applications.

4.2 Assessment of the heuristics

By inspecting the mix of applications that ran on Intrepid (Figure 4a), we observed that two scenarios cover over 95% of the cases: a few large or very-large applications running alone on the whole system, or a mix of small and large applications dividing the machine un-uniformly. We compare the results of the different heuristics over different sets of applications (I/O intensive, computationally intensive, or a mix between the two) following these two scenarios. Figure 5 presents the corresponding results. Simulations were run 200 times on different applications mixes that simulate real scientific applications running on Intrepid, and only the mean values are reported.

We first observe that the Priority variants are, most of the time, less efficient than the original versions, especially when the number of applications running on the system increases. Adding the Priority constraint lessens the flexibility in choosing the set of applications that would maximize the system efficiency. However, the difference in system efficiency and application dilation is small in all studied scenarios. This shows that the heuristics have good results even under the Priority constraint, so that systems that use disks (which at this point represent the large majority of supercomputers) can still benefit from our scheduler.

Another (expected) observation is that MinDilation has better results than MaxSysEff for the Dilation objective, but worse results for the SysEfficiency objective. In particular, with 10 large applications and an average I/O ratio over computation of 20% (Figure 5a), the SysEfficiency of MaxSysEff can be up to 50% larger than that of MinDilation, with a Dilation also up to 50% larger (recall that we want a large SysEfficiency and a small Dilation). The MinMax heuristic (run for $\gamma = 3.7$) is a good trade-off and achieves an intermediate result for both objectives. These results are confirmed, although less visible, in the second scenario (Figure 5b), with many small applications and a few large ones. In Figure 5c, the average I/O ratio over computation is 35%, there are 50 small applications and 5 large ones. In that case, the SysEfficiency of MaxSysEff can be up to 25% that of MinDilation, for a loss in Dilation of 33%. Again, in that case, the MinMax heuristic is a good trade-off.



(a) 10 large applications, ratio of 20%

(b) 50 small and 5 large applications, ratio of 20%

(c) 50 small and 5 large applications, ratio of 35%

Figure 5: Objectives for different mix of applications and IO/computation ratios.

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

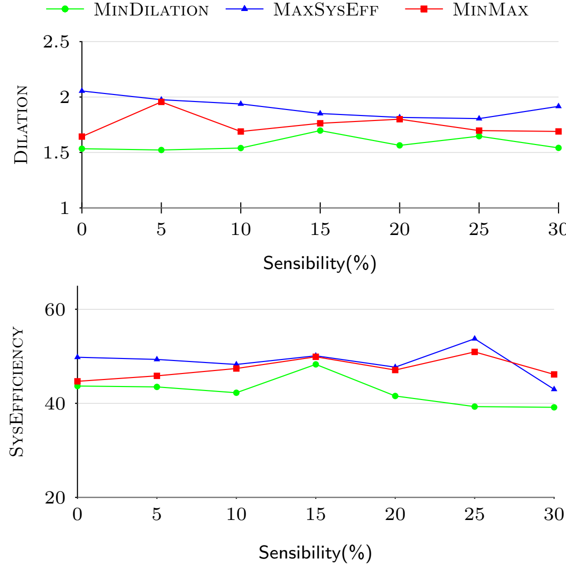


Figure 6: Impact of the sensibility of the computations over SysEfficiency and Dilation of all heuristics.

4.3 Impact of periodicity

As mentioned, based upon the literature and our own verifications on Intrepid, we have assumed so far that applications are periodic. We now discuss the impact of having non-periodic applications in the system. We define the sensibility of an application as $\mathrm{Sens}_w^{(k)} = \frac{\max_i w^{(k,i)} - \min_i w^{(k,i)}}{\max_k w^{(k,i)}}$ and $\mathrm{Sens}_{io}^{(k)} = \frac{\max_i \mathrm{vol}_{\mathrm{io}}^{(k,i)} - \min_i \mathrm{vol}_{\mathrm{io}}^{(k,i)}}{\max_k \mathrm{vol}_{\mathrm{io}}^{(k,i)}}$. For example, for a given application $\mathrm{App}^{(k)}$, if the amount of work between two instances varies from 65 to 102 time units, then $\mathrm{Sens}_w^{(k)} = 1 - \frac{65}{102} \approx 36\%$.

In Figure 6, we study the impact of the sensibility of $w^{(k)}$ for the three heuristics without the Priority constraint. We see that this parameter has almost no impact on the results obtained with periodic applications. This can be explained as follows: the heuristics have no global information about the applications that are being processed, they simply make scheduling decisions according to the information available at each event. We point out that the conclusion is similar when studying the sensibility of the I/O volume.

4.4 Intrepid and Mira simulations

In this section, we focus on comparing the MinMax heuristic and its Priority variant, called Priority for short, with the Intrepid and Mira schedulers as congestion occurs. Due to lack of space, we do not report results for MaxSysEff and MinDilation, but MinMax results always lie between them. Since Intrepid and Mira use burst buffers to improve the behavior of applications with large bursts of I/O, we compare the results on a simulated system that

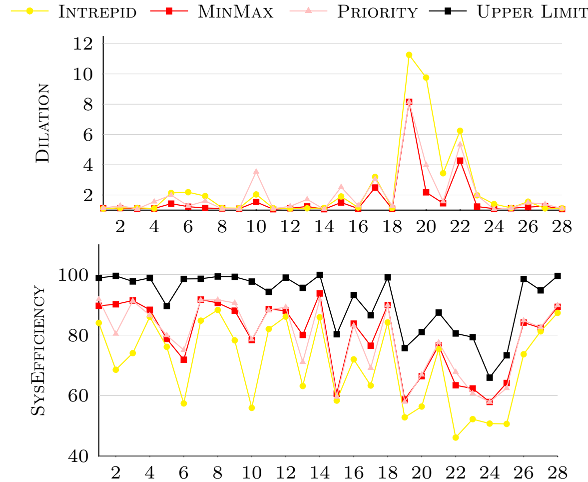


Figure 7: Comparison of the heuristics over the current Dilation and SysEfficiency of Intrepid.

uses MinMax (without burst buffers) with that of a system using burst buffers. Figure 1 presents the congested moments on Intrepid and the induced overhead on I/O throughput seen at the application level. This I/O throughput can be used in the formulas of Section 3 in order to compute SysEfficiency and Dilation on Intrepid. The threshold value γ used for MinMax corresponds to the minimum dilation observed during Intrepid's history for congested moments and it is equal to 3.7.

We have Darshan logs for every congested moment, describing the applications that were running at a given time. We use this information to create the simulation scenario used by our heuristics. The main limitation of the Darshan logs is that they only give information about the total execution time and the total amount of I/O performed by the applications, but do not say anything about their actual behavior. Because most of the applications that run on Intrepid are periodic, we choose to enforce application periodicity by considering that these applications have a random number of iterations, each of a constant execution time and I/O volume. Recall that Section 4.3 has shown that the sensibility does not impact the results, so this hypothesis is not binding. Another limitation with Darshan logs is that they only record around 50% of all the applications running in the system. In most cases when congestion occurs, we did not have access to the information related to the other jobs running in the system. However, we did have information about the coverage of Darshan, so we replicated known applications in order to simulate similar conditions to the usage of the system at the moment of congestion.

Figure 7 presents the SysEfficiency and Dilation obtained on 24 randomly chosen congested moments from the Intrepid system when using the MinMax heuristic. First, it is clear that MinMax behaves better than the current Intrepid scheduler when congestion occurs in all scenarios. On average,

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

the increase in SysEfficiency compared to the congested system efficiency for the MinMax heuristic is 12.58%, slightly higher than the 11.67% obtained for the Priority variant. The gain in Dilation over the current Intrepid scheduler when congestion occurs is on average 35.44% with a peak (in the 20th case) of 348.3%. For the Priority variant, the average gain is 10.53% with a peak of 145.01%. Note that contrary to the SysEfficiency objective, there are cases where the Dilation of the Priority heuristic is worse than the current result of the Intrepid scheduler. Individually, each scenario has different increase values due to different application mixes running in the system at the time. In general, if the volume of I/O done by the application scenario increases, then the difference between our heuristic and the Intrepid scheduler is higher. This is particularly true for the SysEfficiency objective. The aggregated amount of I/O done by the applications running in the system is given by the upper limit curve, which represents the maximum system efficiency achieved when each application can use the maximal bandwidth offered by the system both in the network and for I/O as if it were running alone in the system (see Section 2.2).

The model for our global scheduler does not include burst buffers while Intrepid and Mira use them. Therefore, we extended the model to include virtual burst buffers. The difference compared to what was previously used is the fact that as long as the burst buffers are not full, the applications can resume their execution right after they transferred their I/O volume to the burst buffers. The bandwidth used between the applications and the burst buffers is b while the buffers transfer I/O with a rate of B. Figure 8 presents the SysEfficiency and Dilation obtained on 11 randomly chosen congested moments from the Mira system, using MinMax, Priority, and Priority with burst buffers (called BurstBuffers in the figure). Depending on the I/O behavior of the applications running in the scenario, the advantage of having burst buffers can increase the system efficiency over the classic online scheduler by over 17%. Using our heuristics has similar results to using the the Mira scheduler with burst buffers. However, when using BurstBuffers, the results are much better in all scenarios, having an average gain of 12% and a peak gain of 23.24% over the Mira I/O scheduler.

Because Darshan is not covering all applications running in the system, and also because our model does not include any overhead induced by synchronizing the applications each time they perform I/O, we further validated the results by implementing our heuristics and running them on a real machine. We show in Section 5 that the results obtained in simulation accurately describe what would be obtained if Intrepid or Mira was using our heuristics.

5 Experiments

The study of cross-application interference requires reserving a full machine in order not to be impacted by other applications running in the system at the same time. We have chosen the Vesta machine at Argonne for this purpose. Vesta [16] is a developmental platform for Mira. Its architecture is the same

hal-00983789, version 2 - 29 Apr 2014

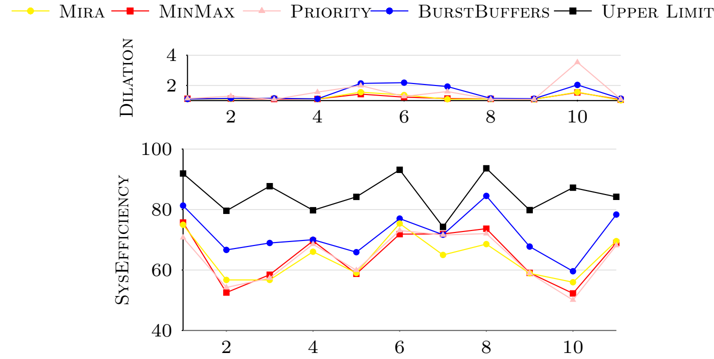


Figure 8: Comparison of the heuristics over the current Dilation and SysEfficiency of Mira.

as Mira's except that it has two compute racks (Mira has 48). A rack has 32 node boards, each of which holds 32 compute cards. Each compute card comprises 16 compute cores of 1600 MHz PowerPC A2 processors with 16GB RAM (1GB/core). In total, Vesta has 2,048 nodes (32,768 compute cores). Applications running on this machine are electrically isolated from each other. This means that even if there are other applications running on the system, their communications will not impact our experiments. Our focus in this section is directed towards write/write interference between multiple applications.

5.1 Setup and measurements

The experiments require a way to control the exact moment when all applications perform I/O. Therefore, we modified the IOR benchmark [17] by splitting its set of processes into groups running independently on different nodes, where each group represents a different application. This way, our implementation of the IOR benchmark allows us to control the access patterns of each application. In addition, because IOR applications are communication-free, we modified them to include some inter-processor communications at each step, in order to make them more similar to typical HPC applications. The added communication is an MPI_Reduce that adds the number of bytes written in the last iteration by each process and simulates the synchronization between different phases of a HPC application.

We made experiments on the modified IOR benchmark and compared the results with the performance of the original IOR benchmark with and without using the option of bypassing I/O buffers. One group of one single process is representing the scheduler and it is responsible for receiving online requests from the rest of the application processes each time they perform an I/O, and confirmations each time the I/O accesses are finished. Since Vesta is using hard disks and it is influenced by the locality of disk access, we implement the Priority variants of the heuristics. Since we have no natural choice for γ in MinMax, we report results for its two extreme cases, namely MaxSysEff ($\gamma =$

hal-00983789, version 2 - 29 Apr 2014

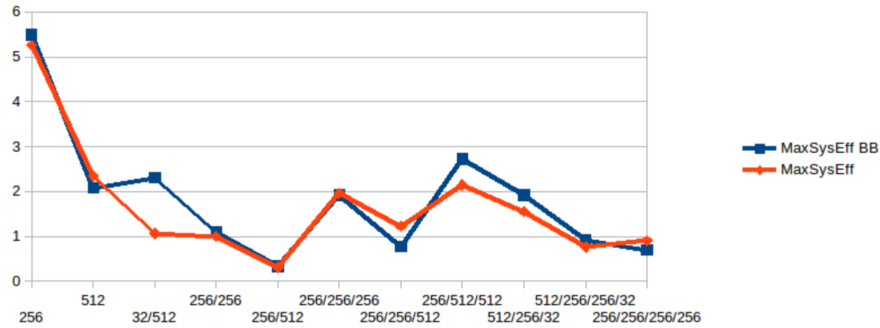


Figure 9: Execution time overhead of our implementation of the IOR benchmark.

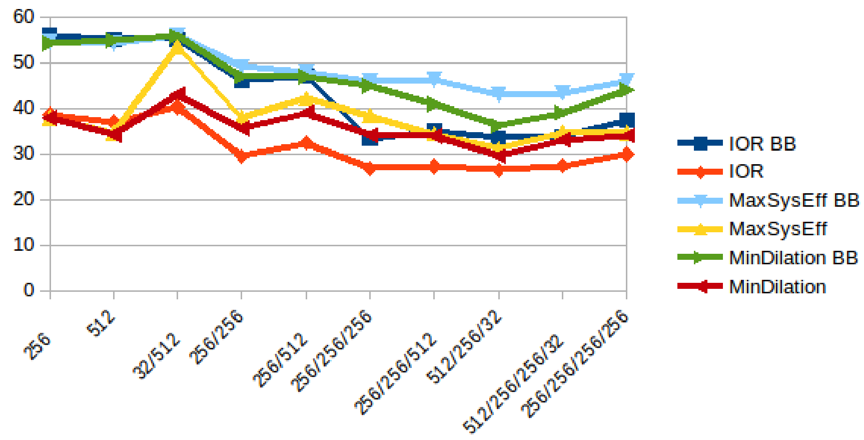
0) and MinDilation ($\gamma = 1$). We implement the heuristics as an additional layer on top of the Vesta I/O scheduler.

In the modified implementation of the IOR benchmark, each application process sends a request to the scheduler thread each time it needs to write some I/O volume. Figure 9 presents the overhead of adding the scheduling thread when no congestion occurs for different scenarios. This overhead was computed by comparing the execution time of one application running the original IOR benchmark with the execution time of our modified version of the IOR benchmark that includes the scheduler. In order to fairly compare the execution time of adding the scheduler without accounting for its benefit in terms of scheduling decisions, in our comparisons, the scheduler always allows all requests to I/O. Depending on the frequency and amount of I/O for each application, the overhead in execution time varies between 1% to 5.3%. In general, for a larger number of applications, the execution time overhead remains under 3%. We account for this idle time as well as the I/O throughput and application delays when computing the system efficiency and application dilation in Section 5.2.

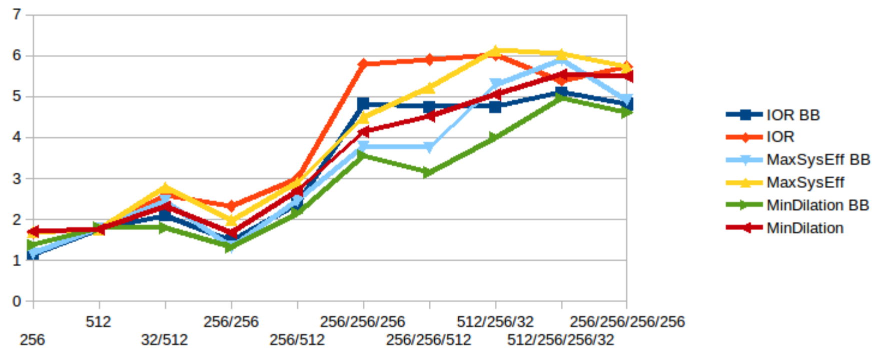
5.2 Results

Figure 10 shows the system efficiency and maximum dilation as seen by all applications running in the system for different scenarios. The horizontal axes present these scenarios in the form x/y/z, where x,y, and z represent the number of nodes used by each application in the system. For example 512/32 means there are two applications running, one on 512 nodes and the other on 32. We made experiments without having any heuristic (results for IOR and IOR BB) and with the modified IOR benchmark using either MaxSysEff or MinDilation. For each case, we ran the application mix either bypassing or using the burst buffers (BB in the name).

The results are very similar to what was seen simulating Mira and confirm what we have observed with the simulations: our heuristics perform very well,



(a) SysEfficiency



(b) Dilation

Figure 10: System efficiency and dilation for different scenarios on Vesta.

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

better than Vesta's I/O scheduler when congestion occurs. Furthermore, the main result of this experimental setup is that with more than 3 applications, our heuristics without burst buffers perform similarly to, and sometimes better than, Vesta's current I/O scheduler with burst buffers when congestion occurs. This superiority of our algorithms over architecture optimizations has benefits from the platform/user perspective (for either objective), but it has an even more important concrete application: if the platform can replace burst buffers with a software scheduler policy, this will most certainly lead to a huge leap in energy efficiency. This is of significant importance since energy consumption is currently one of the main limitation of the race to Exascale.

In general, the MaxSysEff heuristic has larger maximum dilation values than those obtained by letting congestion occur. With the MinDilation heuristic, system efficiency values follow the same curves as with the MaxSysEff heuristic but having, on average, values 5.65% lower. The maximum dilation, however, decreases in all cases showing values smaller than the congested contra-part in all studied scenarios. In general, the MinDilation heuristic has a more significant decrease for the dilation values than it had in the performance values in scenarios when there were more uneven applications (512/32 or 512/256/256/32). We study these scenarios further in the next paragraphs.

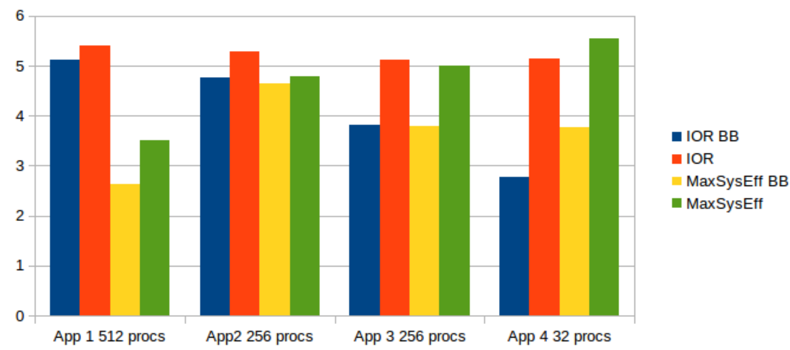
Figure 11 shows the dilation values for each of the four applications running in one of the analyzed scenarios. The small applications are in general more impacted by congestion than the big ones when using the MaxSysEff heuristic, having an increase in their dilation value of 36% compared to the congested dilation. The big applications see a decrease in their dilation of over 48%, which is responsible for the good system performance values. When running the same application mix with MinDilation, the results show an almost uniform decrease in all application dilations compared to the ones obtained running the benchmark without any heuristic, having on average a decrease of 8.4%, and a maximum decrease of 14.5% for the small application.

We can use the MinMax heuristic to obtain the best possible system efficiency without decreasing the application dilation over what would happen during congestion. Depending on the threshold set for this heuristic, a system administrator could tune our heuristic to obtain results within the range of values between the two extremes presented here.

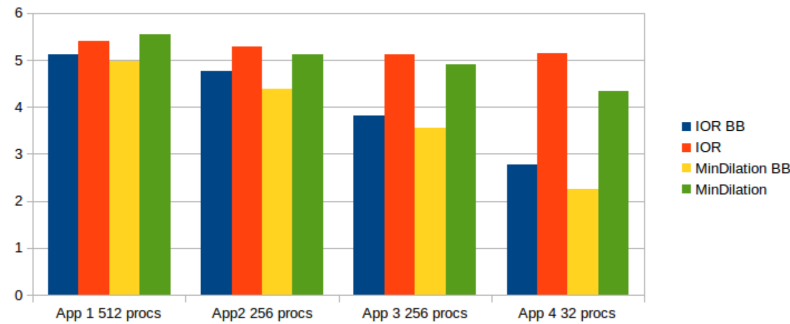
6 Related work

Application performance variability can significantly detract from both the overall performance realized by parallel workloads and the suitability of a given architecture for a workload. In distributed computing, the problem of having performance variability due to sharing resources is well-known and studied. There are numerous papers that analyze this problem for clouds [18, 19, 20]. [19] presents a study of interference specifically for I/O workloads in the cloud in order to understand the performance factors that impact the efficiency and effectiveness of resource multiplexing and scheduling among VMs. In [20], the

hal-00983789, version 2 - 29 Apr 2014



(a) MaxSysEff heuristic



(b) MinDilation heuristic

Figure 11: Dilation values for the applications from 512/256/256/32 scenario.

hal-00983789, version 2 - 29 Apr 2014

authors investigate the sensitivity of measured performance in relation to consolidated server specification of virtual machine resource availability, and burstiness of n-tier application workload. Their results show that an increasingly bursty workload also increases the performance loss among the consolidated servers, however, without being able to offer a solution.

For the HPC community, while many works suggest that I/O congestion is one of the main problems for future scale platforms [1, 21], few paper focus on finding solutions at the platform level. Some papers consider application-side I/O scheduling [2, 3]. In particular, recently, several works focused on using machine learning for auto-tuning and performance studies [22, 23]. However, these solutions do not have a global view of the I/O requirements of the system, and they need to be supported by a platform level I/O management for better results. Cross-application contention has been recently studied in several articles [24, 25, 26]. The study in [24] evaluates the performance degradation in each application program when VMs are executing two application programs concurrently in a physical computing server. The experimental results indicate that the interference among VMs executing two HPC application programs with high memory usage and high network I/O in the physical computing server, significantly degrades application performance. An earlier study in 2005 [25] cites application interference as one of the main problems facing the HPC community. While it proposes ways of gaining performance by reducing variability, minimizing application interference is still left open. [27] is a more general study that analyzes the behavior of the center-wide shared Lustre parallel file system on the Jaguar supercomputer and its performance variability. One of the performance degradations seen on Jaguar was caused by concurrent applications sharing the filesystem. All of these studies highlight the impact of having application interference on HPC systems without, however, offering a solution.

[6] studies the access to disks by multiple applications running in the system by focusing on cases when I/O requests from multiple applications might break the spatial locality of individual programs; this can seriously degrade I/O performance when the data servers concurrently serve synchronous requests from multiple I/O-intensive programs. The authors propose a scheme called IOrchestrator, to improve I/O performance of multi-node storage systems by orchestrating I/O services among programs when such inter-data-server coordination is dynamically determined to be cost effective. Their tool has a global overview of applications in the system and decides which request to perform and in which order, but they simply choose an FCFS ordering. Our implementation focuses on avoiding application interference and provides a variety of heuristics that take into account application history and system properties.

The research closest to our study is [28]. The authors investigate the interference of two applications and analyze the benefits of interrupting or delaying either one in order to avoid congestion. Our study is much more general. It looks at different application mixes and offers a range of options that give good results for two distinct objectives. These results can be used by a system administrator to configure the best solution for their particular machine.

hal-00983789, version 2 - 29 Apr 2014

7 Conclusion and future work

I/O interference of multiple applications running concurrently in the system is one of the main sources of performance variability in HPC systems. We have studied the effects of congestion on application performance and on total system efficiency, and we propose several solutions that minimize the performance degradation. Our global scheduler has a global view of the system and on the past behavior of all applications running at a given time, and dynamically schedules I/O accesses so as to minimize the maximum application dilation and/or to increase the system-wide efficiency.

We show through extensive experiments that our scheduler performs better than current solutions for HPC systems. Moreover, our software scheduler gives very similar results to current architectural enhancements (burst buffers). This is of significant interest since these burst buffers come at an energy price that is important for future exascale platforms. A small limitation of our work is the special case when no I/O congestion occurs on the platform, since our scheduler assumes an extra communication step taken by applications each time they need to perform I/O. There are two directions that we plan to investigate to improve the execution time in this case. First, if the applications were aware of the existence of the scheduler, one thread from each application could be dedicated to communicating with the scheduler process. Another direction is to use a tool to predict the I/O accesses of each application running in the system. Depending on the accuracy of such a tool, the scheduler could be used only when a congestion situation has been predicted.

HPC applications in general are periodic and their behavior is in most cases well known in advance. A periodic scheduler might give even better results than the one proposed in this paper. Periodic schedules would have to be implemented inside the system's job scheduler. It would have a more accurate global view and would be able to compute a complete schedule over a period of given length in advance for all applications, which in return would give a more flexible way of controlling the behavior of the applications. We expect periodic schedulers to be an interesting complement to the online schedulers presented in this paper. Future work will be devoted to assessing the additional gain that periodic schedulers may bring in comparison to online schedulers, and their robustness with respect to the periodicity hypothesis.

Acknowledgments: This research was done in the context of the INRIA-Illinois Joint Laboratory for Petascale Computing. The work was also supported by the U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357, and the ANR Rescue project. A. Benoit and Y. Robert are with Institut Universitaire de France.

hal-00983789, version 2 - 29 Apr 2014

References

[1] R. Biswas, M. Aftosmis, C. Kiris, and B.-W. Shen, "Petascale computing: Impact on future nasa missions," *Petascale Computing: Architectures and Algorithms*, pp. 29–46, 2007.

[2] X. Zhang, K. Davis, and S. Jiang, "Opportunistic data-driven execution of parallel programs for efficient i/o services," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 330–341.

[3] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing variability in the io performance of petascale storage systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[4] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous i/o," in *ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001.

[5] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. Ganger, "Argon: Performance insulation for shared storage servers," in *5th USENIX Conference on File and Storage Technologies (FAST'07)*, 2007.

[6] X. Zhang, K. Davis, and S. Jiang, "Iorchestrator: improving the performance of multi-node i/o systems via inter-server coordination," in *In ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis. IEEE*, 2010.

[7] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[8] W. Kramer, "Blue waters and the future of scale computing and analysis," in *The forth AICS International Symposium*, 2013.

[9] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *FGCS*, vol. 22, no. 3, pp. 303–312, 2004.

[10] R. Sankaran, E. R. Hawkes, J. H. Chen, T. Lu, and C. K. Law, "Direct numerical simulations of turbulent lean premixed combustion," in *Journal of Physics: conference series*, vol. 46, no. 1. IOP Publishing, 2006, p. 38.

[11] R. Nair and H. Tufo, "Petascale atmospheric general circulation models," in *Journal of Physics: Conference Series*, vol. 78, no. 1. IOP Publishing, 2007, p. 012078.

[12] S. Ethier, M. Adams, J. Carter, and L. Oliker, "Petascale parallelization of the gyrokinetic toroidal code," *VECPAR: High Performance Computing for Computational Science*, 2012.

[13] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman *et al.*, "Enzo: An adaptive mesh refinement code for astrophysics," *arXiv preprint arXiv:1307.2265*, 2013.

[14] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel *et al.*, "The universe at extreme scale: multi-petaflop sky simulation on the bg/q," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* IEEE Computer Society Press, 2012, p. 4.

[15] G. H. Bryan and J. M. Fritsch, "A benchmark simulation for moist nonhydrostatic numerical models." *Monthly Weather Review*, vol. 130, no. 12, 2002.

[16] "Cetus and Vesta: Test and Development systems." https://www.alcf.anl.gov/cetus-and-vesta.

[17] H. Shan and J. Shalf, "Using ior to analyze the i/o performance for hpc platforms," *Cray User Group Conference*, 2007.

[18] H. Chiang, R.C.and Huang, "Tracon: Interference-aware schedulingfor data-intensive applicationsin virtualized environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 1349–1358, 2014.

[19] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao, "Who is your neighbor: Net i/o performance interference in virtualized clouds," *IEEE Transactions on Services Computing*, vol. 6, pp. 314–329, 2013.

[20] Y. Kanemasa, Q. Wang, J. Li, M. Matsubara, and C. Pu, "Revisiting performance interference among consolidated n-tier applications: Sharing is better than isolation," *IEEE International Conference on Services Computing (SCC)*, pp. 136–143, 2013.

[21] J. Lofstead and R. Ross, "Insights for exascale io apis from building a petascale io api," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis.* ACM, 2013, p. 87.

[22] B. Behzad, L. H. V. Thanh, J. Huchette, S. Byna, R. A. Prabhat, Q. Koziol, and M. Snir, "Taming parallel i/o complexity with auto-tuning," in *Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2013)*, 2013.

hal-00983789, version 2 - 29 Apr 2014

hal-00983789, version 2 - 29 Apr 2014

[23] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. A. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross *et al.*, "Characterization and modeling of pidx parallel i/o for performance optimization," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis.* ACM, 2013, p. 67.

[24] Y. Hashimoto and K. Aida, "Evaluation of performance degradation in hpc applications with vm consolidation," *IEEE International Conference on Networking and Computing (ICNC)*, pp. 273–277, 2012.

[25] D. Skinner and W. Kramer, "Understanding the causes of performance variability in hpc workloads," *IEEE Workload Characterization Symposium*, pp. 137–149, 2005.

[26] A. Uselton, M. Howison, N. Wright, D. Skinner, N. Keen, J. Shalf, K. Karavanic, , and L. Oliker, "Parallel i/o performance: From events to ensembles," *IEEE IPDPS*, pp. 1–11, 2010.

[27] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," *High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–11, 2012.

[28] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim, "Calciom: Mitigating i/o interference in hpc systems through cross-application coordination," in *IPDPS-International Parallel and Distributed Processing Symposium*, 2014.

hal-00983789, version 2 - 29 Apr 2014

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399