Improving the effectiveness of burst buffers for big data processing in HPC systems with Eley

Orcun Yildiz, Amelie Chi Zhou, Shadi Ibrahim

PII: S0167-739X(17)31730-2
DOI: https://doi.org/10.1016/j.future.2018.03.029
Reference: FUTURE 4041

To appear in: *Future Generation Computer Systems*

Received date : 31 July 2017
Revised date : 20 February 2018
Accepted date : 14 March 2018

Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 00 (2018) 1–17
Future Generation Computer Systems

Improving the Effectiveness of Burst Buffers for Big Data Processing in HPC Systems with Eley

Orcun Yildiz[1], Amelie Chi Zhou[2], Shadi Ibrahim[3]*

[1] *Inria, Univ Rennes, CNRS, IRISA, Rennes, France*

[2] *Shenzhen University, China*

[3] *Inria, IMT Atlantique, LS2N, Nantes, France*

Abstract

Burst Buffer is an effective solution for reducing the data transfer time and the I/O interference in HPC systems. Extending Burst Buffers (BBs) to handle Big Data applications is challenging because BBs must account for the large data inputs of Big Data applications and the Quality-of-Service (QoS) of HPC applications – which are considered as first-class citizens in HPC systems. Existing BBs focus on only intermediate data of Big Data applications and incur a high performance degradation of both Big Data and HPC applications. We present *Eley*, a burst buffer solution that helps to accelerate the performance of Big Data applications while guaranteeing the QoS of HPC applications. To achieve this goal, *Eley* embraces interference-aware prefetching technique that makes reading data input faster while introducing low interference for HPC applications. Evaluations using a wide range of Big Data and HPC applications demonstrate that *Eley* improves the performance of Big Data applications by up to 30% compared to existing BBs while maintaining the QoS of HPC applications.

Keywords: HPC; MapReduce; Big Data; Parallel File Systems; Burst Buffers; Interference; Prefetch.

1. Introduction

With the arrival of the era of Big Data, we have witnessed the emergence of a new scalable data management and processing paradigm through the MapReduce model [1, 2] for data-intensive processing. MapReduce is adopted in both industry and academia due to its simplicity, transparent fault tolerance and scalability. For instance, Carnegie Mellon University is using MapReduce clusters for data analysis on several scientific domains including computational astrophysics, computational neurolinguistics, natural language processing and social networking analysis [3]. The success of MapReduce led to the emergence of new types of applications (e.g., stream data processing, graph processing, analysis of large scale simulation data) where obtaining timely and accurate responses is a must. However, commodity machines-based Big Data environments reach their limit due to being failure-prone and providing moderate performances.

Why Big Data processing on HPC Systems? High performance computing (HPC) systems are known for providing the maximum computing capability. They are equipped with high-speed networks, thousands of nodes with many

*Corresponding author
Email address: `shadi.ibrahim@inria.fr` (Shadi Ibrahim[3])

cores and large memories [4, 5]. For instance, Sunway TaihuLight, No.1 in the top 500 supercomputers list, is a 10,649,600 processor supercomputer with a Linpack performance of 93 petaflop/s[1]. *Given the high performance nature of HPC systems and the increased need of the Big Data community for fast Big Data processing, the Big Data community is rapidly moving towards ways to leverage these HPC systems [6, 7].*

Big Data processing on HPC Systems: Challenges. Introducing Big Data processing to these systems is not trivial; it comes with its own set of challenges. First, one should be aware of the different architectural designs in current Big Data processing and HPC systems. Big Data processing clusters have shared-nothing architecture (e.g., nodes with individual disks attached) and thus they can co-locate the data and compute resources on the same machine (i.e., data locality) when performing the task scheduling. On the other hand, HPC clusters employ a shared architecture (e.g., parallel file systems) [8] which results in separation of data resources from the compute nodes. Big Data systems are mainly optimized to sustain high *data locality* which is the major contributing factor to the application performance [9, 10, 11]. However, the same approach on the data locality can not be applied in HPC systems since all the nodes have an equivalent distance to the input data. In addition, employing a shared parallel file system also results in remote data transfers through network when Big Data applications performing I/O thus resulting in a *higher latency*. Moreover, *I/O interference* (i.e., performance degradation observed by any single application/task running in contention with other applications/tasks on the same platform [12, 13]) is a well-known performance bottleneck for HPC applications due to the large size and shared architecture of HPC clusters. Hence, *the impact of both latency and interference will be amplified, if HPC systems are to serve as the underlying infrastructure for Big Data processing.*

Burst Buffers for Big Data applications in HPC Systems: Current status. Many research efforts have been dedicated to overcome the high latency problem by introducing an intermediate layer — Burst Buffers (BBs) — with high throughput storage devices [14, 15, 16, 17] especially for storing the *intermediate data* (i.e., map output for batch jobs and temporary output produced between stages for iterative jobs). For example, Islam et al. [14] utilized NVRAM as an intermediate storage layer (i.e., burst buffer) between compute nodes and Lustre which improved the application performance by 24%. Wang et al. [17] leveraged SSDs for storing the intermediate data. Chaimov et al. [15] used a separate set of nodes (burst buffer nodes) with NVRAM as a storage space to achieve a better scalability by reducing the latency when reading/writing the intermediate data. Unfortunately, aforementioned works ignore the importance of input data despite the significant amount of work dedicated to improve the application performance by sustaining high input data locality, as we mentioned previously. In addition, these works proposed the adoption of burst buffers in a similar way to the traditional use of burst buffers on HPC systems which aim to minimize the I/O time by absorbing the checkpointing data of scientific applications. In contrast, Big Data applications mostly run in batches therefore there is a continuous interaction with the parallel file system for reading the input data. Lastly, none of these efforts considered the interference problem which can contribute to a significant performance degradation — not only for Big Data applications but also for the first class HPC applications — by up to 2.5x [12]. Hence, *as we argue in this paper, current efforts and solutions to extend burst buffers for Big Data applications in HPC systems may fail in practice to achieve the desired performance and this may hinder the convergence of Big Data processing and HPC.*

Contributions. In an attempt to (effectively) extend Burst Buffers for Big Data applications on HPC systems, in this paper, *we present Eley, a burst buffer solution that aims to accelerate the performance of Big Data applications while guaranteeing the QoS of HPC applications.* A typical job of Big Data applications consists of several iterations (waves) depending on the total input size and the cluster capacity (i.e, the number of map tasks which can simultaneously run in the cluster). For example, if the cluster capacity is set to 100 map tasks where each map task processes one block of 128 MB, a job of 100GB data inputs will be executed in 8 waves. This is common for large scale scientific Big Data applications: during 9 months in a research cluster (i.e., M45) with 400 nodes [18], the amount of processed data was almost 900 TBs and almost 100 of executed jobs have an input data of 4.9 TBs. Thus, *Eley* employs a novel (prefetcher) technique that fetches data inputs of next iterations while computing nodes are still busy processing data inputs of previous one. This allows to have data inputs to be stored on a low-latency device close to computing nodes and therefore results in reducing the latency of reading data inputs of Big Data applications. However, data prefetching may come at a high cost for the HPC applications: the continuous interaction with the parallel file system (i.e., I/O read requests) may introduce a huge interference at the parallel file system level and thus end up with a degraded and unpredictable performance for HPC applications. To this end, we design *Eley* to be interference-aware.

[1] `http://www.top500.org`

Specifically, we equip the prefetcher with five optimization actions including No Action, Full Delay, Partial Delay, Scale Up and Scale Down. It iteratively chooses the best action to optimize the prefetching while guaranteeing the pre-defined QoS requirement of HPC applications (i.e., the deadline constraint for the completion of each I/O phase). We evaluate the effectiveness of *Eley* with both real system evaluations and simulations. With 5% QoS requirement of the HPC application, *Eley* reduces the execution time of Big Data applications by up to 30% compared to the Naive burst buffer solution (denoted as *NaiveBB*) [14, 15, 16, 17] while guaranteeing the QoS requirement. On the other hand, the *NaiveBB* violates the QoS requirement by up to 58%.

In summary this paper makes the following contributions:

- We propose *Eley*, a burst buffer solution for reducing the latency of reading data inputs for Big Data applications while maintaining the QoS of HPC applications. The intuition behind *Eley* is that, on the one hand, Big Data applications are often executed in waves, thus, it is desirable to have data close to compute nodes once required. On the other hand, the heavy I/O requests introduced by Big Data applications to the parallel file system may adversely impact on the performance of the (high priority) HPC applications, thus it is important to alleviate this impact as much as possible (Section 3).

- We introduce interference and performance models for both HPC and Big Data applications, in order to identify the performance gain and the interference cost of the prefetching technique of *Eley* (Section 4). Based on these two models, we propose an interference-aware prefetching technique that iteratively chooses in-between five action to optimize the prefetching, in order to satisfy the QoS requirement of HPC while leading to a good performance of Big Data application (Section 5).

- Evaluations using representative and widely used Big Data and HPC applications demonstrate the effectiveness of *Eley* in obtaining shorter execution time for Big Data applications (shorter map phase) while maintaining the QoS of HPC applications. We believe that these results are encouraging and will help to bring forward the convergence between Big Data and HPC (Section 7).

2. Related Work

Early adoption of Big Data processing on HPC systems. Many research efforts have been dedicated to adopt Hadoop [19], well-known open source MapReduce implementation, on HPC systems [20, 21, 22]. Researchers have discussed solutions to extend data locality to parallel file system (i.e., PVFS [23]) through emulating HDFS [24] on a HPC system by using PVFS [20] or proposing a new storage layer (in-memory storage) on top of PVFS [22]. *Those solutions assume that computing nodes have dedicated disk, this however is not a typical configuration of HPC system.* Moreover, new MapReduce frameworks — beyond Hadoop — have been introduced such as MARIANE [25] and Glasswing [26]. They mainly focus on exploiting multi-core CPU and GPUs. Although aforementioned works can improve the performance of Big Data applications compared to a blind adoption of these applications on HPC systems, *they do not fully address the challenges (i.e., interference, high latency) and thus they are not able to efficiently leverage HPC systems for Big Data processing.*

Burst buffers for Big Data applications in HPC systems. Several works proposed adoption of burst buffers for an efficient Big Data processing on HPC systems. Chaimov et al. [15] employed a NVRAM buffer between compute nodes and Lustre file system in order to improve the scalability of Spark framework. Islam et al. [27] proposed using Memcached as a burst buffer system to integrate HDFS with Lustre file system in a performance-efficient way. They also evaluated the NVRAM performance as a burst buffer system in [14]. Wang et al. [17] performed an experimental study where they investigated the characteristics of Spark on a HPC system with a special focus on the impact of the storage architecture. Based on their findings, they proposed to use SSDs as a burst buffer to store the intermediate data. *As we demonstrated, however, although these studies are indeed important — without considering the interference problem and the latency resulting from the input phase — they do not provide a complete solution for enabling efficient Big Data processing on HPC systems.*

Mitigating I/O interference in HPC systems. I/O interference in HPC systems is an important problem that can seriously impact the performance of HPC applications. For several years researchers have been trying to mitigate this interference problem. For example, Zhou et al. [28] present an I/O-aware batch scheduler that addresses the interference problem at the level of batch scheduling. The batch scheduler schedules and coordinates the I/O requests

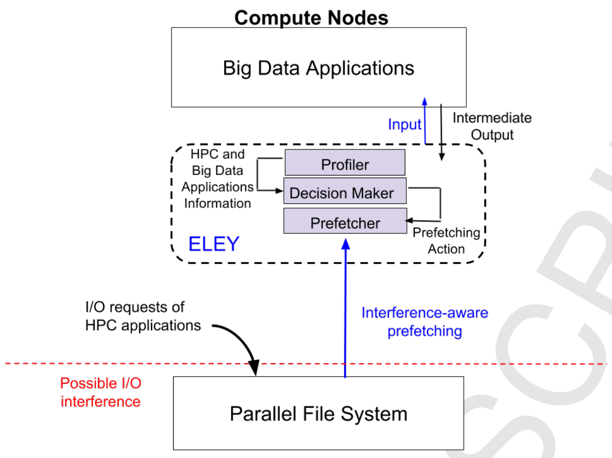


Figure 1. System overview of Eley.

on the fly by considering the system state and I/O activities. Gainaru et al. [29] show the performance degradation due to I/O congestion and propose a new scheduler that tries to eliminate this congestion by coordinating the I/O requests depending on the application's past behaviors and system characteristics. Boito et al. [30] propose AGIOS, an I/O scheduling library for parallel file systems. AGIOS incorporates the applications' access pattern information into the scheduler based on the traces generated by the scheduler itself and uses this information to coordinate the I/O requests in order to prevent congestion to the file system. [31] proposes several I/O scheduling policies at the application level to mitigate I/O interference. These policies include dynamically partitioning the PFS resources among applications and delaying the I/O requests without interrupting the computation by using burst buffers. *Similarly to these application-side scheduling solutions, we introduce an interference-aware prefetching scheme to minimize the performance degradation resulting from the I/O interference between HPC and Big Data applications.*

3. System Overview

In this section, we present an overview of the burst buffer solution (i.e., Eley) and describe our main design principles.

3.1. Burst Buffer Enabled HPC System

With the convergence between Big Data and HPC, many Big Data applications are moving to HPC systems to benefit from their high computation capabilities. Figure 1 shows a typical architecture of HPC system with burst buffer enabled to address the high I/O latency challenge of Big Data applications on HPC. *Eley* is located between compute nodes and parallel file system, where compute nodes only interact with the burst buffer nodes for all I/O phases of the Big Data application, including reading input data and reading/writing intermediate data. Thus, the file system level interference mainly comes from the I/O requests of burst buffer nodes and the HPC applications.

Different from the existing studies on burst buffer, in this paper, we instruct Eley to prefetch the input data of Big Data applications to optimize the I/O performance of those applications. When we co-locate HPC and Big Data applications, prefetching operations may interfere with the I/O operations of HPC applications. Thus, we design a *prefetcher* component in the burst buffer to manage the prefetching of input data in a controlled manner. Specifically, our prefetcher component takes into consideration the file system level interference (i.e., based on the profiled information about the compute and I/O times of both HPC and Big Data applications which is used for estimating the interference) and makes optimization decisions before each prefetching operation, in order to optimize the performance of Big Data applications while satisfying QoS requirements of HPC applications. Then, it initiates the prefetching operations according to the optimized decisions. In each prefetching operation, the prefetcher fetches a subset of the input data (i.e., one wave) for the next wave of computation. By doing so, we can overlap the I/O and computation phases of Big Data application to further hide the high latency of remote data accesses. We introduce the optimization actions and decision making process of the prefetcher in the next section.

3.2. Design Principles

We follow the following principles when designing the burst buffer enabled HPC system, which aim to address the challenges of running Big Data applications on HPC systems without introducing much interference to the HPC applications.

Enabling efficient Big Data processing on HPC systems: The separation of storage resources from the compute nodes in HPC systems requires repetitive data transfer through the network, which results in a high I/O latency. In this paper, we focus on Big Data applications with huge input data sizes which are executed in multiple waves. Thus, the high I/O latency can severely degrade the performance of Big Data applications on HPC systems. We mitigate the latency problem by locating low-latency burst buffer devices (i.e., RAMDisk) close to the compute nodes to reduce the latency of reading data inputs of Big Data applications.

Existing burst buffer studies mainly focus on mitigating the I/O latency on the intermediate data of Big Data applications. However, by analyzing traces collected from three different research clusters, we observe that the intermediate data size is smaller than 20% of the input data size for over 85% of the applications [18]. Thus, it is important to also mitigate the high I/O latency in reading the input data. To this end, we equip our burst buffer with a prefetcher component.

Guaranteeing QoS requirement of HPC applications: Running Big Data applications in HPC systems should not introduce much interference to the HPC applications, which usually involve important scientific simulations. Thus, one of our key design principles is to guarantee the QoS requirements of HPC applications while improving the I/O performance of Big Data applications. We define the QoS requirement of HPC as the deadline constraint for the completion of each I/O phase of HPC applications defined by the users. To this end, we equip our prefetcher with interference-aware data transfer scheme to help satisfying our design objective.

4. Model Formulation

The I/O operations of Big Data applications can seriously degrade the I/O performance of concurrent HPC applications. Thus, in this section, we formally model the impact of interference on the I/O time of both Big Data and HPC applications, in order to optimize the performance of Big Data while guaranteeing the QoS constraint of HPC applications.

4.1. Performance Model for HPC Applications

Our model focuses on HPC applications with periodical behaviors. That is, the applications perform series of computations and periodically checkpoint the computation results with an I/O time of T_{io}. Most of the HPC applications fall into this category of applications [32]. We assume that the profiles of HPC applications (e.g., I/O and computation time when running individually) can be obtained at offline time using solutions such as Omnisc'IO [33]. We define the I/O time of a HPC application when co-locating with a Big Data application as T_{col}.

$$T_{io}^{col} = T_{io}^{alone} + T_{io}^{intf} \times I_{hpc} \tag{1}$$

where T_{io}^{alone} is the time period that the HPC application performs I/O requests individually and T_{io}^{intf} ($T_{io}^{intf} = T_{io} - T_{io}^{alone}$) is the time period that the HPC application is contending with the Big Data application for I/O resources. I_{hpc} is the interference factor, defined as the ratio between the I/O time of the HPC application under contention and the time for it to perform the same I/O operations alone (i.e., $I_{hpc} > 1$).

According to Equation 1, an important parameter for estimating the I/O performance of HPC application is the interference factor. The I/O interference imposed by Big Data applications (i.e., prefetching) is affected by several parameters, including the number of the burst buffer nodes performing prefetching (n_{bb}) and the number of fetching threads per node (n_{tr}). I_{hpc} is also depending on the characteristics of the HPC application itself and the platform where it is running. Thus, given an interfering Big Data application to the HPC (i.e., given a set of n_{bb} and n_{tr} values), we can calculate I_{hpc} for the HPC application by profiling its I/O performance with and without contention from the Big Data application. However, offline profiling for each set of given n_{bb} and n_{tr} values is very costly. To reduce the profiling cost, we decompose the interference into two levels, namely node-level (NI) and thread-level (TI) interference, and assume that they are independent from each other. By profiling the two individually, we reduce the profiling complexity from $O(n \times m)$ to $O(n + m)$, supposing that n and m are the sample sizes for n_{bb} and n_{tr}, respectively.

We profile the node-level and thread-level interference and record $NI(n_{bb})$ and $TI(n_{tr})$ values for different n_{bb} and n_{tr} samples. Specifically, we perform prefetching using different numbers of n_{bb}/n_{tr} while fixing n_{tr}/n_{bb} to the default with the HPC application performing checkpointing at the same time. We set the default value of both n_{bb} and n_{tr} to 1. For example, $TI(2)$ is calculated as $\frac{t_2}{t_1}$, where t_1 and t_2 are the I/O performance of HPC application when one burst buffer is performing prefetching with 1 and 2 threads, respectively. Based on the above analysis, we can calculate the interference factor for HPC as follows.

$$I_{hpc}(n_{bb}, n_{tr}) = TI(n_{tr}) \times NI(n_{bb}) \times I_{hpc}(1, 1) \tag{2}$$

where $I_{hpc}(1, 1)$ is profiled and calculated at offline time.

4.2. Performance Model for Big Data Applications

We provide a performance model for Big Data applications to estimate the prefetching time when using different n_{bb} and n_{tr} values. Similarly to Equation 1, the prefetching time is also impacted by the contention duration with HPC and the interference factor. We model the prefetching time when running Big Data application alone with n_{bb} burst buffer nodes and n_{tr} threads per node as below.

$$T_{pref}(n_{bb}, n_{tr}) = NS(n_{bb}) \times TS(n_{tr}) \times T_{pref}(1, 1) \tag{3}$$

where $NS(n_{bb})$ and $TS(n_{tr})$ represent the system scalability with respect to different number of nodes and number of threads per node, respectively, with the assumption that the node-level and thread-level system scalability are independent. We obtain the scalability values with system profiling using different n_{bb} and n_{tr} samples in the same way as described in the last subsection. For example, $NS(2)$ is calculated as $\frac{t_2}{t_1}$, where t_1 and t_2 are the prefetching time with one and two burst buffer nodes, respectively. Similarly, we obtain $T_{pref}(1, 1)$ by profiling the prefetching time with a single burst buffer node and a single thread.

When co-located with HPC applications, the performance of Big Data applications can be modeled as below.

$$T_{pref}^{col} = T_{pref}^{alone} + T_{pref}^{intf} \times I_{bd} \tag{4}$$

Similarly to Equation 1, I_{bd} is the interference factor for Big Data application, defined as the ratio between the prefetching time with and without co-location with HPC applications. We calculate $I_{bd}(n_{bb}, n_{tr})$ using offline profiling similarly to Eqaution 2. Specifically, we profile $TI(n_{tr})$ and $NI(n_{bb})$ individually using the prefetching time under different n_{bb} and n_{tr} values.

Further, we define the *cost* of prefetching for Big Data applications as the time that the prefetching cannot be overlapped with the computation phase. To optimize the performance of Big Data, we aim to reduce the cost of prefetching in order to hide the I/O latency of reading input data.

$$C_{pref} = \begin{cases} T_{pref}^{col} - T_{cpu} & \text{, if } T_{pref}^{col} \geq T_{cpu} \\ 0 & \text{, otherwise} \end{cases} \tag{5}$$

where T_{pref}^{col} is the time of fetching input data for the next wave and T_{cpu} is the computation time of the current wave.

5. Interference-Aware Prefetching

Prefetching of Big Data applications imposes interference to the I/O operations of HPC applications and can violate the QoS requirement of HPC applications. Thus, we propose a set of optimization actions and iteratively choose the best action to optimize the prefetching, in order to satisfy the QoS requirement of HPC while leading to a good performance of Big Data application. In the following, we first introduce the five optimization actions in the action set and then introduce how to choose good actions to optimize prefetching.

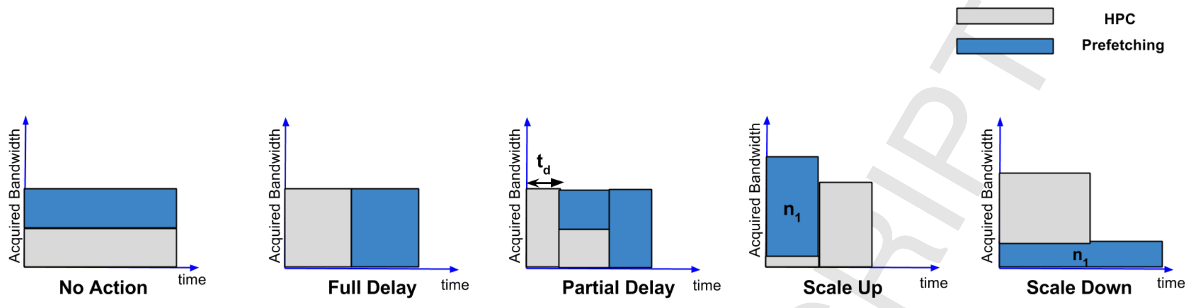


Figure 2. Use cases of five actions.

Name	Description
No Action	Start the prefetching as it is.
Full Delay	Delay prefetching to the end of the I/O phase of HPC application.
Partial Delay	Delay prefetching for a fixed amount of time.
Scale Up	Increase the number of prefetching threads per node.
Scale Down	Decrease the number of total prefetching threads.

Table 1. Details of the actions.

5.1. Action Set

We propose five optimization actions, including *No Action, Full Delay, Partial Delay, Scale Up* and *Scale Down*. These actions are lightweight and common strategies for interference mitigation. Table 1 summarizes the definitions of the five actions. Moreover, we illustrate these actions in Figure 2 where the x-axis represents time and y-axis represents the bandwidth of the parallel file system shared between HPC and Big Data applications. The boxes represent the I/O operations of HPC and Big Data applications. The width of boxes represents the completion time and the height represents the acquired bandwidth of the I/O operations. Note that, in the illustration, both applications start I/O operations at the same time. Next, we present the details of the five actions.

No Action performs the prefetching as it is, immediately after receiving the I/O request. We use it as the baseline to show how the other actions optimize the prefetching operation. Thus, the completion time of the I/O operations for Big Data and HPC are $T_{pref} \times I_{bd}$ and $T_{io} \times I_{hpc}$, respectively.

Full Delay delays the prefetching to the end of the I/O phase of HPC application as shown in Figure 2. While this allows HPC application to perform its I/O operations alone and thus mitigates any interference, it can greatly harm the performance of Big Data application when the HPC application has a long running I/O phase. The completion time of the I/O operations for Big Data and HPC are $T_{pref} + T_{io}$ and T_{io}, respectively.

Partial Delay delays the prefetching for a certain amount of time t_d to meet the QoS requirement of the HPC application. We calculate the optimized I/O completion time T^{col}_{io} for HPC and T^{col}_{pref} for prefetching with t_d delay as follows.

$$T^{col}_{io} = t_d + (T_{io} - t_d) \times I_{hpc} \tag{6}$$

$$T^{col}_{pref} = T_{pref} + (T_{io} - t_d) \times I_{hpc} \times (1 - \frac{1}{I_{bd}}) \tag{7}$$

Scale Up increases the number of fetching threads per node. As shown in Figure 2, increasing the number of total fetching threads can reduce the prefetching time with an increase on the acquired I/O bandwidth. Therefore, while this action reduces the interference time period of HPC and Big Data applications, it can also lead to a larger performance degradation to HPC due to the higher interference factor I_{hpc}. Assume scale up increases the number of fetching threads per node from n_0 to n_1, we calculate T_{pref}, I_{hpc} and I_{bd} values for n_1 threads based on the model defined in Section 4 and calculate the optimized I/O completion time for both applications as follows.

$$T_{io}^{col} = T_{io} + T_{pref} \times I_{bd} \times (1 - \frac{1}{I_{hpc}}) \tag{8}$$

$$T_{pref}^{col} = T_{pref} \times I_{bd} \tag{9}$$

Scale Down decreases the number of total fetching threads from n_0 to n_1. While using less fetching threads can diminish the impact of prefetching on the HPC application due to an expected smaller interference factor (I_{hpc}), it can also lead to a significant poor I/O performance of the Big Data application due to the increased prefetching time (T_{pref}). When n_1 is smaller than the number of burst buffer nodes, we use n_1 nodes and one thread per node for the prefetching. Similarly to scale up, we calculate T_{pref}, I_{hpc} and I_{bd} values for n_1 threads, and calculate the optimized I/O completion time for both applications as follows.

$$T_{io}^{col} = T_{io} \times I_{hpc} \tag{10}$$

$$T_{pref}^{col} = T_{pref} + T_{io} \times I_{hpc} \times (1 - \frac{1}{I_{bd}}) \tag{11}$$

5.2. Decision Making

Given the optimization actions, we iteratively decide the optimal action for prefetching at the beginning of each Big Data iteration, in order to reduce the cost of Big Data applications while satisfying the performance constraint of HPC applications.

When making prefetching decisions, we estimate the cost of Big Data application and the I/O time of HPC application for each action. For example, when estimating the cost of the partial delay action, we first calculate the completion time of both HPC and Big Data applications with different t_d values using Equation 6 and 7. We then select the t_d that leads to the minimum cost of Big Data while satisfying the deadline constraint of HPC for the delay action and return its result as the result of partial delay. If we can find an action with a cost equal to zero for the Big Data application and an I/O time less or equal to the deadline constraint for the HPC application, we simply adopt this action without further checking the other actions. If none of the actions achieves zero cost for the Big Data application, we choose the action with the minimum cost while satisfying the deadline constraint of the HPC application.

6. Experimental Methodology

We evaluate our burst buffer design using both real system experiments and simulations. We further describe the experimental setup for both the real system and simulations.

6.1. Real System Setup

The empirical experiments were carried out on the Grid'5000 testbed [34]. We used the Rennes site; more specifically we employed nodes belonging to the *parasilo* and *paravance* clusters. The nodes in these clusters are outfitted with two 8-core Intel Xeon 2.4 GHz CPUs and 128 GB of RAM. We leveraged the 10 Gbps Ethernet network that connects all nodes of these two clusters. Grid'5000 allows us to create an isolated environment in order to have full control over the experiments and obtained results.

Application	Compute Time(s)	I/O Volume(GB)
Turbulence1 (T1)	70	128.2
Turbulence2 (T2)	1.2	235.8
AstroPhysics (AP)	240	423.4

Table 2. HPC application characteristics for the simulation [37].

6.1.1. System deployment

We used Spark version 1.6.1 to execute Big Data applications. We configured and deployed a Spark cluster using 33 nodes on the *paravance* cluster. One node consists of the Spark master, leaving 32 nodes to serve as slaves of Spark. We allocated 8 GB per node for Spark instance and set the Spark's default parallelism parameter (spark.default.parallelism) to 256 which refers to the number of RDD partitions. Each Spark slave has 16 map tasks thus Spark cluster can execute 512 map tasks in one iteration.

In our burst buffer design, we use low-latency storage devices as a storage space. To emulate this, we used Alluxio version 1.3.1, which exposes RAMDisk of the burst buffer nodes to the compute nodes as an in-memory file system. We configured and deployed 8 burst buffer nodes on the same cluster (*paravance*) as compute nodes to emulate that burst buffer nodes are closer to compute nodes compared to the parallel file system. Each burst buffer node provides approximately 32 GB of storage capacity.

The OrangeFS file system (a branch of PVFS2 [23]) version 2.8.3 was deployed on 6 nodes of the *parasilo* cluster to serve I/O requests from both Big Data and HPC applications. We select 6 PVFS nodes using the same setting as existing studies [12].

6.1.2. Workloads

We selected two representative Big Data workloads including Sort and Wordcount, which are parts of the HiBench [35], a Big Data benchmarking suite. Wordcount is a map-heavy workload with a light reduce phase. On the other hand, Sort produces a large amount of intermediate data which leads to a heavy shuffling, therefore representing reduce-heavy workloads. For both workloads, we use 160GB of input data generated by RandomTextWriter of HiBench suite. Both workloads are executed with five iterations, where each iteration processes input data size of 32GB.

For HPC workloads, we use IOR [36] which is a popular I/O benchmarking tool for HPC systems. We choose this workload for controllable interference between Big Data and HPC to evaluate the effectiveness of Eley. For each set of experiment, we run IOR side by side with the Big Data workloads using the same number of iterations. In each iteration, we use IOR to emulate the I/O requests of HPC applications. Between each request, IOR processes sleep for a given time S to emulate the computation time of HPC applications. We set S to be equal to the computation time (T_{cpu}) of Big Data applications.

6.2. Simulation Setup

We design a deterministic event-driven simulator to simulate the execution of HPC and Big Data applications in a system with the same configuration as our real deployment. We choose three write-intensive scientific applications executed on Intrepid in 2011 [37] as the HPC workloads. Table 2 shows the characteristics of those applications. For Big Data applications, we use the traces collected from a OpenCloud cluster in Carnegie Mellon University [18]. Table 3 presents the characteristics of the eight applications used in our simulation, which are scientific applications that are most likely to be co-located with HPC applications in HPC systems.

We run the HPC applications for 10 iterations in the simulation. For Big Data applications, the number of iterations (waves) depends on the total input size and the cluster capacity. We define the cluster capacity as 50GB in our simulation. Thus we can execute 800 map tasks in one iteration. From our real system evaluation, we observe that the bandwidth of the parallel file system is 4GB/s. Thus, we use this value in our simulation to limit the maximum I/O bandwidth acquired by Big Data and HPC applications to the parallel file system. Based on the model defined in Section 4, we generate I/O requests (i.e., interference factor, data copying/fetching times) by using the profiling values obtained during our real system experiments.

Application	Compute Time(s)	Input Data Size(GB)
1	20	195
2	24	860
3	12	600
4	236	355
5	29	662
6	43	799
7	41.5	1300
8	9.5	13000

Table 3. Big Data application characteristics for the simulation.

6.3. Comparisons

For both real system and simulator-based experiments, we adopt the following state-of-the-art burst buffer solutions as comparisons to *Eley*.

- *NaiveBB.* As in the existing burst buffer solutions [14, 15, 16, 17], *NaiveBB* uses burst buffer only for storing the intermediate data of Big Data applications. While it is already shown that this approach can improve the performance of Big Data applications, different from *Eley*, it does not consider the QoS requirements of HPC applications and the I/O latency problem in the input phase.
- *Eley-NoAction.* This approach performs naive prefetching to copy the input data of Big Data from parallel file system to burst buffer nodes. Different from *Eley*, this approach is not aware of the QoS requirement of HPC applications and does not use any optimization action for the prefetching.

All solutions are evaluated using the same burst buffer storage space (i.e., RAMDisk) for fair comparison. We evaluate the completion time of both Big Data and HPC applications obtained by the three compared solutions to demonstrate the effectiveness of Eley. For Big Data applications, we only report the completion time of the map phase which is the main difference between Eley and the compared solutions. We also compare the resulted interference factors of both applications after applying the three compared burst buffer solutions, which indicate the slowdown (defined as $I - 1$) of the applications due to co-location and I/O contention.

7. Evaluation Results

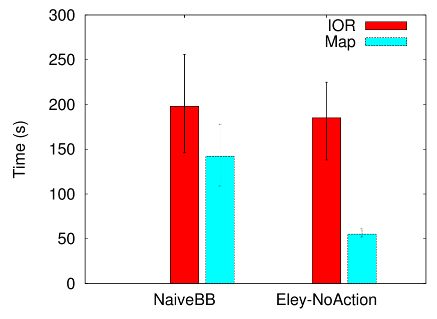
In this section, we present the evaluation results obtained from both real system deployment and simulations.

7.1. Real System Experiments

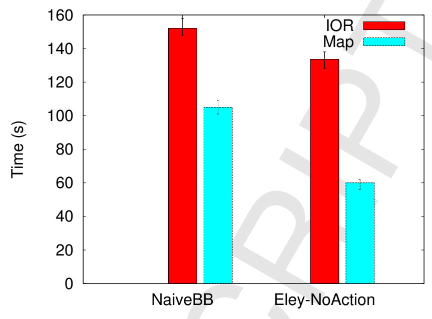
We perform two sets of real system experiments. First, we compare *Eley-NoAction* with *NaiveBB*, the results of which demonstrate that using burst buffer for prefetching the input data of Big Data applications can greatly improve the performance of Big Data while reducing the interference imposed on co-located HPC applications. Second, we compare *Eley* with *Eley-NoAction* and find that our interference-aware model and optimization actions can successfully guarantee the QoS requirement of HPC applications without sacrificing too much of the Big Data performance. Each set of experiments has been executed five times and we report the average values.

7.1.1. Comparing NaiveBB with Eley-NoAction

In this experiment, we execute IOR on a cluster of 32 nodes where one process per node issues a 4GB write request with chunk size of 1MB. Figure 3 demonstrates that prefetching reduces the map time of Sort and Wordcount by 61% and 43%, respectively, compared to *NaiveBB* which reads input data from the parallel file system directly. It can be observed that the I/O time of the IOR workloads are also reduced when using prefetching for Big Data applications, by 7% and 12% when co-locating with Sort and Wordcount, respectively. This is mainly due to the fact that prefetching using burst buffer aggregates the I/O requests sent from a large number of Spark nodes to a small set of burst buffer

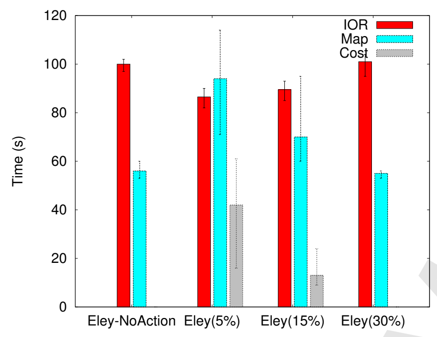


(a) Sort

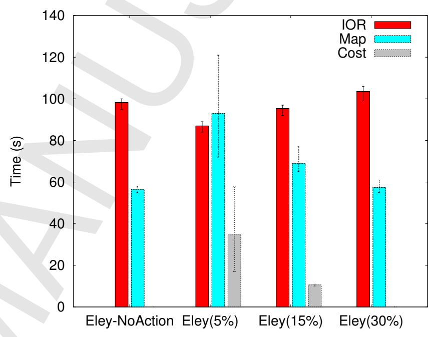


(b) Wordcount

Figure 3. Performance comparison between *Eley-NoAction* and *NaiveBB*.



(a) Sort



(b) Wordcount

Figure 4. Performance comparison between *Eley* and *Eley-NoAction*.

nodes and hence reduces the interference level imposed on IOR workloads. For instance, while 32 compute nodes request the I/O from PVFS in *NaiveBB*, only 8 burst buffer nodes perform prefetching with *Eley-NoAction*. This is inline with the existing studies in the literature [38, 39] which demonstrate that aggregation of I/O requests help to reduce the I/O interference. However, this small improvement might not be enough to meet the QoS requirements of HPC applications. Therefore, we next discuss the effectiveness of our interference-aware prefetching mechanism, *Eley*.

7.1.2. Comparing Eley-NoAction with Eley

	Iter 1	**Iter 2**	**Iter 3**	**Iter 4**	**Iter 5**
IOR Time (s) (*Eley-NoAction*)	17.8	22.4	18.5	20.9	23
IOR Time (s) (*Eley*)	17.8	17.6	17.1	16.3	17.3
IOR Time (s) (Estimated)	17.5	17.9	17	17.9	17
Selected Action	No Action	Partial Delay	Scale Up	Partial Delay	Full Delay
Cost of Sort (s)	0	13	0	9	15

Table 4. Applied actions on prefetching during one run of the Sort application and their respective cost values.

In this experiment, we execute IOR on a cluster of 8 nodes, where one process per node issues a 8 GB write request with chunk size of 1MB. We set three different deadline requirements for IOR workloads, which are 5%, 15% and 30% longer than the completion time of the workloads when executed individually.

Application	NaiveBB (s)	Eley-NoAction (s)
1	151.7	136.6
2	771.3	526
3	379	358.6
4	2018	1925
5	674	487.3
6	979.7	770
7	1518.7	1174
8	6239	6686.7

Table 5. Average map times of Big Data applications with NaiveBB and Eley-NoAction approaches when running together with the three HPC applications.

Figure 4 shows that *Eley* is aware of the QoS requirement of HPC applications and can control the level of interference imposed by Big Data to HPC applications accordingly. We take a closer look at the optimization actions used in each iteration to evaluate the effectiveness of *Eley* on guaranteeing the QoS of HPC applications. Table 4 shows the applied actions during one run of the Sort application with a QoS of 5% (i.e., 17.9 s). We can observe that with the optimization actions, *Eley* is always able to guarantee the QoS requirement of HPC application with low cost of the Big Data application. For example, in iteration 3, *Eley* is able to find an action (i.e., Scale Up) which leads to zero cost for Sort application. The selection of actions depends on both the characteristics of Big Data and HPC applications. For instance, the Scale Down action is not used during the entire execution of Sort application. We believe this is mainly due to the relatively short computation time of Sort compared to its prefetching time (e.g., 12 seconds and 16 seconds per iteration, respectively). Table 4 also shows the estimated IOR times by the model we defined in Section 4 depending on the applied action. The average relative difference between the estimated IOR times and observed IOR times with Eley is 3%. This demonstrates that our model is able to help Eley in guaranteeing the QoS requirements of HPC applications with accurate estimations.

Another observation from Figure 4 is that there is a clear trade-off between the tightness of HPC deadline and the performance of Big Data applications. For example, with the 5% QoS requirement, the cost of Sort and Wordcount applications are 60% and 75%, respectively. The cost reduces to 23% and 21% for Sort and Wordcount, respectively, when the QoS requirement is relaxed to 15%. With a small cost, the performance of the Big Data applications are improved. The cost is also depending on several factors, such as the characteristics of HPC and Big Data applications and the HPC platform on which the applications are executing. In the next subsection, we run a wide-range of applications with our simulator to better study this trade-off.

7.2. Simulation Results

Similarly to the real system experiments, we perform two sets of comparisons using the three burst buffer solutions.

7.2.1. Comparing NaiveBB with Eley-NoAction

Table 5 shows the average map time of the eight Big Data applications when running together with the three HPC applications. We can observe that *Eley-NoAction* can reduce the map time of Big Data applications (except application 8) by up to 32% compared to *NaiveBB*. *Eley-NoAction* achieves performance improvement for Big Data applications mainly by overlapping the input data reading time and map computation time. However, as application 8 has a very small computation time compared to the required prefetching time, it cannot benefit much from the *Eley-NoAction* solution. The prefetching time for the application is 25 seconds while reading the input data directly from the parallel file system is only 12 seconds. This is mainly due to the different number of nodes sending I/O requests in *naiveBB* and *Eley-NoAction*. On the other hand, we find that the improvement on the performance of Big Data applications brought by prefetching is small when the application has a very large computation time. For example, the map computation time of application 4 is 236 seconds per iteration, which is 10 times larger than its prefetching time. Thus, the performance improvement for this application when using *Eley-NoAction* is only 5%.

We evaluate the slowdown ($I_{hpc} - 1$) of the three HPC applications when running together with the eight Big Data applications using *NaiveBB* and *Eley-NoAction* approaches as shown in Figure 5. We have the following observations.

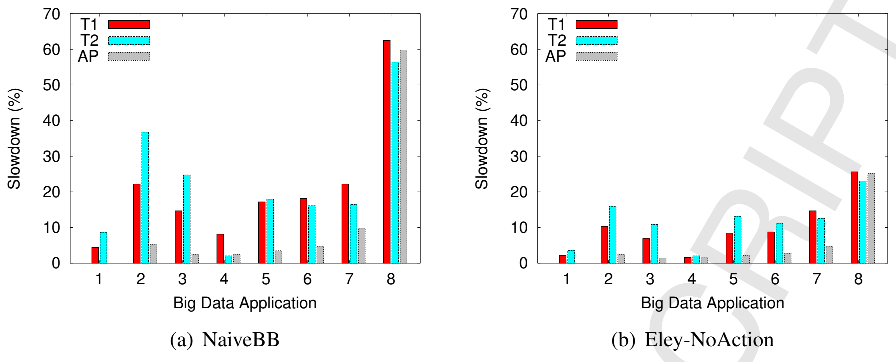


Figure 5. Slowdowns observed for the three HPC applications with NaiveBB and Eley-NoAction.

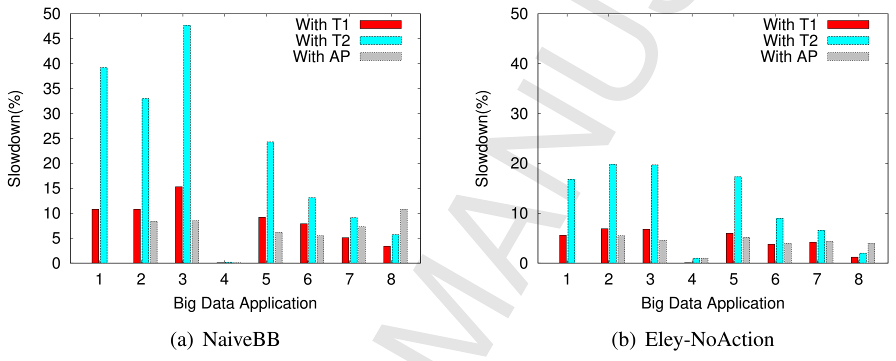


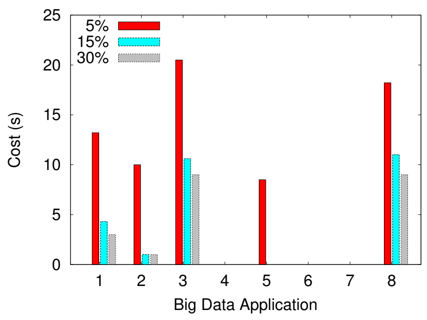
Figure 6. Slowdowns observed for the Big Data applications with NaiveBB and Eley-NoAction.

First, when using *NaiveBB*, most of the HPC applications suffer a lot from the I/O interference. For example, T1 has the highest slowdown of 63% when co-locating with Big Data application 8. Second, the slowdown of the HPC applications are greatly reduced by *Eley-NoAction* compared to *NaiveBB*. For example, the slowdown of T1 is reduced to 26% when running with application 8 using *Eley-NoAction*. Third, the reduction of slowdown varies from application to application. For example, T1 achieves the highest reduction of slowdown with 54% on average while the reduction of slowdown for T2 and AP are both 39% on average. We also observe that application 8 is imposing the highest slowdown to all HPC applications due to its short computation time and frequent I/O requests.

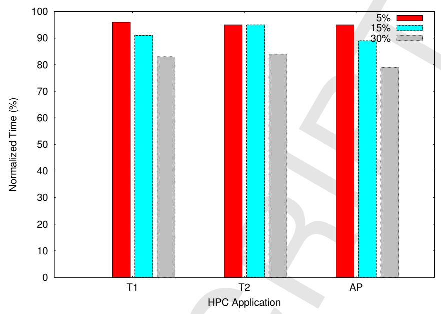
We further study the slowdown ($I_{bd} - 1$) of the Big Data applications using the two burst buffer solutions as shown in Figure 6. We have similar observations as those for HPC applications. First, the *NaiveBB* solution introduces high slowdown to Big Data applications, e.g., 48% for application 3 when co-locating with T2. Second, *Eley-NoAction* can greatly reduce the slowdown of Big Data applications compared to *NaiveBB*. For example, the reduction of slowdown can reach up to 58% on average for application 3. Third, T2 is the HPC application which imposes the highest slowdown to the Big Data applications. This is again due to the fact that T2 has a small computation time and thus its I/O operations can interfere frequently with the I/O operations of Big Data applications.

	Iter 1	**Iter 2**	**Iter 3**	**Iter 4**	**Iter 5**
T2 I/O Time (s) (*Eley-NoAction*)	70	65	68	65	69
T2 I/O Time (s) (*Eley*)	59	59	62	59	59
Selected Action	Scale Down	Full Delay	Partial Delay	Scale Down	Scale Down
Cost of Application 6 (s)	6	0	0	6	6

Table 6. Applied actions on prefetching during the first 5 iterations of the application 6 and T2 and their respective cost values.



(a) Average cost for the Big Data applications



(b) Normalized I/O time of HPC applications respect to QoS

Figure 7. Performance of Big Data and HPC applications with *Eley* approach.

7.2.2. Comparing Eley-NoAction with Eley

We study the effectiveness of *Eley* on satisfying the QoS requirements of different HPC applications with different concurrently running Big Data applications. We set three QoS requirements for HPC applications, which are 5%, 15% and 30% longer than the completion time of the applications when executed individually. Figure 7(a) shows the average cost of the Big Data applications and Figure 7(b) shows the average performance of the HPC applications normalized to the different QoS requirements when optimized with *Eley*.

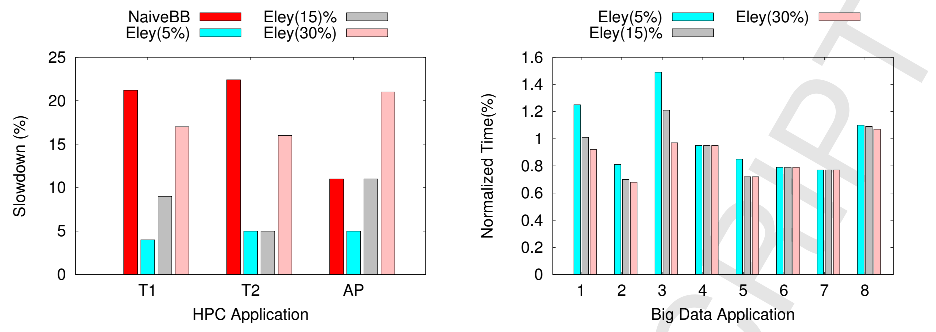
Figure 7(a) shows that the cost of Big Data applications decreases with more relaxed QoS requirements. For Big Data applications with longer prefetching time than computation time, the initial cost before applying any action is high. For example, application 3 and 8 have the highest cost values due to their relatively short computation times. Figure 7(b) demonstrates that *Eley* is able to satisfy the QoS requirement for the three HPC applications under all settings. This shows that our interference-aware mechanism is effective in guaranteeing the QoS for HPC applications. Take the execution of Big Data application 6 with T2 application for example. Table 6 shows the optimization actions selected when the QoS requirement of T2 is set to 5% (i.e., 62 seconds). For all the iterations, *Eley* is able to satisfy the QoS requirement of T2. In two of the iterations, we are able to find actions leading to zero cost of the Big Data application. Different from the real system experiment, we find that the Scale Down action is selected at the last two iterations. This is mainly because application 6 has almost twice longer computation time compared to its prefetching time (recall that Sort has shorter computation time than its prefetching time and thus never uses Scale Down).

7.2.3. Trade-off between QoS of HPC and Performance of Big Data

To further investigate the trade-off between QoS requirements of HPC applications and the performance of Big Data applications, we compared *Eley* with *NaiveBB* under three different QoS requirements: 5%, 15% and 30%. Figure 8(a) shows again that *NaiveBB* degrades the performance of all the three HPC applications due to the I/O interference. On the other hand, *Eley* is able to control this performance degradation depending on the QoS requirements of the HPC applications.

When we look at the performance of Big Data applications in Figure 8(b), we observe that *Eley* can improve the performance of most of these applications even with 5% QoS requirements for HPC applications. We can also see that this performance improvement brought by *Eley* depends on the characteristics of Big Data applications. For instance, when the application has a very small computation time (for the applications 3 and 8), Eley would not be able to fully hide the cost of the taken optimal prefetching action for guarenteeing the QoS of HPC applications.

Figure 8(b) also shows that we can get better performance improvements with *Eley* compared to *NaiveBB* with a bit relaxed QoS requirements for HPC applications. For example, the reduction in the map time of application 2 increases from 19% to 30% when the QoS requirement is relaxed from 5% to 15%. On the other hand, for some applications (e.g., application 4,6 and 7) we do not observe this trade-off. This is mainly due to their relatively larger computation times which give *Eley* more flexibility – even with tighter QoS requirements – in terms of finding optimal prefetching decisions without hurting the performance of these applications. Similar to the observation found in Figure 7(a), *Eley* does not show much benefit compared to *NaiveBB* for application 3 and 8 due to their short computation times.



(a) Average slowdowns observed for HPC applications with NaiveBB and Eley.

(b) Average map times of Big Data applications with Eley under different QoS requirements. Map times are normalized with respect to the NaiveBB.

Figure 8. Performance comparison of Big Data and HPC applications with *NaiveBB* and *Eley* approaches.

8. Conclusion

Big Data analytics frameworks (e.g., Hadoop and Spark) have been increasingly used by many companies and research labs to facilitate large-scale data analysis. However, with the growing needs of users and size of data, commodity-based infrastructure (mostly commonly-used for now) will strain under the heavy weight of Big Data. On the other hand, HPC systems offer a rich set of opportunities for Big Data processing. As first steps toward Big Data processing on HPC systems, several research efforts have been devoted to deploy burst buffer solutions in HPC systems to address the performance challenges of Big Data applications. However, most of those efforts focus on only using burst buffer for intermediate data of Big Data applications, which still suffer from the high I/O latency between compute nodes and parallel file system for Big Data applications with huge input data sizes.

In this paper, we propose *Eley*, a burst buffer solution which takes into consideration both the input data and intermediate data of Big Data applications in HPC systems. Our goal is to accelerate the performance of Big Data applications while guaranteeing the QoS of HPC applications. To achieve this goal, *Eley* is composed of a prefetcher, which prefetches the input data of Big Data applications before the execution of each iteration. By prefetching, we are able to overlap the I/O and computation time to improve the performance of Big Data applications. However, data prefetching may introduce huge I/O interference to the HPC applications and thus end up with a degraded and unpredictable performance for HPC applications. To this end, we design *Eley* to be interference-aware. Specifically, we equip the prefetcher with five optimization actions and propose an interference-aware decision maker to iteratively choose the best action to optimize the prefetching while guaranteeing the pre-defined QoS requirement of HPC applications. We evaluate the effectiveness of *Eley* with both real system experiments and simulations. With 5% QoS requirement of the HPC application, *Eley* reduces the execution time of Big Data applications by up to 30% compared to the Naive burst buffer solution (denoted as *NaiveBB*) [14, 15, 16, 17] while guaranteeing the QoS requirement. On the other hand, the *NaiveBB* violates the QoS requirement by up to 58%.

As future work, we are going to develop a burst buffer system which can alleviate the latency problem in all I/O phases. To this end, we plan to use burst buffer for the output phase and adopt interference-aware strategies to synchronize the output data to the parallel file system. An interesting direction to explore is how different traffic and arrival patterns – described in [40] – may impact the performance of HPC applications. Moreover, we plan to investigate the feasability of employing Eley to improve the performance of Big Data applications under failures [41].

Acknowledgment

This work is supported by the ANR KerStream project (ANR-16-CE25-0014-01), the Shenzhen Science and Technology Foundation (Grant No.JCYJ20150529164656096, JCYJ20170302153955969), Shenzhen University Startup Grant (No.2018062) and Guangdong Pre-national Project (2014GKXM054). The experiments presented in this paper were carried out using the Grid'5000/ALADDIN-G5K experimental testbed, an initiative from the French Ministry of

Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see `http://www.grid5000.fr/` for details).

References

[1] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] H. Jin, S. Ibrahim, L. Qi, H. Cao, S. Wu, and X. Shi, “The MapReduce Programming Model and Implementations,” *Cloud Computing: Principles and Paradigms.*, pp. 373–390, 2011.

[3] “CMU OpenCloud Hadoop Cluster,” http://ftp.pdl.cmu.edu/pub/datasets/hla/dataset.html, Accessed on Feb 2017.

[4] NICS, “Kraken Cray XT5 system, http://www.nics.tennessee.edu/computing-resources/kraken.”

[5] NCSA, “BlueWaters project, http://www.ncsa.illinois.edu/BlueWaters/.”

[6] “Big Data and Extreme-scale Computing (BDEC) Workshop, http://www.exascale.org/bdec/.”

[7] G. Fox, J. Qiu, S. Jha, S. Ekanayake, and S. Kamburugamuve, “Big data, simulations and hpc convergence,” Technical Report· January 2016, DOI: 10.13140/RG. 2.1, Tech. Rep., 2016.

[8] Y. Guo, W. Bland, P. Balaji, and X. Zhou, “Fault tolerant mapreduce-mpi for hpc clusters,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 34.

[9] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, “The power of choice in data-aware cluster scheduling,” in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2014, pp. 301–316.

[10] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, “Scarlett: coping with skewed content popularity in mapreduce clusters,” in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 287–300.

[11] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, “Maestro: Replica-aware map scheduling for mapreduce,” in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 435–442.

[12] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, “On the root causes of cross-application i/o interference in hpc storage systems,” in *IPDPS-International Parallel and Distributed Processing Symposium*, 2016.

[13] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim, “CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination,” in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS ’14)*, Phoenix, AZ, USA, May 2014. [Online]. Available: http://hal.inria.fr/hal-00916091

[14] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda, “High performance design for hdfs with byte-addressability of nvm and rdma,” in *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 2016, p. 8.

[15] N. Chaimov, A. Malony, S. Canon, C. Iancu, K. Z. Ibrahim, and J. Srinivasan, “Scaling spark on hpc systems,” in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2016, pp. 97–110.

[16] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, N. Maruyama, and S. Matsuoka, “A user-level infiniband-based file system and checkpoint strategy for burst buffers,” in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 21–30.

[17] Y. Wang, R. Goldstone, W. Yu, and T. Wang, “Characterization and optimization of memory-resident mapreduce on hpc systems,” in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 799–808.

[18] “Hadoop Workload Analysis,” http://www.pdl.cmu.edu/HLA/index.shtml, Accessed on Jan 2017.

[19] “The Apache Hadoop Project,” http://www.hadoop.org.

[20] W. Tantisiriroj, S. W. Son, S. Patil, S. J. Lang, G. Gibson, and R. B. Ross, “On the duality of data-intensive file system design: reconciling hdfs and pvfs,” in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 67.

[21] “Using Lustre with Apache Hadoop,” http://wiki.lustre.org/index.php/Running_Hadoop_with_Lustre, Accessed on Feb 2017.

[22] P. Xuan, J. Denton, P. K. Srimani, R. Ge, and F. Luo, “Big data analytics on traditional hpc infrastructure using two-level storage,” in *Proceedings of the 2015 International Workshop on Data-Intensive Scalable Computing Systems*. ACM, 2015, p. 4.

[23] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, “PVFS: a Parallel File System for Linux Clusters,” in *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*. Berkeley, CA, USA: USENIX Association, 2000.

[24] “HDFS. The Hadoop Distributed File System,” http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html.

[25] Z. Fadika, E. Dede, M. Govindaraju, and L. Ramakrishnan, “Mariane: Mapreduce implementation adapted for hpc environments,” in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE Computer Society, 2011, pp. 82–89.

[26] I. El-Helw, R. Hofman, and H. E. Bal, “Scaling mapreduce vertically and horizontally,” in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 2014, pp. 525–535.

[27] N. S. Islam, D. Shankar, X. Lu, M. Wasi-Ur-Rahman, and D. K. Panda, “Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store,” in *Parallel Processing (ICPP), 2015 44th International Conference on*. IEEE, 2015, pp. 280–289.

[28] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, “I/O-Aware Batch Scheduling for Petascale Computing Systems,” in *IEEE International Conference on Cluster Computing*, 2015.

[29] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, “Scheduling the I/O of HPC Applications Under Congestion,” in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2015, pp. 1013–1022.

[30] F. Zanon Boito, R. Kassick, P. O. A. Navaux, and Y. Denneulin, “AGIOS: Application-Guided I/O Scheduling for Parallel File Systems,” in *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*. IEEE, 2013, pp. 43–50.

[31] A. Kougkas, M. Dorier, R. Latham, R. Ross, and X.-H. Sun, “Leveraging burst buffer coordination to prevent i/o interference.”

[32] 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.

[33] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross, "Omnisc'io: a grammar-based approach to spatial and temporal i/o patterns prediction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 623–634.
[34] INRIA, "Grid'5000: http://www.grid5000.fr."
[35] "HiBench Big Data microbenchmark suite, https://github.com/intel-hadoop/HiBench."
[36] H. Shan and J. Shalf, "Using IOR to Analyze the I/O Performance for HPC Platforms," in *Cray User Group Conference 2007*, Seattle, WA, USA, 2007.
[37] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2012, pp. 1–11.
[38] M. Dorier, G. Antoniu, F. Cappello, M. Snir, R. Sisneros, O. Yildiz, S. Ibrahim, T. Peterka, and L. Orf, "Damaris: Addressing performance variability in data management for post-petascale simulations," *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 3, p. 15, 2016.
[39] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "DataStager: Scalable Data Staging Services for Petascale Applications," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, ser. HPDC '09. New York, NY, USA: ACM, 2009, pp. 39–48. [Online]. Available: http://doi.acm.org/10.1145/1551609.1551618
[40] W. Miao, G. Min, Y. Wu, H. Wang, and J. Hu, "Performance modelling and analysis of software-defined networking under bursty multimedia traffic," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 5s, pp. 77:1–77:19, Sep. 2016. [Online]. Available: http://doi.acm.org/10.1145/2983637
[41] O. Yildiz, S. Ibrahim, and G. Antoniu, "Enabling fast failure recovery in shared hadoop clusters: Towards failure-aware scheduling," *Future Generation Computer Systems*, vol. 74, pp. 208 – 219, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16300280

Biographies

Orcun Yildiz received his B.Sc. degree in Computer Science from Bogazici University, Turkey and his M.Sc. degree in Distributed Computing from Royal Institute of Technology, Sweden. He is currently a Ph.D. student in INRIA, Rennes-Bretagne Atlantique Research Centre. His research interests include distributed systems, high performance computing and big data management.

Amelie Chi Zhou received her PhD degree in 2016 from School of Computer Engineering, Nanyang Technological University, Singapore. She received her Master's degree in 2011 and Bachelor's degree in 2009 both from Beihang University, China. She is currently a postdoc fellow in Inria-Rennes, France. Her research interests lie in cloud computing, big data processing, distributed systems and resource management.

Shadi Ibrahim is an Inria Research Scientist (CR1) at Inria Rennes Bretagne Atlantique research center. He obtained his Ph.D. in Computer Science from Huazhong University of Science and Technology (HUST) in Wuhan of China in 2011. He has served as a PC member for international conferences in cloud, HPC and parallel and distributed systems (IEEE ICPP, IEEE Cluster, ACM/IEEE CCGrid), and as Program co-chair of ICA3PP 2017, PhD consortium co-chair for the 2014 CloudCom conference and Workshops co-chair for the 2014 ScalCom conference. His current research interests are in cloud computing, Big data management, Data-Intensive computing, virtualization technology, and file and storage systems. His papers are published in prestigious international journals (such as ACM ToPC, IEEE TPDS and FGCS) and proceedings (such as ACM/IEEE SuperComputing, IEEE IPDPS, IEEE ICDCS and IEEE ICPP). He is currently leading the ANR KerStream project and is a member the DISCOVERY Inria Project Lab and JLESC: Joint Laboratory on Extreme-Scale Computing. More: http://people.rennes.inria.fr/Shadi.Ibrahim/