On the Performance of Spark on HPC Systems: Towards a Complete Picture

Orcun Yildiz, Shadi Ibrahim

On the Performance of Spark on HPC Systems: Towards a Complete Picture

Orcun Yildiz[1] and Shadi Ibrahim[2(✉)]

[1] Inria, Univ Rennes, CNRS, IRISA, Rennes, France
[2] Inria, IMT Atlantique, LS2N, Nantes, France
shadi.ibrahim@inria.fr

Abstract. Big Data analytics frameworks (e.g., Apache Hadoop and Apache 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 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 understanding the performance of Big Data applications on these systems. Yet the HPC specific performance considerations have not been fully investigated. In this work, we conduct an experimental campaign to provide a clearer understanding of the performance of Spark, the *de facto* in-memory data processing framework, on HPC systems. We ran Spark using representative Big Data workloads on Grid'5000 testbed to evaluate how the latency, contention and file system's configuration can influence the application performance. We discuss the implications of our findings and draw attention to new ways (e.g., burst buffers) to improve the performance of Spark on HPC systems.

Keywords: HPC · MapReduce · Spark · Parallel file systems
Contention

1 Introduction

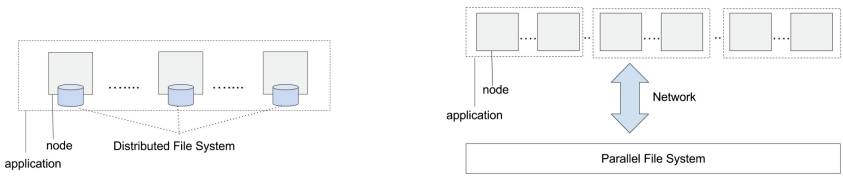
Data is a driving power in almost every aspect of our lives and thus large amounts of data generated everyday. For instance, International Data Research report [6] estimates that the global data volume subject to data analysis will grow by a factor of 50 to reach 5.2 zettabytes in 2025. This huge growth in the data volumes, the deluge of Big Data, results in a big challenge in managing, processing and analyzing these gigantic data volumes.

To benefit from this huge amount of data, different data processing models have emerged [13,20]. Among these models, MapReduce [13,23] has stood out as the most powerful Big Data processing model, in particular for batch processing. MapReduce, and its open-source implementation Hadoop [3], is adopted in both industry and academia due to its simplicity, transparent fault tolerance and scalability. For instance, Yahoo! claimed to have the world's largest Hadoop cluster [7] with more than 100000 CPUs in over 40000 machines running MapReduce jobs.

R. Yokota and W. Wu (Eds.): SCFA 2018, LNCS 10776, pp. 70–89, 2018.
https://doi.org/10.1007/978-3-319-69953-0_5

With the wide adoption of MapReduce in different domains, diverse Big Data applications (e.g., stream data processing, graph processing, analysis of large scale simulation data) have emerged where obtaining timely and accurate responses is a must. This, on the one hand, motivates the introduction of new Big Data analytic frameworks which extend the MapReduce programming model [4,10,11,36]. Such frameworks keep data processing in memory and therefore try to efficiently exploit its high speed. Among these frameworks, Spark [36] has become the de facto framework for in-memory Big Data analytics. Spark is recently used to run diverse set of applications including machine learning, stream processing and etc. For example, Netflix has a Spark cluster of over 8000 machines processing multiple petabytes of data in order to improve the customer experience by providing better recommendations for their streaming services [5]. On the other hand, high performance computing (HPC) systems recently gained a huge interest as a promising platform for performing fast Big Data processing given their high performance nature [1,17]. HPC systems are equipped with low-latency networks and thousands of nodes with many cores and therefore have the potential to perform fast Big Data processing. For instance, PayPal recently shipped its fraud detection software to HPC systems to be able to detect frauds among millions of transactions in a timely manner [26].

However, when introducing Big Data processing to HPC systems, one should be aware of the different architectural designs in current Big Data processing and HPC systems. Big Data processing systems have shared nothing architecture and nodes are equipped with individual disks, thus they can co-locate the data and compute resources on the same machine (i.e., data-centric paradigm). On the other hand, HPC systems employ a shared architecture (e.g., parallel file systems) [19] which results in separation of data resources from the compute nodes (i.e., compute-centric paradigm). Figure 1 illustrates these differences in the design of these two systems. These differences in the design of these two systems introduce two major challenges: Big Data applications will face *high latencies* when performing I/O due to the necessary data transfers between the parallel file system and computation nodes. Moreover, *I/O contention* (i.e., performance degradation observed by any single application/task running in contention with other applications/tasks on the same platform [15,33]) is a well-known problem in HPC systems which often detracts the performance of a single-application



(a) Data-centric paradigm. (b) Compute-centric paradigm.

Fig. 1. The different designs in Big Data and HPC systems.

from the high performance offered by these systems due to their large sizes and shared architecture [16,18,25,33].

In response, several efforts have been conducted to leverage Spark for fast Big Data processing on HPC systems. These works have mainly tried to alleviate the high latency problem by focusing on the *intermediate data* storage (i.e., map output for batch jobs and temporary output produced between stages for iterative jobs) [12,21,28,32,34]. For example, Islam et al. [21] utilized NVRAM as an intermediate storage layer (i.e., burst buffer) between compute nodes and Lustre file system [14]. This brought 24% improvement to the application performance by reducing the latency when reading/writing the intermediate data. However, Big Data applications mostly run in batches and there is a continuous interaction with the parallel file system for reading the input data and writing the output data, thus it is important to study the impact of latency on the performance of Big Data applications by considering the different phases of Big Data applications as input, intermediate and output data. Moreover, none of these efforts considered the contention problem which can contribute to a significant performance degradation by up to 2.5x [33]. Hence, *as we argue in this paper, current efforts and solutions to adopt Spark on HPC systems may fail in practice to achieve the desired performance and this may hinder such adoption.*

Our Contributions. In an effort to complement existing efforts on understanding the performance of Big Data applications on HPC systems, in this paper, we perform an experimental study characterizing the performance of Spark [36] on HPC systems. We use representative Big Data workloads on the Grid'5000 [22] testbed to evaluate how the *latency*, *contention*, and *file system's configuration* impact the application performance. We make the following contributions:

- *A quantitative analysis of the impact of latency on the application performance.* We find that resulting latency during the data movement between compute nodes and parallel file system can degrade the application performance seriously. Specifically, we show evidence that the high latency of reading the input data and writing the output data to the parallel virtual file system (PVFS) [27] have higher impact on performance degradation compared to the intermediate data.
- *The role of contention on the application performance.* Our results show that contention can result in severe performance penalties for Big Data applications on HPC systems due to employing a shared storage system.
- *An analysis of the impact of the file system specific properties on the application performance.* Similar to [12] which shows that metadata operations in Lustre file system [14] create a bottleneck for Spark applications, we demonstrated that synchronization feature of PVFS [27], which can be necessary for providing resilience, can reduce the application performance dramatically by 14x.
- *Towards an efficient adoption of Spark on HPC systems.* We discuss the implications of our findings and draw attention to new ways (e.g., burst buffers) to improve the performance of Spark on HPC systems.

The rest of the paper is organized as follows. Section 2 describes an overview of our methodology and Sect. 3 presents different sets of experiments highlighting the possible performance bottlenecks for Big Data applications on HPC systems. We discuss the implications of our findings to the new ways (i.e., burst buffers) to improve the performance of Big Data applications on HPC systems in Sect. 4. In Sect. 5 we present related work. Finally, we conclude the paper and propose our future work in Sect. 6.

2 Methodology

We conducted a series of experiments in order to assess the impact of the potential issues regarding HPC systems (i.e., latency, contention, file system's configuration) on the performance of Big Data applications. We further describe the experimental environment: the platform, deployment setup, and Big Data workloads.

2.1 Platform Description

The experiments were carried out on the Grid'5000 testbed. 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.

2.2 Spark Deployment

We used Spark version 1.6.1 working with Hadoop distributed file systems (HDFS) version 1.2. We configured and deployed a Spark cluster using 51 nodes on the *paravance* cluster. One node consists of the Spark master and the HDFS NameNode, leaving 50 nodes to serve as both slaves of Spark and DataNodes. We used the default value (number of available cores on the node) for the number of cores to use per each node. Therefore, the Spark cluster can allocate up to 800 tasks. We allocated 24 GB per node for the Spark instance and set Spark's default parallelism parameter (spark.default.parallelism) to 800 which refers to the number of RDD partitions (i.e., number of reducers for batch jobs). At the level of HDFS, we used a chunk size of 32 MB and set a replication factor of 2 for the input and output data.

The OrangeFS file system (a branch of PVFS2 [27]) version 2.8.3 was deployed on 12 nodes of the *parasilo* cluster. We set the stripe size which defines the data distribution policy in PVFS (i.e., analogous to block size in HDFS) to 32 MB in order to have a fair comparison with HDFS. Unless otherwise specified, we disabled the synchronization for PVFS (Sync OFF) which indicates that the incoming data can stay in kernel-provided buffers. We opted for Sync OFF configuration since Spark is also using the memory as a first storage level with HDFS.

2.3 Workloads

We selected three representative Big Data workloads including Sort, Wordcount and PageRank which are part of HiBench [2], Big Data benchmarking suite.

Wordcount is a map-heavy workload which counts the number of occurrences of each word in a data set. The map function splits the input data set into words and produces the intermediate data for the reduce function as a key, value pair with word being the key and 1 as the value to indicate the occurrence of the word. The reduce function sums up these intermediate results and outputs the final word counts. Wordcount has a light reduce phase due to the small amount of the intermediate data.

Sort is a reduce-heavy workload with a large amount of intermediate data. This workload sorts the data set and both map and reduce functions are simple functions which take the input and produce its sorted version based on the key. This workload has a heavy shuffling in the reduce phase due to the large amount of intermediate data it produces.

PageRank is a graph algorithm which ranks elements according to the number of links. This workload updates these rank values in multiple iterations until they converge and therefore it represents the iterative set of applications.

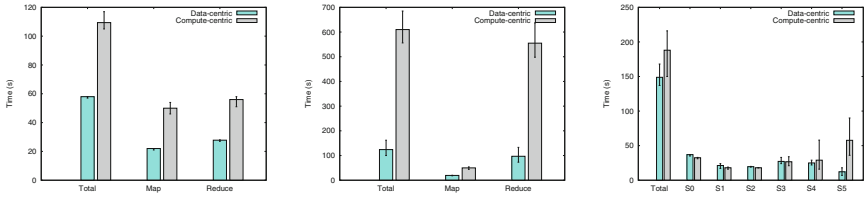
For Sort and Wordcount workloads, we used 200 GB input data set generated with RandomTextWriter in HiBench suite. For the PageRank workload, we also used HiBench suite which uses the data generated from Web data with 25 million edges as an input data set.

3 Experimental Results

In this section, we provide a detailed analysis of the experimental results we obtained which highlights the implications of the potential performance bottlenecks for Big Data applications on HPC systems.

3.1 How Does Latency Affect the Application Performance?

First, we try to understand the impact of the data location on the application performance. While storage resources are co-located with Spark tasks under the data-centric paradigm (i.e., when using Spark with HDFS), Spark tasks need to communicate with the parallel file system either to fetch the input data or to write the output data under the compute-centric paradigm (i.e., when Spark is using PVFS as the storage space). This remote data access results in a higher latency compared to the data-centric paradigm which leverages data locality (i.e., executing tasks on the machines where the input data resides). Figure 2 shows how latency can affect the application performance. Note that, intermediate data is stored locally on the aforementioned settings for Spark in order to focus on the latency resulting from reading the input data in map phase. We explore the intermediate data storage separately in the next subsection.



(a) Wordcount. (b) Sort. (c) PageRank.

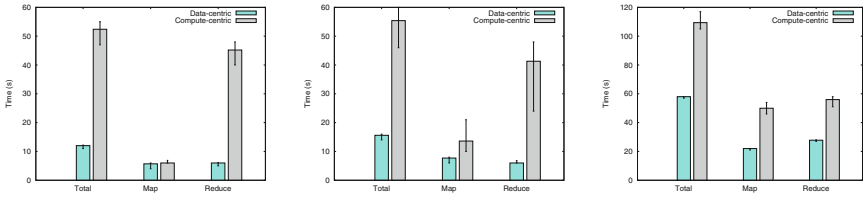
Fig. 2. Performance of Big Data workloads on Spark under data-centric and compute-centric paradigms.

Figure 2(a) displays the execution time of the Wordcount workload for both paradigms with a performance in map and reduce phases. Overall, Wordcount performs 1.9x worse under the compute-centric paradigm compared to the data-centric one. When we look at the performance in each phase, we observe that the performance degradation contributed by the map phase (2.3x) is higher compared to the reduce phase. This stems from the fact that Wordcount has a light reduce phase and generates only a small amount of output data.

Similarly, in Fig. 2(b) we observe that the data-centric configuration outperforms the compute-centric one by 4.9x for the Sort workload. In contrast to Wordcount, the reduce phase is the major contributor to the performance degradation. For the Sort workload, the amount of the output data is equal to the input data thus it suffers from a higher latency in the reduce phase as data is written to the parallel file system. As a result, having a higher latency on both input and output phases led to higher performance degradation for the compute-centric paradigm.

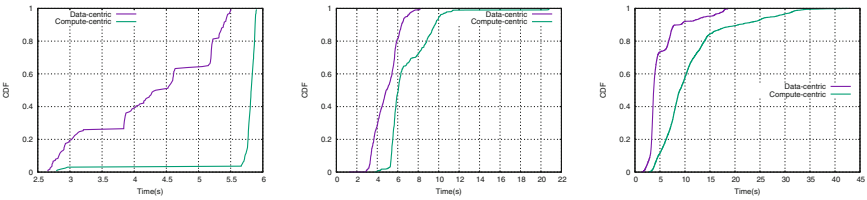
Lastly, we ran the PageRank workload in both settings for Spark and Fig. 2(c) shows the results. Here, performance degradation with the compute-centric paradigm is only 26%. The reason behind this is that I/O phases of the PageRank workload (i.e., Stage 0 and Stage 5 (denoted as S0 and S5)) accounts for a small fraction of PageRank execution time and Spark computes the iterations (i.e., Stage 1, 2, 3 and 4) locally.

The Impact of the Input Data Sizes. We also investigated the impact of the input data size on the application performance. To do so, we ran the Wordcount workload with different input sizes as 2 GB, 20 GB and 200 GB. Figure 3 displays the performance of the Wordcount workload in each phase for data and compute-centric paradigms. Overall, we observe that the impact of I/O latency is only visible in the map phase for the compute-centric paradigm with increasing input sizes: there is a performance degradation for the map phase by 1.2x, 1.8x and 2.3x with 2 GB, 20 GB and 200 GB input sizes, respectively. This is mainly due to the fact that Wordcount is a map-heavy workload which generates a small amount of output data and therefore reduce phase results do not vary significantly with respect to different data sizes. To further investigate



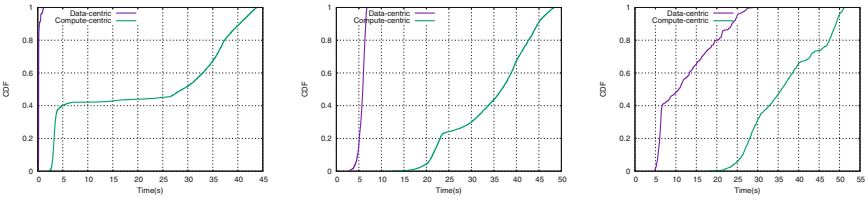
(a) 2 GB. (b) 20 GB. (c) 200 GB.

Fig. 3. Performance of the Wordcount workload with different input sizes.



(a) 2 GB. (b) 20 GB. (c) 200 GB.

Fig. 4. CDFs of running times of map tasks in the Wordcount workload.



(a) 2 GB. (b) 20 GB. (c) 200 GB.

Fig. 5. CDFs of running times of reduce tasks in the Wordcount workload.

these different behaviors in map and reduce phases, we display the CDF of map and reduce task durations in Figs. 4 and 5.

Interestingly, Fig. 4(a) shows that some map task durations are smaller for the compute-centric paradigm compared to the data-centric one. This is due to the fact that Spark employs delay scheduling [35] to increase the chances of a map task to be launched locally for the data-centric paradigm. This delay while launching the map tasks, which results in a performance degradation for the jobs with small input data sizes, is due to the default Spark configuration for the maximum waiting time (i.e., 3 s) in scheduling the map tasks. This is only valid for the data-centric paradigm since there is no data locality objective when scheduling the tasks in the compute-centric paradigm where all the machines have an equivalent distance to the parallel file system. On the other hand, we observe an increase in the map task durations with larger input sizes for the compute-centric paradigm. This results from the higher latency while fetching the input data from parallel file system with larger input sizes.

Another interesting trend we observe is that the maximum map task duration also increases with the increasing data sizes, especially with 200 GB input data size in Fig. 4(c). We believe that this behavior is due to the higher contention with the increased number of concurrent map tasks. It is important to note that there are 33, 594 and 800 concurrent map tasks with 2 GB, 20 GB and 200 GB input sizes. Moreover, we see that this increase is much higher with the compute-centric paradigm which can highlight the severity of the contention problem for this paradigm. We will further explain the impact of the contention on the application performance in Sect. 3.2.

In Fig. 5, we observe a similar trend for the reduce task durations for the compute-centric paradigm. With larger data sizes, we observe an increase in those durations too. This again stems from an increased amount of the remote data transfer while writing the reducer outputs to the parallel file system. Moreover, we discover that there is a high performance variability in the reduce phase and the maximum task duration is quite high even with 2 GB data size. This is due to the static Spark configuration which employs 800 reducers regardless of the input data size. These high number of reducers overload the parallel file system and results in this performance variability. Hence, we do not see the impact of latency in Fig. 3 for the reduce phase. However, when the output data size is large enough as shown for the Sort workload in the previous experiment (Fig. 2(b)), the impact of the I/O latency is quite clear as it results in a significant performance degradation.

For the data-centric paradigm, this time we see that reduce task durations are inlined with the data sizes, different from the map phase. While for the map phase there is an increase in the maximum task duration due to the increased number of concurrent map tasks, for the reduce phase the number of reduce tasks is fixed and the increase in the reduce task durations is mainly due to the increased amount of reducer output with larger input sizes.

Intermediate Data Storage. In Big Data processing systems, intermediate data are typically stored locally. However, nodes in some of the HPC systems may not have individual disks attached to themselves. This gives rise to the question of how to store the intermediate data when running Big Data applications on HPC systems. As a naive solution, we employed PVFS also for storing the intermediate data as well as storage space for input and output data like in the experiments so far. We ran the Sort workload with PVFS since it generates an intermediate data equal to the input data size and thus it is a good fit for evaluating the intermediate data storage for HPC systems. Figure 6 compares the performance of Sort depending on the intermediate data location: local storage (on disk) or remote storage (on PVFS). We see that using PVFS also for storing the intermediate data results in 9% performance degradation.

When we analyze the performance of the Sort workload in each phase, we see that this performance degradation is 16% for the map phase which stems from writing the intermediate data to the parallel file system. For the reduce

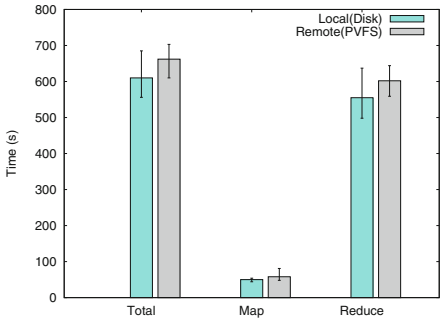


Fig. 6. Impact of the location of intermediate data on the performance of the Sort workload.

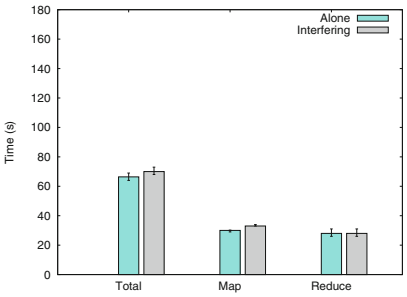
phase, we observe that there is a 8% increase in the completion time due to the additional I/O latency when fetching the intermediate data from PVFS.

Findings. In all of the workloads, we observe that the remote data access to the parallel file system leads to a significant performance degradation, especially for the input and output data. We also confirm that the degree of this performance degradation depends on the characteristics of the workloads and on the input data size.

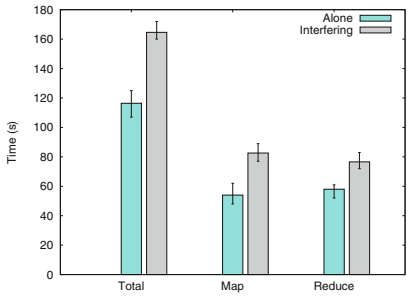
3.2 The Role of Contention

Given the shared architecture of HPC systems, contention is likely to occur when running Big Data applications on a HPC system. To assess the impact of contention on the performance of Big Data applications, we designed the following experiments:

Measuring the contention when running concurrent Big Data applications. Since the storage system is shared by all the nodes, this can create a serious contention problem on the storage path including network, server and storage devices. Here, we ran two Wordcount workloads concurrently under compute and data-centric paradigms by employing the Fair scheduler in Spark. The Fair scheduler allows these workloads to have equal share of the resources in the Spark cluster (i.e., each workload employ 400 tasks which is equal to the half of the cluster capacity). Figure 7 displays the execution times of the Wordcount workload when it runs alone and together with the other identical Wordcount workload for data and compute-centric paradigms. As shown in Fig. 7(a), the performance degradation when running in contention with the other Wordcount workload is negligible with the data-centric paradigm. In contrast, we observe that there is a 41% performance degradation with the compute-centric paradigm when two workloads are running concurrently. This stems from sharing the same parallel file system with compute-centric paradigm while these two workloads perform their I/O operations on their individual storage devices in the data-centric paradigm.



(a) Performance of concurrent Wordcount workloads under a data-centric paradigm.



(b) Performance of concurrent Wordcount workloads under a compute-centric paradigm.

Fig. 7. Performance of concurrent Wordcount workloads under different paradigms.

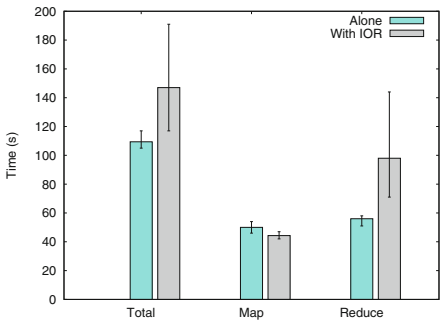


Fig. 8. Performance of the Wordcount workload when running alone and together with IOR workload.

In particular, Fig. 7(b) highlights that this performance degradation is mainly due to the map phase. This is because Wordcount is a map-heavy workload and therefore the number of I/O operations is quite large in the map phase compared to the reduce phase.

Measuring the contention when co-locating HPC and Big Data applications. This contention problem can even become more significant when we consider the ultimate objective of the HPC and Big Data convergence which is co-locating scientific and Big Data applications on a same platform. To emulate this objective, we ran the Wordcount workload alone and together with the IOR workload. IOR [29] is a popular I/O benchmark that allows users to specify different I/O configurations and thus measures the I/O performance of HPC systems. For IOR workload, we employed 224 processes (on a different set of nodes separated from the ones running the Wordcount workload) where each process issues a 512 MB write request in 32 MBs of chunks. Figure 8 shows the execution times of the Wordcount workload for both cases. Due to resource sharing (file system and network) with the IOR workload, there is a 1.4x performance degradation in the

total execution time of the Wordcount workload. When we look at the performance in each phase, we observe that this performance degradation is mainly due to the reduce phase. This stems from the fact that reduce phase performs write operations as the IOR workload and this results in a write/write contention.

Findings. We demonstrate that contention appears as a limiting factor for Big Data applications on HPC systems due to employing a shared storage system.

3.3 Impact of the File System Configuration

Besides the generic problems of HPC systems as latency and contention, we can also encounter performance issues with the file system specific problems when running Big Data applications on HPC systems. For example, [12] reported that metadata operations on Lustre create a bottleneck for Spark applications. Thus, we wanted to investigate file system specific problems that Spark applications can encounter. To this end, we configured PVFS with synchronization enabled (Sync ON). This synchronization feature can be necessary for providing a better reliability guarantee for the clients. To ensure this, each request is immediately flushed to the disk before finalizing the request.

We ran the Wordcount workload with two different synchronization options for PVFS: Sync ON and Sync OFF. Figure 9 shows that Wordcount performs 1.5x worse when synchronization is enabled. We observe that this significant performance degradation mainly stems from the reduce phase. This is expected since the output data is sent to the file system during the reduce phase and each request is flushed to the disk thus resulting in a major bottleneck for the application performance.

We also ran the Sort workload with two different configurations of PVFS and Table 1 shows that Sort performs 4.5x worse when synchronization is enabled. In contrast to Wordcount, we observe a much higher performance degradation with the Sort workload. This is because while Sort is generating a large amount of output data (200 GB as the input data size), Wordcount has a light reduce phase and generates only a small amount of output data.

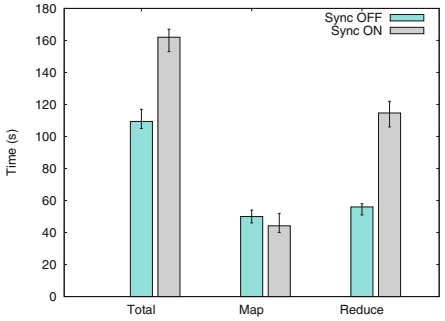


Fig. 9. Performance of the Wordcount workload under different configurations of PVFS.

Table 1. Execution time of the Sort workload and its phases under different configurations of PVFS.

Configuration	Execution time	Map	Reduce
Sync ON	2708.5 s	42.7 s	2665.8 s
Sync OFF	597.6 s	42.6 s	555.0 s

Findings. Parallel file systems are equipped with several features which are important for HPC applications (i.e., synchronization feature in PVFS to provide resiliency, distributed locking mechanism in Lustre to ensure file consistency). However, as we demonstrated in our experiments and also reported earlier in [12,32], these features may bring a significant performance degradation for Big Data applications.

3.4 Burst Buffers: Impact of Their Capacities and Location

We believe that there is a significant potential for improving the performance of Big Data applications using burst buffers. Although burst buffers promise a large potential, leveraging them efficiently for Big Data processing is not trivial. For example, there is a trade-off between the capacity and the throughput of the storage devices that are used in the burst buffers. Although, storage devices such as SSDs or NVRAMs can provide high throughput, they are limited in the *storage capacity*. Moreover, we demonstrated in our experiments that we should tackle all the I/O phases (i.e., input, intermediate and output data) while addressing the latency problem. Therefore, the problem of having limited capacity will be amplified when we try to use the burst buffer for all the I/O phases. By analyzing the traces collected from a research cluster (i.e., M45) [8], we observed that the amount of processed data was almost 900 TBs during 9 months. Hence, data management strategies for the burst buffers will play a crucial role on their efficiency. Similarly, it is important to decide when and which data to evict when running multiple concurrent applications.

Another challenge would be choosing the optimal *deployment location* for the burst buffers. Some of the possible deployment locations are within the compute nodes [32] or using a dedicated set of nodes [12] as burst buffers. While co-locating burst buffers and compute nodes can prevent the aforementioned capacity constraints since compute nodes are greater in size compared to dedicated nodes, this may result in a computational jitter due to sharing of the resources as also reported in [9].

To find out the impact of the aforementioned factors on the application performance when using burst buffers, we emulated a naive adoption of burst buffers by using the *ramdisk* (e.g., /dev/shm/) as a storage space and performed the following experiments:

Measuring the impact of the storage capacity on the application performance. Here, we ran the Wordcount workload with two different storage capacities for

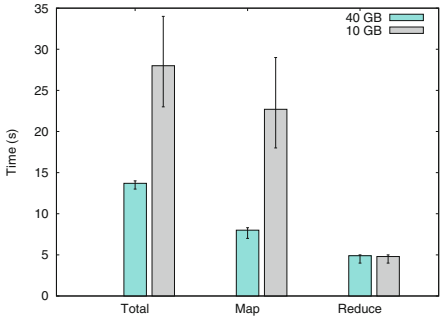


Fig. 10. Impact of the memory capacity of the burst buffer on the performance of the Wordcount workload.

the burst buffer as 40 GB and 10 GB memory. Note that, we used a smaller input data size than previous experiments which has a data size of 20 GB. The burst buffer is employing 5 dedicated nodes. Figure 10 shows the execution time of the Wordcount workload for different burst buffer configurations. We observe a 2.1x performance degradation when the burst buffer has 10 GB storage capacity. When we look at the performance of the workload in each phase, we see that this performance degradation is attributed to the map phase. This results from not having enough space for storing the input data on the burst buffer. Hence, compute nodes have to fetch the input data from the parallel file system thus resulting in a high I/O latency in the map phase. On the contrary, all I/O operations performed between burst buffer nodes and compute nodes when there is enough storage capacity. For the reduce phase, we do not observe any performance degradation since the output data to be written is small enough to fit into the burst buffer storage space, for this workload.

Measuring the impact of the deployment location of the burst buffer. We ran the Wordcount workload with the same configuration as in the previous experiment and deployed the burst buffer in two scenarios: in the first one, the burst buffer is deployed as a disjoint set of nodes and in the second one it is located as a subset of the compute cluster. Figure 11 displays that Wordcount performs better when burst buffer is deployed as a separate set of nodes. We hypothesize the following explanation. When the burst buffer is using the subset of the nodes of the compute cluster, I/O and compute tasks on those nodes conflict with each other thus resulting in a significant performance degradation (38% slowdown). This is in line with the observations reported in [9].

Findings. Our experiments show that the storage capacity and the location of burst buffers can have a significant impact on the performance of Big Data applications. With limited storage capacity, we demonstrate that burst buffers can not mitigate the latency problem fully since compute nodes still need to fetch most of the data from the parallel file system. For the deployment location, we

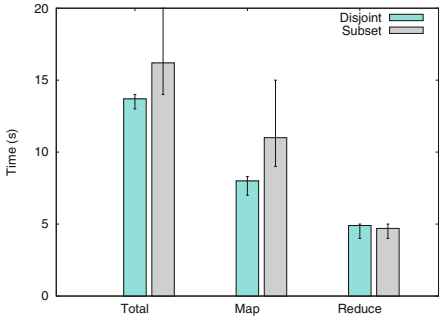


Fig. 11. Impact of the location of the burst buffer on the performance of the Wordcount workload.

observe that co-locating the burst buffer and compute resources on the same node can not be appropriate due to the possible interference among them.

4 Discussion and Implications

Here, we summarize our findings and discuss their implications to the design of burst buffer solutions. Our experiments reveal that Big Data applications encounter serious performance issues when running on HPC systems. First, we show that latency has a significant impact on the application performance and this impact depends on the characteristics of the Big Data applications.

Implications (1). Prior studies [12,21,32] have focused on mitigating the latency resulting from writing and reading intermediate data by introducing an intermediate storage layer (i.e., burst buffer); which is a blind adoption of burst buffers for HPC applications—burst buffers are used to store temporary data (i.e., checkpoints). We observe that using burst buffers for intermediate data can bring an improvement of at most 16%—when intermediate data have the same size as input data. As a result, the latency introduced by intermediate data is not really the bottleneck for a major fraction of Big Data applications: by analyzing traces collected from three different research clusters we observe that the amount of the intermediate data is less than 20% of the input data size for 85% of the applications [8]. On the other hand, we find that the latencies resulting from reading input data and writing output data significantly impact the performance. Thus, it is very important to mitigate the high latency resulting from accessing those data when developing burst buffer solutions. Moreover, prefetching techniques and mechanisms to overlap I/O and computation time could be adopted to further hide the high latency of remote data accesses between compute nodes and the parallel file system.

Second, we demonstrate that contention can severely degrade the performance of Big Data applications on HPC systems.

Implications (2). One could argue that using burst buffers would mitigate the contention problem as well since they are equipped with high throughput

storage devices. However, it is earlier demonstrated that contention is present in the HPC I/O stack regardless of the storage device used (e.g., SSDs, local memory or disk) [33]. In addition, burst buffers are shared by all the nodes as in the parallel file system. Therefore, we believe that we must address the contention problem when developing burst buffer solutions. For instance, we can try to make distinct sets of compute nodes target distinct sets of burst buffer nodes. Moreover, we can further employ well-known I/O aggregation techniques to minimize the contention problem.

We also observe that file system specific features may bring a significant performance degradation for Big Data applications.

Implications (3). Even burst buffers can improve the performance of Big Data applications, they still rely on a parallel file system. Thus, we should tackle the file system specific issues as well for efficient Big Data processing on HPC systems.

Lastly, we confirm that an effective exploitation of burst buffers for Big data applications in HPC systems strongly depends on the size and location settings of burst buffers.

Implications (4). To tackle the limited storage capacity problem, we can develop smarter data fetching techniques for the burst buffer. For instance, instead of trying to fit all the input data set into the burst buffer storage space, we can fetch a subset of the data set (i.e., one wave) as compute cluster computes one wave at a time. For instance, cluster consists of 800 tasks in our experiments and therefore they can only compute 25 GB data at one iteration. In this way, compute tasks can fetch all the data from the burst buffer nodes and therefore the latency problem can be mitigated.

5 Related Work

Several research efforts have been conducted to evaluate the performance of Big Data analytics frameworks on HPC systems. Wang et al. [32] 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 and locality-oriented task scheduling. Tous et al. [31] evaluated the Spark performance on a MareNostrum supercomputer. In particular, they studied the impact of different Spark configurations on the performance of Sort and K-means applications. In [30], the authors compared the performance of MapReduce applications on PVFS and HDFS file systems by using Hadoop framework and give insights into how to emulate HDFS behavior by using PVFS. Li and Shen [24] compared the performance of MapReduce applications on scale-up and scale-out clusters and proposed a hybrid scale-up/out Hadoop architecture based on their findings.

Aforementioned studies provide useful findings towards leveraging HPC systems for Big Data processing. However, they do not illustrate a complete analysis of the potential performance issues (e.g., latency and contention). For the latency

problem, most of the studies focus on the intermediate data storage and ignore the latencies which can occur in other I/O phases. We provide a detailed analysis of the impact of latency on the application performance by giving a breakdown of the latency problem into its different phases (i.e., input, intermediate and output data). Although these studies mention contention as a problem, none of them investigate its impact on the application performance. Hence, we aim to complement those studies by providing a detailed analysis of the impact of latency and contention on the performance of Spark applications. Furthermore, we show potential performance issues specific to different PVFS configurations.

Some works proposed adoption of burst buffers for efficient Big Data processing on HPC systems. Chaimov et al. [12] employed a dedicated set of nodes with NVRAM as the storage space for the intermediate data of Big Data applications. This in turn improved the scalability of the Spark framework compared to the scenario when using Lustre file system as the storage space. Islam et al. [21] proposed a novel design for HDFS which uses NVRAM-based burst buffer nodes on top of a parallel file system for improving the performance of Spark applications. Yildiz et al. [34] present Eley, a burst buffer solution that helps to accelerate the performance of Big Data applications while guaranteeing the performance of HPC applications. Eley employs a prefetching technique that fetches the input data of these applications to be stored close to computing nodes thus reducing the latency of reading data inputs. Moreover, Eley is equipped with a full delay operator to guarantee the performance of HPC applications. Similarly, our findings illustrate that there is a need for burst buffer solutions to alleviate

Table 2. Our major findings on the characteristics of Big Data applications on HPC systems.

The impact of I/O latency
We confirm that I/O latency resulting from the remote data access to the parallel file system leads to a significant performance degradation for all the Big Data workloads. However, in contrary to existing studies [12,21], we demonstrate that intermediate data storage is not the major contributor to this latency problem. We also observe that the impact of this latency problem depends on the characteristics of the Big Data applications (e.g., map-heavy, iterative applications) and on the input data size
The role of contention
We demonstrate that contention appears as a limiting factor for Big Data applications on HPC systems due to employing a shared storage system
The impact of the file system configuration
Parallel file systems are equipped with several features which are important for HPC applications (i.e., synchronization feature in PVFS to provide resiliency, distributed locking mechanism in Lustre to ensure file consistency). However, as we demonstrated in our experiments and also reported earlier in [12,32], these features may bring a significant performance degradation for Big Data applications

the latency problem. In addition, we give insights into designing efficient burst buffer solutions. Specifically, we claim that future burst buffer implementations should be aware of the contention problem and also try to eliminate the latency problem for the input phase and output phase.

6 Conclusion and Future Work

We have recently witnessed an increasing trend toward leveraging HPC systems for Big Data processing. In this paper, we undertook an effort to provide a detailed analysis of performance characteristics of Big Data applications on HPC systems, as first steps towards efficient Big Data processing on HPC systems. Our findings demonstrate that one should carefully deal with HPC-specific issues (e.g., latency, contention and file system configuration) when running Big Data applications on these systems. An important outcome of our study is that negative impact of latency on the application performance is present for all I/O phases. We further show that contention is a limiting factor for the application performance and thus Big Data solutions should be equipped with contention-aware strategies. Lastly, we reveal that enabling synchronization for PVFS in order to provide resilience can create a serious performance bottleneck for Big Data applications.

We summarize our findings in Table 2. We believe that these findings can help to motivate further research leveraging HPC systems for Big Data analytics by providing a clearer understanding of the Big Data application characteristics on these systems.

Acknowledgment. This work is supported by the ANR KerStream project (ANR-16-CE25-0014-01). 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. Big Data and Extreme-scale Computing (BDEC) Workshop. http://www.exascale.org/bdec/
2. HiBench Big Data microbenchmark suite. https://github.com/intel-hadoop/HiBench
3. The Apache Hadoop Project. http://www.hadoop.org
4. Apache Storm (2012). https://storm.apache.org/
5. Apache Spark primer (2017). http://go.databricks.com/hubfs/pdfs/Apache_Spark_Primer_170303.pdf
6. IDC's Data Age 2025 study (2017). http://www.seagate.com/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf
7. Powered by Hadoop (2017). http://wiki.apache.org/hadoop/PoweredBy/
8. Hadoop Workload Analysis. http://www.pdl.cmu.edu/HLA/index.shtml. Accessed Jan 2017

9. Bent, J., Faibish, S., Ahrens, J., Grider, G., Patchett, J., Tzelnic, P., Woodring, J.: Jitter-free co-processing on a prototype exascale storage stack. In: 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–5. IEEE (2012)
10. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: Haloop: efficient iterative data processing on large clusters. Int. J. Very Large Databases **3**(1–2), 285–296 (2010)
11. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K.: Apache flink: stream and batch processing in a single engine. Bull. IEEE Comput. Soc. Tech. Comm. Data Eng. **36**(4), 28–38 (2015)
12. Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K.Z., Srinivasan, J.: Scaling Spark on HPC systems. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, pp. 97–110. ACM (2016)
13. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
14. Donovan, S., Huizenga, G., Hutton, A.J., Ross, C.C., Petersen, M.K., Schwan, P.: Lustre: building a file system for 1000-node clusters (2003)
15. Dorier, M., Antoniu, G., Cappello, F., Snir, M., Sisneros, R., Yildiz, O., Ibrahim, S., Peterka, T., Orf, L.: Damaris: addressing performance variability in data management for post-petascale simulations. ACM Trans. Parallel Comput. (TOPC) **3**(3), 15 (2016)
16. Dorier, M., Antoniu, G., Ross, R., Kimpe, D., Ibrahim, S.: CALCioM: mitigating I/O interference in HPC systems through cross-application coordination. In: Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2014), Phoenix, AZ, USA, May 2014. http://hal.inria.fr/hal-00916091
17. Fox, G., Qiu, J., Jha, S., Ekanayake, S., Kamburugamuve, S.: Big data, simulations and HPC convergence. In: Rabl, T., Nambiar, R., Baru, C., Bhandarkar, M., Poess, M., Pyne, S. (eds.) WBDB -2015. LNCS, vol. 10044, pp. 3–17. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49748-8_1
18. Gainaru, A., Aupy, G., Benoit, A., Cappello, F., Robert, Y., Snir, M.: Scheduling the I/O of HPC applications under congestion. In: International Parallel and Distributed Processing Symposium, pp. 1013–1022. IEEE (2015)
19. Guo, Y., Bland, W., Balaji, P., Zhou, X.: Fault tolerant MapReduce-MPI for HPC clusters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 34. ACM (2015)
20. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. In: Special Interest Group on Operating Systems Review, vol. 41, pp. 59–72. ACM (2007)
21. Islam, N.S., Wasi-ur Rahman, M., Lu, X., Panda, D.K.: High performance design for HDFS with byte-addressability of NVM and RDMA. In: Proceedings of the 2016 International Conference on Supercomputing, p. 8. ACM (2016)
22. Jégou, Y., Lantéri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Iréa, T.: Grid'5000: a large scale and highly reconfigurable experimental grid testbed. Int. J. High Perform. Comput. Appl. **20**(4), 481–494 (2006)
23. Jin, H., Ibrahim, S., Qi, L., Cao, H., Wu, S., Shi, X.: The MapReduce programming model and implementations. In: Buyya, R., Broberg, J., Goscinski, A. (eds.) Cloud Computing: Principles and Paradigms, pp. 373–390. Wiley, New York (2011)
24. Li, Z., Shen, H.: Designing a hybrid scale-up/out hadoop architecture based on performance measurements for high application performance. In: 2015 44th International Conference on Parallel Processing (ICPP), pp. 21–30. IEEE (2015)

25. Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K., Wolf, M.: Managing variability in the I/O performance of petascale storage systems. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12. IEEE (2010)
26. Lopez, I.: IDC talks convergence in high performance data analysis (2013). https://www.datanami.com/2013/06/19/idc_talks_convergence_in_high_performance_data_analysis/
27. Ross, R.B., Thakur, R., et al.: PVFS: a parallel file system for Linux clusters. In: Annual Linux Showcase and Conference, pp. 391–430 (2000)
28. Sato, K., Mohror, K., Moody, A., Gamblin, T., de Supinski, B.R., Maruyama, N., Matsuoka, S.: A user-level infiniband-based file system and checkpoint strategy for burst buffers. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 21–30. IEEE (2014)
29. Shan, H., Shalf, J.: Using IOR to analyze the I/O performance for HPC platforms. In: Cray User Group Conference 2007, Seattle, WA, USA (2007)
30. Tantisiriroj, W., Patil, S., Gibson, G.: Data-intensive file systems for internet services: a rose by any other name. Parallel Data Laboratory, Technical report UCB/EECS-2008-99 (2008)
31. Tous, R., Gounaris, A., Tripiana, C., Torres, J., Girona, S., Ayguadé, E., Labarta, J., Becerra, Y., Carrera, D., Valero, M.: Spark deployment and performance evaluation on the MareNostrum supercomputer. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 299–306. IEEE (2015)
32. Wang, Y., Goldstone, R., Yu, W., Wang, T.: Characterization and optimization of memory-resident MapReduce on HPC systems. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 799–808. IEEE (2014)
33. Yildiz, O., Dorier, M., Ibrahim, S., Ross, R., Antoniu, G.: On the root causes of cross-application I/O interference in HPC storage systems. In: IPDPS-International Parallel and Distributed Processing Symposium (2016)
34. Yildiz, O., Zhou, A.C., Ibrahim, S.: Eley: on the effectiveness of burst buffers for big data processing in HPC systems. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER), pp. 87–91, September 2017
35. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In: Proceedings of the 5th European Conference on Computer Systems, pp. 265–278. ACM (2010)
36. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: HotCloud 2010, p. 10 (2010)