Performance evaluation of data-intensive computing applications on a public IaaS cloud

Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño and Ramón Doallo

Computer Architecture Group, Department of Electronics and Systems, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain
Email: {rreye,taboada,sramos,juan,doallo}@udc.es

The advent of cloud computing technologies, which dynamically provide on-demand access to computational resources over the Internet, is offering new possibilities to many scientists and researchers. Nowadays, Infrastructure as a Service (IaaS) cloud providers can offset the increasing processing requirements of data-intensive computing applications, becoming an emerging alternative to traditional servers and clusters. In this paper, a comprehensive study of the leading public IaaS cloud platform, Amazon EC2, has been conducted in order to assess its suitability for data-intensive computing. One of the key contributions of this work is the analysis of the storage-optimized family of EC2 instances. Furthermore, this study presents a detailed analysis of both performance and cost metrics. More specifically, multiple experiments have been carried out to analyze the full I/O software stack, ranging from the low-level storage devices and cluster file systems up to real-world applications using representative data-intensive parallel codes and MapReduce-based workloads. The analysis of the experimental results has shown that data-intensive applications can benefit from tailored EC2-based virtual clusters, enabling users to obtain the highest performance and cost-effectiveness in the cloud.

Keywords: Data-intensive computing; Cloud computing; Infrastructure as a Service (IaaS); Amazon EC2; Cluster file system; MapReduce

1. INTRODUCTION

In recent years, the computational requirements for large-scale data-intensive computing [1] applications across distributed clusters or data centers have grown significantly in various disciplines including bioinformatics, astronomy or medical image analysis. In the current era of Big Data, characterized by the unprecedented volume of data, these applications are generating and analyzing large data sets, which usually require a high number of computational resources together with the availability of a high-performance cluster file system for scalable performance.

Cloud computing [2] is a relatively recent Internet-based computing model which is gaining significant acceptance in many areas and IT organizations as an elastic, flexible, and variable-cost way to deploy their service platforms using outsourced resources. These resources can be rapidly provisioned and released with minimal management effort. Public cloud providers offer access to external users who are typically billed by consumption using the pay-per-use pricing model.

Infrastructure as a Service (IaaS) is a type of cloud service which dynamically provides, by means of virtualization technologies, on-demand and self-service access to elastic computational resources (e.g., CPU, memory, networking and storage), offering a powerful abstraction that easily allows end users to set up virtual clusters to exploit supercomputing-level power without any knowledge of the underlying infrastructure. Public IaaS providers typically make huge investments in data centers and then rent them out, allowing consumers to avoid heavy capital investments and obtain both cost-effective and energy-efficient solutions. Hence, organizations are no longer required to invest in additional computational resources, since they can just leverage the infrastructure offered by the IaaS provider.

Most popular public cloud providers include Amazon Web Services (AWS) [3], Google Compute Engine (GCE) [4], Microsoft Azure [5] and Rackspace [6]. Nowadays, AWS remains as the top public cloud provider [7], offering the widest range of cloud-based services. In fact, the Elastic Compute Cloud (EC2) service [8] is among the most used and largest IaaS cloud platforms [9], which allows computational resources in Amazon's data centers to be easily rented on-demand. Moreover, Amazon EC2 offers several cloud resources which specifically target High Performance Computing (HPC) environments [10], composed of several virtual machines that are intended to be well suited for highly demanding workloads by offering powerful multi-core CPU resources, improved network performance via a high-speed interconnect (10 Gigabit Ethernet) and enhanced Input/Output (I/O) performance by providing Solid State Drive (SSD) disks.

In this context, the cloud computing paradigm has experienced tremendous growth in the last few years, particularly for general-purpose applications such as web servers or commercial web applications. Furthermore, it has also generated considerable interest both in the scientific community and industry. Thus, cloud computing is becoming an attractive option for distributed computing and HPC due to the high availability of computational resources at large scale. This fact has motivated multiple works that analyze the feasibility of using public clouds, especially Amazon EC2, instead of traditional clusters for running HPC applications [11, 12, 13, 14]. However, most of the previous works are focused mainly on computation- and communication-intensive HPC codes, especially tightly-coupled parallel applications using the Message-Passing Interface (MPI), whereas there are few works that have investigated cloud storage and I/O performance using data-intensive applications (e.g., MapReduce [15] workloads). In addition, previous evaluations have been carried out before Amazon introduced storage-optimized instances [16], which provide with direct-attached storage devices specifically optimized for applications with high disk I/O requirements.

This paper presents a comprehensive study of running data-intensive applications on the leading Amazon EC2 cloud, using storage-optimized instances and conducting a related analysis that takes into account both performance and cost metrics. Hence, multiple experiments have been performed at several layers using a suite of micro-benchmarks and applications to evaluate the full I/O software stack of data-intensive computing, ranging from the low-level storage devices and cluster file systems up to the application level using representative parallel codes implemented on top of common computing frameworks and I/O middleware (e.g., Apache Hadoop [17], MPI-IO [18]). The main experimental results indicate that the unique configurability advantage offered by public clouds, almost impossible to achieve in traditional platforms, can benefit significantly data-intensive applications. Thus, our main conclusions point out that current cloud-based virtual clusters enable end users to build up high-performance I/O systems when using the appropriate resources and configurations.

The rest of the paper is organized as follows: Section 2 describes the related work. Section 3 presents an overview of the software stack for data-intensive computing. Section 4 provides general background of the Amazon EC2 platform and the experimental configuration, briefly describing both the benchmarks and applications used as well as the evaluation methodology. Section 5 presents and analyzes the performance results obtained in the evaluation conducted in this work. Finally, Section 6 summarizes our concluding remarks.

2. RELATED WORK

Recently, there have been a series of research efforts assessing the suitability of using public cloud platforms for HPC and scientific computing. Although there are some works that have evaluated other public IaaS providers such as Microsoft Azure [19] and GoGrid [20], the vast majority of them have assessed the most popular IaaS platform: Amazon EC2 [11, 12, 13, 14, 21, 22, 23]. Most of these previous studies are mainly focused on computation and communication behavior, evaluating only tightly-coupled parallel codes, usually MPI-based applications, which are commonly used in HPC environments, hence characterizing only the performance of CPU and network. As a main conclusion of these works, it has been fairly well-established that communication-intensive MPI codes tend to perform poorly on Amazon EC2, primarily due to the low virtualized network performance.

However, few works have investigated I/O and storage performance on Amazon EC2. Some of them analyzed the suitability of running scientific workflows [24, 25, 26], showing that it can be a successful option as these are loosely-coupled parallel applications. Other works, next presented, have evaluated some I/O aspects of Amazon EC2, but they were carried out before the availability of storage-optimized instances. Evangelinos and Hill [27] reported some I/O storage results using an I/O-intensive workload, but limited to sequential runs on a basic Network File System (NFS) setting. The storage and network performance of the Eucalyptus cloud computing platform was analyzed in [28], confronted with some results from one general-purpose EC2 large instance. Ghoshal et al. [29] compared the I/O performance of Amazon EC2 confronted with Magellan, a private cloud platform, and an HPC cluster. Their results show that NFS performance in Amazon is many orders of magnitude worse than the parallel file system installed in the HPC cluster. Similar studies have been carried out in [30, 31, 32], which have evaluated some parallel codes for NFS and/or PVFS file systems, but only limited to MPI-based applications, as MapReduce-based workloads were not taken into account. Gunarathne et al. [33] presented a MapReduce framework designed on top of the Microsoft Azure cloud, which was evaluated against the Amazon Elastic MapReduce (EMR) service and an

EC2-based Hadoop cluster. More recently, the same authors have introduced a new runtime that supports iterative MapReduce computations [34]. Finally, some other works have evaluated MapReduce frameworks and applications on private cloud platforms such as Nimbus [35] and on traditional HPC clusters [36, 37].

3. OVERVIEW OF DATA-INTENSIVE COMPUTING APPLICATIONS

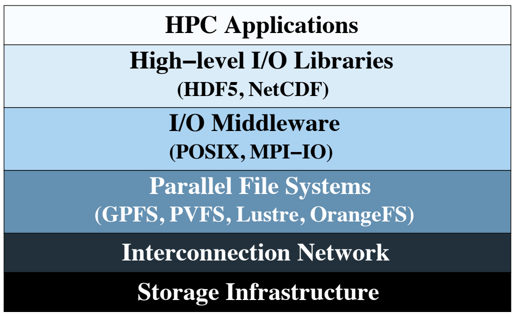
Most current data-intensive applications can be classified into one of the following categories: HPC and Big Data analysis. In both categories, applications are executed across distributed clusters or data centers using multiple compute nodes and handling massive amounts of data, in which the underlying cluster file system is a key component for providing scalable application performance.

On the one hand, HPC is traditionally defined by parallel scientific applications in the fields of science and engineering that rely on low-latency networks for message passing and cluster deployments that usually separate compute and storage nodes. HPC applications are typically large simulations that run for a long time and protect themselves from failures using fault tolerance methods such as checkpointing [38]. These methods involve large-scale data movements as the system state is periodically written to persistent storage in order to be able to restart the application in case of failure. Therefore, checkpointing can be particularity challenging when all processes in the parallel application write to the same checkpoint file at the same time, an I/O access pattern fairly known as N-1 writing [39]. Hence, these HPC applications are considered data-intensive, and they typically rely on a POSIX-based parallel file system for highly scalable and concurrent parallel I/O. In these systems, multiple dedicated storage nodes act as I/O servers to provide a UNIX file system API and expose a POSIX-compliant interface to applications to support a broad range of access patterns for many different workloads. In this scenario, the I/O access pattern is mainly dominated by sequential operations, being random accesses rare in data-intensive HPC applications [40, 41].

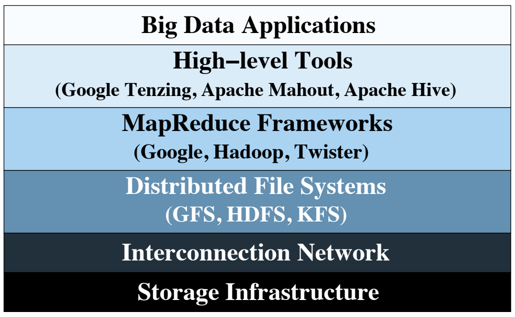
On the other hand, Big Data analysis generally refers to a heterogeneous class of business applications that operate on large amounts of unstructured and semi-structured data, usually implemented using the MapReduce programming model, which was first proposed by Google [15]. In fact, MapReduce has become the most popular computing framework for large-scale processing and analysis of vast data sets in clusters [42], mainly because of its simplicity and scalability. These data-intensive applications are designed to handle data more efficiently than a traditional structured query language to quickly extract value from the data. They include traditional batch-oriented jobs such as data mining, building search indices and log collection and analysis [43], as well as web search and advertisement selection [44]. One key aspect of these applications is that they are aware in advance of the workloads and I/O access patterns, typically relying on a custom, purpose-built distributed file system that is usually implemented from scratch. These file systems are specifically designed to support only one programming model (e.g., MapReduce) in order to provide high scalability with reliability by striping and replicating the data in large chunks across the locally attached storage of the cluster nodes. Hence, by exposing the data layout to the applications, the MapReduce task scheduler is able to co-locate a compute task on a node that stores the required input data [15], thereby relying on cluster deployments that co-locate compute and storage nodes on the same cluster node. However, these file systems feature a simplified design and implementation without providing a POSIX-compliant interface or consistency semantics. Consequently, they work well for MapReduce applications but cannot support traditional HPC applications without changes. Unlike distributed file systems, parallel file systems cannot exploit data locality as they do not generally expose the data layout to the applications, which usually results in a significant performance loss when running MapReduce applications. Hence, the developers of the most popular parallel file systems have demonstrated the feasibility of using them with the MapReduce paradigm by applying some minor modifications and providing simple file system extensions, obtaining comparable performance to distributed file systems [45, 46].

3.1. I/O Hardware and Software Support

Figure 1 shows the I/O software stacks more commonly available on current HPC and Big Data platforms that support data-intensive applications. On the one hand, HPC applications perform I/O at a specific level in the software stack depicted in Figure 1(a). In the upper levels, advanced data models such as HDF5 [47] and NetCDF [48] can provide certain advantages for particular applications. These high-level libraries usually feature a parallel version (Parallel HDF5 [49], Parallel NetCDF [50]) implemented on top of the MPI-IO [18] middleware in order to perform parallel I/O cooperatively among many processes. MPI-IO is specified in the MPI-2 standard and defines a comprehensive API, which is implemented on top of file systems, intended specifically to provide MPI programs with a high-performance, portable and parallel I/O interface. Towards the bottom of the stack, parallel file systems (e.g., GPFS [51], PVFS [52], Lustre [53], OrangeFS [54]) are used to transfer I/O requests and file data across the underlying interconnection network and storage devices. Figure 2(a) shows the most widely extended cluster architecture in HPC platforms, which usually separates compute and storage nodes.

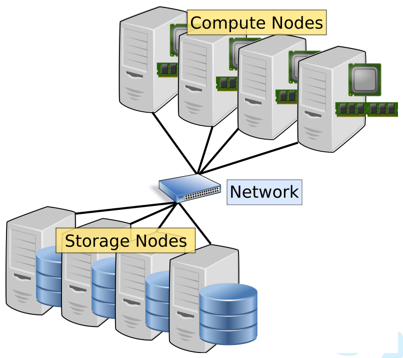


(a) I/O software stack for HPC applications

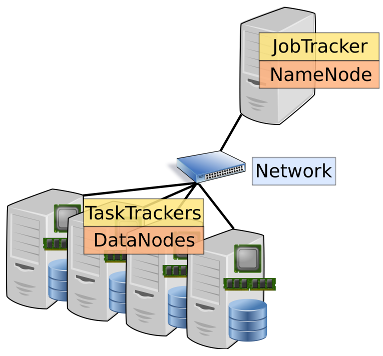


(b) Big Data analysis on top of MapReduce frameworks

FIGURE 1. I/O software stacks for data-intensive computing applications



(a) Traditional HPC cluster architecture



(b) High-level overview of Hadoop and HDFS architecture

FIGURE 2. Hardware support for data-intensive computing applications

On the other hand, Big Data applications can either use a high-level tool or directly a distributed computing framework (e.g., Google MapReduce [15]) to perform their data analysis (see Figure 1(b)). Usually, the high-level tools (e.g., Google Tenzing [55], Apache Mahout [56]) are built on top of a computing framework to allow users write applications that take advantage of the MapReduce programming model without having to learn all its details. Among these frameworks, the Apache Hadoop project [17] has gained significant attention in the last years as a popular open-source Java-based implementation of the MapReduce paradigm derived from the Google's proprietary implementation. As mentioned before, these frameworks generally rely on custom distributed file systems (e.g., GFS [57], HDFS [58], KFS [59]) to transfer I/O requests across the interconnection network and the underlying storage infrastructure. Figure 2(b) shows an overview of a typical Hadoop cluster that stores the data in HDFS, using a master-slave architecture which co-locates compute and storage nodes on the same slave node. Using the Hadoop terminology, the TaskTracker and DataNode processes, which execute the tasks and store the data, respectively, run on the slave nodes. The master node runs the JobTracker and NameNode processes, acting as a single task manager and metadata server.

4. EXPERIMENTAL CONFIGURATION

In this section, the Amazon EC2 platform and the selected instance types are described along with brief descriptions of the representative benchmarks and applications used in the performance evaluation section.

4.1. Amazon EC2 Platform

The Amazon EC2 cloud service currently offers a rich variety of Xen-based virtual machine configurations called instance types [16], which are optimized to fit different use cases. Virtual machines of a given instance type comprise varying combinations of CPU, memory, storage and networking capacity, each with a different price point. One of the key contributions of this paper is the evaluation of the storage-optimized family of EC2 instances, which according to Amazon are specifically intented to be well suited for Hadoop, cluster file systems and NoSQL databases, among other uses.

The storage-optimized family includes two EC2 instance types: High I/O (hi1.4xlarge, hereafter HI1) and High Storage (hs1.8xlarge, hereafter HS1). These instances provide direct-attached storage devices (known as "ephemeral" or local disks) optimized for applications with specific disk I/O and capacity requirements. More specifically, HI1 provides 2 TB of instance storage capacity backed by 2 SSD-based disks, whereas HS1 provides 48 TB across 24 standard Hard Disk Drive (HDD) disks (see Table 1). Generally, EC2 instances can access two additional storage options: (1) off-instance Elastic Block Store (EBS), which are remote volumes accessible through the network that can be attached to an instance as block storage devices; and (2) Simple Storage Service (S3), which is a distributed key-value based object storage system, accessed through a web service interface. However, S3, unlike EBS and ephemeral devices, lacks general file system interfaces usually required by data-intensive applications. Hence, S3 has not been considered in our evaluation since its use is not transparent to applications, and because of its poor performance shown by previous works [24]. Moreover, ephemeral (local) disks perform better than EBS volumes according to [28, 29] due to the overhead caused by the additional network access incurred by EBS. This superior performance of ephemeral disks was assessed even before the availability of storage-optimized instances, which favor significantly the performance of local disks.

In addition, storage-optimized instances provide 8 physical cores that represent 35 EC2 Compute Units (ECUs[1]) of computational power, together with 60.5 and 117 GB of memory for HI1 and HS1, respectively. Moreover, these instances support cluster networking via a high-speed network (10 Gigabit Ethernet), which is also another differential characteristic of these resources. Hence, instances launched into a common placement group are placed in a logical cluster that provides low-latency, full-bisection 10 Gbps bandwidth connectivity between instances in the cluster. However, there is a current limitation that only instances of the same type can be included in a placement group (i.e., a placement group cannot combine HI1 and HS1 instances).

Two additional high-performance instance types which also provide the cluster networking feature together with the 10 Gigabit Ethernet network have been evaluated. On the one hand, the Cluster Compute instance type (cc2.8xlarge, hereafter CC2) is a compute-optimized instance with 16 physical cores, 60.5 GB of memory and 3.4 TB of instance storage capacity across 4 HDDs. On the other hand, the High Memory Cluster Compute instance type (cr1.8xlarge, herefater CR1) is a memory-optimized instance with 16 physical cores, 244 GB of memory and 240 GB of instance storage capacity across 2 SSDs. Note that CC2 and CR1 are among the EC2 instances with the most powerful computational resources (i.e., 88 ECUs). Table 1 summarizes the main characteristics of the selected instance types together with their hourly price for on-demand Linux usage in the North Virginia data center.

[1]According to Amazon one ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or Xeon processor.

4.2. Benchmarks and Applications

The evaluation of data-intensive applications on Amazon EC2 has been conducted using representative benchmarking tools and applications (see details in Table 2) ranging at the different levels shown in the software stacks of Figures 1(a) and 1(b). The first set of experiments consists of a single-instance micro-benchmarking of a local file system, *xfs*, selected as it provides interesting features (e.g., allocation groups, B-tree indexing, metadata journaling) as building blocks for scalable data-intensive infrastructures. This analysis has been carried out using the IOzone benchmark [60] on a single ephemeral disk as well as on multiple disks combined in a single software RAID array with the Linux *mdadm* utility.

After characterizing the performance of the underlying storage infrastructure, both software stacks have been analyzed at the cluster file system level. Regarding the HPC stack, the OrangeFS parallel file system [54] has been evaluated using the IOR benchmark [61] and the MPI-IO interface [18] as representative I/O middleware. OrangeFS is a relatively recent branch of the production-quality and widely extended Parallel Virtual File System (PVFS) [52], but unlike PVFS, under active development and introducing new features. OrangeFS has also been selected because it lacks, to the best of our knowledge, suitable assessments of its performance on a public cloud infrastructure. Regarding the Big Data software stack, the Intel HiBench suite [62] has been used for the evaluation of Apache Hadoop [17], selected as the most representative MapReduce computing framework. The HiBench suite consists of a set of Hadoop programs including both synthetic micro-benchmarks and real-world applications. Hadoop HDFS [58] has been evaluated at the distributed file system level using the Enhanced DFSIO benchmark included in HiBench.

Next, the performance of several data-intensive codes has been analyzed at the application level. On the one hand, the BT-IO kernel [63] and the FLASH-IO code [64], which are implemented on top of the MPI-IO and Parallel HDF5 (PHDF5) libraries, respectively, have been selected as representative I/O-intensive HPC applications. On the other hand, the Sort and WordCount workloads, also included in HiBench, have been selected as they are representative of two widespread kinds of MapReduce jobs: transforming data from one representation to another, and extracting a small amount of information from a large data set. Additionally, the PageRank and Aggregation workloads have been evaluated. They are based on high-level tools built on top of the Hadoop framework. PageRank is an implementation of the page-rank algorithm in Apache Mahout [56], an open-source machine learning library. Aggregation measures the performance of Apache Hive [65] through computing the sum of each group in Hive over a single read-only table.

TABLE 1. Description of the Amazon EC2 instances: CC2, HI1, HS1 and CR1

	CC2	HI1	HS1	CR1
Release Date	November 2011	July 2012	December 2012	January 2013
Instance Family	Compute-optimized	Storage-optimized	Storage-optimized	Memory-optimized
API name	cc2.8xlarge	hi1.4xlarge	hs1.8xlarge	cr1.8xlarge
Price (Linux)	$2 per hour	$3.10 per hour	$4.60 per hour	$3.50 per hour
CPU Model	Intel Xeon E5-2670 Sandy Bridge	Intel Xeon E5620 Westmere	Intel Xeon E5-2650 Sandy Bridge	Intel Xeon E5-2670 Sandy Bridge
#CPUs	2 (Eight-Core)	2 (Quad-Core)	1 (Eight-Core)	2 (Eight-Core)
CPU Speed (Turbo)	2.6 GHz (3.3 GHz)	2.4 GHz (2.66 GHz)	2 GHz (2.8 GHz)	2.6 GHz (3.3 GHz)
#Cores/Threads	16/32 (HT[2] enabled)	8/16 (HT enabled)	8/16 (HT enabled)	16/32 (HT enabled)
Amazon ECUs	88	35	35	88
ECUs per Core	5.5	4.4	4.4	5.5
ECUs per US$	44	11.29	7.61	25.14
L3 Cache size	20 MB	12 MB	20 MB	20 MB
Memory	60.5 GB	60.5 GB	117 GB	244 GB
#Memory Channels	4 (DDR3-1600)	3 (DDR3-1066)	4 (DDR3-1600)	4 (DDR3-1600)
Memory Bandwidth	51.2 GB/s	25.6 GB/s	51.2 GB/s	51.2 GB/s
#QPI Links	2 (4000 MHz)	2 (2933 MHz)	2 (4000 MHz)	2 (4000 MHz)
QPI Speed	8 GT/s	5.86 GT/s	8 GT/s	8 GT/s
Ephemeral Disks	4 × 845 GB (HDD)	2 × 1 TB (SSD)	24 × 2 TB (HDD)	2 × 120 GB (SSD)
Storage Capacity	3.4 TB	2 TB	48 TB	240 GB
Storage per US$	1.7 TB	0.65 TB	10.43 TB	0.07 TB
Interconnect	10 Gigabit Ethernet (Full-bisection bandwidth using Placement Groups)			

[2]The Intel Hyper-Threading (HT) technology is enabled for all instance types. Hence, Amazon announces that these instances have a number of available virtual cores that takes into account hyper-threaded cores (i.e., Amazon states that CC2 instances provide 32 virtual cores). However, the performance increase of using this technology is highly workload-dependent, and may be even harmful in some scenarios. Particularly, we have not seen any improvements using all the available virtual cores in our experiments on Amazon EC2. Therefore, all the configurations shown in this paper have used only one virtual core per physical core.

4.3. Software Settings

The Linux distribution selected for the performance evaluation was Amazon 2012.09.1, as it is a supported and maintained Linux provided by Amazon for its specific usage on EC2 instances. This Linux flavour comes with kernel 3.2.38 and has been tailored for the performance evaluation with the incorporation of the previously described benchmarks: IOzone 3.414, IOR 2.10.3 and HiBench 2.2. For the parallel file system evaluation, OrangeFS version 2.8.7 was used, whereas the selected MPI library was MPICH [66] version 3.0.2, which includes MPI-IO support. In addition, the MPI implementation of the NASA Advanced Supercomputing (NAS) Parallel Benchmarks suite (NPB) [67] version 3.3 was installed for the BT-IO kernel evaluation, whereas the HDF5 data library version 1.8.10 was used for the FLASH-IO code. Regarding the Hadoop experiments, the versions used were Hadoop 1.0.4 (stable), Mahout 0.7 and Hive 0.9.0. The Java Virtual Machine (JVM) was OpenJDK version 1.7.0_19.

Finally, the performance results presented in this paper are averages of a minimum of five measurements for each evaluated configuration. Furthermore, all the experiments have been carried out in the US East EC2 region (i.e., *us-east-1*), corresponding to North Virginia, the main data center, which has the highest availability of the evaluated instance types. The selected availability zone was *us-east-1d*, where, according to previous works [84] and our own experience, there is usually the lowest variability.

5. EVALUATION OF AMAZON EC2 FOR DATA-INTENSIVE COMPUTING

This section presents an in-depth performance and cost analysis of data-intensive computing applications on the selected public IaaS cloud, Amazon EC2, using the representative micro-benchmarks, cluster file systems and kernels/applications described in the previous section.

5.1. Single-Instance Storage Micro-benchmarking

Figure 3 presents the sequential write and read bandwidth (left graphs) using the IOzone benchmark on a single ephemeral disk (two top rows of graphs) and on multiple ephemeral disks combined into a software RAID array (two bottom rows of graphs). The right graphs of the figure show the performance/cost ratio on these scenarios, a productivity metric defined as the bandwidth obtained per invoiced US$. As mentioned before, the underlying local file system is *xfs*, whereas the default chunk size[3] (512 KB) has been used in RAID configurations. Furthermore, the evaluation of CC2, CR1 and HI1 instances has only considered the RAID 0 level (data striping) owing to the low number of available ephemeral disks on these instances (i.e., 2 disks in CR1 and HI1 and 4 disks in CC2). However, HS1 provides up to 24 ephemeral disks and so the RAID 0 level might not be the most reliable solution. Hence, HS1 has been evaluated using two additional RAID levels: RAID 6 and 10. These experiments basically write and read a single data file of 8 GB on *xfs*, and the performance results are shown for a certain block size (i.e., the transfer size of each underlying I/O operation at the file system level) varying from 16 KB up to 16 MB, which is the maximum value supported by IOzone. Finally, the Linux buffer cache was bypassed using direct I/O (O_DIRECT flag) in order to get the real performance (without buffering that might distort the results) of the underlying storage devices.

The results using a single ephemeral disk show that the SSD-based device of the HI1 instance significantly outperforms the rest of instances, especially from 64 KB on, obtaining up to 3 times (for writing, achieving 545 MB/s) and up to 4 times (for reading, achieving 960 MB/s) higher performance and productivity. However, CR1, which also provides SSD-based disks, obtains poor results compared to HI1, and only slightly better than HS1 for the read operation. The CC2 instance type gets the worst results in terms of performance, but it can be a competitive option, at least compared to CR1 and HS1, when taking into account the incurred costs, especially for the write operation.

Regarding software RAID results, HI1 almost doubles the performance of a single ephemeral disk obtaining up to 1060 and 1620 MB/s for writing and reading, respectively. However, HS1 now obtains the maximum bandwidth results achieving up to 2200 MB/s for both operations when using the RAID 0 level and a block size larger than 1 MB, thus taking full advantage of the 24 available disks. In terms of write productivity, HS1 using RAID 0 and the largest block sizes ($\geq$ 4 MB) is again the best option, despite the fact that it is the most expensive instance ($4.6 per hour), whereas its read productivity is slightly lower than HI1. Nevertheless, the RAID 6 level imposes a high performance penalty for the write operation as only 200 MB/s are achieved. Therefore, the RAID 10 level is the midpoint between performance and reliability for HS1, obtaining write bandwidth results very similar to those of HI1, but up to 30% lower productivity. Finally, CC2 and CR1, which cannot rival previous instances for large block sizes, obtain very similar results, which allows CC2 to be more productive than CR1 due to its lower price ($2 vs $3.5 per hour).

As main conclusions, these results have revealed that: (1) HI1 instances clearly provide the best performance and productivity when using a single ephemeral disk. (2) Using RAID 0, the HS1 instance is clearly the best performer thanks to the availability of up to 24 ephemeral disks. However, if the storage reliability is

[3]Chunk size is defined as the smallest "atomic" amount of data that is written to the storage devices.

TABLE 2. Selected benchmarks, kernels and applications

Name	Type	Description
IOzone [60]	I/O Benchmark	IOzone is a popular open-source file system benchmarking tool used in several works [68, 69, 70] that generates and measures a variety of file system operations. It has been ported to many machines and operating systems.
IOR [61]	Parallel I/O Benchmark	IOR is a widely extended benchmarking tool for evaluating parallel I/O performance, as done in [40, 41, 70, 71, 72, 73, 74, 75]. It is highly parameterized and allows to mimic a wide variety of I/O patterns, supporting different APIs (e.g., MPI-IO, HDF5).
Intel HiBench [62]	Benchmark / Application Suite	HiBench is a representative benchmark suite for Hadoop, used for instance in [76, 77, 78]. It consists of a set of typical Hadoop workloads (e.g., Sort, WordCount), HDFS micro-benchmarks (e.g., DFSIO) as well as web search (PageRank), machine learning (K-means) and data analytics (e.g., Hive queries such as Aggregation) real-world applications.
NPB BT-IO [63]	Parallel I/O Benchmark	The NPB suite [67] is the de-facto standard to evaluate the performance of parallel computers, as done in [70, 74, 79, 80, 81] for I/O. BT-IO extends the NPB BT kernel by adding support for periodic solution checkpointing using the MPI-IO interface.
FLASH-IO [64]	Parallel I/O Benchmark	FLASH-IO is a widely used benchmark, e.g. in [50, 75, 81, 82], that mimics the I/O pattern of FLASH [83], a parallel hydrodynamics application. It recreates the primary data structures in FLASH and produces a checkpoint file using the parallel HDF5 library.

critical it will be more reasonable to use the RAID 10 level, which provides comparable performance to HI1 but lower productivity. (3) Although CC2 performance is usually among the worst ones, it achieves competitive productivity results using RAID 0 and a block size ≤ 1 MB, as it is the cheapest instance under evaluation. And (4) the poor performance results obtained by CR1, together with its high price, low productivity and reduced storage capacity (240 GB) points out that it is not a good choice for data-intensive applications, which has led us to discard its evaluation in the remainder of the paper for clarity purposes.

5.2. Performance Analysis at the Cluster File System Level

Two representative file systems have been selected for this analysis: (1) OrangeFS, a parallel file system widely extended in HPC clusters (Section 5.2.1), and (2) HDFS, which is the distributed file system especifically designed for the Hadoop MapReduce computing framework (Section 5.2.2).

5.2.1. Parallel File System Performance

Figures 4 and 5 present the aggregated bandwidth of OrangeFS (top graphs) for write and read operations, respectively, using the MPI-IO interface. These results have been obtained using the IOR benchmark with a baseline cluster configuration that consists of 4 instances acting as I/O servers (i.e., storage nodes of instance type CC2, HI1 or HS1) and multiple instances acting as clients (i.e., compute nodes of instance type only CC2 or HI1), which access the server data through the 10 Gigabit Ethernet network. In these experiments, the number of cores in the cluster has been set to 128, which determines the number of compute nodes being used depending on the client instance type. Hence, each client instance runs 8 (on HI1) or 16 (on CC2) parallel processes (i.e., one MPI process per core), writing and reading collectively a single shared file of 32 GB under different block sizes (from 64 KB to 16 MB), thus simulating the aforementioned N-1 access pattern. For clarity purposes, the graphs only present experimental results using RAID configurations on the storage instances, as they maximize their performance. Furthermore, the usage of RAID in HPC environments is the common rule as it allows to increase performance and/or redundancy. Therefore, the RAID 0 level for CC2 and HI1 storage instances and RAID 10 for HS1 have been selected for this benchmarking.

As shown in Table 3, four different cluster configurations have been evaluated: (1) using 4 CC2 servers and 8 CC2 clients (labeled as CC2-CC2); (2) 4 HI1 servers and 8 CC2 clients (HI1-CC2); (3) 4 HI1 servers and 16 HI1 clients (HI1-HI1); and (4) 4 HS1 servers and 8 CC2 clients (HS1-CC2). Note that the HS1-HS1 configuration (i.e., 4 HS1 servers and 16 HS1 clients) has not been included for clarity purposes, as it provides similar performance than HI1-HI1 but incurring significantly higher costs. The use of CC2 instances as clients in the heterogeneous cluster deployments (i.e., HI1-CC2 and HS1-CC2) has been motivated for their higher number of cores and computational power compared to HI1 and HS1 instances: 16 vs 8 cores and 88 vs 35 ECUs. In addition, their lower price significantly reduces the overall cost of

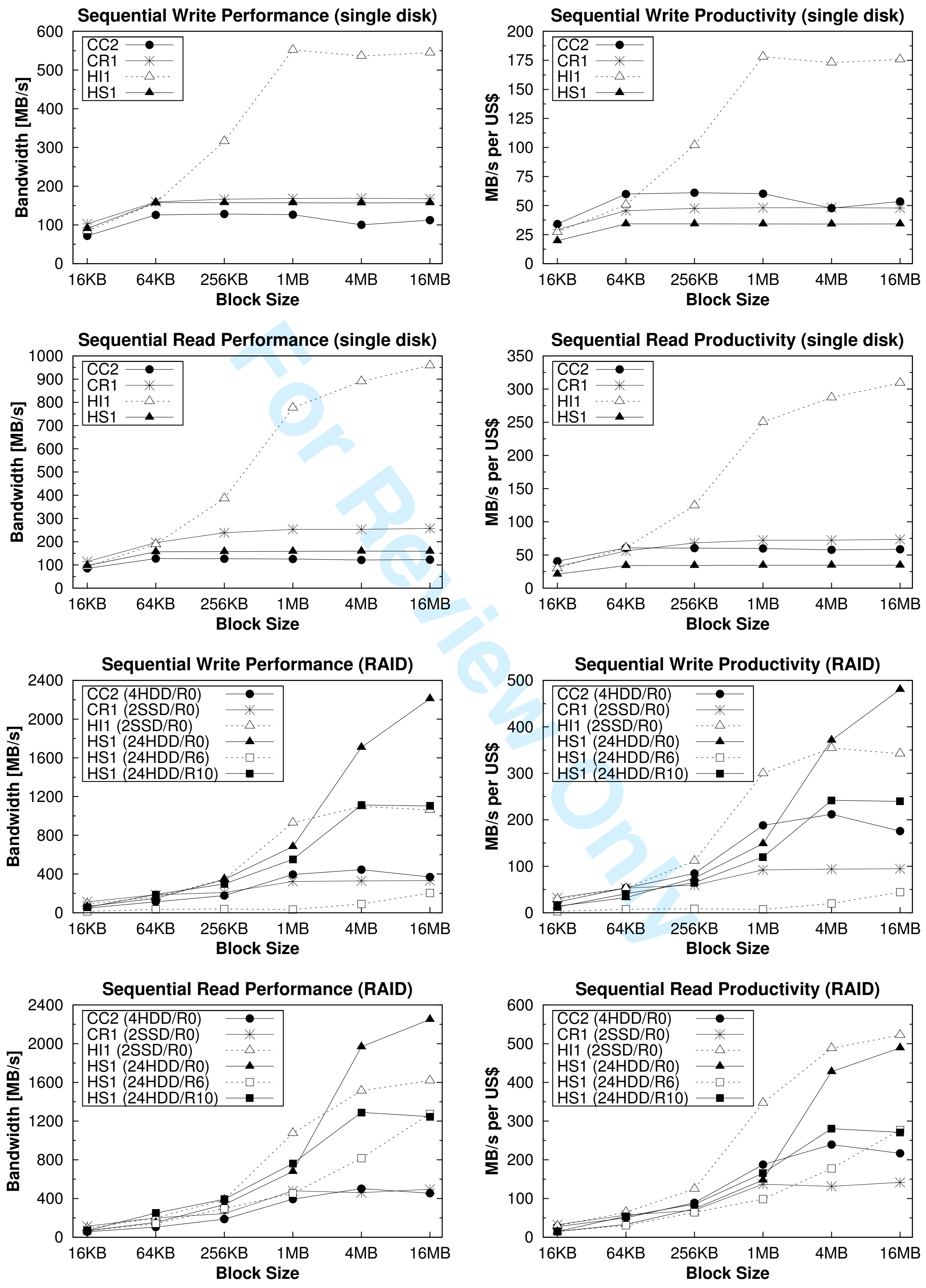


FIGURE 3. Sequential performance and productivity (performance/cost ratio) of ephemeral disks using an 8 GB file

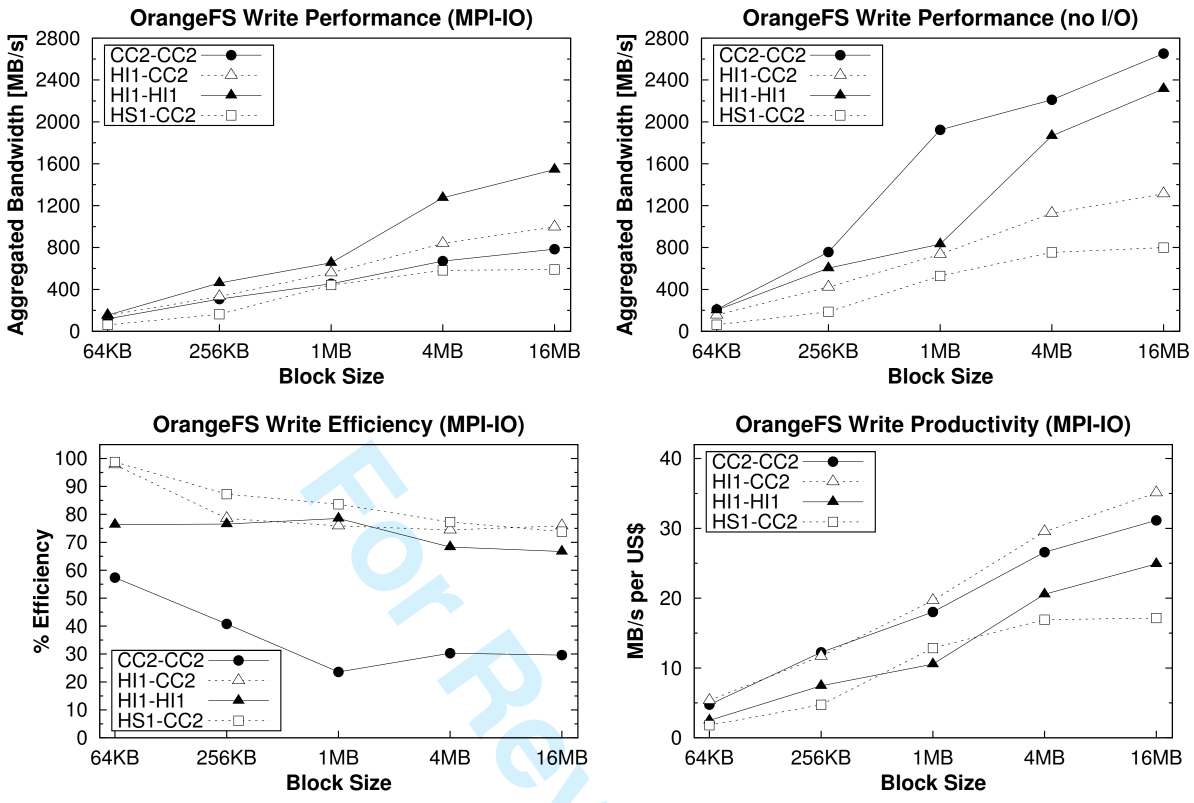


FIGURE 4. OrangeFS write performance, efficiency and productivity using 4 I/O servers and 128 cores

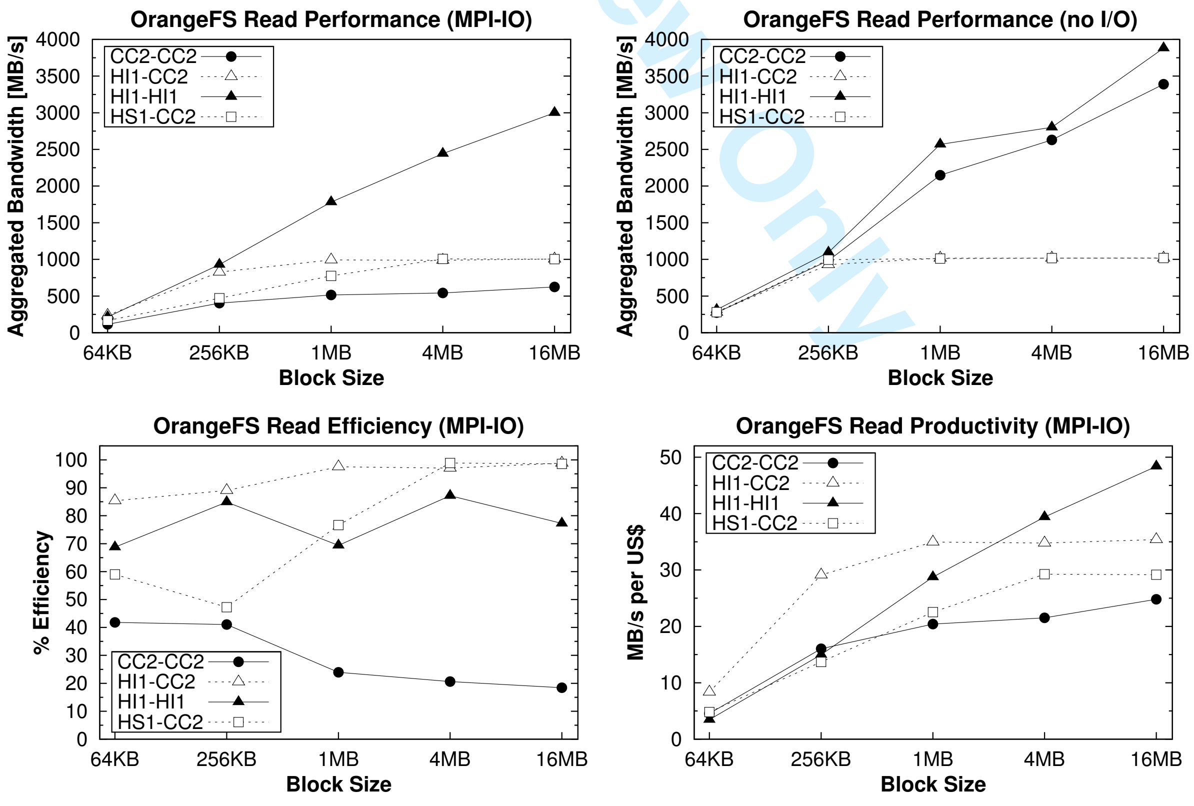


FIGURE 5. OrangeFS read performance, efficiency and productivity using 4 I/O servers and 128 cores

TABLE 3. Hourly cost using on-demand instances of the EC2-based HPC clusters

HPC Cluster	#I/O Server Instances	#Client Instances	#Compute Cores	Hourly Cost
CC2-CC2	4 × CC2	8 × CC2	128	$24
HI1-CC2	4 × HI1	8 × CC2	128	$28.4
HI1-HI1	4 × HI1	16 × HI1	128	$62
HS1-CC2	4 × HS1	8 × CC2	128	$34.4

the cluster deployment on Amazon EC2, which favors their popularity for HPC applications.

Furthermore, the use of storage-optimized instances as clients does not take advantage of the locally attached ephemeral disks. However, the heterogeneous cluster deployments (e.g., HI1-CC2) cannot benefit from being located in the same placement group, which can cause a loss in network performance. In order to analyze the impact of locating nodes outside the same placement group, the "null-aio" TroveMethod[4] available in OrangeFS has been used. This method consists of an implementation that does no disk I/O, and is useful to test the performance of the underlying interconnection network. Hence, the top right graphs in Figures 4 and 5 show the aggregated network bandwidth using the "null-aio" method (labeled as "no I/O"), whereas the top left graphs have been obtained with the default method ("alt-aio", which uses an asynchronous I/O implementation). The "null-aio" method also allows to analyze each cluster configuration in terms of the efficiency metric, shown in the bottom left graphs, which has been defined as the aggregated network bandwidth divided by the aggregated bandwidth when doing I/O. Finally, the bottom right graphs show the productivity results in terms of aggregated bandwidth per invoiced US$, taking into account the hourly cost of the different EC2-based HPC clusters (see the last column of Table 3).

Regarding write performance results (see the top left graph in Figure 4), it can be clearly observed that the use of the largest block size (16 MB) is key to achieve high-performance parallel I/O, mainly using HI1 instances. HI1 server-based configurations achieve the best results, around 1600 and 1000 MB/s for HI1-HI1 and HI1-CC2, respectively, whereas CC2-CC2 obtains only 800 MB/s, although it outperforms the HS1-CC2 configuration (600 MB/s). These results can be explained by the top right graph, where HI1-CC2 and HS1-CC2 obtain the poorest network bandwidths, which significantly limit their overall storage performance. As mentioned before, it is not possible to locate different instance types in the same placement group, so when using CC2 clients together with HI1 and HS1 servers the network performance drops severely. However, the HI1-CC2 cluster achieves slightly higher network bandwidth than HS1-CC2, which can suggest that the physical location of the HI1 and HS1 instances with respect to the clients (CC2) could be different inside the same EC2 region (*us-east-1* in these experiments). Here, the CC2-CC2 cluster achieves the highest network bandwidth (up to 2650 MB/s), but very poor efficiency (around 30%) when using the largest block sizes, as shown in the bottom left graph, because of its limited underlying storage performance. However, the remaining configurations achieve above 65%, being HS1-CC2 the most effective configuration. Taking costs into account (see the bottom right graph), HI1-CC2 becomes the best configuration from 256 KB on, owing to its relatively high performance and the use of client instances that are cheaper than the HI1-HI1 configuration. Even CC2-CC2 seems to be a good choice instead of HI1-HI1, as it is the cheapest cluster under evaluation. Finally, HS1-CC2 is obviously the worst option in terms of productivity due to its high price and low performance, especially when using block sizes > 1 MB.

Regarding read performance (see the top left graph in Figure 5), HI1-HI1 is again the best performer, up to 3000 MB/s of aggregated bandwidth using the largest block size. In this case, HI1-CC2 and HS1-CC2 achieve similar results from 1 MB on, around 1000 MB/s, clearly limited by the interconnection network, as shown in the top right graph. Here, HS1-CC2 requires a large block size (≥ 4 MB) to exploit the underlying network bandwidth, whereas HI1-CC2 does not. The CC2-CC2 configuration presents the worst performance with 625 MB/s, although it obtains up to 3400 MB/s of network bandwidth, showing again very poor efficiencies (below 25%, see the bottom left graph). HI1-HI1 achieves between 70 and 85% of the available network bandwidth, although the maximum efficiency is obtained by HI1-CC2 and HS1-CC2, especially when using block sizes > 1 MB (nearly 98%). This fact allows HI1-CC2 to obtain the best productivity up to a block size of 1 MB (see the bottom right graph). From 4 MB on the highest-performance and most expensive HI1-HI1 configuration offsets its price. CC2-CC2 gets the worst productivity from 1 MB on, even below HS1-CC2.

To sum up, these results have shown that: (1) the HI1-HI1 configuration is the best performer for both write and read operations using any block size; (2) the overall performance of heterogeneous cluster deployments (i.e., HI1-CC2 and HS1-CC2) is severely limited by the network bandwidth, as their instances cannot be located in the same placement group; and (3) taking costs into account, the HI1-CC2 configuration

[4]The TroveMethod parameter specifies how both metadata and data are stored and managed by the OrangeFS servers.

achieves the best productivity, except when reading block sizes > 1 MB, a scenario where HI1-HI1 is the best configuration.

5.2.2. Distributed File System Performance

Figure 6 shows the aggregated bandwidth of HDFS (left graphs) for write and read operations using a baseline cluster that consists of one instance acting as master node (running JobTracker/NameNode) and multiple instances acting as slave nodes (running TaskTrackers/DataNodes), all connected to the 10 Gigabit Ethernet network. As mentioned in Section 4.2, these results have been obtained using the Enhanced DFSIO benchmark included in the Intel HiBench suite, writing and reading 512 files of 512 MB (i.e., 256 GB in total). In these experiments, two different cluster sizes have been evaluated using 8 and 16 slave instances, together with the master node of the same instance type. Therefore, all the instances are located in the same placement group, thus relying on homogeneous cluster deployments. In addition, the right graphs show the productivity results in terms of the aggregated bandwidth per US$, according to the hourly cost of the different EC2-based Hadoop clusters considered in this analysis (shown in the last column of Table 4).

Moreover, each DataNode instance (i.e., slave node) uses all the available locally attached ephemeral disks (e.g., 24 disks for HS1 instances) to store HDFS data. The ephemeral disks have been configured as a Just a Bunch Of Disks (JBOD) (i.e., each disk is accessed directly as an independent drive [5]). This is the most recommended setting for Hadoop clusters, as HDFS does not benefit from using RAID for the DataNode storage. The redundancy provided by RAID is not needed since HDFS handles it by replication between nodes. Furthermore, the RAID 0 level, which is commonly used to increase performance, turns out to be slower than the JBOD configuration, as HDFS uses round-robin allocation of blocks and data across all the available disks.

Regarding Hadoop settings, several configurations have been tested in order to choose the one with the best observed performance. However, the main goal of these experiments focuses on the comparison between different instance types for running Hadoop-based clusters on Amazon EC2 under common setting rules. Therefore, our setup is not intended as a guide to optimize the overall Hadoop performance, which is usually highly workload-dependent. Taking this into account, our main Hadoop settings are as follows: (1) the block size of HDFS is set to 128 MB. (2) The replication factor of HDFS is set to 2. (3) The I/O buffer size is set to 64 KB. (4) The number of map/reduce tasks that are allowed to run in each TaskTracker instance (i.e., slave node) is configured as shown in Table 4. This configuration follows the widespread rule saying that the number of map and reduce tasks should be set to the number of available physical cores in the TaskTracker instance, considering that the DataNode and TaskTracker processes would use 2 hyper-threaded cores. (5) The ratio of map/reduce tasks is 3:1. (6) The available memory to use while sorting files is set to 200 MB. And (7) the compression of the intermediate output data during the map phase is enabled using the *Snappy* codec [85]. This reduces the network overhead as the map output is compressed before being transferred to the reducers.

Concerning write performance results (see the top left graph in Figure 6), the use of 8-slave clusters shows similar HDFS bandwidths, around 1400 MB/s, and thus the CC2-based cluster is clearly the most cost-effective in terms of productivity, as shown in the top right graph. However, storage-optimized instances (i.e., HI1 and HS1) obtain 34% and 44% more aggregated bandwidth, respectively, than CC2 instances when using 16-slave clusters. Nevertheless, CC2 remains as the most productive choice, although followed closely by HI1, whereas the HS1-based cluster, which is the most expensive, remains as the least competitive. Note that while the storage-optimized instances have slightly increased their productivity when doubling the number of slaves, CC2 has decreased 27%. Regarding read results (see the bottom left graph), HS1 obtains the highest bandwidth both for 8 and 16 slaves, around 2920 and 5630 MB/s, respectively. These results are 9% and 70% higher than HI1 and CC2 bandwidths using 8 slaves and 7% and 80% higher using 16 slaves, respectively. Nevertheless, the high performance of the HS1 cluster is not enough to be the best option in terms of productivity due to its high price, even being outperformed by CC2 (see the bottom right graph), the worst cluster in terms of read performance. Here, HI1 is clearly the best choice when taking into account the incurred costs, achieving up to 13% higher productivity than CC2. In this case, all the configurations are able to maintain (or even to increase in the case of HI1) their productivity when doubling the number of slaves.

Another interesting comparison can be conducted taking into account the number of map and reduce tasks. Hence, an 8-slave CC2-based cluster provides the same capacity in terms of map/reduce tasks (i.e., 96/32) as a 16-slave cluster using storage-optimized instances, as can be seen in Table 4. This fact is due to the different number of physical cores provided by each instance type (16 vs 8 cores for CC2 and storage-optimized instances, respectively). For the write operation, the 16-slave HS1-based cluster doubles the performance of the 8-slave CC2-based cluster, but significantly incurring higher costs (the same would apply to HI1). For the read operation, the storage-optimized instances obtain up to 3 times higher bandwidth than CC2, which allows the HI1-based cluster to be the most productive in this case.

[5] We have specified a storage directory for each ephemeral disk using the *dfs.data.dir* property of Hadoop.

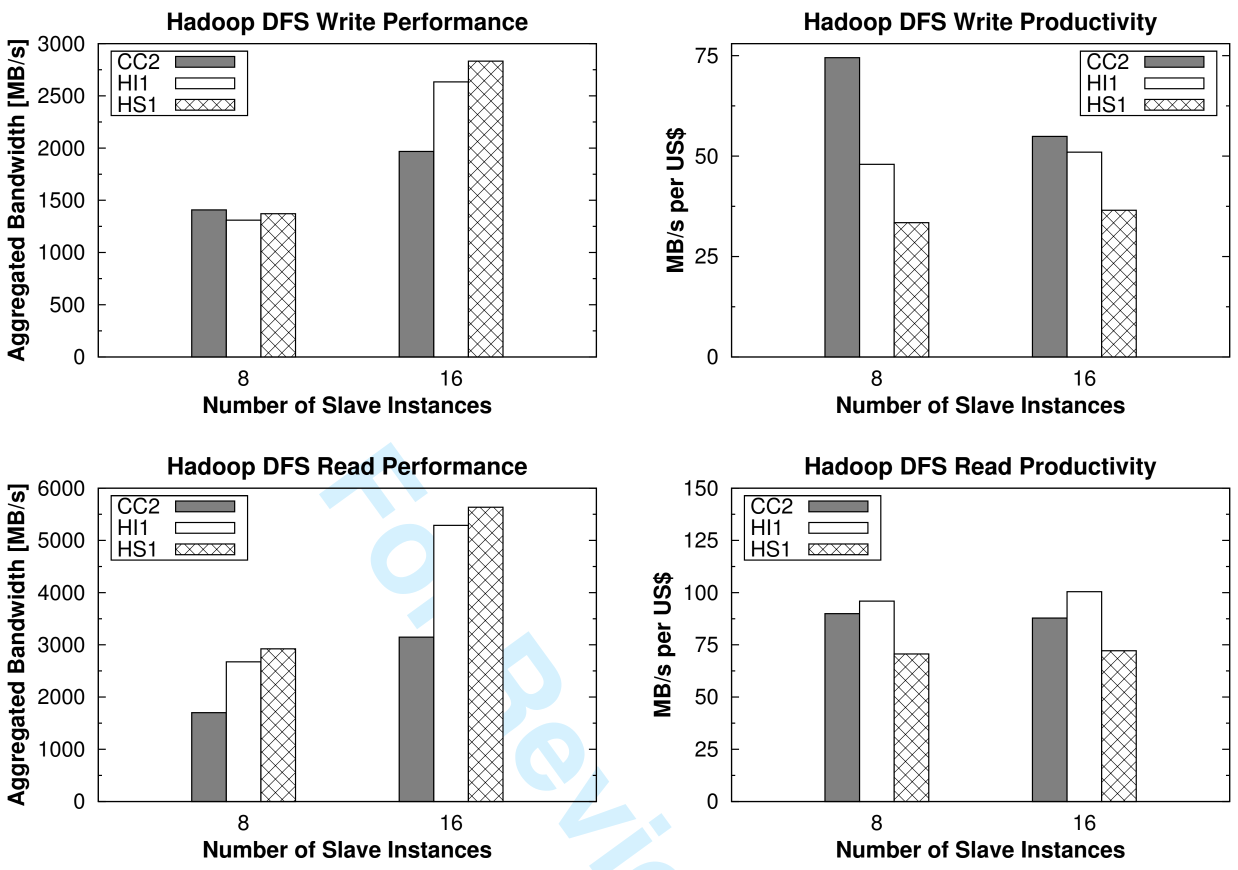


FIGURE 6. HDFS performance and productivity using 8 and 16 slave instances

TABLE 4. Number of map/reduce tasks and hourly cost using on-demand instances of the EC2-based Hadoop clusters

Hadoop Cluster	#Map/Reduce tasks per slave node (8 - 16 slaves)	Hourly Cost (8 - 16 slaves)
CC2-based	12/4 (96/32 - 192/64)	\$18 - \$34
HI1-based	6/2 (48/16 - 96/32)	\$27.9 - \$52.7
HS1-based	6/2 (48/16 - 96/32)	\$41.4 - \$78.2

The main conclusions that can be drawn from these results are: (1) the HS1-based cluster is the best option in terms of HDFS performance, especially when using 16 slaves, but followed closely by HI1; (2) the high price of the HS1 instance discourages its use, favoring HI1 instances as the preferred choice when costs are taken into account without trading off much performance; and (3) comparing clusters in terms of the same map/reduce capacity, the storage-optimized instances provide up to 2 and 3 times higher write and read bandwidths, respectively, than CC2 instances.

5.3. Data-intensive Parallel Application Performance

The performance of two I/O-intensive HPC applications (Section 5.3.1) and four real-world MapReduce workloads (Section 5.3.2) has been analyzed at the application level. As mentioned in Section 4.2, the BT-IO kernel from the NPB suite and the FLASH-IO code have been selected for the HPC software stack evaluation. In addition, the Sort, WordCount, Mahout PageRank and Hive Aggregation workloads have been selected for the Big Data counterpart. Finally, the performance variability of the evaluated workloads is briefly discussed in Section 5.3.3.

5.3.1. I/O-Intensive HPC Applications

The NPB BT kernel solves systems of block-tridiagonal equations in parallel. BT-IO extends the BT kernel by adding support for periodic solution checkpointing using the MPI-IO interface. In these experiments, the NPB class C workload was selected, whereas the I/O size was the "full" subtype. With these settings, all the parallel processes append data to a single file through 40 collective MPI-IO write operations, resulting in an output data file sized around 6.8 GB.

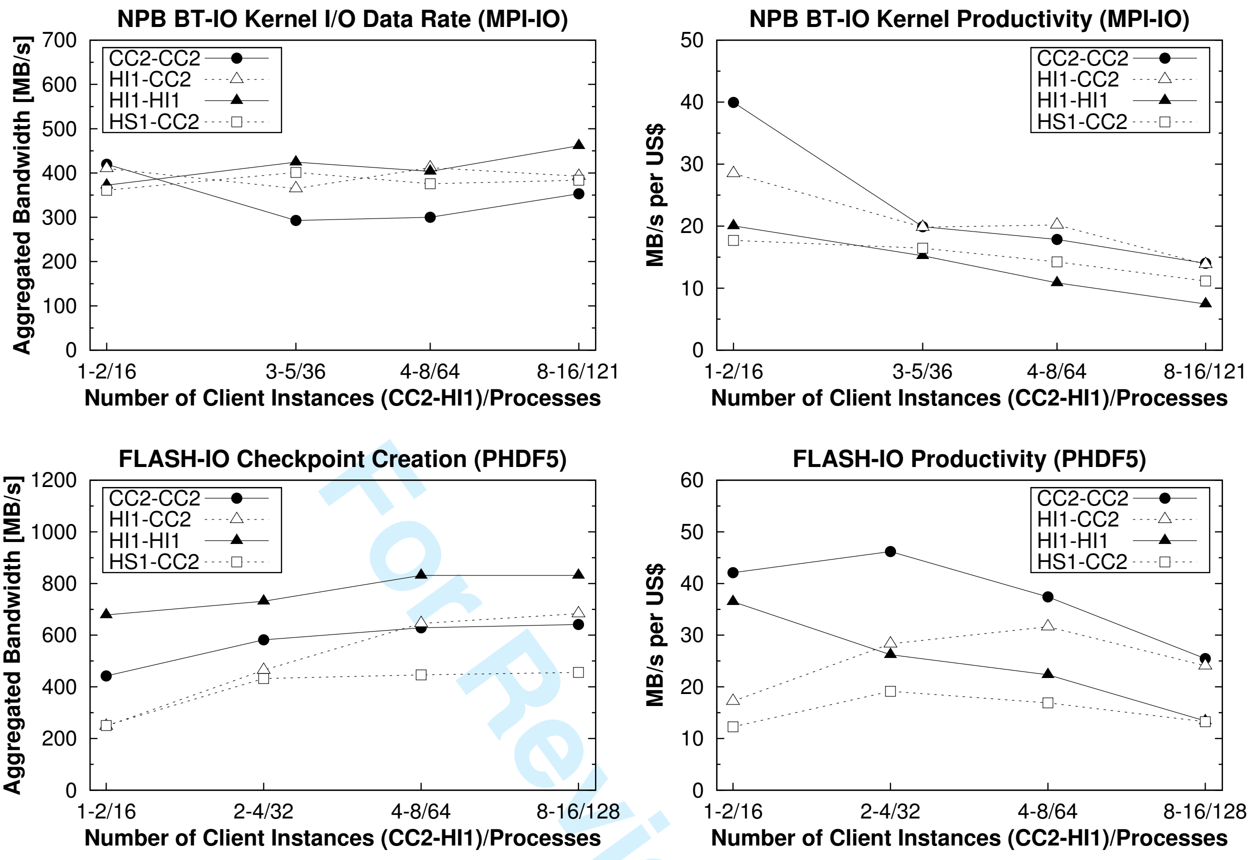


FIGURE 7. NPB BT-IO and FLASH-IO results using OrangeFS with 4 I/O servers

The default I/O frequency, which consists of appending data to the shared output file every 5 computation time steps, was used. Note that this kernel requires that the number of client processes be square numbers. The FLASH-IO kernel simulates the I/O pattern of FLASH [83], a parallel hydrodynamics application that simulates astrophysical thermonuclear flashes in two or three dimensions. The parallel I/O routines of the FLASH-IO code are identical to those used by FLASH, so their performance closely reflects the full parallel I/O performance of the application. This kernel recreates the primary data structures in the FLASH application and produces a checkpoint file sized around 7.6 GB (for 128 clients) using PHDF5, together with smaller plotfiles for visualization and analysis. Finally, the experiments have been conducted under the same cluster configurations previously used in the parallel file system evaluation (see Table 3 in Section 5.2.1).

Figure 7 presents NPB BT-IO (top graphs) and FLASH-IO (bottom graphs) performance and productivity results (left and right graphs, respectively). Regarding BT-IO results, HI1- and HS1-server based configurations achieve the highest bandwidths from 36 processes on, being the HI1-HI1 cluster the best performer in accordance with the previous parallel file system evaluation for the write operation (see the top left graph in Figure 4), obtaining up to 30% higher bandwidth than the CC2-CC2 cluster (460 MB/s vs 350 MB/s for 121 processes). Nevertheless, HI1-CC2 is the configuration with the highest productivity from 36 processes on, as happened before (see the bottom right graph in Figure 4). Here, the performance advantage obtained by HI1-HI1 is not enough to be an interesting cost-effective option, even being outperformed by HS1-CC2. Note that this code is also a computation- and communication-intensive application that performs lots of MPI communication calls, increasing the communication overhead with the number of processes. Hence, MPI communications across client instances share the interconnection network with MPI-IO communications (i.e., I/O operations) between client and server instances. This sharing reduces the available network bandwidth, which limits parallel I/O performance. This limitation becomes more noticeable for CC2-client based configurations, which run 16 parallel processes per instance, due to the poorer ratio between CPU power and network bandwidth. Furthermore, the computation phase of this kernel allows to overlap I/O with computation. This fact boosts the slowest storage configuration for writing, which was HS1-CC2 according to the parallel file system evaluation (see the top left graph in Figure 4), now obtaining more competitive performance results.

Regarding FLASH-IO results, HI1-HI1 shows again the best performance, obtaining more than 830 MB/s for 64 and 128 processes, up to 23% and 30% higher aggregated bandwidth than HI1-CC2 and CC2-CC2 clusters, respectively. The HS1-CC2 cluster gets stucked at 455 MB/s from 32 processes on, being the worst performer as occurred before in the parallel file system evaluation for the write operation. Thus, the underlying network bandwidth significantly limits its overall I/O performance. Note that the FLASH-IO code simply creates the checkpoint file using PHDF5, which is implemented on top of MPI-IO. This means that, unlike the BT-IO kernel, this code is neither computation-intensive nor makes an extensive use of MPI communication routines. Therefore, the FLASH-IO results are more in tune with the write performance at the parallel file system level than the BT-IO results, but using a high-level library (PHDF5) instead of using directly the MPI-IO interface. In terms of productivity, CC2-CC2 becomes the best option, but closely followed by the HI1-CC2 cluster when 64 or more processes are used. HI1-HI1 shows again poor productivity, especially from 32 processes on, which makes it a bad choice when costs are taken into account.

The main conclusions of this evaluation are: (1) in terms of performance, HI1-HI1 remains as the best option for I/O-intensive HPC applications, in accordance with the previous evaluation at the parallel file system level; and (2) in terms of productivity, it is generally advisable to use CC2 instances as clients in order to save costs when using a high number of cores, and combine them with HI1 or CC2 as storage servers to maximize performance.

5.3.2. MapReduce-based Workloads

Figure 8 shows performance (left graphs) and cost results (right graphs) of the selected Hadoop applications from the Intel HiBench suite (see details in Section 4.2). In this case, these workloads do not report any aggregated bandwidth metric as output. Hence, the performance metric reported is the time (in hours) required to run 1,000 executions of each evaluated application. Consequently, the productivity metric reported is the total execution cost in US$, which represents the cost of having each Hadoop cluster running during the number of hours required to complete the 1,000 executions for each experiment. The experiments have been conducted under the same Hadoop settings and cluster configurations previously used in the distributed file system evaluation (see Table 4 in Section 5.2.2). The specific workloads for these applications have been configured as summarized in Table 5. This table shows the total data that are read and written from and to HDFS, which represents the job/map input and job/reduce output, respectively. The last column of the table shows the total data that are transferred during the shuffle phase (i.e., data from output mappers that are read by reducers). As mentioned in Section 5.2.2, these intermediate data are compressed before being transferred, thus reducing the network overhead.

Since the Sort workload transforms data from one representation to another, the job output has the same size as the job input (102 GB), as can be seen in Table 5. Although this workload is I/O bound in nature, it also presents moderate CPU utilization during the map and shuffle phases, especially due to the intermediate data compression, and low CPU utilization and heavy disk I/O during the reduce phase. In addition, considering the large amount of shuffle data (53 GB), even though compressed, it is expected to be the most network-intensive workload under evaluation. Results using 8-slave clusters show that CC2 is able to outperform HI1 and HS1 configurations by 20%, thanks to the higher map/reduce capacity of the CC2 cluster due to the availability of more CPU resources, which seem to be of great importance, especially during data compression. However, CC2 only reduces its execution time by 22% when doubling the number of slaves, whereas HI1 and HS1 reduce it by 40% and 42%, respectively, but far from linear scalability due to the intensive use of the network. This allows storage-optimized instances to slightly outperform CC2 when using 16 slaves, due to their higher HDFS bandwidth, even taking into account that they have half the map/reduce capacity of CC2 (see Table 4). Nevertheless, the cost of the CC2 cluster continues to be the lowest, a pattern that is maintained in the remaining workloads. If the comparison is done using the same map/reduce capacity, the 16-slave clusters using HI1 and HS1 instances outperform the 8-slave CC2 cluster by 24% and 27%, respectively. However, these performance improvements are not enough to turn optimized-storage instances into interesting options when considering the associated costs.

The shuffle data (12.7 MB) and job output (25.5 KB) of WordCount are much smaller than the job input (102 GB), as this application extracts a small amount of information from a large input data set. Consequently, the WordCount workload is mostly CPU bound, especially during the map phase, having very high CPU utilization and light disk/network I/O. Therefore, the CC2 instance type, which provides the largest CPU resources, shows the best performance and productivity results both for 8- and 16-slave clusters. For instance, the 8-slave CC2 cluster achieves up to 1.9 and 2.1 times higher performance than HI1 and HS1 clusters, respectively. These numbers are reduced to 1.7 and 1.8 times when 16-slave clusters are used. Therefore, CC2 obtains again lower execution time reduction (42%) when doubling the number of slaves compared to HI1 (46%) and HS1 (50%, i.e. linear scalability). In this case, the use of 16-slave storage-

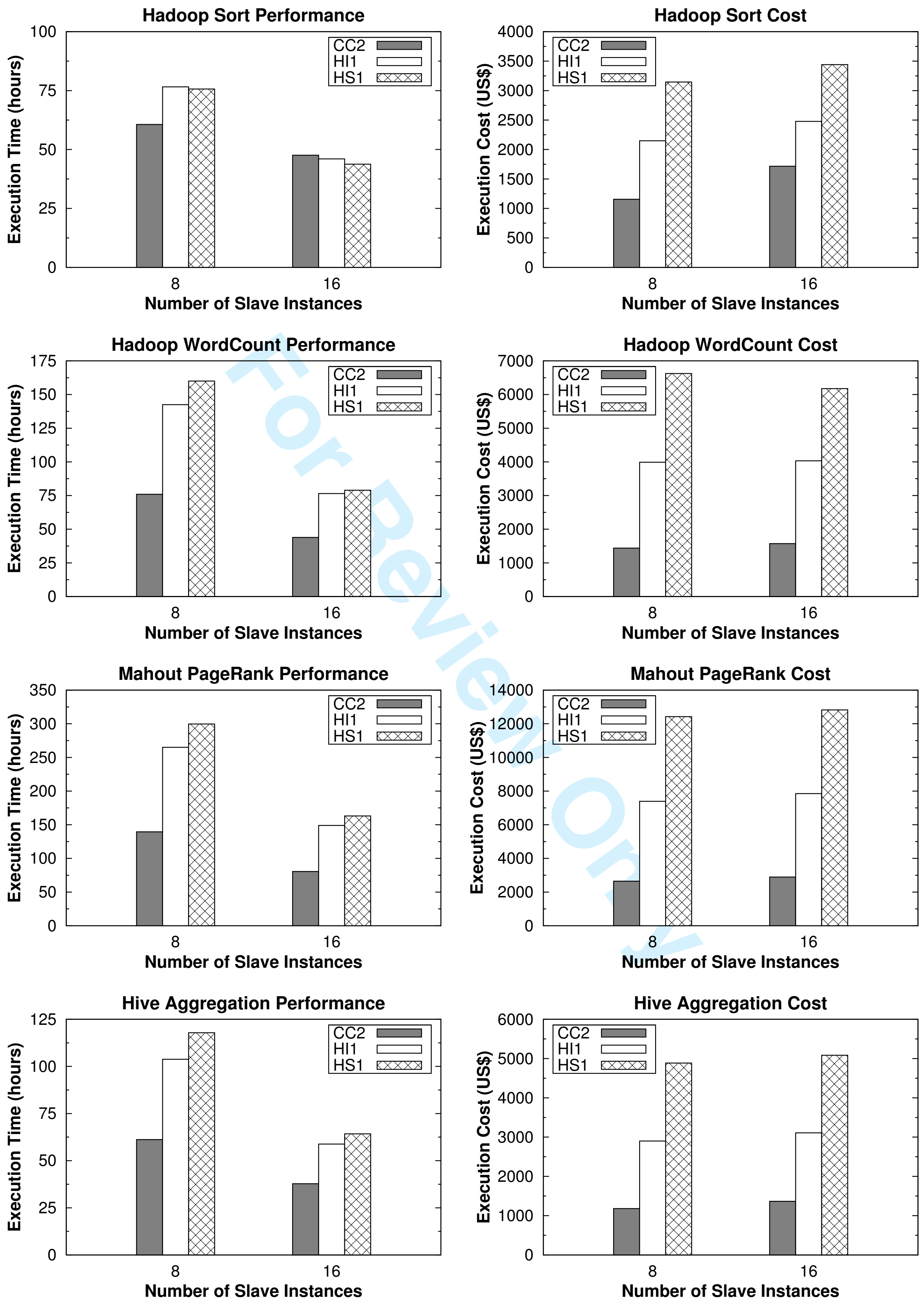


FIGURE 8. Hadoop performance and cost results using 8 and 16 slave instances

TABLE 5. Data size for map, reduce and shuffle phases

Workload	Job/Map Input (HDFS Read)	Job/Reduce Output (HDFS Write)	Shuffle Data
Sort	102 GB	102 GB	53 GB
WordCount	102 GB	25.5 KB	12.7 MB
PageRank	89.8 GB	49.6 GB	4.6 GB
Aggregation	75 GB	14.3 GB	7.5 GB

optimized clusters does not outperform the 8-slave CC2 cluster, as occurred for Sort. In fact, the performance of CC2 using 8 slaves is similar to that of HI1 and HS1 clusters using 16 slaves, but having 2.8 and 4.3 times lower costs, respectively.

The Mahout PageRank and Hive Aggregation workloads share similar characteristics. They are more CPU bound during the map phase and more disk I/O bound during the reduce phase, mostly due to the output to HDFS (49.6 GB and 14.3 GB, respectively). They also present low to medium network I/O, as the amount of shuffle data is relatively small (4.6 GB and 7.5 GB, respectively). Hence, both workloads show very similar results, also similar to those of WordCount. Consequently, the CC2-based cluster is again the best option, both in terms of performance and cost. For instance, CC2 obtains 2.1 times higher performance and 4.6 times lower cost than HS1 when using 8 slaves for the PageRank workload. Furthermore, it is not worth using 16-slave storage-optimized clusters, which provide the same map/reduce capacity than the 8-slave CC2 cluster, as similar or even lower performance is obtained but incurring significantly higher costs.

These results have shown that: (1) storage-optimized instances do not seem to be the most suitable choice for the evaluated MapReduce workloads. As explained before, the main reason is that these instances provide poorer CPU resources than CC2, both in terms of number of physical cores (8 vs 16) and computational power per core (4.4 vs 5.5 ECUs), as shown in Table 1. (2) This fact encourages the use of the CC2-based cluster which, using the same number of slaves, provides twice the map/reduce capacity of clusters based on storage-optimized instances. (3) The CC2-based cluster usually achieves significantly higher performance and lower cost, which offsets the fact that the CC2 instance type provides lower underlying I/O performance, as shown in Section 5.1. And (4) if the instance types are compared using the same map/reduce capacity (i.e., 8-slave CC2 cluster vs 16-slave HI1 and HS1 clusters), only Sort, the most I/O-intensive workload under evaluation, experiences some performance benefit when using storage-optimized instances, but incurring more costs.

5.3.3. Analysis of Performance Variability

Performance unpredictability in the cloud can be an important issue for researchers because of repeatability of results. In this paper, the main guidelines and hints suggested by Schad et al. [84] have been followed in order to minimize the variance and maximize the repeatability of our experiments in Amazon EC2. Examples of these guidelines are always specifying one availability zone when launching the instances (*us-east-1d*, as mentioned in Section 4.3) and reporting the underlying system hardware of the evaluated instances (see Table 1). In this section, a brief analysis of the performance variability at the application level is presented, as it takes into account the likely variability of all the lower levels (i.e., storage infrastructure, interconnection network, cluster file system, I/O middleware and MapReduce frameworks, and high-level I/O libraries and tools), showing its impact on the performance of real applications.

Figure 9 presents the performance variability for the data-intensive applications evaluated in Sections 5.3.1 (left graph) and 5.3.2 (right graph). These graphs show the measure of the mean value and include error bars to indicate the measure of the minimum sample (bottom arrow) and the maximum sample (top arrow). The results are shown using the largest cluster configuration for each corresponding application (i.e., a 128-core cluster for I/O-intensive HPC applications and a 16-slave Hadoop cluster for MapReduce workloads).

As main conclusion, the evaluated MapReduce workloads generally present negligible variance in their performance (see the right graph), except for the Sort application executed on the CC2 cluster. This variability is significantly lower than the one observed for HPC applications, as shown in the left graph. This is motivated by the fact that these MapReduce workloads are more computationally intensive than the BT-IO and FLASH-IO applications, especially due to the intermediate data compression and map/reduce phases. In fact, as mentioned in Section 5.3.1, BT-IO also presents higher computation intensiveness than FLASH-IO and thus its variability is also lower. Among the evaluated instances, HI1 shows the highest variability, particularly for the BT-IO and FLASH-IO codes, which present a write-only access pattern. This fact can be due to the varying overhead of the background tasks associated with the SSD internal architecture. Hence, the inherent variable write performance of SSD disks, especially when using RAID, is due to the internal garbage collection process that can block incoming requests that are mapped to the flash chips currently performing erase operations [86].

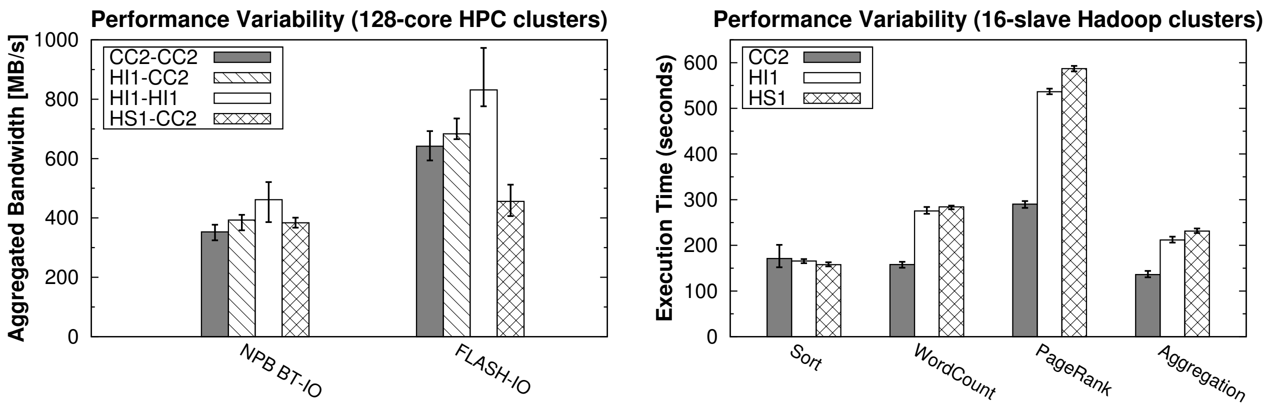


FIGURE 9. Performance variability of data-intensive computing applications

6. CONCLUSIONS

Cloud computing platforms are becoming widely available and are gaining significant popularity in many domains, as a convenient way to virtualize data centers and increase the flexibility in the use of computational resources. Amazon Web Services is the leading commercial public cloud provider, whose EC2 IaaS cloud service provides end users with reliable access to on-demand resources to run their applications. At the same time, scientific research is increasingly reliant on the processing of very large amounts of data. In fact, current data-intensive applications generally demand significant computational resources together with scalable cluster file systems. Public IaaS clouds can satisfy the increasing processing requirements of these applications while offering high flexibility and promising cost savings.

The main contributions of this paper are: (1) extensive assessment of the suitability of using the Amazon EC2 IaaS cloud platform for running data-intensive computing applications; (2) a thorough performance study of the storage-optimized family of EC2 instances, which provide direct-attached storage devices intended to be well suited for applications with specific disk I/O requirements; and (3) a detailed performance, cost and variability analysis of four EC2 instance types that provide 10 Gigabit Ethernet, conducting multiple experiments at several layers and using a representative suite of benchmarking tools (IOzone, IOR, Intel HiBench), cluster file systems (OrangeFS, HDFS), I/O middleware (MPI-IO, HDF5), distributed computing frameworks (Apache Hadoop), I/O-intensive parallel codes for HPC (NPB BT-IO and FLASH-IO) and MapReduce workloads for Big Data analysis (Sort, WordCount, PageRank, Aggregation).

The analysis of the experimental results points out that the unique configurability and flexibility advantage offered by Amazon EC2, almost impossible to achieve in traditional platforms, is critical for increasing performance and/or reduce costs. Hence, this paper has revealed that the suitability of using EC2 resources for running data-intensive applications is highly workload-dependent. Furthermore, the most suitable configuration for a given application heavily depends on whether the main aim is to obtain the maximum performance or, instead, minimize the cost (or maximize the productivity). Therefore, one of the key contributions of this work is an in-depth and exhaustive study that provides guidelines for scientists and researchers to increase significantly the performance (or reduce the cost) of their applications in Amazon EC2. Finally, our main outcomes indicate that current data-intensive applications can benefit from tailored EC2-based virtual clusters, enabling end users to obtain the highest performance and cost-effectiveness in the cloud.

FUNDING

This work was supported by the Ministry of Economy and Competitiveness of Spain and ERDF funds of the European Union [Project TIN2013-42148-P]; by the Galician Government [Project GRC2013/055]; by the FPU Program of the Ministry of Education of Spain [ref. AP2010-4348]; and by an Amazon Web Services (AWS) Research Grant.

REFERENCES

[1] Gorton, I., Greenfield, P., Szalay, A., and Williams, R. (2008) Data-intensive computing in the 21st century. *Computer*, **41**, 30–32.

[2] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009) Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, **25**, 599–616.

[3] Amazon Web Services Inc. `http://aws.amazon.com/`. [Last visited: July 2014].

[4] Google Inc. Google Compute Engine. `https://cloud.google.com/products/compute-engine/`. [Last visited: July 2014].

[5] Microsoft Corporation. Microsoft Azure. http://azure.microsoft.com/. [Last visited: July 2014].

[6] Rackspace Inc. Rackspace Public Cloud. http://www.rackspace.com/. [Last visited: July 2014].

[7] Top 10 cloud platforms of 2013. http://yourstory.com/2013/12/top-10-cloud-platforms-2013/. [Last visited: July 2014].

[8] Amazon Web Services Inc. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2. [Last visited: July 2014].

[9] Top 20 cloud IaaS providers of 2014. http://www.crn.com/slide-shows/cloud/240165705/the-20-coolest-cloud-infrastructure-iaas-vendors-of-the-2014-cloud-100.htm/pgno/0/1. [Last visited: July 2014].

[10] Amazon Web Services Inc. High Performance Computing using Amazon EC2. http://aws.amazon.com/ec2/hpc-applications/. [Last visited: July 2014].

[11] Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J., and Wright, N. J. (2010) Performance analysis of high performance computing applications on the Amazon web services cloud. *Proc. 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10)*, Indianapolis, IN, USA, 30 November–3 December, pp. 159–168. IEEE Computer Society, Washington, DC, USA.

[12] Mauch, V., Kunze, M., and Hillenbrand, M. (2013) High performance cloud computing. *Future Generation Computer Systems*, **29**, 1408–1416.

[13] Napper, J. and Bientinesi, P. (2009) Can cloud computing reach the TOP500? *Proc. Combined Workshops on UnConventional High Performance Computing Workshop plus Memory Access Workshop (UCHPC-MAW'09)*, Ischia, Italy, 18–20 May, pp. 17–20. ACM, New York, NY, USA.

[14] Walker, E. (2008) Benchmarking Amazon EC2 for high-performance scientific computing. *USENIX;login:*, **33**, 18–23.

[15] Dean, J. and Ghemawat, S. (2008) MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, **51**, 107–113.

[16] Amazon Web Services Inc. Amazon EC2 instance types. http://aws.amazon.com/ec2/instance-types/. [Last visited: July 2014].

[17] Apache Hadoop. http://hadoop.apache.org/. [Last visited: July 2014].

[18] Thakur, R., Gropp, W. D., and Lusk, E. (1999) On implementing MPI-IO portably and with high performance. *Proc. 6th Workshop on I/O in Parallel and Distributed Systems (IOPADS'99)*, Atlanta, GA, USA, 5 May, pp. 23–32. ACM, New York, NY, USA.

[19] Roloff, E., Birck, F., Diener, M., Carissimi, A., and Navaux, P. O. A. (2012) Evaluating high performance computing on the Windows Azure platform. *Proc. 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, Honolulu, HI, USA, 24–29 June, pp. 803–810. IEEE Computer Society, Washington, DC, USA.

[20] He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010) Case study for running HPC applications in public clouds. *Proc. 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*, Chicago, IL, USA, 21–25 June, pp. 395–401. ACM, New York, NY, USA.

[21] Expósito, R. R., Taboada, G. L., Ramos, S., Touriño, J., and Doallo, R. (2013) Performance analysis of HPC applications in the cloud. *Future Generation Computer Systems*, **29**, 218–229.

[22] Mehrotra, P., Djomehri, J., Heistand, S., Hood, R., Jin, H., Lazanoff, A., Saini, S., and Biswas, R. (2013) Performance evaluation of Amazon elastic compute cloud for NASA high-performance computing applications. *Concurrency and Computation: Practice and Experience (in press)*, http://dx.doi.org/10.1002/cpe.3029.

[23] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2009) A performance analysis of EC2 cloud computing services for scientific computing. *Proc. 1st International Conference on Cloud Computing (CLOUDCOMP'09)*, Munich, Germany, 19–21 October, pp. 115–131. Springer-Verlag, Berlin, Germany.

[24] Juve, G., Deelman, E., Berriman, G. B., Berman, B. P., and Maechling, P. (2012) An evaluation of the cost and performance of scientific workflows on Amazon EC2. *Journal of Grid Computing*, **10**, 5–21.

[25] Vecchiola, C., Pandey, S., and Buyya, R. (2009) High-performance cloud computing: A view of scientific applications. *Proc. 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN'09)*, Kaoshiung, Taiwan, 14–16 December, pp. 4–16. IEEE Computer Society, Washington, DC, USA.

[26] Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2011) Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems*, **22**, 931–945.

[27] Evangelinos, C. and Hill, C. N. (2008) Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. *Proc. 1st Workshop on Cloud Computing and Its Applications (CCA'08)*, Chicago, IL, USA, 22–23 October, pp. 1–6. ACM, New York, NY, USA.

[28] Shafer, J. (2010) I/O virtualization bottlenecks in cloud computing today. *Proc. 2nd Workshop on I/O Virtualization (WIOV'10)*, Pittsburgh, PA, USA, March 13, pp. 5:1–5:7. USENIX Association, Berkeley, CA, USA.

[29] Ghoshal, D., Canon, R. S., and Ramakrishnan, L. (2011) I/O performance of virtualized cloud environments. *Proc. 2nd International Workshop on Data Intensive Computing in the Clouds (DataCloud-SC'11)*, Seattle, WA, USA, 12–18 November, pp. 71–80. ACM, New York, NY, USA.

[30] Zhai, Y., Liu, M., Zhai, J., Ma, X., and Chen, W. (2011) Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications. *Proc. 23rd ACM/IEEE Supercomputing Conference (SC'11, State of the Practice Reports)*, Seattle, WA, USA, 12–18 November, pp. 11:1–11:10. ACM, New York, NY, USA.

[31] Expósito, R. R., Taboada, G. L., Ramos, S., González-Domínguez, J., Touriño, J., and Doallo, R. (2013)

Analysis of I/O performance on an Amazon EC2 cluster compute and high I/O platform. *Journal of Grid Computing*, **11**, 613–631.

[32] Liu, M., Zhai, J., Zhai, Y., Ma, X., and Chen, W. (2011) One optimized I/O configuration per HPC application: Leveraging the configurability of cloud. *Proc. 2nd ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'11)*, Shanghai, China, 11–12 July, pp. 1–5. ACM, New York, NY, USA.

[33] Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. (2010) MapReduce in the clouds for science. *Proc. 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10)*, Indianapolis, IN, USA, 30 November–3 December, pp. 565–572. IEEE Computer Society, Washington, DC, USA.

[34] Gunarathne, T., Zhang, B., Wu, T.-L., and Qiu, J. (2013) Scalable parallel computing on clouds using Twister4Azure iterative MapReduce. *Future Generation Computer Systems*, **29**, 1035–1048.

[35] Moise, D. and Carpen-Amarie, A. (2012) MapReduce applications in the cloud: A cost evaluation of computation and storage. *Proc. 5th International Conference on Data Management in Cloud, Grid and P2P Systems (Globe 2012)*, Vienna, Austria, 5–6 September, pp. 37–48. Springer-Verlag, Berlin, Germany.

[36] Fadika, Z., Govindaraju, M., Canon, R., and Ramakrishnan, L. (2012) Evaluating Hadoop for data-intensive scientific operations. *Proc. 5th IEEE International Conference on Cloud Computing (CLOUD'12)*, Honolulu, HI, USA, 24–29 June, pp. 67–74. IEEE Computer Society, Washington, DC, USA.

[37] Zhang, C., De Sterck, H., Aboulnaga, A., Djambazian, H., and Sladek, R. (2009) Case study of scientific data processing on a cloud using Hadoop. *Proc. 23rd International Conference on High Performance Computing Systems and Applications (HPCS'09)*, Kingston, ON, Canada, 14–17 June, pp. 400–415. Springer-Verlag, Berlin, Germany.

[38] Elnozahy, E. N., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002) A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, **34**, 375–408.

[39] Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., and Wingate, M. (2009) PLFS: A checkpoint filesystem for parallel applications. *Proc. 21st ACM/IEEE Supercomputing Conference (SC'09)*, Portland, OR, USA, 14–20 November, pp. 21:1–21:12. ACM, New York, NY, USA.

[40] Carns, P. H., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R., and Ross, R. B. (2011) Understanding and improving computational science storage access through continuous characterization. *Proc. 27th IEEE Symposium on Mass Storage Systems and Technologies (MSST'11)*, Denver, CO, USA, 23–27 May, pp. 1–14. IEEE Computer Society, Washington, DC, USA.

[41] Shan, H., Antypas, K., and Shalf, J. (2008) Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. *Proc. 20th ACM/IEEE Supercomputing Conference (SC'08)*, Austin, TX, USA, 15–21 November, pp. 42:1–42:12. IEEE Press, Piscataway, NJ, USA.

[42] Doulkeridis, C. and Norvag, K. (2014) A survey of large-scale analytical query processing in MapReduce. *The VLDB Journal*, **23**, 355–380.

[43] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., and Liu, H. (2010) Data warehousing and analytics infrastructure at Facebook. *Proc. 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*, Indianapolis, IN, USA, 6–10 June, pp. 1013–1020. ACM, New York, NY, USA.

[44] Borthakur, D. et al. (2011) Apache Hadoop goes realtime at Facebook. *Proc. 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*, Athens, Greece, 12–16 June, pp. 1071–1080. ACM, New York, NY, USA.

[45] Ananthanarayanan, R., Gupta, K., Pandey, P., Pucha, H., Sarkar, P., Shah, M., and Tewari, R. (2009) Cloud analytics: Do we really need to reinvent the storage stack? *Proc. 1st Workshop on Hot Topics in Cloud Computing (HotCloud'09)*, San Diego, CA, USA, June 15, pp. 15:1–15:5. USENIX Association, Berkeley, CA, USA.

[46] Tantisiriroj, W., Son, S. W., Patil, S., Lang, S. J., Gibson, G., and Ross, R. B. (2011) On the duality of data-intensive file system design: Reconciling HDFS and PVFS. *Proc. 23rd ACM/IEEE Supercomputing Conference (SC'11)*, Seattle, WA, USA, 12–18 November, pp. 67:1–67:12. ACM, New York, NY, USA.

[47] HDF5 home page. `http://www.hdfgroup.org/HDF5/`. [Last visited: July 2014].

[48] Rew, R. and Davis, G. (1990) NetCDF: An interface for scientific data access. *IEEE Computer Graphics and Applications*, **10**, 76–82.

[49] Parallel HDF5 home page. `http://www.hdfgroup.org/HDF5/PHDF5/`. [Last visited: July 2014].

[50] Li, J., Liao, W.-K., Choudhary, A., Ross, R. B., Thakur, R., Gropp, W. D., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003) Parallel netCDF: A high-performance scientific I/O interface. *Proc. 15th ACM/IEEE Supercomputing Conference (SC'03)*, Phoenix, AZ, USA, 15–21 November, pp. 39:1–39:11. ACM, New York, NY, USA.

[51] Schmuck, F. and Haskin, R. (2002) GPFS: A shared-disk file system for large computing clusters. *Proc. 1st USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, 28–30 January, pp. 231–244. USENIX Association, Berkeley, CA, USA.

[52] Carns, P. H., Ligon III, W. B., Ross, R. B., and Thakur, R. (2000) PVFS: A parallel virtual file system for Linux clusters. *Proc. 4th Annual Linux Showcase & Conference (ALS'00)*, Atlanta, GA, USA, 10–14 October, pp. 317–328. USENIX Association, Berkeley, CA, USA.

[53] Lustre File System. `http://www.lustre.org/`. [Last visited: July 2014].

[54] Orange File System (OrangeFS). `http://www.orangefs.org/`. [Last visited: July 2014].

[55] Chattopadhyay, B., Lin, L., Liu, W., Mittal, S., Aragonda, P., Lychagina, V., Kwon, Y., and Wong, M. (2011) Tenzing a SQL implementation on the MapReduce framework. *Proc. VLDB Endowment (PVLDB)*, **4**, 1318–1327.

[56] Apache Mahout. `http://mahout.apache.org/`. [Last visited: July 2014].

[57] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003) The Google file system. *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, USA, 19–22 October, pp. 29–43. ACM, New York, NY, USA.

[58] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010) The Hadoop distributed file system. *Proc. 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'10)*, Incline Village, NV, USA, 3–7 May, pp. 1–10. IEEE Computer Society, Washington, DC, USA.

[59] Kosmos Distributed Filesystem (KFS). `https://code.google.com/p/kosmosfs/`. [Last visited: July 2014].

[60] IOzone filesystem benchmark. `http://www.iozone.org/`. [Last visited: July 2014].

[61] Shan, H. and Shalf, J. (2007) Using IOR to analyze the I/O performance of HPC platforms. *Proc. 49th Cray User Group Conference (CUG'07)*, Seattle, WA, USA, 7–10 May. CUG, Oak Ridge, TN, USA.

[62] Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. (2010) The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. *Proc. 2nd IEEE Workshop on Information & Software as Services (WISS'10)*, Long Beach, CA, USA, 1–6 March, pp. 41–51. IEEE Computer Society, Washington, DC, USA.

[63] Wong, P. and Van der Wijngaart, R. F. (2003) NAS parallel benchmarks I/O version 2.4. Technical Report NAS-03-002. NASA Ames Research Center, Moffett Field, CA, USA.

[64] FLASH I/O benchmark routine. `http://www.ucolick.org/~zingale/flash_benchmark_io/`. [Last visited: July 2014].

[65] Apache Hive. `http://hive.apache.org/`. [Last visited: July 2014].

[66] MPICH: High performance and widely portable implementation of the MPI standard. `http://www.mpich.org/`. [Last visited: July 2014].

[67] NASA. NASA Advanced Supercomputing (NAS) Parallel Benchmarks (NPB). `http://www.nas.nasa.gov/publications/npb.html`. [Last visited: July 2014].

[68] Polte, M., Simsa, J., and Gibson, G. (2008) Comparing performance of solid state devices and mechanical disks. *Proc. 3rd Petascale Data Storage Workshop (PDSW'08)*, Austin, TX, USA, November 17, pp. 1–7. IEEE Press, Piscataway, NJ, USA.

[69] Kasick, M. P., Gandhi, R., and Narasimhan, P. (2010) Behavior-based problem localization for parallel file systems. *Proc. 6th International Conference on Hot Topics in System Dependability (HotDep'10)*, Vancouver, BC, Canada, October 3, pp. 1–13. USENIX Association, Berkeley, CA, USA.

[70] Méndez, S., Rexachs, D., and Luque, E. (2012) Evaluating utilization of the I/O system on computer clusters. *Proc. 18th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*, Las Vegas, NV, USA, 16–19 July, pp. 366–372. UCMSS, San Diego, CA, USA.

[71] Yu, H., Sahoo, R. K., Howson, C., Almasi, G., Castanos, J. G., Gupta, M., Moreira, J. E., Parker, J. J., Engelsiepen, T. E., Ross, R. B., Thakur, R., Latham, R., and Gropp, W. D. (2006) High performance file I/O for the Blue Gene/L supercomputer. *Proc. 12th International Symposium on High-Performance Computer Architecture (HPCA-12)*, Austin, TX, USA, 11–15 February, pp. 187–196. IEEE Computer Society, Washington, DC, USA.

[72] Méndez, S., Rexachs, D., and Luque, E. (2011) Methodology for performance evaluation of the input/output system on computer clusters. *Proc. 13th IEEE International Conference on Cluster Computing (CLUSTER'11)*, Austin, TX, USA, 26–30 September, pp. 474–483. IEEE Computer Society, Washington, DC, USA.

[73] Saini, S., Talcott, D., Thakur, R., Adamidis, P., Rabenseifner, R., and Ciotti, R. (2007) Parallel I/O performance characterization of Columbia and NEC SX-8 superclusters. *Proc. 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS'07)*, Long Beach, CA, USA, 26–30 March, pp. 1–10. IEEE Computer Society, Washington, DC, USA.

[74] Méndez, S., Rexachs, D., and Luque, E. (2012) Modeling parallel scientific applications through their input/output phases. *Proc. 4th Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS'12)*, Beijing, China, September 28, pp. 7–15. IEEE Computer Society, Washington, DC, USA.

[75] Sigovan, C., Muelder, C., Ma, K.-L., Cope, J., Iskra, K., and Ross, R. B. (2013) A visual network analysis method for large-scale parallel I/O systems. *Proc. 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*, Boston, MA, USA, 20–24 May, pp. 308–319. IEEE Computer Society, Washington, DC, USA.

[76] Islam, N. S., Lu, X., Rahman, M. W., and Panda, D. K. (2014) SOR-HDFS: A SEDA-based approach to maximize overlapping in RDMA-enhanced HDFS. *Proc. 23rd ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC'14)*, Vancouver, BC, Canada, 23–27 June, pp. 261–264. ACM, New York, NY, USA.

[77] Wu, D., Luo, W., Xie, W., Ji, X., He, J., and Wu, D. (2013) Understanding the impacts of solid-state storage on the Hadoop performance. *Proc. 1st International Conference on Advanced Cloud and Big Data (CBD'13)*, Nanjing, China, 13–15 December, pp. 125–130. IEEE Computer Society, Washington, DC, USA.

[78] Dimitrov, M., Kumar, K., Lu, P., Viswanathan, V., and Willhalm, T. (2013) Memory system characterization of big data workloads. *Proc. 1st Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware (BPOE'13)*, Silicon Valley, CA, USA, October 8, pp. 15–22. IEEE Computer Society, Washington, DC, USA.

[79] Worringen, J., Träff, J. L., and Ritzdorf, H. (2003) Fast parallel non-contiguous file access. *Proc. 15th ACM/IEEE Supercomputing Conference (SC'03)*, Phoenix, AZ, USA, 15–21 November, pp. 60:1–60:18. ACM, New York, NY, USA.

[80] Thakur, R., Gropp, W., and Lusk, E. (1999) Data sieving and collective I/O in ROMIO. *Proc. 7th Symposium on the Frontiers of Massively Parallel Computation (FRONTIERS'99)*, Annapolis, MD, USA, 21-25

February, pp. 182–189. IEEE Computer Society, Washington, DC, USA.

[81] Coloma, K., Choudhary, A., Liao, W.-K., Ward, L., Russell, E., and Pundit, N. (2004) Scalable high-level caching for parallel I/O. *Proc. 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, NM, USA, 26–30 April, pp. 96b. IEEE Computer Society, Washington, DC, USA.

[82] Ching, A., Choudhary, A., Coloma, K., Liao, W.-K., Ross, R. B., and Gropp, W. D. (2003) Noncontiguous I/O accesses through MPI-IO. *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03)*, Tokyo, Japan, 12–15 May, pp. 104–111. IEEE Computer Society, Washington, DC, USA.

[83] Fryxell, B., Olson, K., Ricker, P., Timmes, F. X., Zingale, M., Lamb, D. Q., MacNeice, P., Rosner, R., Truran, J. W., and Tufo, H. (2000) FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophysical Journal Supplement*, **131**, 273–334.

[84] Schad, J., Dittrich, J., and Quiané-Ruiz, J.-A. (2010) Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endowment (PVLDB)*, **3**, 460–471.

[85] A Hadoop library of Snappy compression. `http://code.google.com/p/hadoop-snappy/`. [Last visited: July 2014].

[86] Kim, Y., Lee, J., Oral, S., Dillow, D. A., Wang, F., and Shipman, G. M. (2014) Coordinating garbage collection for arrays of solid-state drives. *IEEE Transactions on Computers*, **63**, 888–901.