Exploiting Analytics Shipping with Virtualized MapReduce on HPC Backend Storage Servers

Cong Xu, Robin Goldstone, Zhuo Liu, Hui Chen, Bryon Neitzel, and Weikuan Yu, *Senior Member, IEEE*

Abstract—Large-scale scientific applications on High-Performance Computing (HPC) systems are generating a colossal amount of data that need to be analyzed in a timely manner for new knowledge, but are too costly to transfer due to their sheer size. Many HPC systems have catered to in situ analytics solutions that can analyze temporary datasets as they are generated, i.e., without storing to long-term storage media. However, there is still an open question on how to conduct efficient analytics of permanent datasets that have been stored to the backend persistent storage because of their long-term value. To fill the void, we exploit the analytics shipping model for fast analysis of large-scale scientific datasets on HPC backend storage servers. Through an efficient integration of MapReduce and the popular Lustre storage system, we have developed a Virtualized Analytics Shipping (VAS) framework that can ship MapReduce programs to Lustre storage servers. The VAS framework includes three component techniques: (a) virtualized analytics shipping with fast network and disk I/O; (b) stripe-aligned data distribution and task scheduling and (c) pipelined intermediate data merging and reducing. The first technique provides necessary isolation between MapReduce analytics and Lustre I/O services. The second and third techniques optimize MapReduce on Lustre and avoid explicit shuffling. Our performance evaluation demonstrates that VAS offers an exemplary implementation of analytics shipping and delivers fast and virtualized MapReduce programs on backend Lustre storage servers.

Index Terms—Analytics shipping, hadoop, mapreduce, HPC, lustre

1 Introduction

HPC on supercomputers has been a widely leveraged paradigm to tap the power of parallel data processing and computing resources for Big Data. For example, simulation codes of Center for Plasma Edge Simulation (CPES) [3] and the Sandia Combustion Research Facility (CRF) [9] are producing hundreds of terabytes of data in a week. Because of the sheer size of scientific data, it is very costly to move them back and forth on HPC systems for the purpose of analytics or visualization. Minimizing data movement is also essential to reducing the power consumption of future exascale computing systems.

These constraints have driven many HPC systems towards in situ analytics solutions that can analyze temporary datasets, when they are created in memory and before being stored to long-term storage media. However, there is still an open question on how to conduct efficient analytics of permanent datasets that have been stored to the backend storage because of their long-term value without causing too much movement.

We take on an effort to exploit the analytics shipping model for fast analysis of large-scale persistent scientific datasets. The key idea is to allow scientific users to ship their analytics programs to the backend storage servers of HPC systems and process the persistent datasets therein without retrieving them back to the compute nodes. The disruptive nature of adding analytics workloads to the storage servers triggers several critical questions. These include (1) how to isolate the analytics programs from the critical I/O services at the storage servers? (2) how much performance interference and resource contention the analytics programs will cause to the I/O services? (3) what mismatches, if any, there are between the requirements of analytics programs and those system features available at the backend storage servers? and (4) how to mitigate these mismatches and achieve efficient analytics?

MapReduce [15] is a popular analytics programming model. Various implementations of MapReduce such as Hadoop [2] and Spark [30] have been deployed by many organizations. Lustre [8] is a typical storage system used on HPC systems, offering parallel I/O services. In this paper, we develop an analytics shipping framework using MapReduce as a representative analytics model and Lustre as a representative HPC backend storage system. With such a framework, we undertake an effort to systematically explore the aforementioned four questions. First of all, HPC systems and their storage servers are deployed using a compute-centric paradigm, which is distinctly different from the data-centric paradigm that is adopted by the MapReduce-style analytics model. In addition, HPC storage servers are dedicated to the I/O services, which can be disrupted if the storage servers are to host analytics workloads.

Several challenges must be addressed for the development of an effective analytics shipping framework, including (1) the need of isolating analytics programs from the storage services on the backend storage servers, (2) the need of distributing analytics datasets for the

- *C. Xu, Z. Liu, H. Chen, and W. Yu are with the Auburn University, Auburn, AL 36849, USA. E-mail: {congxu, zhuoliu, hchen, wkyu}@auburn.edu.*
- *R. Goldstone is with the Lawrence Livermore National Lab, Livermore, CA 94550, USA. E-mail: goldstone1@llnl.gov.*
- *B. Neitzel is with Intel, Santa Clara, CA, USA. E-mail: bryon.s.neitzel@intel.com.*

Manuscript received 16 Sept. 2014; revised 14 Dec. 2014; accepted 4 Jan. 2015. Date of publication 6 Jan. 2015; date of current version 16 Dec. 2015. Recommended for acceptance by R. Brightwell.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2389262

exploitation of locality in task scheduling, and (3) the need of minimizing data movement for intermediate data generation and shuffling.

To address these challenges, we have developed three techniques including (a) virtualized analytics shipping with fast network and disk I/O; (b) Lustre stripe-aligned data distribution and task scheduling; and (c) pipelined intermediate data merging and reducing. The first technique provides the needed isolation between MapReduce analytics programs and the I/O services. The second and third techniques mitigate the mismatches of data distribution, shuffling, and task scheduling between MapReduce and Lustre, and avoid the explicit shuffling phase in MapReduce. These techniques are together integrated into a VAS framework. We have conducted a systematic set of tests to evaluate the performance of analytics programs and the impact to I/O services. Our results demonstrate that the VAS framework offers an exemplary implementation of analytics shipping and delivers fast and virtualized MapReduce programs on backend Lustre storage servers.

The rest of the paper is organized as follows. Section 2 provides the background of YARN MapReduce and Lustre. We then describe the challenges for shipping analytics to backend storage servers in Section 3. Section 4 describes our design of virtualized analytics shipping. Section 5 provides our experimental results, followed by related work in Section 6. Finally, we conclude the paper in Section 7.

2 Background

In this section, we provide an overview of the MapReduce framework in YARN and the Lustre parallel file system.

2.1 An Overview of YARN MapReduce

YARN is the next-generation of Hadoop's compute platform [1], [25]. It can support various programming models such as MapReduce and MPI. There are three main components in the YARN MapReduce framework, including the ResourceManager (RM), NodeManager (NM), and the ApplicationMaster (AM). The RM is responsible for allocating resources, which are abstracted as *containers*, to applications. A NM monitors and reports its container's resource usage (cpu, memory, disk, and network) to RM. The per-application AM is responsible for the job's whole life cycle management including resource negotiation, task deployment, status monitoring and reporting, till the application exits.

A MapReduce job starts an AM that negotiates with the RM for containers. Once the containers are allocated, the AM will communicate with the corresponding NMs to launch map tasks (MapTasks). Each MapTask reads an input split from the HDFS [21] and generates intermediate data in the form of Map Output Files (MOFs). Each MOF contains as many partitions as the number of reduce tasks (ReduceTasks). ReduceTasks are launched after the generation of some MOFs. Each fetches its partition from every MOF. Meanwhile, the background merging threads will merge and spill intermediate data to local disks. In the reduce phase, the merged intermediate data will be processed by the reduce function for final results, which will be written back to HDFS.

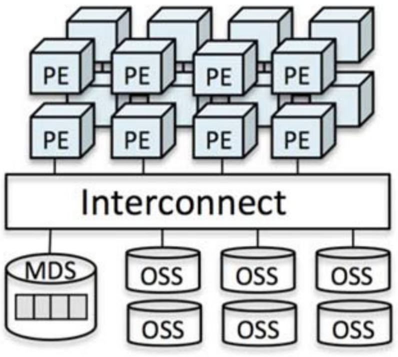


Fig. 1. Diagram of a typical HPC system architecture.

For minimizing data movement, MapReduce adopts a data-centric paradigm to co-locate compute and storage resources on the same node to facilitate locality-oriented task scheduling. In YARN's case, a job's MapTasks will be scheduled to the containers where the input data splits are located.

2.2 Overview of HPC and Lustre Backend Storage System

Fig. 1 shows a diagram of typical HPC systems. Such systems are constructed using a compute-centric paradigm where compute nodes and storage servers belong to separated groups. The core of such systems consists of a large collection of compute nodes, *i.e.*, processing elements (PEs), which offer the bulk of computing power. Via a high-speed interconnect, these PEs are connected to a parallel file system from the storage backend for data I/O. With such a compute-centric paradigm, tasks on compute nodes of HPC systems are, in general, equally distant from the backend storage system.

Lustre is a popular backend storage system used on many HPC systems. A typical Lustre file system [29] consists of a metadata server (MDS), a metadata target (MDT), object storage servers (OSS), object storage targets (OST) and Lustre clients. The MDS makes metadata stored in one or more MDTs available to Lustre clients. Each MDS manages the names and directories in the Lustre file system and provides network request handling for local MDTs. The OSS provides file I/O service, and network request handling for one or more local OSTs. A typical configuration is an MDT on a dedicated node, two or more OSTs on each OSS node. File striping is the key feature of Lustre for distributing the segments of a single file across multiple OSTs. Specifically, each file is divided into multiple fixed-size stripes, which are placed onto multiple OSTs in a round-robin manner. Thus a user can access a striped file's data from multiple OSTs concurrently.

In addition, Lustre provides fine-grained parallel file services with its distributed lock management. To guarantee file consistency, it serializes data accesses to a file or file extents using a distributed lock management mechanism. Because of the need for maintaining file consistency, all processes first have to acquire locks before they can update a shared file or an overlapped file block. Thus, when all processes are accessing the same file, their I/O performance is dependent not only on the aggregated physical bandwidth from the storage devices, but also on the amount of lock contention among them.

3 Challenges for Shipping Analytics to Backend Storage Servers

In this section, we discuss the challenges associated with shipping analytics to backend storage servers on HPC systems.

3.1 Isolating Analytics from I/O Services

Backend storage servers on HPC systems are typically dedicated to I/O services. Shipping analytics to these servers will inevitably disrupt this dedication. Concerns can arise from a variety of aspects, including resource contention, performance interference, and integrity and availability of data hosted by the storage servers. There is a serious need to isolate analytics from I/O services when the backend storage servers are used to host both analytics programs and I/O services. In this paper, we consider the users and their analytics programs to the HPC systems are trustworthy, which is a reasonable assumption considering the rigidity and secure practices in allocating user accounts and computer time allocations on HPC systems. Thus we focus on the issues of resource contention and performance interference.

There are many approaches such as chroot [4], virtualization [6], [10], [11] to isolating analytics from I/O services. The most appropriate approach should allow all the flexibility of scheduling and tuning analytics programs on the backend storage servers while, at the same time, providing rich mechanisms to alleviate resource contention and mitigate performance interference. For its rich functionalities and versatile execution, we consider the use of virtual machines to be the most suitable for analytics shipping. While virtualization provides handy mechanisms for isolating analytics and segregating allocated resources, it is critical to have a thorough examination on its performance implication to both I/O services and analytics, and accordingly mitigate the performance overhead where applicable.

3.2 Data Distribution and Exploiting Task Locality

As reviewed in Section 2, HPC systems and their storage servers are deployed using a compute-centric paradigm, which is distinctly different from the data-centric paradigm that is adopted by the MapReduce-style analytics model. Therefore, we need to address the mismatches of data distribution and task scheduling between MapReduce-based programs and backend storage servers. First, in terms of data distribution, the datasets of analytics programs need to be distributed in a manner that can closely match the pattern used by MapReduce. In the case of YARN, we need to emulate a distribution pattern of HDFS such that analytics datasets are split into blocks and distributed to all Lustre storage servers. Second, in terms of task scheduling, we need to extract the knowledge of data distribution and bestow this information to YARN for its scheduler to launch analytics tasks with the best data locality.

3.3 Intermediate Data Placement and Shuffling

Another key difference between the execution of MapReduce programs and the configuration of HPC systems is the management of intermediate data. MapReduce systems such as YARN are usually configured with local disks attached to the compute nodes. As reviewed in Section 2, this allows MapReduce programs to store their intermediate data temporarily on local disks. But an HPC storage system such as Lustre is usually a shared file system. This means that MapReduce programs, when shipped to Lustre servers, have Lustre as a global storage space to store any intermediate data.

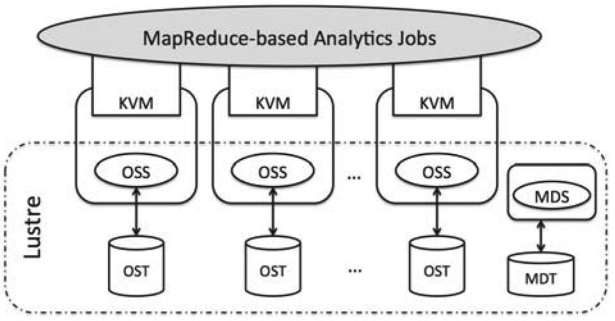


Fig. 2. MapReduce-based analytics shipping model.

Intuitively, placing intermediate data as MOFs on Lustre seems to bring a lot of convenience because all MOFs are globally available to any MapReduce task. ReduceTasks no longer need to have an explicit shuffling stage to fetch data segments from the MOFs. However, this can have several ramifications. First, the generation of many temporary MOFs can present a huge burst of metadata operations to the often bottlenecked Lustre MDS [19]. Second, the striping pattern of a MOF, i.e., the placement of its stripes, needs to be decided carefully. Should all stripes be placed on a Lustre OST on the same node with the MapTask, on another OST closer to some reduce task, or across all OSTs? Finally, when many ReduceTasks are reading from the same MOFs on Lustre, they can cause a burst of lock operations. The resulting lock contention may significantly degrade the speed of data shuffling, causing an overall performance loss. Moreover, in original MapReduce's shuffle phase, repetitive merge operations are usually conducted due to limited memory capacity. By shipping MapReduce analytics to HPC systems to work with HPC backend storage servers, can we mitigate such overheads through a better merging and reducing scheme?

Taken together, we need to address all challenges for an effective analytics shipping framework.

4 Developing an Efficient Virtualized Analytics Shipping Framework

To tackle the aforementioned challenges, we propose a *VAS* framework in this paper. As shown in Fig. 2, we leverage KVM [6] to segregate MapReduce jobs from the I/O services running on OSS nodes. The isolation of KVM ensures that the performance stability and system reliability of Lustre storage servers are protected from the MapReduce programs. To be specific, each OSS hosts one virtual machine (VM), which installs YARN and Lustre client related drivers to interact with Lustre file system. Thus the collection of KVMs on all OSSes form a virtualized cluster, to which the analytics programs are shipped.

TABLE 1
Throughput Performance for Different Communication Cases

Case #	Source	Destination	Interface	BW (Gbps)
1	hostA-VM	hostA	VIRTIO	**15.2**
2	hostA-VM	hostA	SR-IOV	7.1
3	hostA-VM	hostB	VIRTIO	4.3
4	hostA-VM	hostB	SR-IOV	**6.1**
5	hostA-VM	hostB-VM	VIRTIO	7.2
6	hostA-VM	hostB-VM	SR-IOV	**9.2**

Simply porting the MapReduce programming model—YARN—into computing-intensive HPC environment to work with Lustre file system can only deliver sub-optimal performance due to distinct features of Lustre and HDFS [16]. We develop three main techniques in VAS to achieve efficient analytics shipping. First, we optimize the performance of network and disk I/O via dynamic routing configuration of VMs. Second, we develop stripe-aligned data distribution and task scheduling for fast data accesses in MapReduce. Finally, we propose a new technique that can avoid the explicit shuffling and pipeline the merging and reducing of intermediate data in our VAS framework.

4.1 Fast Network and Disk I/O

While we utilize KVM for segregating MapReduce from I/O services on Lustre servers, a major concern is the need of appropriate techniques for delivering efficient network and disk I/O to the VMs.

4.1.1 Network Virtualization

There are two widely used I/O options in KVM: VIRTIO and single root I/O virtualization (SR-IOV). VIRTIO is a virtualization standard for network and disk device drivers. It performs much better than "full virtualization" because VIRTIO does not require the hypervisor to emulate actual physical devices. SR-IOV provides VMs with direct access to network hardware resources, eliminating the overhead of extra memory copies. A major difference between VIRTIO and SR-IOV network configuration is the use of switch-bypassing: in VIRTIO a bridge is created for the VM-to-host communication without going through the switch. In SR-IOV, all communication has to go through the switch, even when a VM communicates with its host.

To obtain optimal network performance, we have firstly evaluated both technologies for three different connection scenarios. All measurements are conducted using Mellanox ConnectX-2 QDR 10G Host Channel Adaptors, which are connected by a 10G Ethernet switch. By leveraging iperf benchmark, we test the point-to-point bandwidth of each connection scenario for VIRTIO and SR-IOV. The results are demonstrated in Table 1.

The results of Cases 1 and 2 reveal that VIRTIO bridge networking can achieve about 15.2 Gbps bandwidth, which is much faster than SR-IOV method because VIRTIO does not go through the network switch. In VIRTIO bridged network communication, VM exchanges data with its local Physical Machine (PM) directly through the intra-node bridge between them, so that network flow does not go outside of the node.

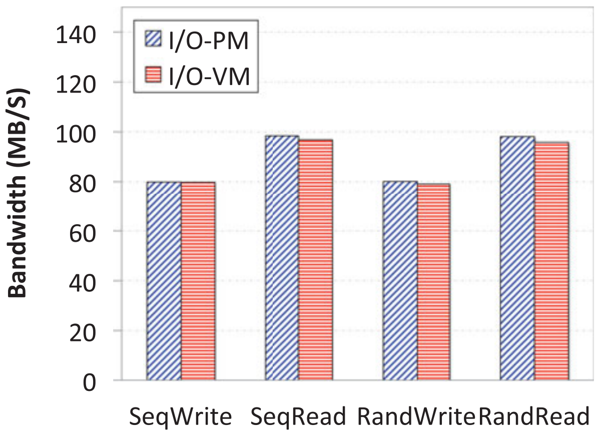


Fig. 3. Disk I/O performance of PM and VM.

However, SR-IOV outperforms VIRTIO in terms of remote network connection. For example, in Case 4, SR-IOV is able to achieve 6.1 Gbps bandwidth for the connection between the VM and the remote Physical Machine, and 9.2 Gbps for remote VM-to-VM communication (Case 6). This is close to the peak link rate. Such performance differences are because the software-based approach of VIRTIO requires redundant memory copies for sending/receiving packages in the host. In contrast, SR-IOV allows the direct exchange of data between VMs and thus avoids extra memory copies.

Since VIRTIO and SR-IOV can achieve superior performance for the local VM-to-PM (Case 1) and remote VM-to-VM (Case 6) scenarios, respectively, we adopt a dynamic routing policy for network traffic of VMs. To implement this policy, we firstly ensure that the routing table of both VM and PM are set up appropriately. By default, the routing scheme uses SR-IOV. If the routing scheme detects that the source and destination reside on the same node, it switches to VIRTIO.

4.1.2 Fast Disk I/O in VMs with VIRTIO

For disk I/O, we apply VIRTIO for attaching storage devices to the KVM. To evaluate its effectiveness, we have conducted experiments to measure I/O performance for both the VM and PM. We use the IOzone benchmark. As shown in Fig. 3, the results indicate that VIRTIO delivers an I/O performance for the VMs that is comparable to that for the PMs under various access patterns such as sequential reads/writes and random reads/writes.

4.1.3 KVM's I/O Interference to Lustre Storage Servers

A major concern for the VAS framework is the performance interference to the I/O services of Lustre storage servers. To investigate this issue, we leverage IOzone benchmark to simulate heavy I/O workload and measure the aggregated I/O bandwidth of three cases: Lustre-alone, Lustre with a concurrent YARN program running on the physical machines (Lustre-YARNinPM), and Lustre with a virtualized YARN program running on KVM virtual machines (Lustre-YARNinVM). For the last two cases, 100 YARN TeraSort jobs are launched one by one on 8 Lustre storage servers, at the same time heavy I/O operations are conducted

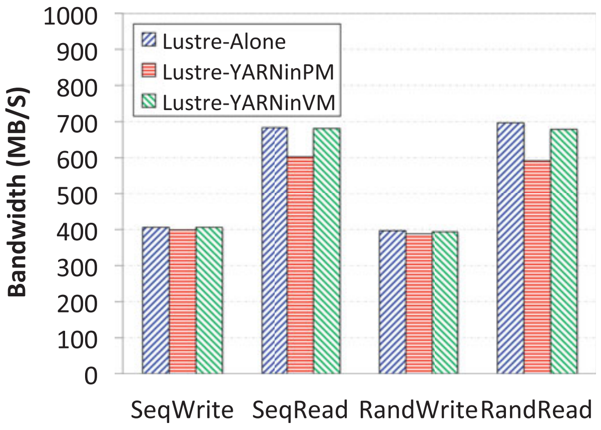


Fig. 4. Using KVM to alleviate YARN's interference to Lustre.

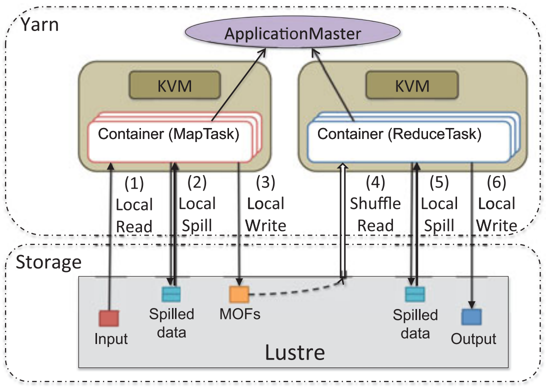


Fig. 5. End-to-end data path for MapReduce on Lustre.

by IOzone Benchmark on remote Lustre clients to evaluate Lustre's I/O bandwidth. Note that, in this test, both Lustre-YARNinPM and Lustre-YARNinVM are using separate disks for the HDFS instead of sharing the same storage disks with Lustre OSTs.

Fig. 4 shows the aggregated bandwidth of Lustre storage servers. Running YARN directly on PMs incurs a bandwidth degradation of 11.9 and 15.2 percent for sequential reads and random reads, respectively. In contrast, running YARN in VMs has negligible impact to the Lustre servers. To be specific, the time to start and configure KVM on OSS nodes is about 25 seconds. Since the virtualized VMs can stand by for serving ad-hoc analytics requests, such time overhead is minimal. In addition, since the Lustre write bandwidth is not sensitive to caching effects [20], the write bandwidths for all three cases are comparable.

The performance of Lustre storage servers has not been affected much when launching Yarn in VMs, because KVM provides very good resource segregation. KVM is able to restrict the resource usage of VMs of Yarn (especially memory and CPU usage), thereby reserving sufficient resources for the storage servers. As a result, the performance interference of YARN to Lustre storage servers has been mitigated.

4.2 Stripe-Aligned Data Distribution and Task Scheduling

The locality of task scheduling is an important factor that affects the amount of data movement during analytics. To exploit locality in VAS, we need to have a thorough understanding of its data movement. Fig. 5 shows the end-to-end path of data movement when running MapReduce on top of Lustre in our VAS framework. There are six main steps of data I/O, all happening on Lustre. A MapTask is involved in the first three steps: (1) initially reading the input splits, (2) spilling data during the map phase, finally (3) merging and storing the intermediate data as a MOF on the Lustre. A ReduceTask is involved in the last three steps: (4) fetching and shuffling data from MOFs, (5) merging data segments and spilling some merged segments to Lustre as needed, and (6) reading spilled segments and writing final results at the end of the reduce phase. A partial enhancement to this path may not result in an overall performance improvement, e.g., improving the locality in Step 1 as done in [16]. For an efficient analytics shipping framework, we need to provide a complete optimization of this end-to-end data path. In the rest of this subsection, we elaborate our strategies for data placement and task scheduling in Steps 1, 2, 3, and 6.

For both the input datasets and the temporary spilled data, we need to determine the placement and distribution on Lustre. On Lustre, it translates to a decision on the choice of stripe size and stripe width of MapReduce data files. Setting the stripe size smaller than the size of MapReduce input split will force a MapTask to read from many OSTs, resulting in the loss of locality and an increase of I/O interference. Using a stripe size larger than the split size can force multiple MapTask to read from the same OST, reducing the parallelism and increasing the lock contention. Our tests indicate that it is beneficial to choose the Lustre stripe size equal to the input split size. We adopt this as the default configuration when a MapReduce program is allowed to generate and distribute its own input data. To work with the datasets that are generated a priori by scientific applications, we instrument our framework for the MapTasks to extract the distribution of Lustre stripes and retrieve them accordingly.

Lustre records the distribution of stripes among physical machines, which cannot be directly translated to the actual VMs. To make the YARN scheduler aware of the correspondence of VMs to PMs, we create a configurable file with a mapping table of PMs and VMs. With this mapping table, we use the Lustre "lfs" tool inside the YARN MapReduce resource manager to distribute data in a stripe-aligned manner and place the initial input and the newly generated data in the end-to-end path (Steps 1, 2, 3, 5, 6) on the local OSTs. With the stripe-aligned distribution, in Step 1 of the end-to-end path, by default we enable the YARN scheduler to assign a MapTask preferentially to the storage server that has the data split, i.e., the stripe is placed on the OST local to the MapTask. When the stripe size is smaller than the data split, we place the MapTask to the storage server that hosts the most number of stripes for the targeted split.

Besides the preferred use of locality-based striping for Step 1, we use this policy exclusively for the intermediate data as they are generated and retrieved for Steps 2, 3, 5 and 6. In these steps, spilled intermediate data are temporarily written to Lustre and soon read by the same task. Placing them in a stripe-aligned manner on the local OST can reduce network and disk I/O to a remote OST. In our

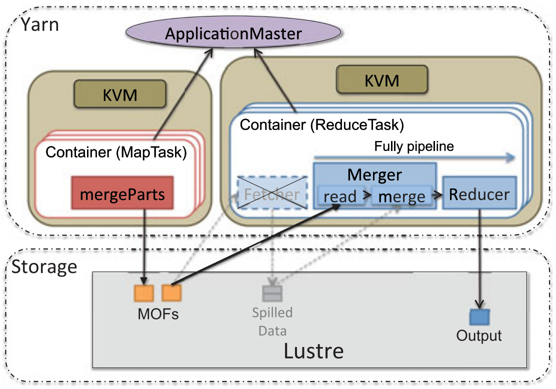


Fig. 6. Pipelined merging and reducing of intermediate data without explicit shuffling.

VAS framework, we have implemented a new class LustreFileSystem in YARN, which extracts the input file location from Lustre, then pass this information to YARN. Note that Steps 4 and 5 for shuffling and merging of intermediate data are discussed next (see Section 4.3).

4.3 Pipelined Merging and Reducing of Intermediate Data

We take advantage of the fact that HPC storage systems provide a global shared file system such as Lustre. We are aware that some HPC storage systems such as Gordon [5] have solid state drives, which can break this assumption. However, we do not want to limit our analytics shipping framework to the environment with SSDs, and leave the exploration of SSDs in future work. We focus on exploiting the feasibility of analytics shipping in a typical Lustre environment.

With all the MOFs available to any task on Lustre, we can avoid an explicit shuffling stage in a ReduceTask. Even though there is still a need to bring the data into memory, merge them, and process them through the reduce function, a ReduceTask can go through an implicit shuffling step, i.e., reading data from MOFs through file I/O operations.

We describe our management of intermediate data based on the data movement steps shown in Fig. 5. For Steps 4 and 5, we develop a technique called *pipelined merging and reducing of intermediate data* to mitigate the need of spilling and retrieving data. This technique fully leverages the fact that intermediate data stored as MOFs are available to all tasks without explicit shuffling. Specifically, we eliminate the explicit fetching step in a ReduceTask and form a new pipeline of data merging and reducing. As shown in Fig. 6, the optimized merger directly reads intermediate data generated by MapTasks from Lustre, and merges them into a stream of key-value pairs to the final reduce function. This new strategy can avoid the spilling of data in Step 4 and eliminate all the associated I/O operations.

To form a full pipeline of merging and reducing of data, the new merger does not start until all MOFs are available, similar to the strategy in [26]. In addition, we introduce two threads, one on file I/O and the other on merging data. These two threads are implemented in the Fetcher file class of YARN. The read thread in the merge function reads MOFs from Lustre directly without explicit shuffling. These threads are pipelined to avoid the serialization of I/O and computation. Double buffers are provided in both threads to facilitate this pipeline. For example, in the file I/O thread, one buffer is provided for buffering data from Lustre, the other for buffering data ready to the merging thread.

5 Evaluation

In this section, we evaluate the performance of our VAS framework compared to the original YARN. We first describe our experimental environment, then we provide our experimental results in detail.

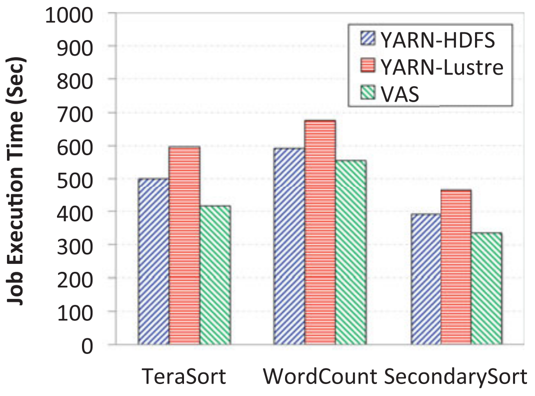
5.1 Experimental Setup

Our experiments are conducted in a 22-node cluster. Each node contains dual-socket quad-core 2.13 GHz Intel Xeon processors and 8 GB DDR2 800 MHz memory, along with 8X PCI-Express Gen 2.0 bus. Each node has a 500 GB, 7200 RPM Western Digital SATA hard-drivers. All nodes are equipped with Mellanox ConnectX-2 QDR Host Channel Adaptors, which are configured to run in the 10G mode and connect to a 48-port 10G switch. These nodes are also connected to a 48-port Gigabit Ethernet switch. We install InfiniBand software stack and Mellanox OFED version 2.0, with SR-IOV enabled.

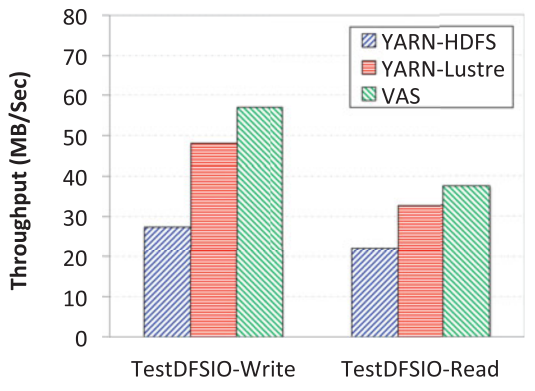
In our evaluation, all experiments are based on Hadoop 2.0.4 and Lustre 2.5.0. Each container is allocated 1 GB memory and the maximum memory configured for the virtual machine on each node is 6 GB. The replica ratio is configured to be 1. A group of benchmarks have been leveraged for performance measurements, including TeraSort, WordCount, SecondarySort and TestDFSIO in the YARN distribution. Among them, TeraSort and SecondarySort generate a large amount of intermediate data. WordCount produces only a small quantity of intermediate data. TestDFSIO benchmark is designed to test the raw performance of a file system, in which the write and read throughput are measured. These diverse benchmarks are used to evaluate our framework from a variety of aspects.

5.2 Overall Performance

Fig. 7 compares the overall performance of YARN on HDFS (YARN-HDFS), YARN on Lustre (YARN-Lustre) and our VAS framework (VAS) for the four benchmarks. Fig. 7a shows the job execution time of TeraSort, WordCount and SecondarySort. For all three benchmarks, YARN-Lustre performs the worst. That is because simply porting MapReduce to Lustre cannot fully leverage the advanced features of Lustre and suffers from lock contention for the access of intermediate data. YARN-HDFS is up to 16.2 percent faster than YARN-Lustre because it leverages input locality for fast data movement. On average, our VAS framework achieves an execution time 12.4 and 25.4 percent better than YARN-HDFS and YARN-Lustre, respectively. Specifically, VAS outperforms YARN-HDFS and YARN-Lustre by 16.8 and 30.3 percent, respectively, for TeraSort. The improvement of VAS over YARN-HDFS is mainly due to our pipeline technique for intermediate data. Such improvement is more significant for the benchmarks with a lot of



(a) Job Execution



(b) TestDFSIO

Fig. 7. The overall performance of three schemes.

intermediate data such as TeraSort than those with little such as WordCount.

These performance gains demonstrate the strengths of our techniques in stripe-aligned data placement, task scheduling, and pipelined merging and reducing. The first two techniques help to improve the locality of the VAS and reduce the network transmission, thus increasing the MapTask read and ReduceTask write performance. The last two techniques are able to eliminate I/O operations in ReduceTask, leading to faster data processing.

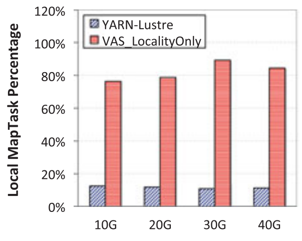
To shed light on how different schemes can leverage Lustre for I/O, we further measure the average I/O throughput using TestDFSIO. As shown in Fig. 7b, VAS achieves 14.6 and 69.7 percent better read bandwidth, and 18.7 and 108 percent better write bandwidth, compared to YARN-Lustre and YARN-HDFS, respectively. In general, DFSIO displays better performance on Lustre than on HDFS. This is because of the I/O capability of Lustre. In addition, the performance improvement of VAS over YARN-Lustre comes from our exploitation of data locality and optimization on intermediate data shuffling. Since HPC uses parallel file systems such as Lustre instead of HDFS, in the rest of this paper, we focus on the comparison between VAS and YARN-Lustre, omitting the case of YARN-HDFS.

5.3 Locality Improvement and the Benefit

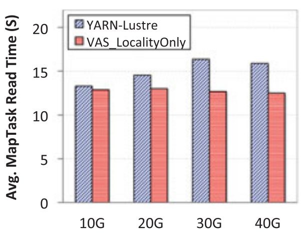
By default, VAS allocates a MapTask to an OSS with the entire split available locally as a stripe or the OSS with the most number of stripes if the input data split is striped. In addition, MapTasks will be controlled to dump its spill data and MOFs to local OSTs. Similarly, ReduceTasks write any of its spill/merge data and output to local OSTs. Here we focus on the comparison between YARN-Lustre and our VAS framework that is enabled with only the locality optimization (VAS-Locality-Only), i.e., without our technique on pipelined merging as detailed in Section 4.3. We run TeraSort on 10 slave nodes with input data varying from 10 GB to 40 GB. During the tests, we measure the percentage of local MapTasks, and profile the performance of I/O operations and job execution time. Fig. 8a shows the percentage of local MapTasks. As shown in the figure, YARN-Lustre randomly launches MapTasks, achieving only 11.6 percent in terms of the percentage of local MapTasks. In contrast, our VAS framework achieves high percentage of local MapTasks by effectively managing the distribution of data splits. VAS-Lustre schedules 82.1 percent of MapTasks with its data on the local OST.

To examine the performance benefits of data locality, we have measured the average input read time by MapTasks. As shown in Fig. 8b, the performance gain is not significant for 10 GB data size. This is because the virtual VMs are connected with 10 Gigabit Ethernet. Data locality cannot bring much benefit when the data size is small. As the data size increases to 40GB, the MapTask reading time in VAS outperforms YARN by as much as 21.4 percent. On average, the reading performance of MapTasks in VAS is 15 percent better than YARN-Lustre due to improved data locality.

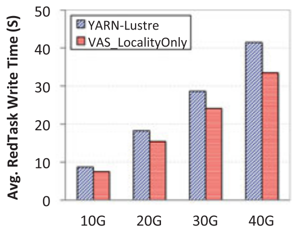
Fig. 8c presents the write performance of ReduceTasks. In YARN, since the number of ReduceTasks is fixed while the number of MapTasks is proportional to the size of input data. The writing time of ReduceTasks grows with an increasing size of total data. As shown in the figure, VAS-Locality-Only leads to 19.3 percent better write performance than YARN-Lustre when the data size is 40 GB. This



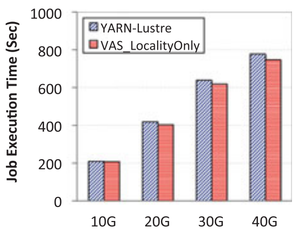
(a) Percentage of Local Tasks



(b) MapTask Read Time

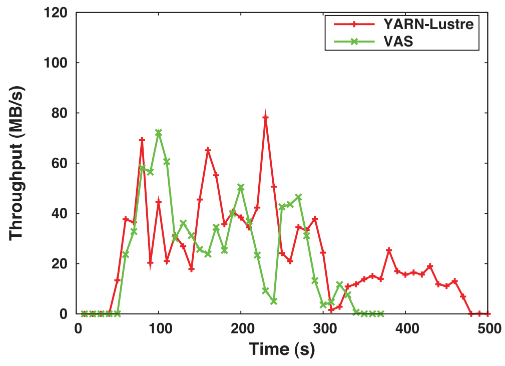


(c) ReduceTask Write Time

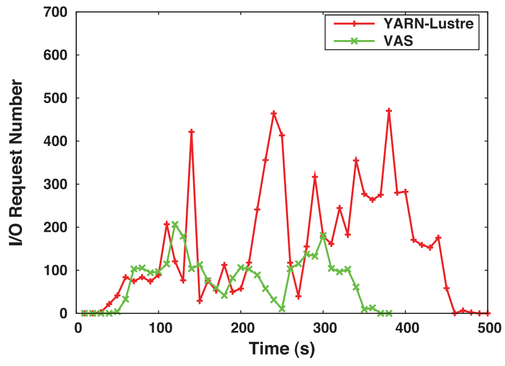


(d) Job Execution Time

Fig. 8. Locality improvement.



(a) Network Throughput



(b) Number of I/O Requests

Fig. 9. Profile of resource utilization.

performance improvement comes from the better locality in spilling data and storing output. Moreover, the result also demonstrates that leveraging data locality over Lustre can obtain better write performance for the ReduceTasks.

Fig. 8d shows the job execution time. VAS-Locality-Only demonstrates up to 4 percent better job execution time than YARN-Lustre for a varying amount of input data. The overall percentage of improvement on the job execution time is much smaller because the weights of the MapTask reading time and the ReduceTask writing time are quite low in the total job execution time.

5.4 Network and Disk I/O Utilization

We also examine the effectiveness of network and disk I/O utilization. These experiments are conducted with TeraSort running on 30 GB dataset. Fig. 12 shows the network throughput and I/O request numbers during the MapReduce job execution. In Fig. 9a, VAS cuts down the total network throughput by up to 26.7 percent since it introduces the stripe-aligned I/O and task scheduling, which allows both MapTasks and ReduceTasks to access data locally rather than resorting to remote storage targets in most cases, thus reducing the network transmission overhead, particularly in Steps 1, 2, 3, 5 and 6 as shown in Fig. 5).

Fig. 9b shows the I/O request numbers during execution. VAS significantly reduces the disk I/O request numbers by 62.3 percent on average compared to YARN-Lustre. This is because the avoidance of explicit shuffling in VAS, which eliminates the associated data spilling and retrieval. Therefore, at the beginning of job execution (namely the map phase), I/O requests in the two cases are very close to each

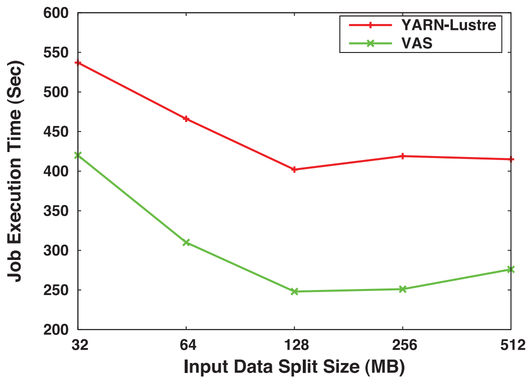


Fig. 10. The impact of input split size.

other. When the job further progresses to the reduce phase, we can observe that the I/O request number of VAS becomes much lower than YARN-Lustre. These results indicate that our techniques not only improves the shuffle phase but also provides an efficient pipeline of data merging and reducing, resulting in fast job execution.

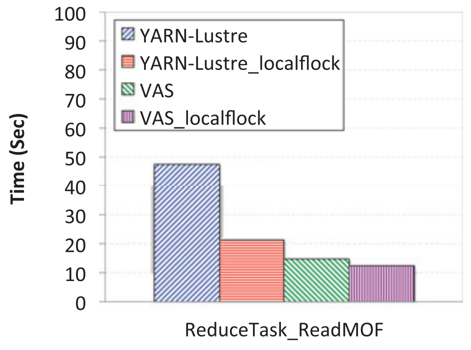
5.5 Tuning of Input Split Size

To gain more insights on the impact of the input split size, we measure the performance of TeraSort with the default striping configuration, i.e., when each MapReduce split is equal to a Lustre stripe. In our VAS framework, MapTasks in YARN take input data from Lustre. The choice of the input split size represents an important tradeoff between data access sequentiality and task execution parallelism. To be specific, a split size too small can increase scheduling overheads, while a split size too big may reduce the concurrency of task execution. Fig. 10 shows the execution time of TeraSort for YARN-Lustre and VAS with the input split size varying from 32 MB to 512 MB. Both schemes perform best when input split size is set as 128 MB. These results suggest that 128 MB provides the best tradeoff between task parallelism, management overhead and I/O granularity.

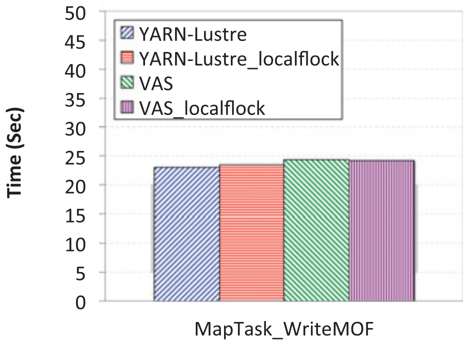
5.6 The Impact of Lock Management

Lustre distributed lock management is known as a critical factor for the parallel I/O performance. In our VAS framework, many I/O operations from many map and reduce tasks overlap with each other, thereby presenting a scenario of parallel I/O. To alleviate the impact of Lustre lock contention on the central MDS, there is an option (*localflock*) to delegate the management of Lustre locks to each OSS.

We have also measured the I/O performance of four different execution cases: YARN-Lustre, YARN-Lustre with localflock enabled (YARN-Lustre-localflock), VAS, and VAS with localflock enabled (VAS-localflock). Fig. 11 shows the MOF reading time for ReduceTasks and the MOF writing time for MapTasks when running TeraSort on 30 GB data. Fig. 11a shows that VAS with the localflock option (VAS-localflock) can improve the performance of reading intermediate files in ReduceTasks by up to 55.2 and 15.9 percent, compared to YARN-Lustre-localflock and VAS, respectively. To be specific, YARN-Lustre-localflock reduces the read time of MOFs by 69 percent compared to YARN-Lustre. This performance difference is due to the serious



(a) MOF Reading Time in ReduceTasks



(b) MOF Writing Time in MapTasks

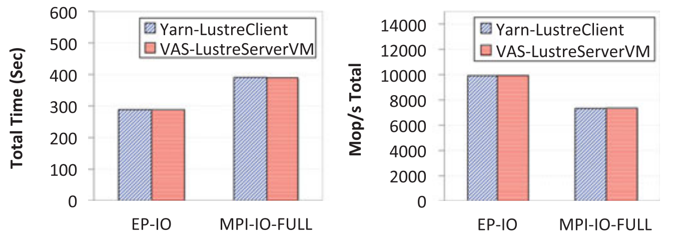
Fig. 11. Impact of Lustre lock management.

lock contention when many ReduceTasks are reading the same MOFs. The localflock option can greatly alleviate the impact of such contention. However, the option of localflock does not have a significant performance impact for the VAS framework. This can be explained by two reasons. First, our pipelined merging/reducing technique reduces the frequency of I/O operations. Second, VAS has modified the original MOF format. In the original YARN MapReduce, each MapTask generates one MOF file while ReduceTasks fetch their segments according to the offsets in the MOF file. In the VAS framework, each MapTask creates separate files, one for each ReduceTask. This avoids the lock contention issue on a shared MOF file.

As depicted in Fig. 11b, VAS-localflock slightly increases the average write time. This is because VAS leads to more metadata operations because of the creation of more intermediate files.

5.7 Scientific Applications

In this subsection, we run an HPC application—NPB BT-IO [13] on Lustre clients, at the same time launch a MapReduce analytic application—K-Means [7] in YARN. Two cases are evaluated. In the first case, the MapReduce programs are shipped to Lustre Storage Servers (VAS-LustreSeverVM). This configuration represents the scenario in which a MapReduce program is shipped to the backend storage servers. In the second case (VAS), an additional cluster of Lustre Clients (YARN-LustreClient) is configured to run MapReduce programs. This configuration represents the scenario in which a MapReduce program runs at the computing partition and has to retrieve data from HPC backend storage servers.



(a) Total Execution Time (b) Throughput

Fig. 12. The performance of NAS BT-IO when MapReduce programs are running on the backend storage servers.

NPB (NAS Parallel Benchmarks) are derived from computational fluid dynamics (CFD) applications. BT-IO is one of the NPB benchmarks that basically solves a set of nonlinear equations and writes the entire solution field storage every five time steps. Two I/O patterns EP-IO and MPI-IO-FULL are measured in this evaluation. EP-IO is Embarrassingly Parallel I/O, in which every processor writes its own file. MPI-IO-FULL adopts collective I/O to write a single file. K-Means clustering is a technique of vector quantization that aims to partition datasets into k clusters, in which each dataset of the cluster obtains the nearest mean. The dataset adopted in this measurement is a set of around 20,000 postings to 20 different newsgroups. We utilize a MapReduce program to solve the K-Means problem.

Fig. 12a and 12b show the job execution time and throughput of BT-IO for both EP-IO and MPI-IO-FULL cases, when a concurrent MapReduce program is running. As shown in the figure, shipping MapReduce programs to Lustre storage servers has negligible impact to the I/O performance of NAS BT-IO. This is because the virtualization technique in VAS provides good resource segregation, with little performance interference to Lustre storage servers.

We also measure the performance of MapReduce program (K-Means) when it is shipped to HPC storage servers (VAS-LustreSeverVM) compared to the case when it is not shipped (YARN-LustreClient). Again, in both cases, a concurrent I/O program (BT-IO) is running. As shown in Fig. 13, The VAS-LustreSeverVM configuration outperforms YARN-LustreSeverVM when the BT-IO program is running in either EP-IO or MPI-IO-Full mode. These evaluation results demonstrate that our VAS framework is able to deliver better analytics performance for MapReduce programs when they are shipped to

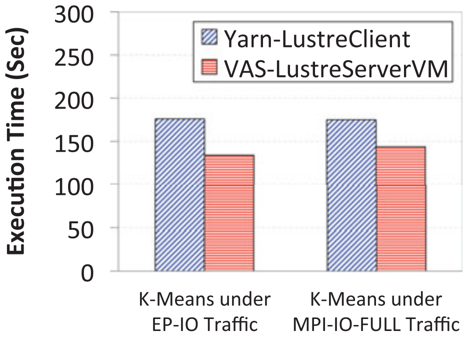


Fig. 13. The performance of K-Means when BT-IO is generating I/O traffic to HPC storage servers.

the HPC storage servers, avoiding the costly data movement from storage servers to the computing partition.

6 Related Work

Similar to our work, the *active storage* implementation in [18] was also designed to exploit the idle computation power in the storage nodes for running some general programs. Differently, our VAS framework leveraged the MapReduce programming model for parallel and efficient data analysis.

In order to avoid enormous data migration back and forth in the supercomputer, there have been many studies providing in situ data analytics along with scientific simulations [27], [31]. For example, Tiwari et al. [24] implemented an active flash prototype to conduct analysis on the solid-state drives before simulation results are stored back to disks. In addition, some other analysis techniques have been developed for better coordination with the large-scale scientific simulations. Among them, Bennett et al. [14] exploited the DataSpaces and ADIOS frameworks for supporting efficient data movement between in situ and in transit computations.

There have been several attempts of integrating MapReduce based data analytics model into compute-centric High Performance Computing environment to work with distributed file systems. For example, Ananthanarayanan et al. [12] compared the performance of HDFS with a commercial cluster filesystem (IBM's GPFS) for a variety of MapReduce workloads. Maltzahn et al. [17] showed the feasibility of executing Hadoop with the Ceph file system. Tantisiriroj et al. [22], [23] integrated PVFS with Hadoop and compared its performance with Hadoop on HDFS. However, the evaluation results showed that, after carefully tuning the parameters of the distributed file systems, Hadoop could only deliver similar performance as Hadoop on HDFS.

Wilson et al. [28] has explored a different approach to enable Hadoop MapReduce atop traditional HPC storage infrastructure. The authors have chosen to implemented a file system called Replicating Array of Independent NAS File System (RainFS). In contrast, our work pursues a new run-time system that ships MapReduce programs without having to replace proven HPC storage systems such as Lustre with a new file system.

In [16], performance studies were conducted to compare Hadoop with HDFS and Hadoop with Lustre. The results showed that Hadoop with Lustre performed significantly worse than Hadoop with HDFS mainly due to the inefficient intermediate data access and processing on Lustre. In most cases, integrating Hadoop with parallel file systems so far demonstrated unsatisfactory performance. That is because simply porting Hadoop to HPC systems would suffer from issues such as resource contention and performance interference, etc. Our virtualized analytics shipping framework on Lustre is able to address such issues through a set of new techniques and demonstrate more efficient analysis than Hadoop on HDFS.

7 Conclusion

Simulation codes on large-scale HPC systems are generating gigantic datasets that need to be analyzed for scientific discovery. While in-situ technologies are developed for extraction of data statistics on the fly, there is still an open question on how to conduct efficient analytics of permanent datasets that have been stored to HPC backend storage. In this paper, we have exploited the analytics shipping model for fast analysis of large-scale persistent scientific datasets on backend storage servers of HPC systems without moving data back to the compute nodes.

We have undertaken an effort to systematically examine a series of challenges for effective analytics shipping. Based on our examination, we have developed a Virtualized Analytics Shipping framework as a conceptual prototype, using MapReduce as a representative analytics model and Lustre as a representative HPC backend storage system With the VAS framework, we have also provided several tuning and optimizations including fast network and disk I/O through dynamic routing, stripe-aligned data distribution and task scheduling, and pipelined intermediate data merging and reducing. Together, these techniques have realized an efficient analytics shipping implementation, which supports fast and virtualized MapReduce programs on backend Lustre storage servers. In future, we plan to investigate the benefits of our VAS framework to short and ad-hoc analytics jobs, and its applicability to large-scale leadership computing facilities.

Acknowledgments

The authors would like to acknowledge the contributions of Yandong Wang while he was a student at Auburn and for the discussions they had with Eric Barton and Omkar Kulkarni from Intel. This work was supported in part by awards from Intel and Lawrence Livermore National Laboratory, and by the NSF awards 1059376, 1320016, 1340947 and 1432892. Weikuan Yu is the corresponding author of the article.

References

[1] Apache Hadoop NextGen MapReduce (YARN). [Online]. Available: http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/YARN.html, 2014.

[2] Apache Hadoop Project. [Online]. Available: http://hadoop.apache.org/, 2014.

[3] Center for Plasma Edge Simulation. [Online]. Available: http://cims.nyu.edu/cpes/, 2014.

[4] Chroot. [Online]. Available: https://wiki.archlinux.org/index.php/Change_Root, 2015.

[5] Gordon: Data-Intensive Supercomputing. [Online]. Available: http://www.sdsc.edu/supercomputing/gordon/, 2015.

[6] Kernel Based Virtual Machine. [Online]. Available: http://www.linux-kvm.org/page/Main_Page, 2014.

[7] KMeans over MapReduce. [Online]. Available: http://cmj4.web.rice.edu/MapRedKMeans.html, 2014.

[8] Lustre 2.0 operations manual. [Online]. Available: http://wiki.lustre.org/images/3/35/821-2076-10.pdf, 2011.

[9] The Combustion Research Facility at Sandia National Laboratories. [Online]. Available: http://crf.sandia.gov/, 2014.

[10] The Xen Project. [Online]. Available: http://wiki.xen.org/wiki/Main_Page, 2015.

[11] VMware, Inc. [Online]. Available: http://vmware.com/, 2015.

[12] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari, "Cloud analytics: Do we really need to reinvent the storage stack," presented at the *Proc. 1st USENIX Workshop Hot Topics Cloud Comput.*, San Diego, CA, USA, 2009.

[13] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, 1991.

[14] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *Proc. High Perform. Comput., Netw., Storage Anal. Int. Conf.*, Nov. 2012, pp. 1–9.

[15] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Opear. Syst. Des. Implementation—Volume 6*, Berkeley, CA, USA, 2004. USENIX Association, pp. 10–10.

[16] D. Luan, S. Huang, and G. Gong, "Using Lustre with Apache Hadoop," *Sun Microsystems Inc.*, 2009.

[17] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. J. Nelson, S. A. Brandt, and S. Weil, "Ceph as a scalable alternative to the hadoop distributed file system," *login: USENIX Mag.*, 2010.

[18] J. Piernas, J. Nieplocha, and E. J. Felix, "Evaluation of active storage strategies for the lustre parallel file system," IN *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. New York, NY, USA: ACM, pp. 28:1–28:10, 2007.

[19] K. Ren, Q. Zheng, S. Patil, and G. Gibson, "Indexfs: Scaling file system metadata performance with stateless caching and bulk insertion," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Piscataway, NJ, USA: IEEE Press, pp. 237–248, 2014.

[20] H. Shan and J. Shalf, "Using IOR to analyze the I/O performance for HPC platforms," In Cray Users Group Meeting (CUG) 2007, Seattle, Washington, USA, 2007.

[21] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*. Washington, DC, USA: IEEE Computer Society, pp. 1–10, 2010.

[22] W. Tantisiriroj, S. Patil, and G. Gibson, "The crossing the chasm: Sneaking a parallel file system into hadoop," in *Proc. Petascale Data Stroage Workshop*, 2008.

[23] W. Tantisiriroj, S. W. Son, S. Patil, S. J. Lang, G. Gibson, and R. B. Ross, "On the duality of data-intensive file system design: Reconciling hdfs and pvfs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 67. .

[24] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. J. Desnoyers, and Y. Solihin, "Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines," in *Proc. 11th USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2013, pp. 119–132.

[25] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, New York, NY, USA, 2013, pp. 5:1–5:16.

[26] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal, "Hadoop acceleration through network levitated merge," in *Proc. 2011 Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 57.

[27] B. Whitlock, J. M. Favre, and J. S. Meredith, "Parallel in situ coupling of simulation with a fully featured visualization system," in *Proc. 11th Eurographics Conf. Parallel Graph. Vis.*, Aire-la-Ville, Switzerland, 2011, pp. 101–109.

[28] E. H. Wilson, M. T. Kandemir, and G. Gibson, "Will they blend?: Exploring big data computation atop traditional hpc nas storage," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 524–534.

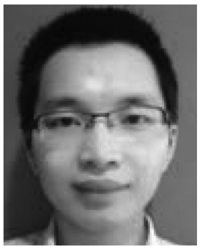
[29] W. Yu, J. S. Vetter, R. S. Canon, and S. Jiang, "Exploiting lustre file joining for effective collective I/O," presented at the *7th Int. Conf. Cluster Comput. Grid*, Rio de Janeiro, Brazil, May 2007.

[30] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2Nd USENIX Conf. Hot Topics Cloud Comput.*, Berkeley, CA, USA, 2010, pp. 10–10.

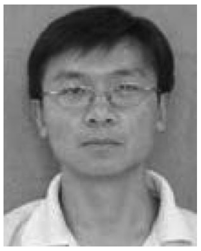
[31] F. Zhang, S. Lasluisa, T. Jin, I. Rodero, H. Bui, and M. Parashar, "In-situ feature-based objects tracking for large-scale scientific simulations," in *Proc. SC Companion*, IEEE Computer Society, 2012, pp. 736–740.

Cong Xu received the Bachelor's degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Master's degree from Auburn University, Auburn, AL, in 2012. He is currently working toward the PhD degree at the Parallel Architecture and System Laboratory in the Department of Computer Science and Software Engineering at Auburn University. His research interests include high performance computing, cloud computing, and big data analytics.

Robin Goldstone received the Master's Degree in computer science from California State University, Chico, CA. She is currently a computer scientist working in the High Performance Computing (HPC) division at Lawrence Livermore National Laboratory (LLNL), Livermore, CA. She is also a member of LLNLs HPC Advanced Technologies Office where she is involved in technology evaluation and planning for next generation HPC systems, as well as developing novel architectures for Data Intensive computing. She was previously involved in the deployment of some of the world's fastest supercomputers, including ASCI White and BlueGene/L.



Zhuo Liu received the Bachelor's degree from Huazhong University of Science and Technology, Wuhan, China, in 2007. He is currently working toward the PhD degree at the Parallel Architecture and System Laboratory in the Department of Computer Science and Software Engineering at Auburn University, Auburn, AL. His research interests include cloud computing, big data analytics, parallel I/O and storage systems.



Hui Chen received the Bachelor's and the PhD degrees from Beijing University of Posts and Telecommunications, Beijing, China, in 2006 and 2012, respectively. He is currently a postdoctoral researcher in the Parallel Architecture and System Laboratory in the Department of Computer Science and Software Engineering at Auburn University, Auburn, AL. Before coming to Auburn University, he worked in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences for two years. His research interests include cloud computing, energy efficiency management of data center and big data processing.

Bryon Neitzel received the Master's degree in computer science from Florida Atlantic University, Boca Raton, FL. He is currently the director of engineering in the High Performance Data Division at Intel Corporation, Santa Clara, CA. He leads development of Exascale IO software teams which are responsible for exploiting new hardware technologies to improve HPC IO. He also leads teams that develop Hadoop solutions on the Lustre filesystem, and previously led the Lustre software engineering team for the last four years.

Weikuan Yu received the Bachelor's degree in genetics from Wuhan University, Wuhan, China, and the Master's degree in developmental biology from the Ohio State University, Columbus, OH, where he also received the PhD degree in computer science in 2006. He is currently an associate professor in the Department of Computer Science and Software Engineering at Auburn University, Auburn, AL. Prior to joining Auburn, he served as a research scientist for two and a half years at Oak Ridge National Laboratory until January 2009. He is also a joint professor in the Departmental of Biological Sciences at Auburn University. At Auburn University, he leads the Parallel Architecture and System Laboratory for research and development on high-end computing, parallel and distributing networking, storage and file systems, as well as interdisciplinary topics on computational biology. He is a senior member of the IEEE and member of the ACM.