Data Management Strategies for Scientific Applications in Cloud Environments

Devarshi Ghoshal[1], Valerie Hendrix[2], Eugen Feller[23], Christine Morin[3], Beth Plale[1], Lavanya Ramakrishnan[2]
[1]School of Informatics and Computing, Indiana University, Bloomington, IN, USA
[2]Inria Centre Rennes - Bretagne Atlantique, 35042 Rennes Cedex, France
[3]Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA
Email: dghoshal@cs.indiana.edu, vchendrix@lbl.gov,eugen.feller@inria.fr,
christine.morin@inria.fr,plale@cs.indiana.edu, lramakrishnan@lbl.gov

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

Abstract

Clouds are increasingly being used for running data-intensive scientific applications. However, science applications need to contend with the I/O and network performance characteristics of cloud environments. Additionally, managing data effectively and efficiently over these cloud resources is challenging due to the myriad storage choices with different performance-cost trade-offs, complex application choices, complexity associated with elasticity and failure rates. In this paper, we evaluate various aspects of data management strategies in cloud environments. Our evaluation is performed in the context of two frameworks - Hadoop and FRIEDA and conducted on four cloud testbeds - FutureGrid, ExoGeni, Grid5000, Amazon. Our experiments highlight the different performance implications of storage, file system, provisioning choices in the context of data management strategies.

1 Introduction

Cloud resources are increasingly gaining popularity for data-intensive applications due to their ease of management and on-demand scalability. Infrastructure as a Service (IaaS) cloud services provide control over provisioning of hardware resources and selection of software stacks.

Cloud resources present a number of challenges for data-intensive applications. First, scientific applications often have large amounts of data and data might be generated at experimental facilities (e.g., Advanced Light Source, Large Hydron Collider) and the data needs to be moved to cloud environments. Previous work has also shown that the I/O and network can often be bottlenecks in cloud environments [1]. Thus, it is critical to manage both data management and movement carefully in cloud environments to alleviate this problem. Second, cloud environments provide a myriad of temporary and permanent storage resources (e.g., transient local disks, block store volumes and object stores) and file systems. Each of these resources has various performance, price and size trade-offs [1, 2]. Thus, users need to pick and compose their storage choices. There is a limited understanding today on what might be optimal choices for an application.

Previous work has investigated various aspects of managing the compute resources and associated software stack for scientific applications in IaaS environments [3, 4]. However, the work in storage and data management has been largely focused on storage services for storing and retrieving data [5, 6, 7]. Applications running in cloud environments need to manage data transfer in and out of cloud resources, across the nodes of the virtual cluster and across multiple sites. The transient nature of virtual machines, the different performance and cost trade-offs of storage options, elasticity and failure rates make cloud data management difficult. In addition, different applications have different data characteristics that can also affect the data management decisions. For example, BLAST, a bioinformatics application relies on a database that needs to be available to each task. In contrast, an image analysis pipeline that compares images with other images in the set, lends itself to data partitioning across nodes. Thus, we need a cloud data management framework that is not only flexible to application needs but provides ways to manage the cloud characteristics.

In this paper, we evaluate various aspects of data management strategies in cloud environments. This evaluation is performed in the context of two frameworks for data-intensive applications on four cloud testbeds (FutureGrid, ExoGeni, Grid5000 and Amazon). We use Hadoop [8] and FRIEDA [9] as the frameworks for our testing. Hadoop is an open source implementation of MapReduce [10]. FRIEDA (**F**lexible **R**obust **I**ntelligent **E**lastic **DA**ta Management) framework provides a two-level execution mechanism separating the data control from the execution phases. We evaluate and compare data management strategies on different machines and resource types. Specifically, in this paper:.

- We evaluate the effect of storage options and file systems on both FRIEDA and Hadoop application.
- We evaluate effect of different data management strategies that use provisioned resources intelligently for data-intensive jobs.
- We provide a discussion on effective data management for scientific applications in cloud environments.

The rest of this paper is organized as follows. In Section 2 we describe the Hadoop and FRIEDA data management frameworks. We describe our evaluation methodology in Section 3. We present our evaluation in Section 4. We discuss the implication of the results in Section 5 and present related work in Section 6. Finally, we present our conclusions in Section 7.

2 Background

In this section, we briefly describe both Hadoop and FRIEDA and the pipeline strategies implemented in both frameworks.

2.1 Hadoop

Hadoop is a software framework for managing data-intensive applications in distributed environments based on the MapReduce programming model. It is designed to scale up to thousands of nodes, where each node

	Grid'5000	**Amazon Web Services**	**FutureGrid**	**ExoGENI**
Cloud System	Snooze	Amazon EC2	OpenStack Grizzly	Eucalyptus with virtio
Instance Descriptions	s1.small: 1 VCORE 1GB RAM, 40GB disk s1.large: 4VCORES, 2GB RAM, 40 GB disk	m1.small: 1 VCORE, 1.7GB RAM, 160 GB disk m1.large: 2 VCORES, 3.7GB RAM, 840GB disk	m1.small: 1 VCORE, 2GB RAM, 20GB disk m1.large: 4 VCORE, 8GB RAM, 80GB disk	m1.xlarge: 1 VCORE, 4GB RAM, 64GB disk
Storage Options	local disk	local disk	local ephemeral disk and persistent block store	local ephemeral disk and iSCSI persistent disk
OS	Debian sid	CentOS 6.5	Ubuntu 12.10	Debian 6(squeeze) v1.0.10

Table 1: Summary of the test bed configurations.

offers both local computation and storage. Its fault-tolerant architecture manages both job and data failures by replicating data to multiple nodes. We describe the components of Hadoop as per its release 0.20.0.

HDFS. The Hadoop distributed file system (HDFS) is a scalable, portable and distributed filesystem that can store large files across multiple nodes. It achieves reliability by replicating the data across multiple nodes. It achieves higher performance due to the awareness of the data location. This is helpful because it enables local computation by avoiding expensive data transfers during computation.

MapReduce Engine. MapReduce engine consists of one JobTracker and several TaskTrackers. Client applications submit MapReduce jobs to the JobTracker. The JobTracker pushes work out to available TaskTracker nodes within a cluster and keep the processing as close to the data as possible. Since the MapReduce engine run atop HDFS, the JobTracker knows which node contains the data, and the spatial locality of the other nodes. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a TaskTracker fails or times out, that part of the job is rescheduled on another node.

Scheduling. Hadoop has a pluggable scheduler with a number of scheduling algorithms. By default Hadoop uses a FIFO scheduler. For this scheduler, a JobTracker pulls the oldest job first from the work queue. Other scheduling algorithms, like fair scheduling and capacity scheduling, can also be used. Alternate schedulers can be added based on the implementation requirements. We use Hadoop with the FIFO scheduler in our evaluation.

Data Management in Clouds. In this paper, we use Hadoop for managing data on cloud platforms for understanding both performance and scalability on different cloud configurations for provisioning of resources. Hadoop provides a transparent data management framework that relies heavily on its filesystem and the semantics of the MapReduce jobs. While using HDFS as its underlying filesystem, Hadoop distributes the data to several nodes and schedules the map tasks accordingly. But due to its random distribution of data blocks, an expensive data movement and transfer occurs during the shuffle and reduce phases of a MapReduce job. A MapReduce job in Hadoop comprises of five stages. a) Job submission tells the JobTracker that the job is ready for execution, b) Job initialization creates a list of tasks encapsulated within a job based on the distribution of data across the nodes, c) Task assignment requires the JobTracker to assign a new task to the TaskTracker, d) Task execution deals with the TaskTracker running the task assigned to it and reporting the progress until the task completes and e) Job completion is notified to the JobTracker once the last task for a job is complete.

It is evident that the job assignment and execution largely depends on the distribution and management of data across the cluster. This is neither controlled by the user nor by the semantics of the program in execution.

2.2 FRIEDA

FRIEDA provides a two-level execution mechanism that separates the data control from the execution plan [9]. The controller and partition generation algorithm set up the environment for data management and program execution. The execution plane is based on the master-worker paradigm where the master and the workers manage execution. The controller first starts the master and initializes it with the partition strategy to be used for execution and then starts and initializes the workers. This separation of concerns allows for flexible implementation of different data management strategies within the same framework as suitable to specific application and resource needs.

Our focus in this paper is on high-throughput and data-intensive applications with negligible communication and hence we focus largely on the master-worker programming paradigm. The master decomposes the problem into small tasks and distributes these tasks for execution on the workers. The communication is between the master and the workers and the master is responsible for aggregating the partial results to produce the final result. We assume that there is no specific task affinity to a certain machine i.e., tasks are divided equally among all virtual machines (VMs).

Our framework supports two modes of data partitioning and distribution strategies: a) pre-determined and b) real-time. In a pre-determined strategy, data is pre-partitioned and moved to the storage on individual workers before the start of the execution. In real-time mode, each worker receives the data during execution. Our previous work, contrasts the benefits of these two approaches [9].

2.3 Pipelining Strategies

We have developed two pipelining strategies which differ in the concurrent execution of the stages of the pipeline for both Hadoop and FRIEDA.

Clubbed Pipeline: In the clubbed pipeline strategy, the last two stages are clubbed together into a combined data transfer and execution phase where each stage gets exclusive access to the network. This is a more conservative approach to pipeline data-intensive jobs.

Full Pipeline: Clubbed pipelines can result in under-utilization of resources when the requirements (movement to HDFS + task execution) of individual subtasks do not utilize the provisioned resources to its full. To avoid under-utilization, second stage of one subtask can be executed concurrently with third stage of another subtask. In some cases, this could result in over-utilization of resources and can adversely affect the performance.

As mentioned before, there are two types of placement strategies in FRIEDA - pre-determined and real-time. In pre-determined mode, each individual chunk is transferred and then processed by FRIEDA. Thus, the pre-determined mode implements the clubbed pipeline where the transfer and execution phases are clubbed together. The processing of different chunks do not overlap with each other. In real-time mode, data is transferred and processed at real-time by FRIEDA. This mode uses the full pipeline since internally data transfer and processing of individual chunks can overlap and share network resources.

3 Methodology

In this section, we detail our experiment methodology with a focus on the platform setup, workloads, and the experiment scenarios. Application use cases were performed on ExoGENI, FutureGrid, Grid5000 and Amazon using the FRIEDA Execution framework. The applications and their input data were setup using FRIEDA and run with real-time data partitioning unless otherwise specified.

3.1 Testbeds

Our platform setup comprises VMs from public and private cloud environments like Grid‘5000, Amazon Web Services, FutureGrid and ExoGENI (see Table 1).

Amazon Web Services. We used Amazon EC2 to provision VMs for our public cloud experiments. We used VMs of types m1.small and m1.large. Small instances are configured with one virtual core (VCORE), 1.7 GB RAM, and 160 GB local disk. Large instances have two VCORES, 3.7 GB RAM, and 840 GB local disk. All instances are running the CentOS Linux distribution.

App	Instances	Input Sizes	Storage Configurations
FutureGrid experiments			
BLAST	24 m1.small 6 m1.large	2500 7500	local on worker block storage on worker(m1.large only)
ImgComp	24 m1.small 6 m1.large	50 100 200 400	local on master with NFS block storage on master with NFS
Amazon EC2 experiments			
ATLAS	24 m1.small 12 m1.large	400 800	local on worker
ImgComp	24 m1.small 12 m1.large	50 100	
ExoGENI experiments			
BLAST	16 m1.xlarge	7500	local and iSCSI disks on worker
ImgComp	16 m1.xlarge	30 50	
Hadoop Wikipedia	16 m1.xlarge	10 30 60 90 180	

Table 2: Science Application Experiments. All FutureGrid and Amazon EC2 experiments ran 24 concurrent tasks which required a different number of VMs depending on instance type and IaaS provider. Storage was either provided at the worker or using a shared filesystem on the master. ExoGENI experiments ran 16 concurrent tasks on 16 m1.xlarge instances. Input sizes for ATLAS and ImgComp are measured in total gigabytes for all input files. BLAST input sizes refers to the total number protein sequences queries.

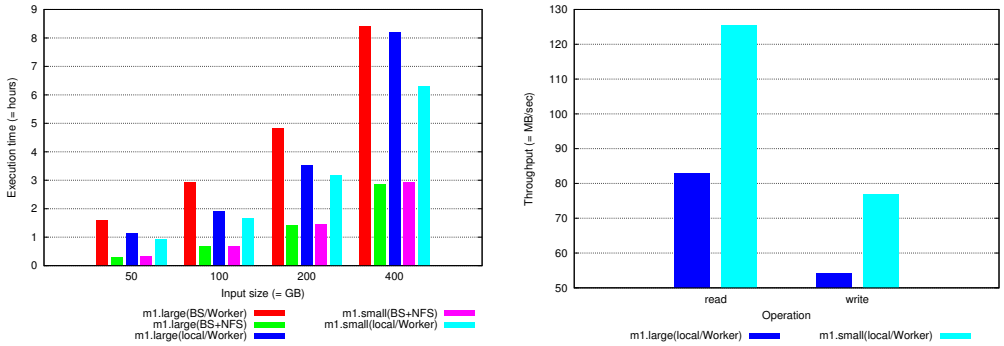
FutureGrid. We use a private FutureGrid reservation of OpenStack Grizzly. The deployment consists of four compute servers and one OpenStack management server. Each compute server has two Intel Xeon X5550 2.66 GHz CPUs (4 cores per CPU), 2.5 TB of local ephemeral disk storage, and 24 GB RAM. The servers are interconnected using 1 Gb Ethernet. The management node has a block store service and 2.5 TB of shared storage allocated for providing volumes over iSCSI.

ExoGENI. We also use the ExoGENI cloud testbed [11] that provides a widely distributed experimental networked infrastructure-as-a-service (NIaaS) platform for evaluating data management and resource provisioning optimizations. We use the Open Resource Control Architecture (ORCA) control framework software for allocating and managing compute resources and layer 2 (data link layer) network links. Individual ExoGENI deployment consists of 16 xlarge instances, each with 64 GB local disk and 100 GB high-speed iSCSI disks, 4 GB RAM and x3650 M4 servers with SandyBridge with 10G NICs. We provision all the VM instances with a 1 Gbps network link unless stated otherwise. The VMs are managed using Eucalyptus with virtio.

Grid'5000. VM provisioning on Grid'5000 is managed through the Snooze [12] cloud management system across 15 Dell PowerEdge R720 servers of the taurus cluster. Each server is equipped with two Intel Xeon E5-2630 2.3 GHz CPUs (6 cores per CPU), 32GB of RAM, and 598GB of disk space with 10 Gb Ethernet and running Debian wheezy with a 3.2.0-4-amd64 kernel. All VMs are using the virtio disk and networking drivers. We have created two types of VM instances: s1.small and s1.large. Small instances are configured with one virtual core (VCORE), 1GB of RAM, and 40GB of disk space. Large instances have two VCORES, 2GB of RAM, and 40GB disk space.

3.2 Workloads

For understanding the performance of data-intensive jobs on clouds we consider two kinds of software infrastructure, each with a set of applications and benchmarks.



(a) ImgComp on FutureGrid

(b) IOR MPIIO on FutureGrid

Figure 1: (a) ImgComp with 24 concurrent tasks, time by input data size. This shows image comparisons using 24 concurrent tasks on differing input data and virtual machine sizes. Local disk is faster than block storage. NFS is significantly faster than data movement; (b) Large instances are slower than small in data movement scenarios which is mirrored in our IOR MPIIO tests on 24 processes. In this test, we used 24 processes reading/writing a file per process to local disk which confirms that small instances are faster than large for both read and write in this setup;

1) Hadoop: Hadoop provides a distributed computing framework on a cluster where users either have very little or no control over the distribution of data. We use three applications: 'filter', 'merge' and 'reorder', on large volumes of Wikipedia data. The filter application indexes individual Wikipedia pages, the reorder application replaces <timestamp> tags with <date+time> tag and the merge application indexes individual lines on a Wikipedia page. 'Filter' reads more data than it writes, 'merge' writes more data than it reads whereas 'reorder' reads and writes equal amount of data. For all the tests, the amount of input data is same, but the amount of intermediate and final output data differs.

2) FRIEDA: We use three workloads in our evaluation: ImgComp (Image Comparison), BLAST (Basic Local Alignment Search Tool), and ATLAS (A Toroidal LHC Apparatus). FRIEDA provides a framework to experiment with different storage and data management strategies through the use of various storage devices, file systems and storage configurations. Our experiments focus on evaluating the different storage choices (local, block store) in the context of scientific applications (ImgComp, BLAST, and ATLAS). We demonstrate the impact of storage and file system options that drive the policies in FRIEDA.

Image Comparison Application (ImgComp). We use data from a light source beamline for ImgComp. The application compares images to determine the similarity. It requires two files for every execution and requires moving the image files to the cloud before operating on them. Data and compute-wise ImgComp application deals with many image files for every computation. The size of each image file is approximately 11MB. The input size of a test refers to the total size in GB of all the image files to be compared.

Basic Local Alignment Search Tool (BLAST). BLAST, a bioinformatics application, allows comparison of primary biological sequences for different proteins against a sequence database. We measured input size by the total number of database protein sequence queries. BLAST test cases had two input sizes: 2500 and 7500 protein sequence queries. These were split into files that contained 10 sequences each. Each task execution received a single input file which was used to query against a reference database residing on configured storage.

ATLAS. We ran an ATLAS user analysis application to enable parallel processing of D3PD [13] ATLAS datasets. The dataset we use is composed of multiple D3PD files. Each task execution requires one dataset and outputs a set of histograms. The size of each D3PD file is approximately 1.5GB. The input size of a test refers to the total size in GB of all the D3PD files to be processed.

3.3 Scenarios

In our experiments, we consider the following storage classes of devices: a) local disks on virtual machines which are pre-mounted at boot-up time, b) block store devices where storage is exposed as a physical device that end-user has to prepare (i.e., create partitions and format) and, c) object stores where the user might specify that the output should be stored. The first two classes might have size limits associated with them as configured by a site administrator.

In the application experiments in Section 4.1, we ran 24 concurrent tasks which required a different number of VMs depending on instance type and IaaS provider. Storage was either provided at the worker or using a shared filesystem on the master. On FutureGrid, we ran ImgComp and BLAST test cases (Table 2) with 24 concurrent tasks varying input sizes, storage types and VM instance types (24 m1.small or 6 m1.large). Storage was was either provided locally on the worker or remotely on the master over a network file system (NFS). A limitation of 10 block store volumes prevented us from testing 24 m1.small workers on FutureGrid. On Amazon EC2, we ran ImgComp and ATLAS test cases (Table 2) with 24 concurrent tasks across 24 m1.small and 12 m1.large instances. Local ephemeral disks on the workers were used in all the experiments.

Input files were placed on block storage and attached to the FRIEDA master during data placement. FRIEDA's real-time data partitioning behaves differently with local and shared file systems. If the workers used a local file system, the FRIEDA master delivered the file via secure copy upon a worker request. However, in the case of NFS, the master hosts an NFS server mounting the block storage volumes containing the input data. In this case, all FRIEDA workers participate as NFS clients obtaining input files via NFS.

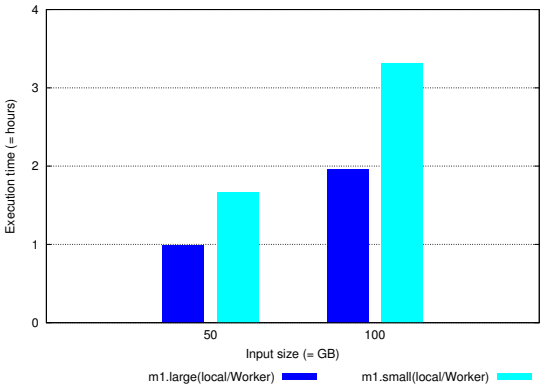


Figure 2: ImgComp on Amazon EC2 m1.small and m1.large instances. Execution time is faster on m1.large instances than on m1.small instances.

Our Hadoop evaluation on Grid'5000 investigates the Hadoop performance using three file systems: NFS, GlusterFS, and the Hadoop Distributed File System (HDFS). To conduct the experiments we have deployed Hadoop v0.20.2 on the hercule cluster. The first hercule server acted as the JobTracker and the file system (NFS and GlusterFS) master services. Each task tracker was configured with 10 maps and 1 reduce. The block size was set to 128MB. We run our Wikipedia data processing application for each file system with 50, 100, and 150GB input size. In total 60 maps and 3 reduce slots were used.

On ExoGENI, we used 16 m1.xlarge compute instances for running both ImgComp and BLAST applications as well as Hadoop Wikipedia processing (Table 2). We used both local and iSCSI disks to evaluate relative storage performance and scalability.

4 Evaluation

In this section, we describe the results of our evaluation. Specifically, we cover the results from our experiments:

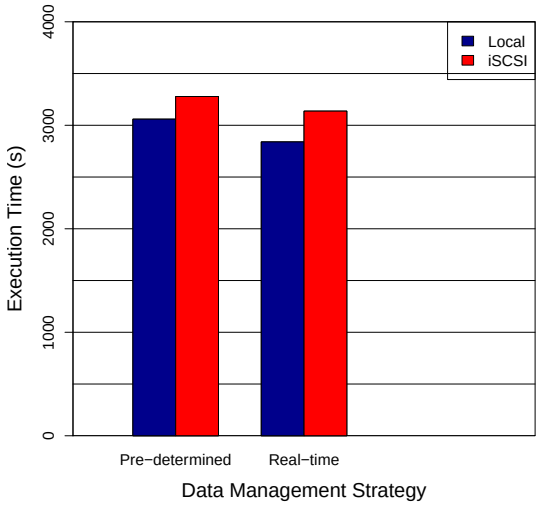
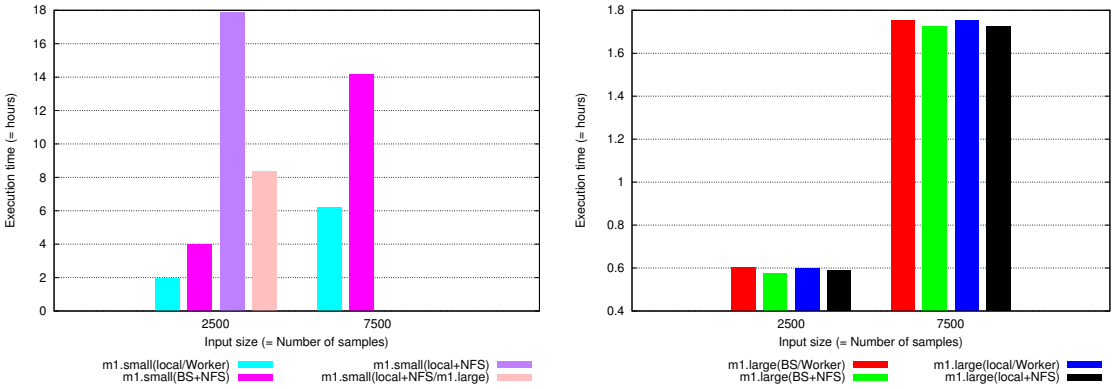


Figure 3: Performance evaluation of ImgComp with the two data management strategies in FRIEDA using different storage options at ExoGENI. Local disk performs better than iSCSI for both data management strategies.



(a) Small Instances

(b) Large Instances

Figure 4: BLAST executions ran 24 concurrent processes varying the storage scheme. (a) NFS is considerably slower than data movement to workers. The worst case scenario is sharing the master's local disk over NFS. The larger instance size of the master cuts the execution time in half. (b) Execution time scales linearly with input size with local disk being slightly faster than block storage.

- We evaluate the storage and filesystem options with the FRIEDA and Hadoop applications on all four testbeds
- We study the data management strategies (pre-determined and real-time) in the context ImgComp and BLAST.
- We evaluate the effects of network provisioning on ImgComp and BLAST applications.
- We study the effects of pipelining in both Hadoop and FRIEDA.

4.1 Effect of Storage and FileSystem Options

In this section, we discuss the effect of different storage options of ImgComp, BLAST, ATLAS and Hadoop applications on the four testbeds.

ImgComp ImgComp is a read-intensive application. The image comparison algorithm receives two beamline image files as input and outputs whether a match was found.

Figure 1 shows the results of running 24 concurrent tasks and plots the execution time by input data size for the differing compute and storage configurations from Table 2. Execution time scales linearly as the data size increases.

Figure 1a shows the results on FutureGrid. It shows that overall NFS (`BS+NFS`) was faster for all input sizes and instance types than transferring data to the worker. Among cases where data is on the worker (`BS/Worker`, `local/Worker`), local disk is faster than block storage. In the NFS cases, large instances were slightly faster than small.

For ImgComp, the cost of data transfer makes a difference as the input data size increases. The worst case for ImgComp, was using large instance types and moving data from master to block storage volumes on workers (`m1.large (BS/Worker)`). Data movement test cases seem to scale at the same rate until 400GB input size. At 400 GB, `m1.large (local/Worker)` performance starts closing in on `m1.large (BS/Worker)`. This could be point at which the network begins to get saturated by data movement and writing to the shared block store server. In this case, small instances were faster than large instances.

To understand the cause of this better, we ran IOR MPIIO read/write tests with 24 concurrent processes (see Figure 1b). The tests wrote a file per process to local disk. Small instances outperformed large on both read and write. This is likely due to the interplay of the virtual machine load with the virtualization overhead coming out of a single virtual machine. This might be the result of contention between the four processes that are writing to the VM local disk in the case of large instances that we don't see in the small instances.

Figure 2 shows the ImgComp performance on Amazon EC2 using 24 concurrent processes on m1.small and m1.large instances. As it can be observed, execution time is faster on m1.large instances and is proportional to the input data size. This is different than the results we observed on Futuregrid.

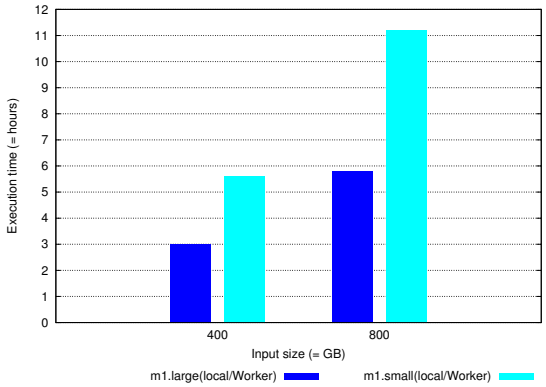
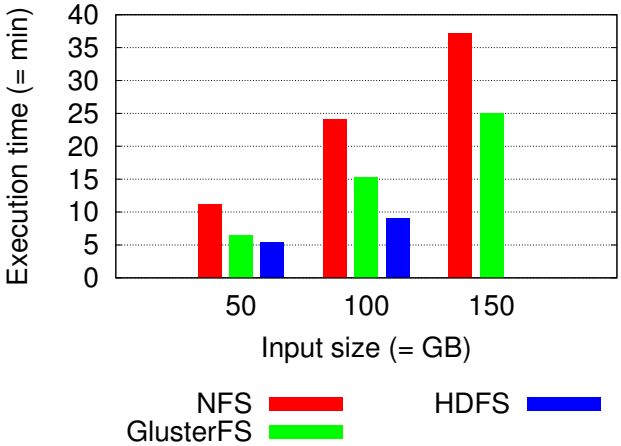


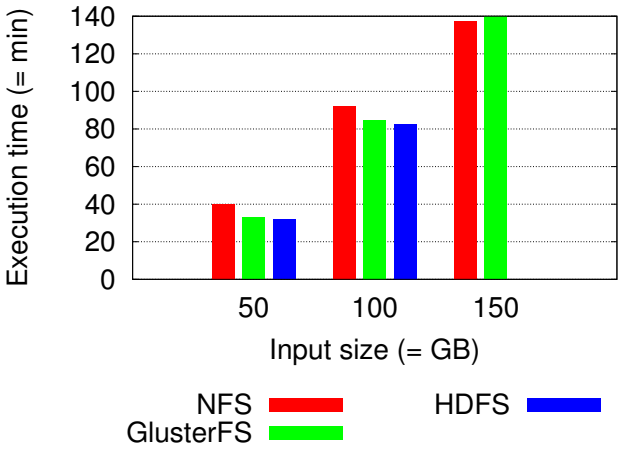
Figure 5: ATLAS data processing on m1.small and m1.large instances for 400 and 800 GB of data. Execution time on m1.large instances is faster than on m1.small instances and doubles with increased data size.

A closer inspection of the VM specifications shows that Amazon instance types are configured differently than the Futuregrid reservation. Amazon VMs have less RAM per core than their Futuregrid counterparts. Furthermore the Futuregrid m1.small VCORE counts matched and the m1.large did not. Amazon m1.large VCORE counts where half (2) that of Futuregrid (4). In keeping our experiments consistent by always running the science applications with 24 concurrent processes, this difference in configuration resulted in running 12 VM instances on Amazon versus 6 for Futurgrid. This reduced the contention for disk as seen previously in the Futuregrid experiments which resulted in the m1.large local disk experiments being slower than m1.small.

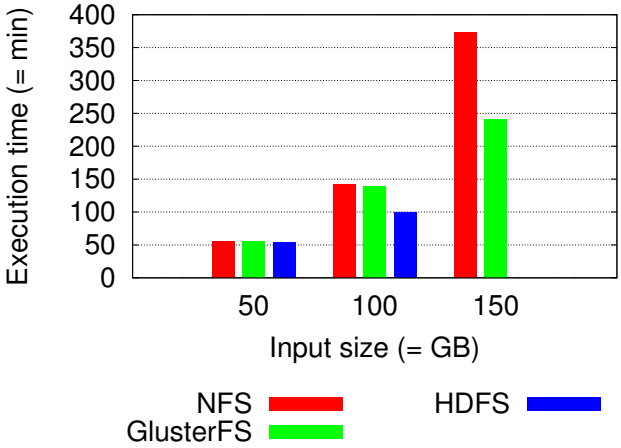
We also used different storage options at ExoGENI to evaluate different data management strategies. Figure 3 shows that both data management strategies for ImgComp application processing 30GB of input



(a) filter



(b) reorder

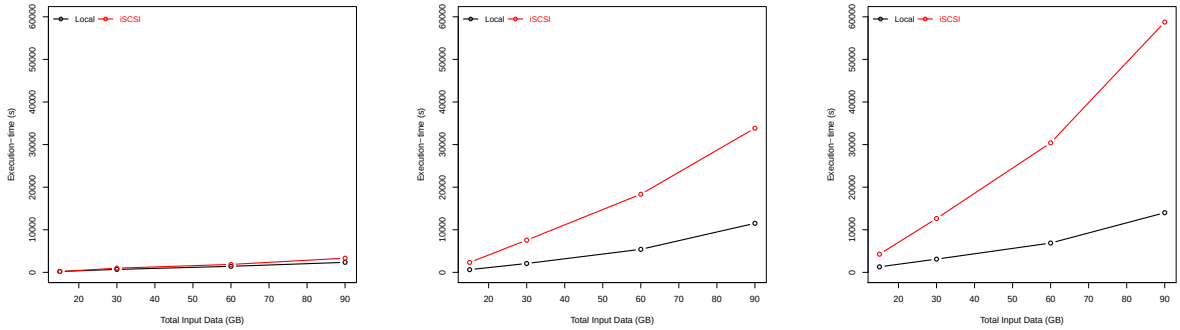


(c) merge

Figure 6: Hadoop Wikipedia data processing for three data-intensive operations on Wikipedia data on physical servers. HDFS performs the best due to localilty for the read-intensive filter operation. NFS server performs the worst for the write-intesive merge operation.

data have better performance on local as compared to iSCSI disks. The results also suggest that iSCSI disk performance can be improved by selecting the appropriate data management strategy.

BLAST BLAST is a memory-intensive application. A single input file with a list of protein sequences is used to query a 6 gigabyte reference database. The total size of the largest input case (7500 protein sequences) is 3 megabytes. All executions ran 24 concurrent tasks with varying storage scheme and virtual machine type. Most of the experiments failed to fit the 6 GB reference database in memory due insufficient RAM on the VMs.



(a) Filter Benchmark (b) Reorder Benchmark (c) Merge Benchmark

Figure 7: Performance of Hadoop applications on different storage options without pipelining strategies. For large datasets without pipelining, write-intensive operations like (b) reorder and (c) merge perform significantly worse on iSCSI disks than on local disks. But, iSCSI disks can process larger datasets due to extra storage.

Figure 4 plots BLAST protein queries total execution time by total number of protein sequences. Note, the time scale between the small (left figure) and large (right figure) instances are very different. The range of execution time for small instances was between two and eighteen hours while large instances all finished in less than two hours.

BLAST performance is very slow on small instances which is in part due to insufficient memory for the 6 GB BLAST reference database. Data sharing with NFS is considerably slower than data movement to workers. Due to these exceedingly long run times, the input size of 7500 protein sequences for local disk over NFS test cases were not run.

In cases where data was shared through the NFS server, sharing the master's local disk over NFS can lead to contention with the BLAST reference database. We were able to cut the time by a little more than half from 18 to 8 hours by changing the master to a large instance somewhat alleviating this contention. The lower performance can be explained by the limited memory and CPU resources available on the small instances. There is contention for memory resources between the NFS clients and BLAST database. Increasing the instance size of the master alleviated some of this contention. The block storage over NFS was dramatically faster than local disk over NFS at 4 hours versus 8 and 18. Moving the I/O off of the VM to block storage reduced resource contention inside the VM.

Figure 4 shows that for large instances tripling the input size tripled the execution time and that NFS is slightly faster than transferring data to the worker. Large instances have more memory and execute four workers (1 per core). Having four workers was an advantage for the BLAST application which benefited from sharing the in memory BLAST database.

ATLAS. The ATLAS application was only run on Amazon. Figure 5 shows the ATLAS execution time on m1.small and m1.large instances on Amazon EC2. As it can be observed, execution time is faster on m1.large instances and increases proportionally with the data size. This demonstrates that the application benefited from the extra memory per core provide by the Amazon m1.large instance type.

Hadoop. Figure 6 depicts the results from our Hadoop experiments on Grid5000. As it can be observed, HDFS performs the best in all the experiments. This is not surprising Hadoop applications can greatly benefit from the HDFS local read optimizations and data locality. However, when considering the reorder

and merge operations for smaller inputs (i.e., 50 and 100 GB), NFS and GlusterFS do not yield significant performance degradation. On the other hand, filtering performs significantly better using HDFS for all inputs as it is dominated by reads. The reorder operation, performs similarly for both NFS and GlusterFS for all input sizes, with GlusterFS beeing slightly better. Finally, when considering the merge operation, GlusterFS outperforms NFS at larger inputs (i.e. 150GB) due to its better write performance.

We evaluated Hadoop performance on both local and iSCSI disks at ExoGENI. The total local disk space was around 1 TB (64 GB on 16 VM instances), but we could go only upto 120 GB of total input data with total DFS usage of around 600 GB during execution. Figure 7 shows comparison of the three Hadoop applications (filter, merge and reorder) with local and iSCSI disks without any pipelining strategy. For filter applications, the difference between iSCSI and local disks is minimal since this application reads a lot but has minimal writes. Though we observe that at larger data sizes, the gap between local and iSCSI disks starts increasing. In contrast for reorder and merge applications, we observe that the iSCSI disks start to perform poorly when handling larger amounts of input data size. Reorder and merge are write-intensive applications and are hence affected by the network latency to the shared network storage.

4.2 Efffect of Data Management Strategies

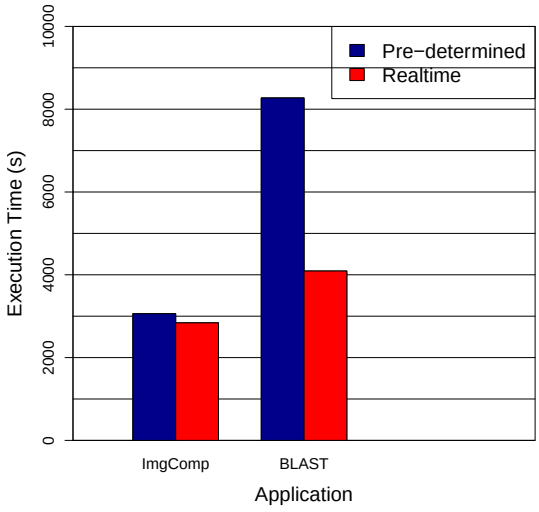
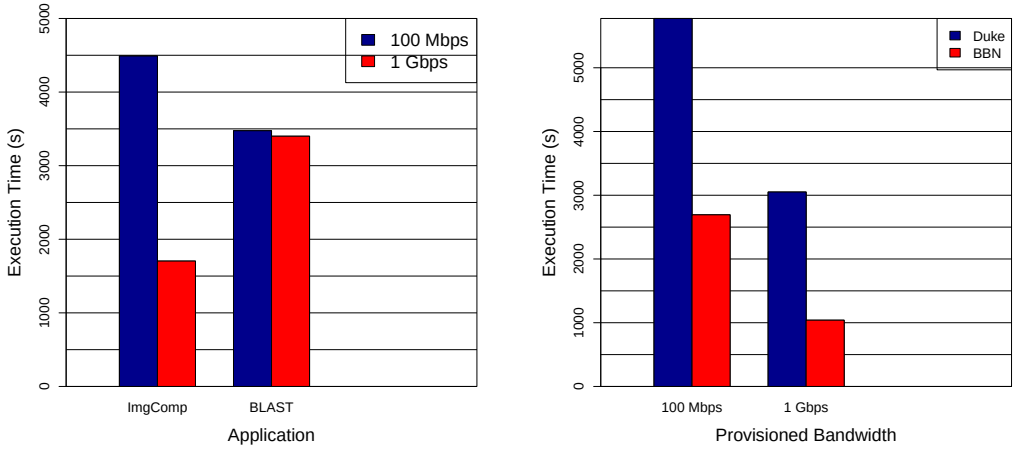


Figure 8: Performance variation of ImgComp and BLAST using the two different data management strategies in FRIEDA on ExoGENI. Whereas BLAST achieves significant performance improvement using real-time data management, ImgComp achieves similar performance with both strategies due to high provisioned network bandwidth.

Figure 8 shows performance of ImgComp and BLAST for the two data management strategies in FRIEDA on ExoGENI. The results show ImgComp execution with 30GB of total input data and BLAST execution with 7500 protein sequences. For ImgComp, both strategies have almost similar performance. Real-time partitioning performs 2× better than pre-determined partitioning for the BLAST workload due to higher degree of load imbalance between tasks in the workload.

4.3 Effect of Network Provisioning

Figure 9a shows the impact of bandwidth provisioning on BLAST and ImgComp. In this experiment, BLAST processes 7500 sequences and ImgComp processes 50GB of data. It shows that BLAST has minimal or no gain from network provisioning of 100 Mbps vs 1Gbps since it is a compute- and memory-intensive application. The ImgComp application shows significant improvement of 2.5× with higher bandwidth network



(a) Effect of Network Provisioning

(b) Site-wise Performance

Figure 9: Effect of variable network provisioning. a) Applications like ImgComp see significant improvement in performance for high provisioned network. b) Evaluation of ImgComp at different sites and with different network bandwidths shows that even with lower bandwidths higher performance can be achieved at BBN with similar instances due to better local I/O performance.

provisioning since the application moves a large amount of data. Therefore, it is important for systems to perform intelligent provisioning based on application profiles.

Figure 9b shows the performance ImgComp processing 30GB of input data at two different sites with different provisioned network bandwidth. The results show that with same instance-types and provisioned network, one site outperforms the other. This may be attributed to higher disk throughput at the BBN site. This suggests that if inter-site performance varies for similar instance-types, cost of provisioning can still be reduced without sacrificing application performance by intelligently selecting appropriate resources (for example, lower bandwidth networks at BBN instead of Duke).

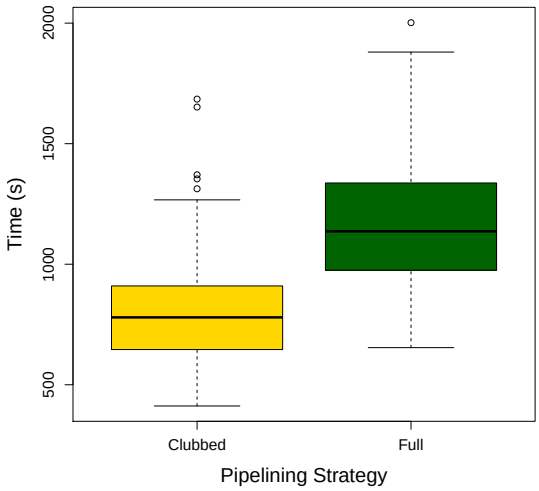
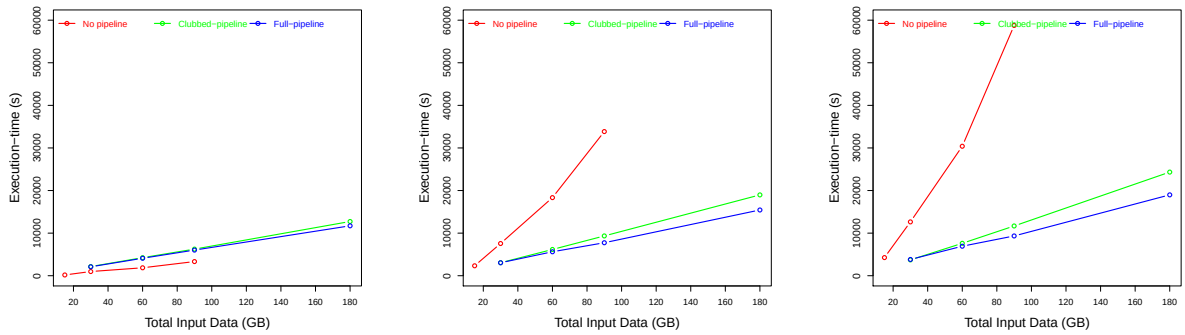


Figure 10: Times to transfer data to HDFS using the two pieplining strategies. Since full pipelineing strategy shares network resources for data transfer and execution stages of the pipelines, it takes longer for transferring chunks of data to HDFS using the full pipelining strategy as compared to the clubbed pipelining.

4.4 Effect of Pipelining

Our initial tests showed that data-intensive Hadoop jobs tend to fail on VMs due to various errors [14, 15]. The errors included checksum errors, missing blocks and disk space. Our pipelining strategy streams data into each stage of the pipeline as required thereby increasing the stability, scalability and performance of data-intensive jobs.

We divided the total data into smaller sizes of approximately 15 GB for a Hadoop cluster with worker nodes of approximately 64 GB of local storage. Each data set is then processed in separate pipelines. Figure 10 shows a comparison for transferring data to HDFS using the two pipelining strategies. Full pipelineing strategy shares network resources for data transfer and execution stages of the pipelines. This results in lower transfer rates using full pipelining as compared to the clubbed pipelining. But Figure 11 shows that the overall performance of the Hadoop benchmarks improve using full pipelining strategy. This is due to higher overlap between data transfer and job execution phases which slows the individual data-transfer phase but the overall performance is improved. The results also depict that pipelining improves both performance and scalability of the data-intensive Hadoop jobs as compared to running the jobs without pipelining. The performance of Hadoop applications on cloud environments improved by a factor of almost 5× by pipelining in comparison to normal execution in most cases. The filter application shows better performance with direct use of iSCSI disks rather than using pipelines. This is because filter reads a lot of data and reduces the data size in the map phase. Hence, there is less data that is getting shuffled in the shuffle/reduce phase of the job and thus, pipelining offers no benefits.



(a) Filter Benchmark (b) Reorder Benchmark (c) Merge Benchmark

Figure 11: Effect of pipelining on Hadoop applications. For read-intensive Hadoop job (a) pipelining improves scalability but the performance is lower as compared to directly using iSCSI disks for small to moderate data-sizes. For write-intensive Hadoop jobs (b) and (c) pipelining performs and scales better than using iSCSI disks due to efficient management of network and storage resources by reducing the number of shuffle/reduce operations for an individual sub-task.

Figure 12 shows the results of pipelining the stages in the ImgComp application in FRIEDA. For the ImgComp application, using the pre-determined data management strategy and pipelining the transfer and execution stages of the application lifecycle improved the performance with increased data size as compared to directly using the iSCSI disks. The figure shows that pipelining continues to follow the linear trend whereas external storage (iSCSI disks) starts performing poorly for larger data sizes. This is due to the network bottleneck between the compute nodes and the external storage. Local disks are not suitable for large data sets due to the limits on size.

5 Discussion

In this paper, we studied the effect of data management strategies for scientific applications running in cloud environments over a number of resource and application characteristics. Our experimental evaluation on four cloud environments running scientific applications in FRIEDA and Hadoop demonstrate that application's

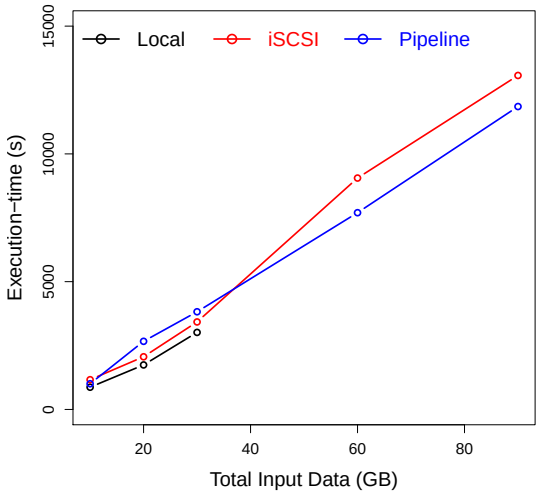


Figure 12: Effect of pipelining on the Image Comparison Application. Local disks do not scale whereas iSCSI performance degrades with increasing data size due to lower disk speed and large volume of data transfer over unprovisioned network. Pipelining shows a linear trend by utilizing the network and disk resources efficiently.

characteristics need to be considered in conjunction with resource characteristics. The summary of our results are:

- It is important to consider combined storage and compute provisioning due to the interplay between I/O and memory for different application patterns.
- The appropriate storage choice combined with filesystem is highly application-dependent and depends on I/O patterns, setup available on site, scale of VMs and data size.
- Using effective data management strategies over provisioned network can significantly improve the performance of data-intensive applications.
- Performance of local disks on VMs over provisioned networks outperforms external storage devices but are often limited in size in cloud environments.
- The behavior of different cloud instances can also vary across various cloud providers and between different sites of a cloud federation.
- If a large data-intensive job can be divided into smaller tasks, then pipelining improves the overall performance of the application.
- A combination of parallel and pipelined processing scales and performs better than just adding more compute nodes which might result in network and disk saturation.

In this section, we discuss various aspects of a) resource provisioning and b) data management. Finally, we summarize what future data and application management frameworks for science applications on clouds need to consider.

5.1 Resource Provisioning

In cloud environments, resources are provisioned based on the need of the application. Since network is a key bottleneck in virtual cloud environments, provisioning network resources is important for obtaining better

performance. Currently, network provisioning is only provided on the ExoGENI testbed. The performance of applications can be improved when the network between the external storage and the VM is provisioned.

Today, there are limited storage provisioning options in the cloud environments. The interplay between storage and network provisioning will need to be considered when handling large data volumes in cloud environments. Cloud systems will need to provide more flexible, real-time, on-demand resource provisioning mechanisms.

5.2 Data Management

We use Hadoop and FRIEDA as the two data management frameworks for executing applications on clouds. Hadoop's data management makes it suitable to process data locally due to Hadoop's locality-aware computation where the the data to be processed is selected based on its locality to a node. But scientific applications running on cloud environments need flexibility in both resource management and data management in terms of file formats, input sources and output sinks. FRIEDA provides this flexibility by separating the control from the execution and also allows executing various applications without any source code modification. Cloud environments require data management frameworks to be agile and intelligent and adapt to various resource conditions and application initiated change such as scaling of resources.

Running data-intensive applications on cloud environments require data to be partitioned and distributed across resources. These applications can transfer large data-sets efficiently over provisioned networks. Different data placement strategies as provided by FRIEDA can utilize these resources based on the type of application to improve the performance.

Scalability, performance and reliability are issues for data-intensive applications in virtual environments. We show that data-intensive Hadoop jobs tend to fail more frequently when run on cloud environments even at modest data sizes [15, 16]. We used a set of pipelining strategies by dividing a job into smaller sub-tasks and splitting a single job into different stages. Pipeline effectively uses provisioned resources, specifically network resources in our case.

Our results also show that for smaller data-sets, performance is poor with pipelining. This can be attributed to pipeline overheads when there isn't sufficient overlap between the data transfer and computation phases.

5.3 Design of Data management frameworks

Our work highlights three important things that are to be considered in future for scientific applications running in cloud environments. First, the performance of applications is affected by a large number of factors including application characteristics. Thus, frameworks such as FRIEDA that allow users to plug-in application- and resource-specific policies are vital. Second, dividing a large data-intensive job into smaller tasks and pipelining the data-transfer and execution phases based on available storage and network bandwidth helps in improving both scalability and performance. It is important to use resources effectively by intelligent provisioning and data management rather than over-provisioning and under-utilizing them. Finally, executing the application with different configurations and associated models are needed to build automated planning and execution frameworks. The ease of use of the FRIEDA framework makes it easy to configure and study the various choices (as in our experimentation).

6 Related Work

Many frameworks, programming models, algorithms and tools have evolved to manage data-intensive applications in cloud environments. However, to the best of our knowledge there is no previous work that studies and manages the interplay of provisioning, partitioning and pipelining for data-intensive applications in cloud environments.

Provisioning. Wang et al. [17] showed that virtualization causes abnormal variations in network performance on cloud platforms. A study on network failures and their impact on various applications is shown in [18]. This shows the importance of provisioning network resources for guaranteeing performance on cloud environments. Previous work has looked at heuristics, algorithms and techniques[19, 20, 21, 22] for resource

provisioning and orchestrating end-to-end applications in cloud environments. Power-aware resource provisioning in cloud environments has been studied for reducing energy consumption of datacenters [23]. Efficient resource provisioning algorithm for scientific workflows in clouds has been proposed in [24]. In this paper, we study the impact of provisioning based on application needs.

Hadoop on Clouds. Hadoop [25] [10] and MPI are used to manage scalable large-scale parallel applications. Fadika et al. [26] evaluates Hadoop performance on HPC platforms for data-intensive science operations. There have been many proposals to improve the existing MapReduce framework for adaptive data placement on heterogeneous Hadoop clusters [27]. Modifications to the Hadoop data management system [28] and in-memory computations [29] have been proposed to improve performance on Cloud environments. However, there is still a lack of generic techniques to distribute data and execute applications. Apache Hadoop YARN [30] is shown to provide better and stable performance than previous versions of Hadoop when running on cloud platforms. Appuswamy et al. [31] evaluate scaling up and scaling out mechanisms for running analytic jobs using Hadoop for achieving better and cost-effective performance on clouds. But all of these techniques require modifications to application code or middleware. Our work focuses on mechanisms to improve performance of an application without modifying it. Our strategy involves an intelligent interplay of provisioned and unprovisioned resources combined with different data management and distribution strategies.

Data Management. Distributed data management has been considered in the context of grid environments including tools for optimized wide area data transfer [32] [33], replica management [34], metadata catalog and data discovery systems [35, 36]. Distributed storage systems like MosaStore [37] provide configurable application-specific storage optimizations for scalable data management. Cost-effective and high performance data management for geographically distributed datacenters has been proposed in [38]. Recent research has also focused distributed data management by facilitating interoperability between clouds and other heterogeneous systems [39]. StorkCloud [40] provides optimized data transfers as a cloud service. In our work, we describe a combination of data management and pipelining strategies for moving the data within and across cloud sites over provisioned resources in order to obtain better performance and lower failure rates for data-intensive applications.

Workflow tools. Scientific workflow management systems [41] manage huge amount and complex processing of data. Deelman et al. [42] highlights several challenges in data management for data-intensive scientific workflows. But, none of the wokflow tools provide flexible mechanisms to partition the data. Moreover, workflow tools rely on existing locations of data and/or move data where there are dependencies. FRIEDA supports only data-parallel tasks. However, FRIEDA provides a flexible interface that can be used by workflow tools to control parts or all of its workflow execution.

7 Conclusions

Infrastructure-as-a-Service (IaaS) cloud model provides a flexible and composable model to manage resources for scientific applications. However, the storage options with different characteristics and the transient nature of the environment result in unique storage and data management challenges. In this paper, we evaluate various data management strategies in the context of FRIEDA and Hadoop to understand storage, file system and provisioning options. We discuss the various factors that play into data management policies on cloud environments. We run three scientific applications using FRIEDA and three data operations using Hadoop on four cloud test beds - FutureGrid, ExoGENI, Grid'5000 and Amazon EC2. Our results showed that a combination of different data management strategies and provisioning techniques resulted in improved performance and scalability.

Acknowledgments

This work was funded by the Office of Science, Office of Advanced Scientific Computing Research (ASCR) of the U.S. Department of Energy under Contract Number DE-AC02-05CH11231. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This paper uses FutureGrid which is based upon work supported in part by

the National Science Foundation under Grant No. 0910812. The ExoGENI work was performed using the National Science Foundation under Grant No. 1032873.

References

[1] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/O performance of virtualized cloud environments," in *The 2nd international workshop on Data intensive computing in the clouds*, ser. DataCloud-SC '11. New York, NY, USA: ACM, 2011, pp. 71–80.

[2] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas, "Seeking supernovae in the clouds: a performance study," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 421–429.

[3] K. Keahey, P. Armstrong, J. Bresnahan, D. LaBissoniere, and P. Riteau, "Infrastructure outsourcing in multi-cloud environment," in *Proceedings of the 2012 workshop on Cloud services, federation, and the 8th open cirrus summit*, ser. FederatedClouds '12, 2012.

[4] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 50:1–50:12.

[5] J. Bresnahan, K. Keahey, D. LaBissoniere, and T. Freeman, "Cumulus: an open source storage cloud for science," in *Proceedings of the 2nd international workshop on Scientific cloud computing*, ser. ScienceCloud '11, 2011, pp. 25–32.

[6] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, "Pesto: online storage performance management in virtualized datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:14.

[7] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 3, pp. 311–336, 2011.

[8] "Apache Hadoop. http://hadoop.apache.org."

[9] D. Ghoshal and L. Ramakrishnan, "FRIEDA: Flexible Robust Intelligent Elastic DAta management in cloud environments," in *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1096–1105.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[11] I. Baldine et al., "ExoGENI: A multi-domain infrastructure-as-a-service testbed," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2012, vol. 44, pp. 97–113.

[12] E. Feller, L. Rilling, and C. Morin, "Snooze: A scalable and autonomic virtual machine management framework for private clouds," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, ser. CCGRID '12, 2012, pp. 482–489.

[13] F. Legger, F. Fassi, A. Pacheco Pages, and A. Stradling, "The ATLAS Distributed Analysis System," CERN, Geneva, Tech. Rep. ATL-SOFT-PROC-2013-024, Oct 2013.

[14] D. Ghoshal and L. Ramakrishnan, "Provisioning, placement and pipelining strategies for data-intensive applications in cloud environments," in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, ser. IC2E '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 325–330. [Online]. Available: http://dx.doi.org/10.1109/IC2E.2014.66

[15] F. Dinu and T. E. Ng, "Understanding the effects and implications of compute node related failures in hadoop," in *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '12. New York, NY, USA: ACM, 2012, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287108

[16] J. Dean, "Experiences with mapreduce, an abstraction for large-scale computation," in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '06. New York, NY, USA: ACM, 2006, pp. 1–1. [Online]. Available: http://doi.acm.org/10.1145/1152154.1152155

[17] G. Wang and T. S. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *The 29th Conference on Information Communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1163–1171.

[18] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *The 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 15:1–15:17.

[19] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic resource provisioning in cloud systems with availability goals," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. New York, NY, USA: ACM, 2013, pp. 1:1–1:10.

[20] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *Services Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 164–177, 2012.

[21] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *The 2011 International Conference on Cloud and Service Computing*, ser. CSC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–10.

[22] I. Baldine et al., "Networked cloud orchestration: A geni perspective," in *GLOBECOM Workshops (GC Wkshps)*, dec. 2010, pp. 573 –578.

[23] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '09. New York, NY, USA: ACM, 2009, pp. 1:1–1:6.

[24] M. Rodriguez Sossa and R. Buyya, "Deadline based resource provisioning and scheduling algorithmfor scientific workflows on clouds," vol. PP, no. 99, 2014, pp. 1–1.

[25] D. Borthakur, *The Hadoop Distributed File System: Architecture and Design.* The Apache Software Foundation, 2007.

[26] Z. Fadika, M. Govindaraju, S. R. Canon, and L. Ramakrishnan, "Evaluating hadoop for data-intensive scientific operations." in *IEEE CLOUD*, R. Chang, Ed. IEEE, 2012, pp. 67–74.

[27] J. et al. Xie, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Atlanta, Georgia, April 2010, pp. 1–9.

[28] K. Krish, A. Khasymski, A. R. Butt, S. Tiwari, and M. Bhandarkar, "Aptstore: Dynamic storage management for hadoop," *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1, pp. 33–41, 2013.

[29] Z. Ma, K. Hong, and L. Gu, "Volume: Enable large-scale in-memory computation on commodity clusters," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, Dec 2013, pp. 56–63.

[30] V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet another resource negotiator," in *The 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.

[31] R. Appuswamy et al., "Scale-up vs scale-out for hadoop: Time to rethink?" in *The 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 20:1–20:13.

[32] W. Allcock et al., "The globus striped gridftp framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005.

[33] A. Shoshani, A. Sim, and J. Gu, "Storage resource managers: Middleware components for grid storage," 2002.

[34] A. Chervenak et al., "The Globus Replica Location Service: Design and Experience," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, no. 9, pp. 1260 –1272, sept. 2009.

[35] A. Rajasekar, R. Moore, C.-Y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, "irods primer: Integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[36] G. Singh et al., "A metadata catalog service for data intensive applications," in *The 2003 ACM/IEEE conference on Supercomputing*, ser. SC '03. New York, NY, USA: ACM, 2003.

[37] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, "The case for a versatile storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 10–14, Mar. 2010.

[38] R. Tudoran, A. Costan, R. Wang, L. Bougé, and G. Antoniu, "Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers," in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Chicago, États-Unis, May 2014.

[39] I. Kelley, "A distributed architecture for intra- and inter-cloud data management," in *Proceedings of the 5th ACM Workshop on Scientific Cloud Computing*, ser. ScienceCloud '14. New York, NY, USA: ACM, 2014, pp. 53–60.

[40] T. Kosar, E. Arslan, B. Ross, and B. Zhang, "StorkCloud: Data transfer scheduling and optimization as a service," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, ser. Science Cloud '13. New York, NY, USA: ACM, 2013, pp. 29–36.

[41] I. J. Taylor, E. Deelman, and D. B. Gannon, *Workflows for e-Science: Scientific Workflows for Grids*. Springer, Dec. 2006.

[42] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows," in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, May 2008, pp. 687–692.