FRIEDA: Flexible Robust Intelligent Elastic Data Management in Cloud Environments

Devarshi Ghoshal
School of Informatics and Computing, Indiana University
Bloomington, IN
Email: {dghoshal}@cs.indiana.edu

Lavanya Ramakrishnan
Lawrence Berkeley National Laboratory
Berkeley, CA
Email: {lramakrishnan}@lbl.gov

***Abstract*—Scientific applications are increasingly using cloud resources for their data analysis workflows. However, 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 and complexity associated with elasticity, failure rates in these environments. The explosion in scientific data coupled with unique characteristics of cloud environments require a more flexible and robust distributed data management solution than the ones currently in existence. This paper describes the design and implementation of FRIEDA - a Flexible Robust Intelligent Elastic Data Management framework. FRIEDA coordinates data in a transient cloud environment taking into account specific application characteristics. Additionally, we describe a range of data management strategies and show the benefit of flexible data management approaches in cloud environments. We study two distinct scientific applications from bioinformatics and light source image analysis to understand the effectiveness of such a framework.**

I. Introduction

Scientific processes are continously generating and analyzing large data sets to derive scientific insights. Cloud environments have been considered suitable for many of the high-throughput data analysis workflows from various scientific domains including bioinformatics, nuclear physics, etc [1] [2] [3]. Cloud computing technologies have largely evolved to process and store large data volumes of web and log data. Many of the design principles used in the context of Internet data do not directly translate for use of clouds for scientific data. The experiences from Magellan [4], [5], a DOE funded project evaluating the use of clouds for scientific workloads, point to significant challenges when managing data in cloud environments.

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 makes 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.

Recent work [6] addresses the challenges associated with provisioning and orchestrating end-to-end resources and provisioning the network between cloud sites. However, application-specific data management is an open challenge in these environments and the cloud data-management additionally needs to be "network topology aware" in federated cloud sites. In recent years, MapReduce [7] and its open-source implementation Hadoop [8] have taken the approach that data management can be transparent to the user and the framework can transparently provide data locality to the tasks at runtime. While this works well for a certain class of applications, it often is less optimal for applications that don't fit the paradigm [9], [10].

In this paper, we describe our design and implementation of FRIEDA (**F**lexible **R**obust **I**ntelligent **E**lastic **DA**ta Managment) framework. FRIEDA provides a two-level execution mechanism that separates the data control with the execution phases. 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. FRIEDA supports two data management schemes a) pre-paritioning of data, and b) real-time, that facilitates various optimizations and trade-offs related to robustness, storage selection, elasticity, performance and cost necessary in cloud environments

Specifically, in this paper we make the following contributions:

- We present the design and implementation of FRIEDA and show that the separation of concerns between the data control and execution enables various cloud data management strategies.
- We describe a range of data management strategies for data parallel applications in cloud environments implemented in FRIEDA.
- We evaluate our data management strategies with two scientific applications and demonstrate the effectiveness of our flexible data management approach.

The rest of this paper is organized as follows. We describe the architecture of FRIEDA in Section II and the various data management strategies in Section III. We present performance

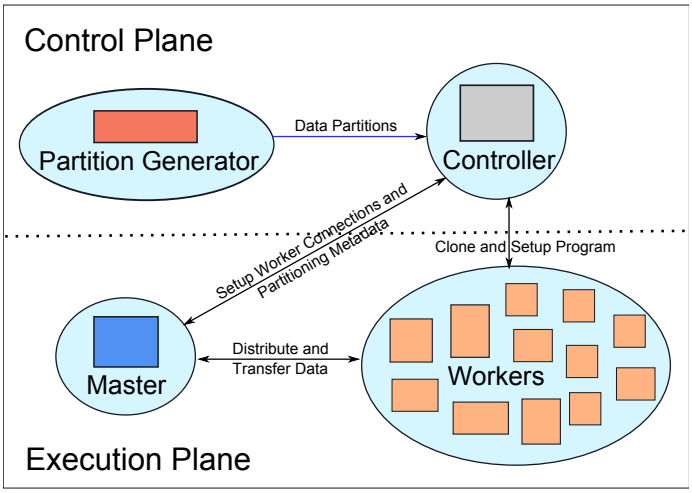
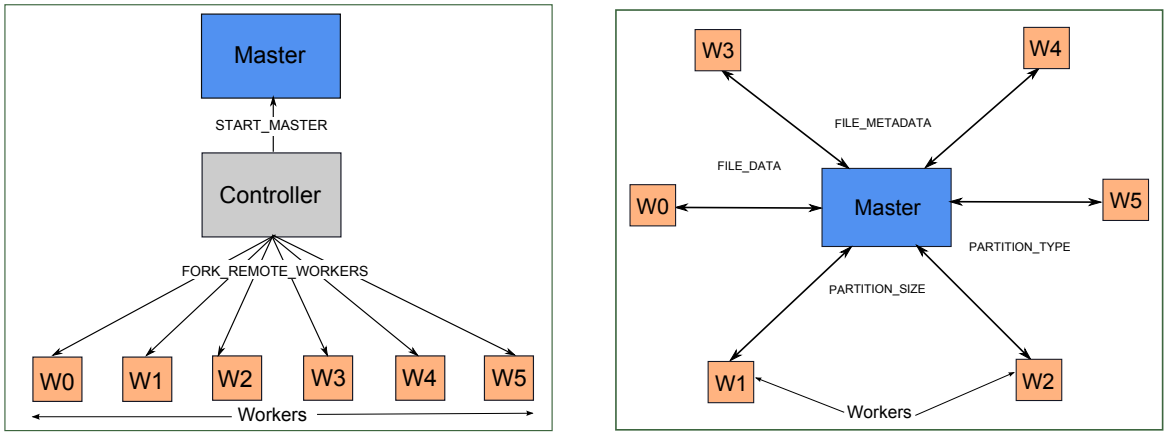


Fig. 1: FRIEDA System Architecture: The figure shows the two plane architecture of FRIEDA. The control plane is responsible for setup of the system components and the execution plane handles the mechanics of data-management in real-time. The partition generator partitions the list of input files to be distributed to the tasks. The controller is responsible for controlling the setup for data transfer and program execution. The master is responsible for distributing and transferring data to respective workers. The workers, upon receiving the required data, do the computation and return the results.



(a) Control Plane

(b) Execution Plane

Fig. 2: FRIEDA Planes: The two figures above show the different views with respect to the two planes in FRIEDA. a) The controller manages the configuration and communication setup. b) The data distribution and program execution occurs at the execution plane.

results in Section IV and discuss considerations for use of FRIEDA in Section V. We discuss related work in Section VI and present our conclusions in Section VII.

II. FRIEDA: System Architecture

Figure 1 shows the system architecture of FRIEDA. FRIEDA provides a data management framework for cloud environments through a two plane architecture: a) control plane, and b) execution plane. The control plane enables us to implement flexible data management strategies for different applications in FRIEDA. The architecture allows us to implement and plug-in different data management strategies in the execution plane while using much of the core framework for data management. The data management and execution in FRIEDA is handled by three components - controller, master and workers.

The controller (and the partition generation algorithm) are in the control plane and are responsible for setting up the environment for data management and program execution. The master and the workers operate in the execution plane and manage the execution of the program.

The separation of the controller from the execution enable us to implement many of the cloud specific policies and decision processes (e.g., storage selection, elasticity) in the control plane. The current implementation of FRIEDA has a master-worker execution implementation. However, FRIEDA is flexible enough to be able to interface with other execution environments as appropriate for applications and/or resource

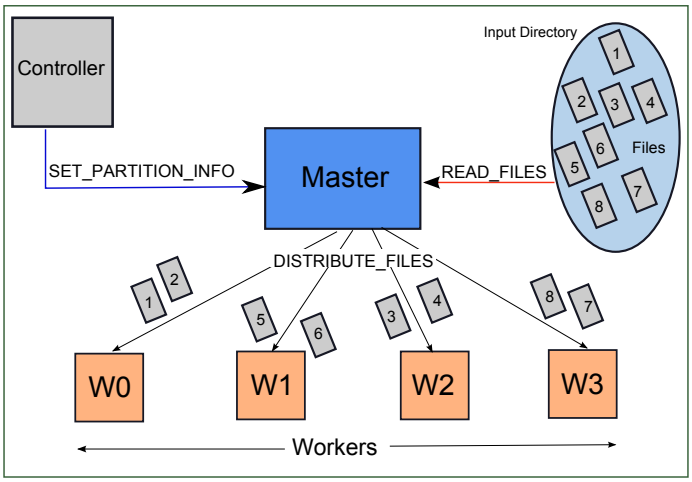


Fig. 3: Data Partitioning in FRIEDA: The diagram shows the operations for distributing files between different workers. The controller communicates the data management information to the master. The master distributes and transfers the files to the respective workers based on the selected data management strategy.

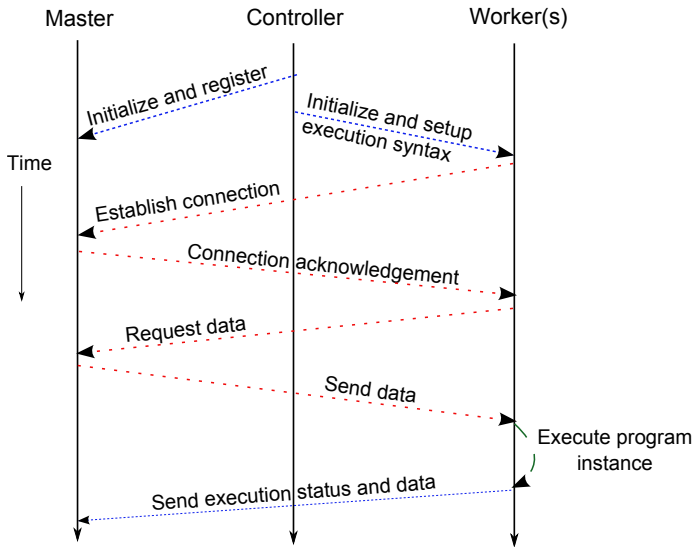


Fig. 4: Component Interaction and Event Sequence: The diagram shows the sequence of events and the interactions between the controller, master and workers in the the FRIEDA system.

environments.

In the next section, we present the two planes and the system components in greater detail.

A. *Control Plane*

The control plane is responsible for setting up the configurations for data transfer and process execution. The configuration setup at the control plane generates the partitioned data-set for the workers. The control plane is also responsible for maintaining the state of the master, and the workers. It sets up the workers too.

A control plane view of the system is shown in Figure 2a. The 'controller' is the primary actor in the control phase and manages the master and the workers. The controller provides the "intelligence" in our system. It communicates with the data partitioning algorithm to set up the data partition and distribution logic between the master and the workers. During the control phase, the controller first spawns the master and informs the master of the partition strategy (e.g., real-time, pre-partition). Next, the controller forks the remote workers on the available nodes. After this point, the master and the workers communicate directly.

B. *Execution Plane*

An execution level view of the system is shown in figure 2b. The 'master' uses the controller's directive and partitions and transfers the data to the target nodes. The master process needs to run close to the source of the input data to prevent additional latency.

Once the set of inputs are obtained by a worker, it executes the program. Every worker continues to receive data and

executes programs until all the inputs are processed. The workers are all symmetrical i.e., all workers perform identical work on different data.

Data transfer is initiated between the source (master) and the targets (workers) in the execution plane. Depending upon the kind of data partitioning mechanism, distributed execution across the workers is managed by the master. FRIEDA looks similar to task farming frameworks but has additional capabilities to control data management.

C. Execution Stages

As mentioned earlier, FRIEDA's controller sets up the master and workers and then hands control over to the master. The master has two stages in its execution lifetime - a) data transfer, and b) process execution.

In the data transfer phase, data is transferred to the target locations (i.e., workers). In our prototype, we use scp to transfer files. Future work will consider other protocols including GridFTP [11].

The master sends the data and asks the workers to execute the tasks. Each worker might execute more than one task i.e., it will ask the master for more work as long as the worker/virtual machine is available and/or there is still data available to be processed.The number of workers running on each nodes depends on the multicore setting in the control phase. If multicore computation is enabled then every node will have as many workers as there are cores. A program is cloned as many times as there are cores on the workers. The number of execution instances of a program, therefore, becomes equal to the total number of worker cores in the system. Currently, FRIEDA supports multicore computation only for homogeneous systems.

Typically, the phases are sequential, i.e., process execution starts only when the transfer of data is completed. For real-time data partitioning strategies, the phases are interleaved.

It is important to note that the 'process execution' phase is similar to a task-farmer and is responsible for executing a sequential program in parallel by distributing the data and parallelizing the program execution. FRIEDA does not modify any program code nor do we provide a separate programming model for writing programs. FRIEDA provides more flexible controls on data partitioning, distribution, and computation while executing in a distributed parallel environment.

D. Communication Protocol

Figure 4 summarizes the sequence of communication between the three system components - controller, master and worker. The controller first starts the master and initializes it with the partition strategy to be used for execution. The controller and master have an open channel that will allow the controller to change the execution configuration at the master at run-time. This communication channel enables us to change the configuration without requiring the master to be restarted. Additionally, dynamic decisions such as elasticity can be relayed to the master through this channel.

The controller then starts the workers and initializes the workers with the execution syntax of the program. For example, if 'app' is the program that needs to be executed and takes arg1 and arg2 as params and inp1 as input, then the execution command is sent to the workers as *app arg1 arg2 $inp1*, where $inp1 is replaced by the location of the file at runtime.

The workers then connect to the master and receive the data they are assigned to process. Once the workers complete the execution instances of the program with all the data they received, both the execution results and the status can be transferred to the master or left behind on the workers as the application might desire. In our evaluation, we consider only the case of local output. Information on any failed worker gets reported to the controller allowing the controller to initiate remediation measures.

E. Data Partitioning

The performance and cost of data movement in the cloud environments makes it important to consider carefully partitioning strategies for the data. The 'partition generator' module at the control level generates file access groupings based on the syntax of the program execution. The partitioning scheme determines the number of input files that will be used for every program instance. The module creates a list of files for each instance of the program. We support three basic schemes (described below) but the design allows other schemes to be easily added.

The three basic pairwise groupings that can be generated using the partition generator are listed below.

- One to all: one file in the input directory is paired with all the rest of the files to be passed as arguments to the program.
- Pairwise adjacent: two adjacent files from the list are paired together and passed as arguments to the program.
- All to all: every file is paired with all other files to be used as arguments to the program.

If a specific partitioning and grouping mechanism is not selected, every instance of the program execution takes one input as a file.

F. Implementation

The current implementation of FRIEDA is in Python using the Python-twisted framework [12]. Twisted is an event driven network engine written in Python. We used it in building our framework since it is well supported and stable. Also, Twisted supports asynchronous communication between the actors that provides greater flexibility across the system components.

Scientific applications have different input specifications and execution parameters. Data partitioning divides the set of files within the directories into groups and sends the information to the controller. This file is used by the master to process the files from the input directories and build the actual runtime execution command for the program to be executed. The actual execution command is built by filling in the variables with appropriate file names as and when the workers receive the data from the master (Figure 3).

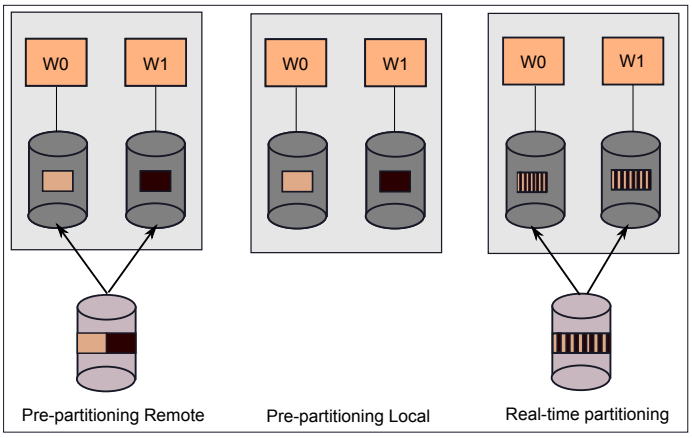


Fig. 5: Data Management Classes in FRIEDA. There are three major classes of how data is partitioned and managed a) data is read from remote disks based on some pre-defined partitioning b) data is local to computation, and, c) real time partitioning and distribution strategy where every worker receives the data as it requests for it.

FRIEDA can execute any data-parallel program, if the program is installed and configured on the worker nodes.

The partitioning of the data is performed based on the configuration parameters sent by the controller. In case of pre-partitioning, the groups of files that will be processed by every worker is determined by the master at the beginning. Based on the partitioning algorithm, the files are transferred to the workers. For real-time mode, the transfer is 'lazy'- the master doesn't transfer a file until a worker asks for it.

The real-time mode inherently handles load balancing and process skew. Worker nodes that are heavily loaded, process less compared to the nodes which are lightly loaded. The controller can set up the workers to create as many instances of the program as there are cores.

III. Data Management Framework

FRIEDA gives users the control over the data partitioning and distributing across the nodes within a virtual cluster. In this section, we describe the strategies currently supported in FRIEDA.

A. Data Partitioning

Cloud offers different storage options with different performance, reliability, scalability and cost trade-offs [13] [14]. Data management strategy needs to account for the storage options that might be available to an applicatin at a given resource provider site.

Every virtual machine has a local disk that provides the fastest I/O. However local disk space is very limited typically in the order of a few gigabytes. Thus, typically local disks are best used for application codes and checkpoint/stage intermediate data. Additionally, various cloud providers provide a way to use block store volumes and/or external storage volumes within the virtual machines. Applications that need to operate on shared data might mount shared file systems. External storage, like iSCSI disks or any other network storage, provide means to handle and store large amounts of data which can be shared across the network as well. For our evaluation, we focus on local and networked disks for comparison.

There are trade-offs between locally placing the data versus dynamically fetching the data from remote sources. It is important to consider eliminating the network bottleneck and making best use of computation resources available. Based on how the resources are configured in a cluster and the execution behavior of the experiment, data can be partitioned in different ways as described below.

No partitioning. The naive approach to data management would be to make the complete dataset available on every compute node. In this mode, we do not partition the data. This model is well suited for applications where every computation relies on a common data set (e.g., a database).

Pre-partitioning. In pre-determined and homogeneous workloads, optimal solutions can be found by pre-partitioning the data before the computation starts. Thus, every node only has the data it needs for its computation thus saving on data movement costs and possible synchronization overheads. This method works best if every computation is more or less identical.

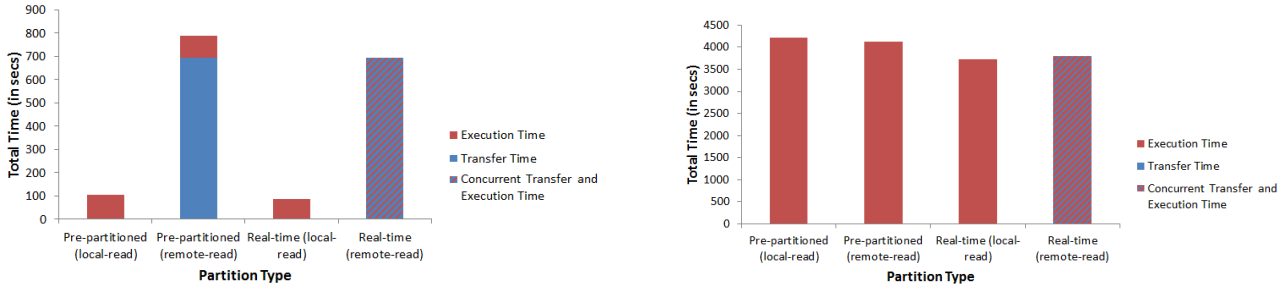
Real-time partitioning. The real-time partitioning of data is designed to suit experiments where each computation is not identical or the compute resources are heterogeneous or in elastic environments where additional resources might become available during the execution. This partitioning strategy inherently load-balances since overloaded nodes get less data to process.

B. Data Management

In this section, we describe the different options in FRIEDA for task and data partitioning. The combination affects the data management strategy and can impact application performance. Figure 5 shows the data management strategies in FRIEDA. FRIEDA supports pre-partitioning of data where data is available locally and/or on a networked storage. Additionally, FRIEDA supports real-time data partitioning and distribution.

TABLE I: Effect of Data Parallelization

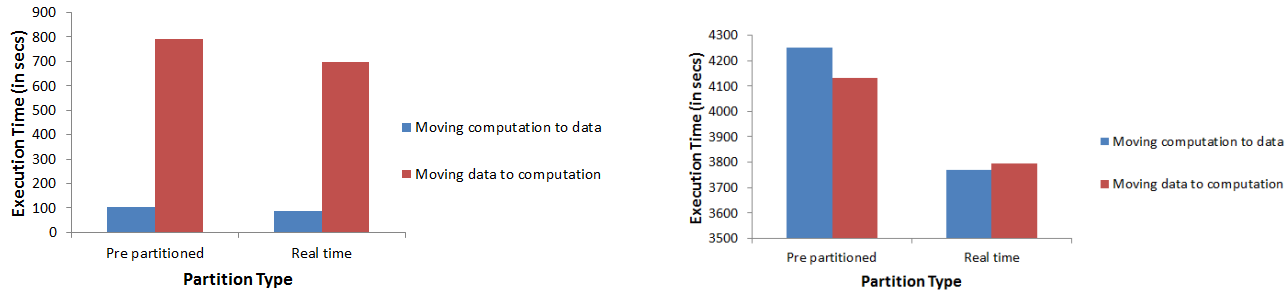
Application	Sequential (s)	Pre-partitioned Data Parallelization (s)	Real-time Data Parallelization (s)
ALS	1258.80	789.39	696.70
BLAST	61200	4131.07	3794.90



(a) ALS Performance

(b) BLAST Performance

Fig. 6: Effect of Different Partitioning



(a) ALS Performance

(b) BLAST Performance

Fig. 7: Effect of Data Movement

Pre-Partitioned Task and Common Data. In this mode, tasks are pre-partitioned but the entire data-set is pre-distributed to all the nodes. Although processing the data locally will be much efficient than over the network, transferring all the data to every node is expensive and not a practical solution for most applications.

Pre-Partitioned Task and Data. In this strategy, partitioning happens before computation begins. Each computation unit processes the data-set assigned to it. This strategy helps applications where processing the data at the source is impossible due to resource constraints. Every partition is transferred to the respective compute nodes before the computation begins. In this case, the total execution cost is the transfer time of the data to the nodes plus the execution time of each computation.

Real-Time Task and Data Partitioning. A real-time task and data paritioning handles data partitioning and distribution dynamically when the computation unit asks for it. It is inherently load-balanced, specifically if every computation task does not consume same amount of resources or if computation happens in a heterogeneous environment. This type of partitioning is also capable of utilizing the network in an efficient manner since data transfer can be overlapped with computation over a shared network.

IV. Performance Results

In this section, we present the performance results and comparisons for different data management strategies in FRIEDA. We study two scientific applications and their data sets with different data access patterns within our framework.

For our experimental and performance analysis, we consider local and remote storage. In addition, we consider two partitioning strategies - pre-partition and real-time.

A. Experiment Setup

Machines. All our test results are performed on the ExoGENI testbed at Duke. We use ORCA [15] [16] and Flukes [17] to launch our virtual machines on the testbed. We use 4 c1.xlarge instances where every VM has 4 QEMU virtual cores and 4 GB memory. The code was compiled using python ver 2.6.6. We provision the network bandwidth between the nodes to be 100 Mbps for the reported set of experiments. We bandwidth limited our applications since this is often true in shared public cloud environments.

Workload.

We consider two separate applications with different data and compute requirements - an image comparison program and BLAST.

BLAST (Basic Local Alignment Search Tool) [18] is a bioinformatics application. BLAST is used to compare primary biological sequences of different proteins against a sequence database. In a BLAST run, a sequence file is compared against a large database under a set of specified parameters.

We use data from a light source beamline. The data consists of a set of images. The simple program we use here basically compares images to see similarity between the images. The image analysis requires two files for every execution.

Data and compute-wise the image comparison deals with large image files for every computation whereas BLAST compares small protein sequence against a large database. We used 1250 images and 7500 protein sequences in our evaluation.

B. Results

Table I shows the effect of data parallelization using both pre-partitioning and real-time partitioning techniques compared to a sequential execution on a single virtual machine. Parallelization over cloud resources provides about $2\times$ speedup for the image analysis and $15\times$ speedup for BLAST.

Figure 6 shows the data transfer and execution times for the overall performance for different data partitioning types for the light source image analysis and BLAST. Figure 6a shows the effect of different schemes for the image analysis. As expected, the local reads are faster since they assume the data is already available on the virtual machines. Data could be packaged in the virtual machine image thus being available when the VM boots up. However, this has significant trade-offs since the local disk is fairly limited on these resources. Additionally, this would work only for static data that does not change often since data change would either mean the image has to be recreated or transferred again.

In case of the pre-partitioned remote data, the transfer and execution happens sequentially and thus displays the worst performance. In the real-time remote data case, the overlap of the stages improves performance. It is obvious that the data transfer strategy for ALS can significantly affect the overall performance of the application.

Figure 6b shows the execution time for BLAST. However, BLAST shows different trends than ALS. In BLAST, comparing n sequences to a database containing m sequences require approx. $(n * m)$ comparisons. This computationally intensive operation makes the transfer time significantly less important compared to the execution time for every comparison. Additionally, the data movement costs are dominated by the backend database that needs to be available on every node. However, BLAST benefits from the inherent load balancing in FRIEDA in the real-time strategy since every task might have different computation cost than the other based on the match of the search.

Our results show that real time partitioning performs better than its pre-partitioned counterpart but the benefit depends on the characteristics of a specific application.

Figure 7 compares the approaches of moving data to the computation and moving computation to the data. An important question for any application is whether to move the data closer to the computation or vice-versa. And, as described earlier this depends on both the data access pattern and the computational requirements of the application. The results in figure 7a show better performance for image analysis application when the computation moves closer to the data since the cost of moving the data is higher than the cost of computing over the data. In contrast, Figure 7b shows that BLAST is almost insensitive to the placement of computation or data.

Thus, for applications such as BLAST where computation is predominant over data size, data or computation movement is not as significant as the load balancing and partitioning over the data set. However, for applications such as image analysis where the size of the data matters, data movement and data placement plays a pivotal role in determining the performance of the application. *The idea behind developing a framework like FRIEDA is to provide flexibility in data management and data placement.*

V. Discussion

The advantage of FRIEDA comes from its ability to handle both data and computation in heterogeneous environments. In this section, we discuss various considerations in the design and use of FRIEDA for managing data-parallel applications in cloud environments.

A. FRIEDA Properties

FRIEDA's data management framework is unique from other distributed data management frameworks due to four characteristics (as the name suggests) to address specific challenges in cloud environments.

Flexible. Scientific applications have different data characteristics in terms of file formats, input sources and output sinks. Any data management framework needs to be flexible to handle these diverse requirements in conjunction with specific resource characteristics. FRIEDA achieves this flexibility by separating the control from the execution and allowing a plug-and-play model. In the performance evaluation, the two applications we considered have different set of input files and parameters. The ability to handle both these applications demonstrates that FRIEDA can be used to execute any application without any modification of the original source code.

Robust. Cloud environments often rely on commodity hardware and have been shown to have availability fluctuations [14]. A data management framework needs to be robust to handle both virtual machine failures and network performance variations. Scientific applications running on HPC systems and or dedicated clusters have typically not handled these classes of failures in their design. Gracefully handling these failures and either correcting or reporting it back to the user is critical in these environments. FRIEDA controller keeps track of all the errors from the workers. In real-time mode FRIEDA is capable of automatically isolating the failed workers from doing further computation. However, it still does not implement a general fault recovery mechanism, i.e., it

is not capable of automatically restarting the failed task. In future work, we plan to develop additional fault tolerance and recovery mechanisms in FRIEDA. Additionally FRIEDA, like many other management frameworks has a single point of failure if the controller or master fails. Future work, will address the monitoring and recovery of the maser through the controller-master communication channel.

Intelligent. There is a fine balance between making dynamic decisions based on the resource status and allowing users to control the process. 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. FRIEDA's controller is the "intelligence" in the system. While in our initial implementation we have limited policies, the design allows various decision processes to be added seamlessly. In real-time mode, FRIEDA does automatic load balancing and data distribution. Future work will investigate the ability to select the best data management strategy based on past executions of an application.

Elastic. On-demand elasticity is considered to be one of the strengths of cloud environments. This has implications in terms of data management since additional available resources could require a different data management strategy. Additionally, if resources are going to disappear, snapshots of the data need to be captured. The controller in FRIEDA handles the addition and removal of workers. Addition of any new worker goes through the controller which establishes the connection between the master and the workers.

B. FRIEDA Extensions

The architecture of FRIEDA is extensible. FRIEDA allows users to plug-in other partitioning and data managent algorithms and protocols.

Data Management Strategy. Defining a new data management strategy is equivalent to defining a new protocol for data transfer and computation. Since, data transfer protocols are decoupled with execution protocols, various combinations can be used to effectively use the network, I/O and compute resources of the system.

Partition Generation. The 'partition generator' utility is a separate component in FRIEDA. Any new grouping of input files can therefore be generated by extending it, irrespective of the core FRIEDA system.

C. Integration with other system components

Our current evalution of FRIEDA is focused on virtual machines. However, there is nothing in the design of FRIEDA that precludes it from being used in other environments. FRIEDA's design thus makes it possible to use in other container transient environments or in federated cloud environments or in hybrid environments that have a mix of resources from HPC and cloud environments.

FRIEDA's control plane can be used with other execution mechanisms such as batch queue systems, Hadoop, etc. However, the flexibility of the algorithms supported will rely on the execution system characteristics. Similarly, FRIEDA is capable of working with a myraid of storage options including local disks, iSCSI disks, EBS volumes etc. FRIEDA only supports data-parallel tasks. However, it is possible for a higher-level workflow engine to interact with FRIEDA to control parts or all of its workflow execution.

VI. Related Work

FRIEDA provides application the ability to control data partitioning and distributing in cloud environments. In this section, we summarize some related work.

Data management in programming models. The requirement for data intensive applications to distribute and process data in parallel has given rise to many frameworks and programming models. Hadoop is an extremely popular programming model for distributed data processing [19], [7]. MPI [20] is another standard for writing portable and scalable large-scale parallel applications. Both these mechanisms need programs to be written using their specific program semantics. Hadoop provides minimal control over data distribution among the nodes. MPI, on the other hand, provides full control over data management and communication across different compute nodes. But, none of these techniques provide a generic solution to distribute data and execute applications without instrumentation or rewriting them. Hadoop streaming provides solutions to execute any existing script or program. The FRIEDA framework is a generic framework to execute any script or program by combining the concept of data parallelization, data grouping and, data distribution strategies without any instrumentation at the application-level.

There have been many proposals to improve the existing MapReduce framework for adaptive data placement on heterogeneous Hadoop clusters [21]. Dryad [22] is a general-purpose distributed execution engine for coarse-grain data-parallel applications.

Distributed Data management. Various aspects of distributed data management have been considered in the context of distributed environments/grid environments including tools for optimized wide area data transfer [11] [23], replica management [24], metadata catalog and data discovery systems [25] [26]. However, the characteristics of cloud environment (e.g., elasticity, transient nature of storage) present unique challenges which necessitates the need to revisit the data management framework design.

Data parallelization and task farming approaches have been shown to significantly reduce execution times for embarrassingly parallel applications like BLAST [27], [28].

Workflow tools. Scientific workflow management systems [29] manage huge amount and complex processing of data. Deelman et. al [30] 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 support data management for other workflow patterns.

VII. Conclusions & Future Work

In this paper, we have described FRIEDA, a framework for managing data in distributed transient environments such as clouds. FRIEDA provides different mechanisms for managing transfer and storage of data. The flexible two-plane architecture of FRIEDA provides the ability to plug-in different policies and methodologies for managing data. Thus, an existing application can be plugged into FRIEDA and appropriate data management strategy can be selected, requiring no change to the application. Our evaluation shows that any data-parallel application can manage the data effectively and efficiently over cloud resources using FRIEDA using an appropriate data management strategy.

Our future work will focus on introducing additional fault tolerant mechanisms to handle and recover from failures at run-time. Currently, elasticity requires some interaction with the controller; our goal is to make addition and removal of workers from the system transparent to the user. Finally, we envision FRIEDA to be adaptive to not only different real-time situations but also have adaptation strategies that uses past historical information.

VIII. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1032873. This work was also supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We would like to thank Ilia Baldine, Jeff Chase, Anirban Mandal, Paul Ruth, Victor J. Orlikowski for help with the ExoGENI testbed.

References

[1] S. Canon, S. Cholia, J. Shalf, K. Jackson, L. Ramakrishnan, and V. Markowitz, "A performance comparison of massively parallel sequence matching computations on cloud computing platforms and hpc clusters using hadoop," in *Using Clouds for Parallel Computations in Systems Biology Workshop, Held at SC09*, 2009.

[2] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413421

[3] K. Keahey, T. Freeman, J. Lauret, and D. Olson, "Virtual workspaces for scientific applications," *J. Phys.: Conf. Ser.*, vol. 78, no. 1, pp. 012 038+, 2007. [Online]. Available: http://dx.doi.org/10.1088/1742-6596/78/1/012038

[4] L. Ramakrishnan, P. T. Zbiegel, S. Campbell, R. Bradshaw, R. S. Canon, S. Coghlan, I. Sakrejda, N. Desai, T. Declerck, and A. Liu, "Magellan: Experiences from a science cloud," ser. ScienceCloud, 2011, pp. 49–58. [Online]. Available: http://doi.acm.org/10.1145/1996109.1996119

[5] "Magellan final report," http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf.

[6] I. Baldine, Y. Xin, A. Mandal, C. Renci, U. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin, "Networked cloud orchestration: A geni perspective," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, dec. 2010, pp. 573 –578.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

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

[9] Y. Gu and R. L. Grossman, "Lessons learned from a year's worth of benchmarks of large data clouds," in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, ser. MTAGS '09. New York, NY, USA: ACM, 2009, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/1646468.1646471

[10] E. Dede, M. Govindaraju, D. Gunter, and L. Ramakrishnan, "Riding the elephant: Managing ensembles with hadoop," in *4th Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*, 2011.

[11] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "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, pp. 54–. [Online]. Available: http://dx.doi.org/10.1109/SC.2005.72

[12] (2012) The twisted website. [Online]. Available: http://twistedmatrix.com/trac/

[13] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/o performance of virtualized cloud environments," in *Proceedings of the second international workshop on Data intensive computing in the clouds*, ser. DataCloud-SC '11. New York, NY, USA: ACM, 2011, pp. 71–80. [Online]. Available: http://doi.acm.org/10.1145/2087522.2087535

[14] 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. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851538

[15] "ORCA. https://geni-orca.renci.org/trac/."

[16] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, and A. Yumerefendi, "Beyond virtual data centers: Toward an open resource control architecture," in *International Conference on the Virtual Computing Initiative*, ser. ICVCI 2007, 2007.

[17] "ORCA Flukes. https://geni-orca.renci.org/trac/wiki/flukes."

[18] BLAST. [Online]. Available: http://blast.ncbi.nlm.nih.gov/

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

[20] Open mpi. [Online]. Available: http://www.open-mpi.org/

[21] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "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.

[22] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys '07. New York, NY, USA: ACM, 2007, pp. 59–72. [Online]. Available: http://doi.acm.org/10.1145/1272996.1273005

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

[24] A. Chervenak, R. Schuler, M. Ripeanu, M. Ali Amer, S. Bharathi, I. Foster, A. Iamnitchi, and C. Kesselman, "The globus replica location service: Design and experience," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, no. 9, pp. 1260 –1272, sept. 2009.

[25] 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. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00233ED1V01Y200912ICR012

[26] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman, "A metadata catalog service for data intensive applications," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, ser. SC '03. New York, NY, USA: ACM, 2003, pp. 33–. [Online]. Available: http://doi.acm.org/10.1145/1048935.1050184

[27] A. Krishnan, "Gridblast: a globus-based high-throughput implementation of blast in a grid computing framework," *Concurrency Computat.: Pract. Exper.*, vol. 43, no. 2, p. 16071623, Apr. 2005.

[28] R. D. Bjornson, A. H. Sherman, S. B. Weston, N. Willard, and J. Wing, "Turboblast(r): A parallel implementation of blast built on the turbohub," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, ser. IPDPS '02. Washington, DC,

USA: IEEE Computer Society, 2002, pp. 325–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645610.661393

[29] I. J. Taylor, E. Deelman, and D. B. Gannon, *Workflows for e-Science: Scientific Workflows for Grids*. Springer, Dec. 2006. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/1846285194

[30] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows."