Scalable State Management for Scientific Applications in the Cloud

Tonglin Li[1], Ioan Raicu[1,2], Lavanya Ramakrishnan[3]
tli13@hawk.iit.edu, iraicu@cs.iit.edu, lramakrishnan@lbl.gov
[1]Computer Science Department, Illinois Institute of Technology, Chicago, IL, USA
[2]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
[3]Advanced Computing for Science, Lawrence Berkeley National Laboratory, Berkeley, CA, USA

ABSTRACT
The data generated by scientific simulations and experimental facilities is beginning to revolutionize the infrastructure support needed by these applications. The on-demand aspect and flexibility of cloud computing environments makes it an attractive platform for data-intensive scientific applications. However, cloud computing poses unique challenges for these applications. For example, cloud computing environments are heterogeneous, dynamic and non-persistent which can make reproducibility a challenge. The volume, velocity, variety, veracity and value of data combined with the characteristics of cloud environment make it important to track the state of execution data and application's entire lifetime information to understand and ensure reproducibility. This paper proposes and implements a state management system (FRIEDA-State) for high-throughput and data-intensive scientific applications running in cloud environments. Our design addresses the challenges of state management in cloud environments and offers various configurations. Our implementation is built on top of FRIEDA (Flexible Robust Intelligent Elastic Data Management), a data management and execution framework for cloud environments. Our experiment results on two cloud test beds (FutureGrid and Amazon) show that the proposed solution has a minimal overhead (1.2ms/operation at a scale of 64 virtual machines) and is suitable for state management in cloud environments.

Categories and Subject Descriptors
D.0 [Software]: General; H.4 [Information Systems Applications]: Miscellaneous; H.2 [Database Management]: Miscellaneous.

General Terms
Management, Measurement, Performance, Design, Experimentation

Keywords
Data management, state management, provenance, cloud computing, scientific computing

I. INTRODUCTION

Data analysis is central to next-generation scientific discoveries. Cloud is as an emerging platform and increasingly attractive to scientists due to its flexibility and convenience. But cloud environments are typically transient. Virtual machine instances are terminated after applications complete execution. Users cannot leave data and/or revisit the resource setup to diagnose discrepancies. In the cloud environment, users have the responsibility to capture everything before the virtual machines are shutdown.

Big data scientific applications need to track every step of the scientific process, data access and environment for lineage, reconstruction, validity and reproducibility purposes. It is important to know the environment in which the applications run (e.g., floating point operations could give different results on different machines). Users might also wish to "rerun" some (e.g., only what failed) or all of the tasks.

Provenance tools have tracked workflow and data lineage at various levels (e.g., operating system [21], file systems[2], databases[3], cloud storage [4] and workflow tools ([5][6][7][8][9][10][11]). Many monitoring tools ([12][13][14]) have been developed to monitor real-time system changes. These systems provide methods to collect, aggregate, and query monitoring data. However, this data is often insufficient for reproduction since they do not capture human knowledge. Furthermore, state management in cloud environments needs to tackle additional challenges due to its characteristics. First, the transient nature of the environments makes it important to capture metadata and state at various levels. Second, the performance and reliability characteristics of virtual machines is important to consider in the design of the collection system. Finally, different clock drifting rates on physical machines make it hard to have a unified time view for the end-user to rebuild meaningful semantics.

In this paper, we propose FRIEDA-State, a state management system for cloud environments. We use the term state to represent the metadata from both execution framework and applications. FRIEDA-State addresses the transient nature, performance concerns and clock drifting issue in its design. FRIEDA-State is currently implemented atop of FRIEDA [15], a data management and execution framework for cloud environments, which supports a high-throughput and data-intensive scientific applications,. We present a key-value based collection system to manage state in dynamic transient environments. We design and implement a vector clock based event-ordering mechanism to address the clock drifting issue,

FRIEDA-State collects static and dynamic state data. Static state data is the information that doesn't change

when the system is running (e.g., CPU/Memory info, environment variables and software stack information). Dynamic state data, on the other hand, changes during application running, such as the information on details of the input file that is processed, the time taken for a machine to finish execution or failure of jobs.

Specifically, the contributions of our work are:

- *Design and implementation of FRIEDA-State, a state management system for scientific applications running in cloud environments, with lightweight capturing, efficient storage and vector clock-based event ordering*
- *Evaluation on multiple platforms (FutureGrid and Amazon EC2) at scales of up to 64 VMs; results show good efficiency with minimal overhead (1.2ms/operation at 64-node scales)*

II. Background

In this section, we describe the background required to understand the design considerations in FRIEDA-State.

A. Use cases

Scientific applications need to track their scientific process for a number of reasons including a) real-time monitoring b) tracking data lineage c) validation of results d) reconstruction or repeating some or all of the experiments and, e) reproducibility of research results. Users might want to track their configuration and environment settings and repeat some or all of the experiment or validate a certain result (e.g., floating point). The state information might also be used for post-execution analysis. For example, the users might like to query job statistics and understand why some jobs took longer than others. Users might want to rerun the same experiment and/or run the same experiment with slightly different parameters.

B. Challenges

Next, we discuss the challenges on *state collection*, *storage*, and *event synchronization*.

State collection and storage: State information is generated on each of VMs and multiple VMs are part of an application execution. High capture latency may degrade the application performance. Thus, scalable collection of data is important in the design of FRIEDA-State. Information aggregation and appropriate storage mechanisms are also important and different solutions might have different trade-offs. Centralized storage system (e.g., databases) could result in concurrent read/write bottlenecks and be the source of single-point failures. Distributed solutions often suffer from high operation latency and often require extra dedicated hardware.

Event Synchronization: Cloud environments are dynamic. Virtual machines may not run on the same physical machines. This implies that the physical time clocks may not run at exactly the same speed because of the slight difference between crystal oscillators on different machines thus result in drifting [16]. With different drifting rates, at the end of a long run, the base-time gap between each virtual machine could be big. The drifting issue is serious in large-scale distributed systems and is even more serious due to transient nature of clouds. Synchronized bootstrap time clocks may not be guaranteed in distributed cloud environments. It is important that the events/states captured on the machines is unified for the end-user to build meaningful semantics.

C. FRIEDA framework

Our state management system is built on top of FRIEDA[15]. FRIEDA is a Flexible Robust Intelligent Elastic Data Management framework. FRIEDA manages the lifecycle of data that includes storage planning and provisioning, data placement and application execution of scientific applications in cloud environments.

FRIEDA enables users to plug-in flexible data management strategies for different application patterns by separating data control from execution. FRIEDA supports a Master-Worker execution model. There are three major components in FRIEDA architecture, namely controller, master and workers. The *controller* takes charge of environment setup and configurations for data management and application execution. The *master* is responsible for managing application execution and data distribution. The *workers* accept data and computation jobs from the master and execute them locally. After all workers finish their jobs, the framework will collect output data from all nodes. State management system collects information from the resource provisioning and execution phases.

III. System Design and Implementation

Figure 1 shows the system architecture of the state management system (FRIEDA-State). FRIEDA-State has a collection component in each of the main FRIEDA actors – the controller, master and worker. FRIEDA-State works in two phases: capturing and storage. It allows multiple storage solutions to be plugged into the framework to meet different usage needs. The runtime state capture component collects two types of state: static and dynamic. The static states are collected mainly from a configuration YAML file, which is used in FRIEDA to configure the virtual machines (e.g. it defines the roles and the setups of the master and workers). The YAML file is populated in the state management system from the controller node once the experiment starts. The remaining static information (e.g. system information) is collected from the worker virtual machines directly. Dynamic states are captured from the FRIEDA framework through built-in functions. Once captured, states are encapsulated in key-value pairs and pushed to one of three storage solutions that is selected by the user. FRIEDA-State currently supports raw files, Cassandra or DynamoDB (on Amazon Web Services).

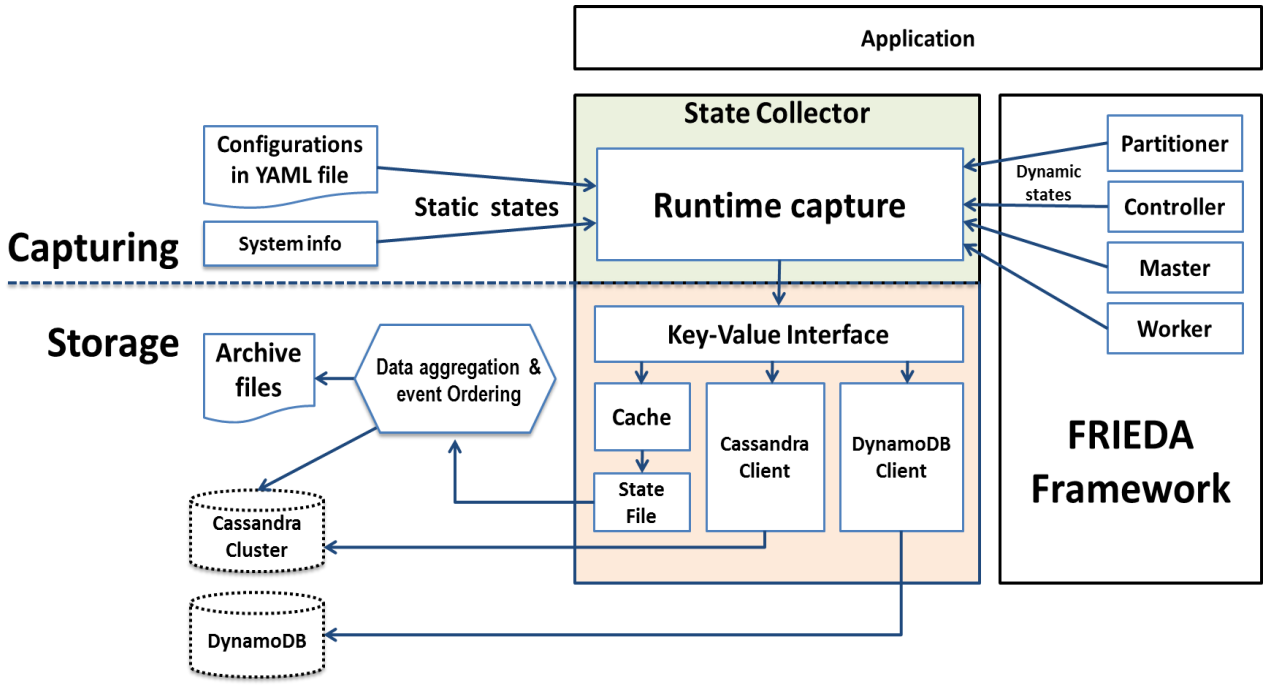

Figure 1. FRIEDA-State system architecture. Capturing is the first layer. State collector captures static states from two source, configuration YAML file and system information. Dynamic states are captured from FRIEDA-State functions, which are called in FRIEDA framework. Captured states are encapsulated into the form of key-value pairs and pushed to one of three storage solutions as selected by the user.

A. *State Description*

Each state in our system has the following fields. The *state name* is used as the key and the rest are used as values. Examples are provided in Table 1 and Table 2.

State name: This is used to represent the type of event.

State information: This field captures the state content.

Role: This field captures the source of the event or the role of the host (i.e., master or worker),

Hostname/IP: This captures the identity of the host where the state was collected.

Logical timestamp: We set a field for logical time for ordering the events captured from distributed nodes. The logical timestamps is used as a part of vector clock. (More details in Section III.F)

Local timestamp: The local timestamp is also captured that indicates the time when the event is captured on a local host and be used to order events within a virtual machine.

B. *Static state capture*

Static state represents the data that will not change during application execution. In FRIEDA, most of the static states are covered in a configuration YAML file. The YAML file includes platform name, image ID, instance type, authentication information, application details. The YAML file allows users to setup environment, software installation and the application running details. The YAML file is loaded into memory and stored as structured data items and then dumped into a data store or state file as a record. Other static states, such as hardware info (CPUINFO/MEMINFO), software stack information and so on are captured when the virtual machines are launched. Table 1 shows several examples of the static states captured on a FRIEDA controller node.

Table 1. Example of captured static states. The item "meta" stores serialized configuration the YAML file. The cpuinfo and ENV are collected from the virtual machines directly.

State name	State information	Role	Hostname/IP	Local timestamp
meta	{"actions": {"worker_data_directories": {"params": {"user": "root", "data_dirs": ["/data", "/data/output"] }, "template": "/N/u/pmantha//FRIEDA//resources/templates/data.pp.tpl"}, "master_data_directories": {" ….	controller	server-9c735efe	2013-10-14 05:50:06.966210
cpuinfo	'processor\t: 0\nvendor_id\t: GenuineIntel\ncpu family\t: 6\nmodel\t\t: 2\nmodel name\t: Intel Core 2 Duo P9xxx …	controller	server-9c735efe	2013-10-14 05:50:07.002920
ENV	OS_INSTANCE_TYPE=m1.small\nMASTER_PRIVATE_IP=10.35.23.19\nINPUT_DATA_DIR=/data/ input\nWORKER_IDS=i-000009a2 i-000009a3\nMAIL=/var/mail/root\nSSH_CLIENT=192.168.11.1	controller	server-9c735efe	2013-10-14 05:50:07.028955

Table 2. Example of captured dynamic states. The item Master logical clock presents the reference vector clock value sent from master node. Note that the three states have the same master clock value, because they all happened after a master message sent to the worker and before another one received.

State name	State information	Role	Hostname/IP	Local logical clock	Master logical clock	Local timestamp
connectionMade	to_10.35.23.19	worker	server-5ef1ae8e	1376459408044136	1376459408108778	2013-10-14 05:50:08.044155
Processing_start	__FILE_METADATA__ 20111110_112213_TI1_AFTER_478.tif	worker	server-5ef1ae8e	1376459408046341	1376459408108778	2013-10-14 05:50:08.046360
Processing_end	__FILE_METADATA__ 20111110_112213_TI1_AFTER_478.tif	worker	server-5ef1ae8e	1376459410394276	1376459408108778	2013-10-14 05:50:10.394558

C. Dynamic state capture

Dynamic state information represents the data that changes during the run-time of applications. For example, the identification of input files processed by a worker and the duration of the execution are captured by FRIEDA-State. We capture two types of information a) communication events and b) application execution details. Table 2 shows examples of the dynamic states captured on a FRIEDA worker node.

All communication events, such as connection made (and to which machine), data received (and from where), etc., are captured. These events not only describe the communication itself, but also carry vector clock information for later event reordering (section III.F). Application execution and data flow details include the commands executed on each worker, I/O operations and application execution time etc. This can be used to track the application execution and to analyze run-time problems.

D. State Storage

The essence of state management is to capture data in distributed environment and store for future queries. For designing such a scalable system with low latency, the major concern is storage architecture. The state operation latency must be very low to prevent degradation of the application performance. Scalability is also important since the storage system could be a bottleneck when serving many clients for writing and/or query. FRIEDA-State currently supports three storage options: files, Cassandra data store and DynamoDB (on Amazon). This allows users and applications to select the right storage while accounting for the tradeoffs for their needs.

Files: This mode uses files for capturing state. Captured states are first written to files, which are later aggregated from all machines at the end of execution. Files as a storage mechanism provides some advantages over key-value stores and databases. First, simple memory-file operation is significantly faster than single node key-value stores, due to the fact that memory-file operation executes sequential writes while key-value stores execute writes randomly (hash table or B-Tree). Second, file-based mechanisms do not require any additional services and hence does not add any overheads on the nodes. Third, merging files are simple and fast since the files are already naturally ordered due to sequential writes on each node. Therefore sorting the state files has a linear time complexity. Files are not good to query on, but they are easy to manipulate and archive.

We capture states from all FRIEDA components on VMs and store them in an in-memory cache before being flushed to disks. The cache size is customizable to address the tradeoffs between robustness and performance. If application fails, the states can still be found on those VMs since they are flushed to disk. For even better durability, it can write to a distributed file system or a block store that can survive beyond the life of the virtual machine, based on the configuration. Finally all states files are copied to a target machine for merging. If application fails, states are still saved within the FRIEDA-State framework. But if VMs or FRIEDA fails during execution, users might lose unsaved states.

Cassandra: We include Cassandra as one of the storage solutions [17], as it provides rich features for managing semi-structured data. It is easy to plug-in other NoSQL databases in FRIEDA-State. Users control the number of Cassandra instances according to their performance and capacity needs. Cassandra could share the virtual machines with the application or run on a separate cluster. Key-value stores are known to work well when deployed on dedicated machines. Practically, users can use a private cluster to host the Cassandra cluster. They can also setup a dedicated virtual cluster on cloud to serve the requests. In this case, users will need to periodically move state data to a more permanent storage.

DynamoDB: The third storage solution is based on DynamoDB, a NoSQL database available on Amazon Web Services. With this type of cloud databases services, users don't have to deploy a software stack to run and configure those data stores, but have to pay extra money for the service.

E. Storage architectures

FRIEDA-State supports multiple storage architectures: centralized, distributed and local storage. This sub-section will describe each of these architectures in detail.

Centralized Storage Solution: In our current implementation, centralized solution is based on a single node key-value store or database. When all nodes in the system generate states, collecting and storing can result in a storage bottleneck. Depending on the application type, the rate of state generation can be different. If the data generated is minimal and at a low rate, centralized storage solution will work perfectly in practice. An important advantage of centralized solution is that all events can be naturally ordered as they arrive at the storage server and assigned a timestamp based on server's local time. The solution naturally provides persistence of the state data beyond the lifetime of the virtual machines.

Distributed storage solution: Similar to centralized solution, FRIEDA-State support distributed storage solution through NoSQL databases (e.g., Cassandra). High write concurrency is a big challenge for all types of storage systems. Distributed storage solutions, such as distributed databases, key-value stores can serve large amounts of write requests and spread them to many nodes to achieve scalable performance and load balancing[18][19]. In this type of solution, a group of dedicated data storage servers will be started prior to application execution. States generated on any node in the system will be written remotely to the data store. The latency of this operation depends on the data store solution and could be up to a few milliseconds [18]. To deploy data stores on all the nodes that will generate states will not help much on performance, because running the data stores consumes extra CPU and memory resources, and messages still need to be sent between all VMs over the network.

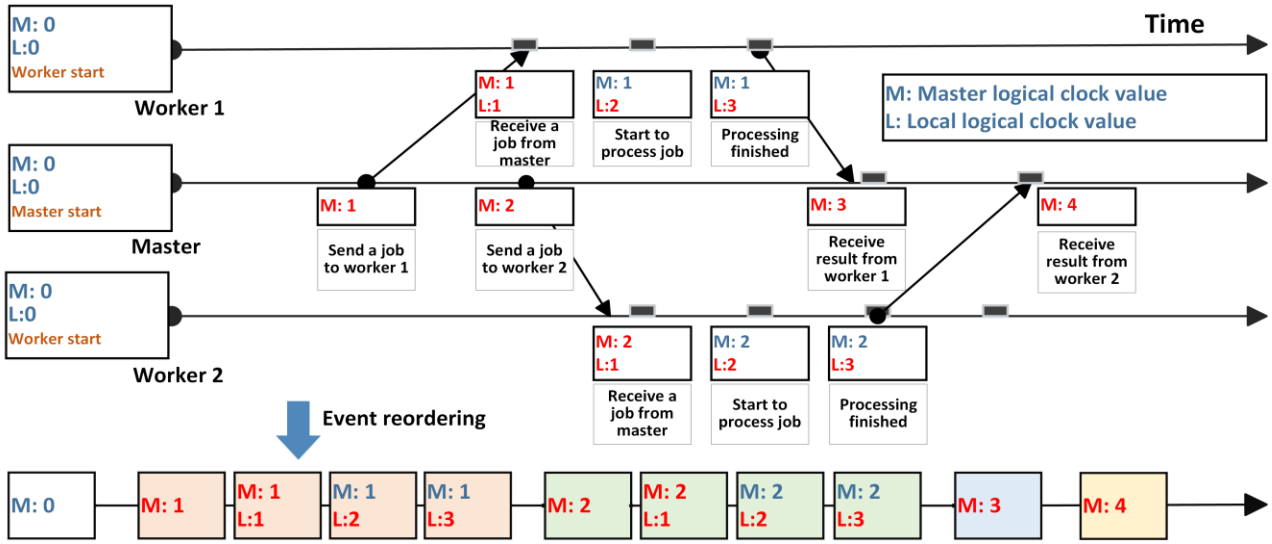

Figure 2. Event reordering example. Three machines have their local clock and maintain a vector clock. Each event will increase the local clock value; each received message will update other's clock value in their local vector. The master's clock is used to maintain the time order when reordering the events. Sorting by master clock M value, all the events are divided into 5 groups. Just sort the groups will order all the event

Local storage solution: We implemented this solution based on traditional files. Local storage eliminates network latency (the major part of operation latency).. Considering the data collected is likely to be queried after an application run, offline storage solutions are reasonable. Writing to local disk/memory is extremely fast compared to remote access and no extra resources are consumed by data store programs on VMs. The hard part of using local storage is aggregating data from many VMs and to merge into a form that offers a single query interface. File-based solution is not perfect though. If users want to query the states during the system running, extracting the desired records is complicated and slow.

F. Event reordering

Distributed event ordering is an important topic in large-scale distributed system design. The goal is to keep a global logical order of events based on timestamps. In FRIEDA-State, we use modified vector clock[20] to maintain time order. FRIEDA-State uses 1-to-n communication pattern since communication only happens between master and workers. We use the master node clock as major clock and all workers logically synchronize to it using vector clock. By comparing attached master clock value in communication messages, we can tell which event happened earlier. If the master clock reads are same, then these events occurred in the same machine. It is trivial to order events within one machine by sorting the local timestamps. When using file-based storage solutions, events are sequentially written to files and thus are naturally ordered. Each state record has two fields for vector clock: one for local clock, another for master clock. Events on master node have same values for both clocks.

In the beginning, all logical clocks are set to be 0. Once a new state is captured and stored, the corresponding clock value is increased. The local clocks increase naturally along with the events happening, and the master clock can only be updated when a message is received that contains a new master clock value. Workers' states collection can be divided into independent event groups by master clock value. In each group, events are naturally ordered and do not interleave with those in other groups. The possible causal relation between different groups, if exist, is determined by master clock. Thus, the problem of reordering and merging different events is reduced to sorting the event groups. Sorting groups is simple and has the same time complexity as the merging phase in merge-sort, namely O(n), where n is number of groups. For example, in Figure 2, sorting by master clock M value, all the events are divided into five groups. Each group presents an atomic sequence in a machine. Since the inner events of a group are naturally ordered, the reordering is efficient.

IV. Evaluation

In this section, we describe the performance of the state management system, with different storage solutions.

A. Testbeds

- **FutureGrid Sierra [21]**: a research purpose public cloud, experiments used up to 16 virtual machines.
- **Amazon EC2 cloud [22]**: up to 128 c1.medium virtual machines, each has 2 virtual CPU, 5 elastic compute units and 1.7GB memory.
- **DataSys**: an 8-core x64 server at IIT: dual Intel Xeon quad-core w/ HT processors, 48 GB RAM. This machine is used for experiment to study the merging overhead (Section IV.E)

B. Scientific Workloads

We use two applications that are representatives of scientific workloads using cloud environments: *Image Comparison* and *Event Processing*. Image Comparison compares an image with other images in the set to find similarities. These applications are representative of typical data processing scientific workflows.

C. Experiment Setup

We use the same workload for three storage solutions,

respectively based on files, Cassandra and DynamoDB. For synthetic benchmarks, on each state client, we send 10K requests in a tight loop to simulate an extremely operation-intensive scenario. Each request consists of 20 bytes key and 80 bytes value. Both key and value are randomly generated.

File-based solution: For file-based solution, the key-value pairs are saved to a local file on each client. Next, all these files are copied to a shared NFS directory, located on a dedicated VM where the files are merged. This is a simple solution for demonstrating state aggregation. Apparently its vulnerability to single point failures and the bottleneck can be addressed by well-known techniques such as mirroring or parallel file systems. We measure the time of writing to files, moving files to NFS server and merging events. We amortize the cost of file moving to state storage to get the average equivalent latency per state.

Cassandra-based solution: We use 1 to 8 Cassandra servers on dedicated VMs, and send requests from 1 to 128 state clients on VMs.

DynamoDB-based solution: DynamoDB is a service provided by Amazon. The data servers don't need to be deployed on the VMs. The VMs only need to communicate to the remote Amazon data stores and this has a minimal performance impact on local VMs. We provision the maximum available throughput for DynamoDB, which is 10K ops/sec. Up to 128 VMs on Amazon EC2 are used as state clients to send requests.

D. Metrics

The metrics measured and reported are average latency and throughput.

Average Latency: We consider the average latency as per request to write a state to data stores, measured in microseconds. Note that the latency includes the round trip communication and storage access time. Measuring latency for Cassandra and DynamoDB is straightforward, but file-based solution needs more care. We use the formula below to calculate the average equivalent latency t_{ave} for file-based solution, where t_w is the average file write latency, T_{moving} and $T_{merging}$ are the total time spent on moving and merging respectively, n is the total number of operations.

$$t_{ave} = t_w + (T_{moving} + T_{merging})/n$$

Throughput: The number of operations the system can handle over some period of time, measured in Operations per seconds (Ops per second).

E. Synthetic benchmark

Capture overhead: We conduct micro benchmarks on scales of up to 128 VMs. Note that the latency of file-based state management includes amortized cost for file moving, reordering and merging. Cassandra data stores crashed frequently and cannot serve requests at a scale of 8 servers with 128 clients. Similarly, DynamoDB's maximal throughput is reached at this scale and started to give errors, thus we only show results at a scale of 64 clients.

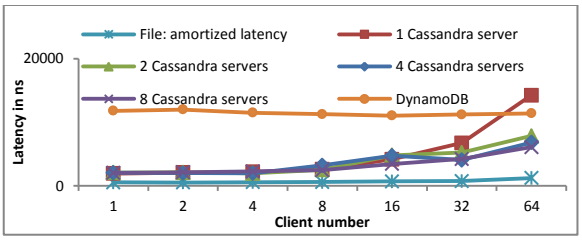

Figure 3. File-based solution has lower average latency. Cassandra performance decreases with the scale. DynomoDB latency doesn't change much with scale, but failed 128 clients test.

Figure 3 shows that file-based state solution has significant advantage over other two capture methods. When clients number increases, moving files to a single server causes contention. But this cost gets amortized across all requests. A single node Cassandra is saturated with 8 clients. On larger scales, multiple servers show some benefits but it is still limited, compared to the file based approach. DynamoDB shows very stable performance when facing different client request pressure before it is saturated, but it's at least three times slower than Cassandra at most scales.

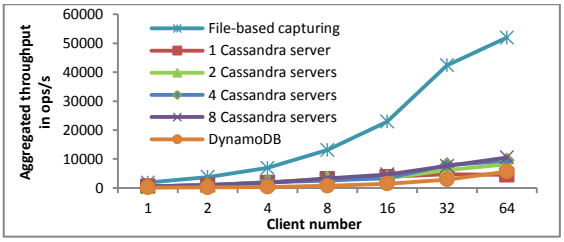

Figure 4. System throughput comparison. File-based solution obtains the maximum aggregated performance increases with scale.

Similar to the latencies, file-based solution delivers significantly higher throughput than other two. At 64 nodes, file-based solution achieves 52K ops/s, which is five times faster than a dedicated eight nodes Cassandra cluster and 18 times faster than DynamoDB based solution.

Events reordering and file merging overhead: On an 8-core Xeon server, we generate up to 512 state files. Each state file contains 10000 state records. Using a simple single thread merging program, 4 files cost 16ms, and 512 files cost 8209 ms. This can be further improved with more sophisticated merging algorithms in the future.

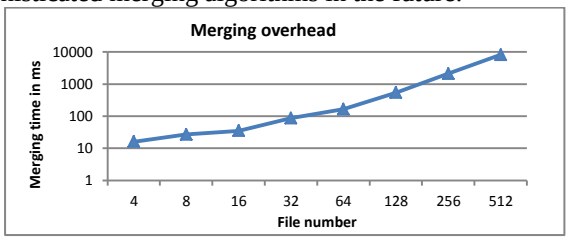

Figure 5. File merging time becomes longer with number of files.

File-based state management: We measured the time for capturing state and writing to file, moving and merging files respectively. We set an in-memory cache to boost the disk write performance. As shown in Figure 6 a full-size cache setting brings around 10% performance increase.

Since the state capturing on a local machine doesn't involve any contention, the latency is actually constant,

around 500us. Simultaneously moving a large number of files can cause contention, either on network or disks. The time spent on moving files keeps increasing even when the time is amortized. Better methods to aggregate data will be needed when running at larger scales. Merging overhead increases as well, but still negligible compare to other two overheads.

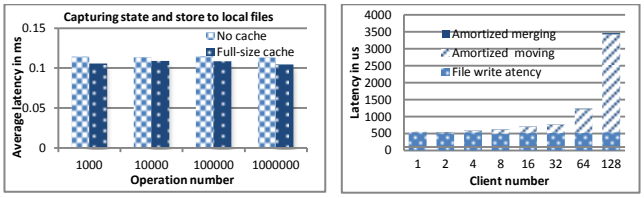

Figure 6. Left: File write operation scales well. Full-size cache brings around 10% performance gains. Right: File-based storage write latency is constant while merging time slightly increases. The amortized moving time increases exponentially.

F. *Scientific applications*

With integrated state management system in FRIEDA, we run two scientific applications (Image Comparison and Event Processing) to evaluate the overall performance impact of state collecting on real applications. Both applications are evaluated on FutureGrid [21] system.

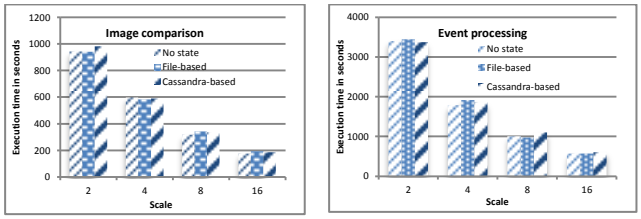

Figure 7. Image comparison and event processing performance

Although in synthetic benchmarks we observed huge difference of performance among different storage solutions, in application tests, we see no significant difference (state-introduced overhead is less than 5%). This is mainly because the micro benchmark tests execute operations in a tight loop while real applications have sparser and random patterns, so the total time spent on state management is very low compared to the application running time.

V. Related work

A. *Provenance*

Traditional data provenance represents the change history of data objects. Previous works on data provenance [23] have addressed different aspects, from operating system [21] to file systems[2], from databases [3] to cloud storage [4]. In our previous work [24], we have shown that distributed key-value stores can boost performance. Karma [7] provenance framework gives a set of tools for collecting provenance from workflow and process. Milieu [25] focuses on provenance collection for scientific experiments in HPC systems.

B. *Monitoring*

Monitoring gives users a perspective that combines resource utilization, cost efficiency and performance. Previous work has focused on runtime model and attempt to reach the balance between runtime overhead and monitoring capability[13]. Earlier works include Ganglia [14], a distributed monitoring system for clusters and grid systems. FRIEDA-State is event-driven i.e., it does not proactively go to fetch information, and hence is more efficient.

C. *Key-value stores*

Key-value stores (or distributed hash tables) are widely used as building blocks in many production systems, such as Amazon shopping cart with Dynamo[26], Facebook with Memcached [30]. Active key-value store projects include Cassandra[28], ZHT [18][19], Riak [27] and CouchDB [29]. This approach has many advantages, such as simplified API, encapsulated communication methods, the promise that to inherit desired features from key-value stores such as load balance, fault tolerance and scalability.

D. *Unsynchronized Time Clocks and Event Ordering*

In large scale distributed systems, unsynchronized clocks and drifting issue are inevitable. Based on different time baselines, it's hard to build meaningful semantics from distributed events or logs without synchronization or logical clock mechanisms.

Synchronization to a standard time source (atomic clock or GPS clock) is simpler. Typical cases are Precision Time Protocol [32] and NTP [33]. In recent projects, Google Spanner [31] adopts similar way to offer a synchronized clock to global scale databases and offers 5ms accuracy in global scales. Many works have been done for distributed event ordering. Beside Lamport's timestamp [34], Vector Clock [20] is another popular approach in today's systems.

VI. Conclusion

Scientific applications are increasingly using cloud environments and need a way to track the application's entire lifetime information both for monitoring and ensuring reproducibility. We propose and implement a state management system (FRIEDA-State) for a broad type of scientific applications running in cloud environments. FRIEDA-State has an innovative design that allows various storage mechanisms to be plugged-in while providing different trade-offs in durability, performance and usability.

In this paper, we discussed our implementations based on files, Cassandra and DynamoDB respectively and evaluated them on two cloud platforms. The evaluation showed that FRIEDA–State has very low overhead even when running at a scale of 64 virtual machines. File-based storage solution offers significantly better performance than key-value stores (e.g. Cassandra) on moderate scales. Furthermore, in some conditions, file-based storage is better than cloud databases services (e.g. DynamoDB) as well, in terms of latency and aggregated throughput. The major part of overhead of file-based storage solution is file moving, when using a centralized data server. Further scalability can be achieved with better merging algorithms for file-based systems or deploying larger number of NoSQL data nodes. We expect that as we increase scale

into 100s and 1000s of VMs, that the centralized data server will become a bottleneck, and distributed key-value stores would begin to offer better performance.

ACKNOWLEDGMENTS

This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This work used the Future Grid testbed funded by the National Science Foundation under Grant No. 0910812. This work used Amazon AWS resources and research grant. We would like to thank our colleagues for their generous help, namely Pradeep Mantha, Eugen Feller and Valerie Hendrix at Lawrence Berkeley National Laboratory and Dongfang Zhao at Illinois Institute of Technology.

REFERENCES

[1] Frew, J., M Etzger, D., Slaughter, P. Automatic capture and reconstruction of computational provenance. Concurrency and Computation: Practice and Experience 20 (April 2008), 485–496

[2] Muniswamy-Reddy, K.-K., H Olland, D. A., B Raun, U., Seltzer, M. Provenance-aware storage systems. In Proceedings of the 2006 USENIX Annual Technical Conference

[3] Peter Buneman and Wang-Chiew Tan. 2007. Provenance in databases. In Proceedings of the SIGMOD '07. ACM, New York, NY, USA, 1171-1173

[4] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. 2010. Provenance for the cloud. In *Proceedings of FAST'10*. USENIX Association, Berkeley, CA, USA, 15-14.

[5] Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor von Laszewski, Ioan Raicu, Tiberiu Stef-Praun, Mike Wilde. "Swift: Fast, Reliable, Loosely Coupled Parallel Computation", IEEE Workshop on Scientific Workflows 2007

[6] Jihie Kim, Ewa Deelman, Yolanda Gil, Gaurang Mehta, Varun Ratnakar. Provenance Trails in the Wings/Pegasus Workflow System. Concurrency and Computation: Practice and Experience, Special Issue on the First Provenance Challenge, 2007

[7] Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru. 2006. Performance evaluation of the karma provenance framework for scientific workflows. In Proceedings of IPAW'06, Luc Moreau and Ian Foster (Eds.). Springer-Verlag, Berlin, Heidelberg, 222-236

[8] Ioan Raicu, Ian Foster. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", Computer Science Dept., University of Chicago, Doctorate Dissertation, March 2009

[9] Y. Zhao, I. Raicu, S. Lu, X. Fei. "Opportunities and Challenges in Running Scientific Workflows on the Cloud", IEEE CyberC 2011

[10] Michael Wilde, Ioan Raicu, Allan Espinosa, Zhao Zhang, Ben Clifford, Mihael Hategan, Kamil Iskra, Pete Beckman, Ian Foster. "Extreme-scale scripting: Opportunities for large task-parallel applications on petascale computers", SciDAC 2009

[11] Ioan Raicu, Ian Foster, Yong Zhao, Alex Szalay, Philip Little, Christopher M. Moretti, Amitabh Chaudhary, Douglas Thain. "Towards Data Intensive Many-Task Computing", book chapter in "Data Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management", IGI Global Publishers, 2009

[12] Jin Shao, Hao Wei, Qianxiang Wang, and Hong Mei. 2010. A Runtime Model Based Monitoring Approach for Cloud. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD '10). IEEE Computer Society, Washington, DC, USA, 313-320.

[13] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run- time monitoring of instances and classes of web service com- positions," in Web Services, 2006. ICWS '06. pp. 63 –71

[14] Matthew L Massie, Brent N Chun, David E Culler, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Computing, Volume 30, Issue 7, July 2004, Pages 817-840

[15] Devarshi Ghoshal and Lavanya Ramakrishnan. 2012. FRIEDA: Flexible Robust Intelligent Elastic Data Management in Cloud Environments. In *Proceedings of SCC '12*. IEEE Computer Society, Washington, DC, USA, 1096-1105

[16] Rafail Ostrovsky and Boaz Patt-Shamir. 1999. Optimal and efficient clock synchronization under drifting clocks. In PODC '99. ACM, New York, NY, USA, 3-12

[17] Lavanya Ramakrishnan, Pradeep K. Mantha, Yushu Yao, Richard S. Canon, "Evaluation of NoSQL and Array Databases for Scientific Applications". DataCloud Workshop 2013

[18] Tonglin Li, Raman Verma, Xi Duan, Hui Jin, and Ioan Raicu. 2011. Exploring distributed hash tables in HighEnd computing. *SIGMETRICS Perform. Eval. Rev.* 39, 3 (December 2011), 128-130

[19] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. 2013. ZHT: A Light-Weight Reliable Persistent Dynamic Scalable Zero-Hop Distributed Hash Table. In *Proceedings of IPDPS '13*. IEEE Computer Society, Washington, DC, USA, 775-787

[20] Mattern, F. (October 1988), "Virtual Time and Global States of Distributed Systems", in Cosnard, M., Proc. Workshop on Parallel and Distributed Algorithms: Elsevier, pp. 215–226

[21] FutureGrid https://portal.futuregrid.org/, 2014

[22] Amazon EC2 Cloud http://aws.amazon.com/ec2/, 2014

[23] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. 2005. A survey of data provenance in e-science. *SIGMOD Rec.* 34, 3 (September 2005), 31-36. DOI=10.1145/1084805.1084812

[24] Dongfang Zhao, Chen Shou, Tanu Malik and Ioan Raicu. Distributed data provenance for large-scale data-intensive computing. IEEE CLUSTER '13

[25] You-Wei Cheah , Canon, R. , Plale, B. ; Ramakrishnan, L. Milieu: Lightweight and Configurable Big Data Provenance for Science, Big Data (BigData Congress), 2013

[26] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. "Dynamo: Amazon's Highly Available Key-Value Store." SIGOPS Operating Systems Review, 2007

[27] Riak, http://docs.basho.com/riak/latest/, 2013

[28] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 35-40

[29] CouchDB, http://couchdb.apache.org/, 2014

[30] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski. 2013. Scaling Memcache at Facebook. In nsdi'13, USENIX Association, Berkeley, CA, USA, 385-398

[31] James C. Corbett, Jeffrey Dean, Michael Epstein, etc. 2012. Spanner: Google's globally-distributed database. In proceeding of OSDI'12. USENIX Association, Berkeley, CA, USA, 251-264

[32] Precision Time Protocol, http://www.ieee1588.com/, 2014

[33] David L. Mills (12 December 2010). Computer Network Time Synchronization: The Network Time Protocol. Taylor & Francis

[34] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (July 1978), 558-565