Milieu: Lightweight and Configurable Big Data Provenance for Science

You-Wei Cheah+, Richard Canon*, Beth Plale+, Lavanya Ramakrishnan*
+School of Informatics and Computing Indiana University, Bloomington, IN
Email: {yocheah, plale}@cs.indiana.edu
*Lawrence Berkeley National Laboratory, Berkeley, CA
Email: {scanon, lramakrishnan}@lbl.gov

***Abstract*—The volume and complexity of data produced and analyzed in scientific collaborations is growing exponentially. It is important to track scientific data-intensive analysis workflows to provide context and reproducibility as data is transformed in these collaborations. Provenance addresses this need and aids scientists by providing the lineage or history of how data is generated, used and modified. Provenance has traditionally been collected at the workflow level often making it hard to capture relevant information about resource characteristics and is difficult for users to easily incorporate in existing workflows. In this paper, we describe Milieu, a framework focused on the collection of provenance for scientific experiments in High Performance Computing systems. Our approach collects provenance in a minimally intrusive way without significantly impacting the performance of the execution of scientific workflows. We also provide fidelity to our provenance collection by allowing users to specify three levels of provenance collection. We evaluate our framework on systems at the National Energy Research Scientific Computing Center (NERSC) and show that the overhead is less than the variation already experienced by these applications in these shared environments.**

***Keywords*-Provenance, High Performance Computing, Database**

I. Introduction

Data volumes and complexity is growing rapidly in scientific collaborations. For example, the Large Hadron Collider (LHC) projects 16PB/year, the Large Synoptic Survey Telescope (LSST) projects 6PB/year of raw data and 3PB/year of catalog data and climate scientists project collections of hundreds of exabytes are expected by 2020 [1]. As assumptions and educated guesses are used to reduce or reorganize the data sets, it is necessary to capture data provenance to provide context and reproducibility. Data provenance is the lineage of the data and brings a historical perspective to data products.

High Performance Computing (HPC) environments have traditionally been used for large monolithic simulations running on thousands of nodes. More recently, HPC systems are being used to process large volumes of data through analysis workflows. It is important to track the runs and data to enable lineage tracing of experiments and data. Users of HPC systems currently maintain provenance indirectly. They embed timestamps or other simple descriptions within file names and/or through other manual methods. Since current approaches are usually ad-hoc, these methods are error-prone and do not scale to large data volumes and systems.

Provenance or tracking lineage of data and computation has traditionally been done in the context of workflows [6]. However, provenance collection at the workflow level is not very flexible to user needs and makes it difficult to capture resource characteristics. Additionally, it is not accessible to a large number of users who rely on scripts to manage their computation and data on HPC systems. Our work addresses this gap by providing a minimally intrusive, light-weight provenance collection and storage system for HPC applications.

In this paper, we present Milieu, a provenance collection and storage framework. Milieu, as the name suggests, is designed to operate in the background being minimally-intrusive and light-weight. Provenance collection and query systems have been closely tied in the provenance systems so far. Thus, provenance collection and storage systems are designed and optimized for a range of queries anticipated and/or provenance queries are limited by what is available in the storage. However, as we scale up to large systems and data volumes, it is necessary to support semi-structured provenance collection and more complex provenance analysis. Milieu achieves this by operating in an envisioned tiered architecture where storage for collection and storage for analysis might be different.

Milieu is designed to be flexible (i.e., support various provenance data models), extensible (i.e., can be easily expanded to support other resource environments), semi-transparent (i.e, user initiates provenance but instrumentation is automated) and scalable (i.e., able to support large provenance collections on petascale systems).

Specifically, the contributions of this paper are:

- We present an architecture and implementation of Milieu: a provenance collection and storage framework for scientific applications running on HPC systems.
- We evaluate Milieu on two large systems at the National Energy Research Scientific Computing Center (NERSC) including a petascale system and show that the provenance overhead is minimal and within the normal variation experienced by the applications on the systems.
- We present and evaluate our query interface for provenance collection and provide a framework for provenance analysis support.

II. Overview

Figure 1 shows the envisioned architecture for provenance collection and querying for scientific applications. We propose a tiered architecture for provenance collection and analysis where the storage for provenance collection is separated from the storage model for querying and analysis.

A. *Tiered Provenance System*

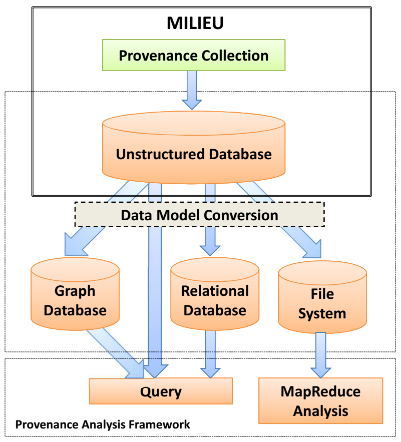


Fig. 1: Overview of Provenance Collection and Analysis Framework. Milieu is the first tier where provenance data is collected in unstructured or semi-structured format. The second tier is more focused on provenance analysis and appropriate storage options might be used. This architecture allows us to provide the best optimized storage model for collection and specific analysis types.

Milieu, the focus of our paper, centers on the provenance collection aspects of this architecture. Provenance is collected from HPC systems through a collection mechanism that is minimally intrusive. A mixture of provenance about data products and provenance of resources and system environment are also collected. The user is presented with the choice of collecting provenance from multiple levels (described in more detail in Section III).

The amount of provenance data may vary according to the application and also the instrumentation done by the user. Since provenance is captured in all forms and sizes, Milieu does not store provenance in a well-structured manner, prioritizing instead the ability to write the captured provenance in a fast and efficient manner. Milieu provides a way to collect all raw provenance data from the systems. Provenance stored in the provenance buffer can then be queried directly or further processed and transformed into different formats and more structured provenance. Associating structure with the data is left to downstream provenance systems (e.g., Karma [18] or other data analysis tools (e.g., graph analysis or MapReduce/Apache). By separating provenance collection storage and query storage, optimizations will be possible at both levels.

B. *Use Cases*

We consider two use cases for our provenance collection system. The first use case is for user job submissions. In the first case, provenance needs to be tracked for data processed in job scripts that are submitted to the batch queue systems on HPC systems. We need to track the job submission, execution and completion so system administrators and users can track every step of the execution. Additionally, users often perform a myriad of data tasks on the command prompt. Tasks might include data preparation for the run and moving data to archival systems.

C. *Design Goals*

Milieu is based on the following design principles:

Support for various provenance data models. Provenance systems have so far relied on structured data models that are mirrored by a relational database in the backend. Provenance collection systems capture a set of records at known points in the execution cycle (e.g., workflow started). However, job scripts on HPC systems vary widely and hence it is hard to develop a strong-schema based data model. Thus, it is important that we support semi-structured provenance data models from different users, systems and applications.
Low overhead. As we process large volumes of data, high performance is critical. Hence it is very important that provenance collection has minimal impact on application's performance.
Semi-transparent collection. Provenance collection needs to be minimally intrusive to the user's workflows. It is important that provenance collection be easily initiated by the user but mostly transparent during execution. Automated instrumentation of user's scripts are necessary at the scales of the workflows we aim to address.
Support user annotations. In addition to automated instrumentation, it is often necessary to capture user's notes or metadata. Thus, we need an interface that allows users to add their notes about the experiments and data either before, during or after a run is over.
Staged provenance levels. HPC applications tend to vary significantly in their needs and use. For example, routine simulation runs might need just a base level of provenance to know jobs start and finish times and other basic characteristics of the data. However, some applications or use scenarios may need a higher granularity of data collected during execution.
Scalability. Simulation sizes and data collections on HPC systems are rapidly growing. In turn, provenance data is also rapidly growing making it necessary to scale up provenance collection and storage mechanisms. This is going to be especially true as we lead into the exascale era.

III. Milieu

Figure 2 shows the design of Milieu to capture and query provenance. The user submits a job to a batch queue system, which is then executed when resources become available. The provenance instrumentation module in Milieu first captures the original job script and stores information of the script in our data store. Next, it instruments the job script with provenance calls and the modified job script is then submitted to the queue. During execution, the instrumented provenance

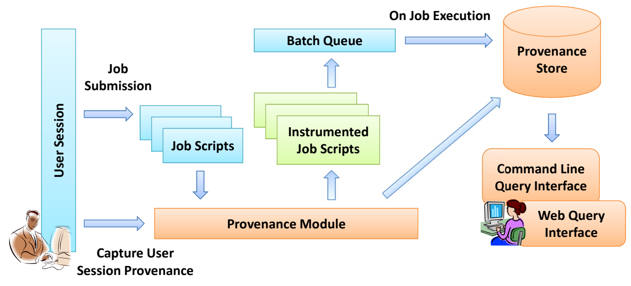


Fig. 2: Architecture of provenance framework. The activation of the collection mechanism triggers user session provenance and provenance from job script executions to be captured, which is stored in a MongoDB data store. The stored provenance can be retrieved either through a command line interface or via a web query interface.

calls load provenance data into the provenance store. The provenance data is then accessible through a command-line query and web interface.

With provenance capabilities built into HPC systems, the user can rely on captured provenance to track their jobs and data. Moreover, scientists can use the provenance system as their scientific 'notebook' for keeping notes of their applications. Users can also perform searches and identify similarities or differences between their experiments. In addition, system administrators of HPC systems can also benefit from the use of provenance by leveraging the collected provenance to help with the debugging of scientific applications.

A. *Collection*

The collection component that we have designed is responsible for the capture of provenance of job script executions and those of user sessions. The provenance capture of a user session is collected in its entirety by our framework. For the provenance collection, we devised three levels of provenance capture:

Level 1 - First level provenance consists of basic provenance. These includes information about the script, the outputs of the job run, basic environment about where the job was submitted and user annotations. This is essentially provenance that a scientific user would be interested in.

Level 2 - The second level includes all of the provenance from level 1 and additionally provides more detailed resource information of the resources involved for the computation. Provenance at this level are more of interests to system administrators and/or for detailed analysis of resource usage for a particular class of problems.

Level 3 - In level three, we capture everything captured at level 2 with the addition of provenance in the form of detailed traces of I/O calls for commands in the job script. We anticipate this level to be used rarely to collect detailed I/O information for data sets and/or debugging.

The collection levels in our framework are designed based on current user requirements. These requirements were gathered based on discussions with our users, where they are generally scientists or system administrators. However, we anticipate that the exact implementation of these levels might vary by deployment and resource types and Milieu is extensible to accommodate these changes.

B. *Storage*

The captured provenance is diverse and varied. Thus, the data store must be capable of supporting the storage of semi-structured provenance documents. In our current implementation, we use MongoDB, a NoSQL data store for our storage needs. MongoDB is a scalable, high-performance data store that is open-source and developed in C++. MongoDB is ideal for our needs due to its native support for semi-structured data, its support for sharding. MongoDB also has the capability for performing complex aggregate analysis using MapReduce through both its native MapReduce framework as well as the MongoDB-Hadoop connector that will allow rich provenance analysis tools to be developed leveraging Milieu.

In our system, captured provenance is grouped around job IDs (unique identifiers assigned by the batch queue system) or file identifiers (location based). However, each entry in the data store is not constrained by a specific format and may vary among different jobs or files. This enables us to use a single store across possibly multiple systems and user sessions (e.g., job vs shell).

C. *Query*

In Milieu, the resulting provenance can be accessed through query interfaces that support a simple query language centered around the notion of job IDs and file names. The query language is designed to abstract the underlying language used at the data store level and allows the user to query the provenance in an intuitive manner without knowing too much about how the underlying data is stored. In our framework, both command line and a graphical web interface is designed for different users. The command line interface is meant to be used by users who are comfortable with the command line. The web interface, on the other hand, is meant to be akin to "Google" of provenance collection, supporting fairly open-ended provenance queries on the collection.

IV. Implementation

Our initial implementation of Milieu was built for NERSC HPC systems and we detail our implementation in this section.

A. *Collection*

The Milieu collection module is available to the user by loading the module for provenance. Modules is a simple environment management tool used on HPC systems that allows setting up of the environment through a simple set of commands (e.g., module load, module unload).

For the provenance capture from job executions, the user needs to submit their job scripts via a command, *qsub-prov*, which is a wrapper command around the original *qsub* command (part of the Portable Batch System). We use a different name for the job submission so that the early users of our system are aware of their participation in the provenance. Our eventual goal is to make this completely transparent to the end-users.

```
#!/bin/csh
#PBS -j oe
#PBS -q regular
#PBS -N GTCcloud
#PBS -l walltime=00:20:00
#PBS -l nodes=8:ppn=8
#PBS -S /bin/csh
module load python
python $USER/provenance/source/python_scripts/
    timestamp.py $USER/provenance/source/
    python_scripts/dbconfig.py run.pbs 'begin
    script' yocheah $PBS_JOBID
qstat -f $PBS_JOBID > $USER/GSCRATCH/GTC/qstat
    .$PBS_JOBID

set verbose
cd $PBS_O_WORKDIR
cp ./gtc.input.64MPItasks.cloud gtc.input
record_provenance ``Edit date'' ``2012-01-01''
strace -tt -o $USER/GSCRATCH/GTC/strace.out
    .1.31543.20120927185010 mpirun -n 64 ./
    gtcmpi
rm gtc.input
cp $PBS_JOBID.OU `pwd'/output
    .31543.20120927185010
echo `$USER/provenance/source/bin/
    qsub_file_insert hopper12 yocheah
    $PBS_JOBID 3 $USER/provenance/source/
    python_scripts/file_insert.py $USER/
    provenance/source/python_scripts/dbconfig.
    py $USER/GSCRATCH/GTC/qstat.$PBS_JOBID
    $USER/GSCRATCH/GTC/strace.out
    .0.31543.20120927185010 $USER/GSCRATCH/GTC
    /strace.out.1.31543.20120927185010 $USER/
    GSCRATCH/GTC/strace.out
    .2.31543.20120927185010 $USER/GSCRATCH/GTC
    /output.31543.20120927185010'
python $USER/provenance/source/python_scripts/
    timestamp.py $USER/provenance/source/
    python_scripts/dbconfig.py run.pbs `end
    script' yocheah $PBS_JOBID
```

Listing 1: Example of an instrumented PBS script. Highlighted texts are instrumentation added or modified by the provenance framework or user.

```
$ cmdline_pc
# begin user session
...
# end user session
$ exit
```

Listing 2: Provenance capture for a user session.

The *qsub-prov* command also takes an optional integer parameter that corresponds to the level of provenance capture (akin to the levels that are specified in a logger). When a job script passes through this wrapper, the original copy of the script is kept as provenance along with other static information about the system, such as the host name, IP addresses, user credentials, and a host of other information. The PBS headers of the script are also extracted, parsed and stored as well. An instrumented copy of the script is then forwarded to the PBS queue (Listing 1). Depending on the level of provenance capture that was set by the user, the script is instrumented at different granularities. The instrumentation enables a range of things to be captured including the output of a job, information in the PBS queue during execution, to more detailed provenance such as straces of individual calls and also outputs from the Integrated Performance Monitoring (IPM) [19] used at NERSC. IPM is an application profiling framework that provides details such as the time spent by the application computing and communicating using MPI.

The more detailed level of provenance provides information about the nodes that were involved in a distributed MPI job and also more detailed information about system calls if necessary. The design decision to enable provenance in various levels is driven by the provenance demands that we foresee for different users and also the flexibility that allows us to conserve storage and reduce instrumentation overheads where possible.

In order to specify the calls that the user wants to strace, the tag *record_trace_provenance* has to be added in front of the intended call. Additional user provenance can also be captured in the form of name value pairs by adding a line *record_provenance name value* in the users job scripts, as shown in the listing.

Provenance is pushed off to storage in three stages, namely pre-execution, the start of execution, and end of execution. Once a job is submitted, initial provenance about the node where the submission happens and the provenance of the PBS headers are stored. Once the job is executed, runtime provenance such as the execution variables, nodes on which the job was distributed to and fields such as timestamps are captured. Finally, straces (if enabled) and outputs are put into files and stored right before the job is terminated. This is done in stages to minimize the number of database I/O operations and to also minimize the performance impact to the actual job.

Provenance about a users session is collected through a wrapper *cmdline_pc* that uses the shell command *script*. Basic information about the system on which the session is initiated and the timestamps of the start and end of the session are also recorded. When a user is done with their session, the captured provenance is stored in a file, which is then put into storage on our provenance Mongo database (MongoDB).

We implemented our collection tools using a mixture of Python and Shell scripts. In all cases, the Shell scripts were used to interact at the front end, since this is native in a HPC environment.

B. Storage

We group all provenance documents under a single provenance collection. For a single job ID, multiple provenance documents may exist in MongoDB. Mandatory fields ("columns" in the traditional sense) for each document include an associated job ID, user ID and timestamp. Most documents contain two fields with one acting as a description and a second field containing the value associated with that description. The provenance store also has information about the systems. The system information is more structured and will contain more fields, such as the host name, IP address, user environment variables, etc. Thus, we are able to use a single collection for different types of data.

For file management, we use the MongoDB GridFS specification for storing the actual file contents due to its ease of use, metadata and large file support. GridFS uses two collections for storing data. Large data objects are split into small chunks, which are then stored in the *chunks* collection, while metadata that describes the data object is stored in the *files* collection.

C. Query and Access

Milieu provides two query interfaces a) command-line and, b) web interface, that allows the user to query and make simple edits to the MongoDB provenance storage. The web query interface uses Django [9] (version 1.3). Our query components support basic queries such as querying for the name of a file, the name of a job, and any other attribute that is associated with a job or data file. In both interfaces, we also provide the capability for users to add annotations to the captured provenance. These annotations are added to the existing provenance in the form of name value pairs.

```
[Provenance Query$] jobid . (field,walltime) (
    value,00:10:00)
```

Listing 3: Example of a query to return all job IDs that have walltime 00:10:00 in the command line.

```
[Provenance Query$] file MILC
```

Listing 4: Example of a query to return all filenames that contain MILC.

The implemented query language is mainly focused on the retrieval of provenance centered around jobIDs and file names. Regular expression queries are also supported. A sample of a query to return job IDs with a specific matching criteria is given in Listing 3 and an example for querying all files in the provenance store that has file name of the MILC application is given in Listing 4. We anticipate that for more complex queries and analysis, some of the specialized tools such as graph query languages and MapReduce will be used.

V. Experiments

We evaluated our provenance collection and query mechanisms through examining the performance of our provenance framework. We use three applications from the NERSC-6 application benchmark suite [2]. A range of problem sizes and core counts were used to capture mid-range to large-scale codes. Our experiments were evaluated on two NERSC systems, namely Carver and Hopper, a petascale system.

A. Testbed Setup

Carver is an IBM iDataPlex system with 1202 compute nodes. The compute nodes are a mix of 2.67GHz Intel Quad-core Nehalem processors with 24GB and 48GB of memory, two six-core Westmere 2.67GHz processors with 48GB of memory and, four eight-core Nehalem-EX 2.00GHz processors with 1TB of memory. All nodes are connected using 4X QDR InfiniBand technology.

Hopper is a Cray XE6 peta-flop system consisting of 6384 nodes. Each node consists of 2 twelve-core AMD 'Magny-Cours', with 2.1GHz processors per node for a total of 24 cores with 64GB of memory. The compute nodes are connected via a custom high-bandwidth, low latency network provided by Cray in a 3D torus topology.

In all experimental cases, we used the regular queue of the system for evaluation, which is used by the users of these systems. The storage and query mechanisms were hosted on a single Carver node. A single MongoDB instance (version 2.0.6) was deployed for storing the collected provenance along with the Django based web query interface.

Workload. For the evaluation of provenance collection, we performed experiments using three applications from the NERSC-6 applications benchmark suite [2]: GTC, MILC, and PARATEC. Based upon previous research [2], these applications represent a significant portion of NERSC workloads.

GTC short for 3D Gyrokinetic Toroidal Code, is a 3-dimensional particle-in-cell (PIC) code used to study micro-turbulence in magnetically confined toroidal fusion plasmas. We used version 2 of the GTC code with a large problem size that involves 66,455,552 number of grid points as input on Hopper and 2 million grid points on Carver.

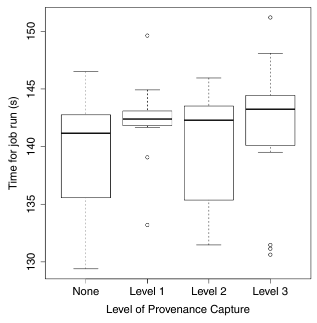
MILC or MIMD Lattice Computation is used in part to study quantum chromodynamics (QCD), the theory of the subatomic "strong" interactions responsible for binding quarks into protons and neutrons and holding them together in the nucleus. We used version 7 of the MILC code in our experiments using the extra large input lattice size of 64x64x64x144 on Hopper. A smaller version using a lattice size of 32x32x16x18 with 2 quark flavors, four trajectories and eight steps per trajectory was used on Carver for our evaluation.

The PARAllel Total Energy Code (PARATEC) benchmark code performs *ab-initio* quantum-mechanical total energy calculations using pseudopotentials, a plane wave basis set and uses an all-band (unconstrained) conjugate gradient (CG) approach for solving Density Functional Theory's (DFT) Kohn-Sham equations. The version we used is based off the NERSC-6 input that contains 6 conjugate gradient iterations and only 250 atoms in a diamond lattice configuration. This input does not allow any aggregate over the transposed data.

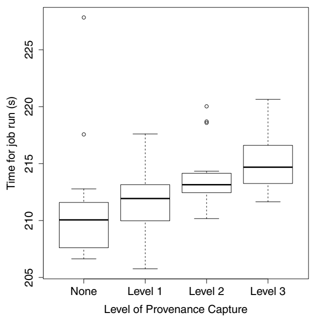
The applications were executed on Carver using 8 nodes with 8 processes per node. On Hopper, GTC was executed using 2048 cores and MILC was executed using 4096 cores. We conducted our experiments by comparing the different levels of provenance capture along with the base case of just running the application without any provenance capture. For each application, we perform 15 measurements without any provenance capture and also 15 measurements for each level of provenance capture (levels 1–3). We only performed 7 Hopper measurements for each scenario since jobs are more expensive than their Carver counterparts both resource-wise and time-wise. All results were generated from job runs during normal production time on the NERSC machines discussed above.

B. Evaluation of Provenance Collection

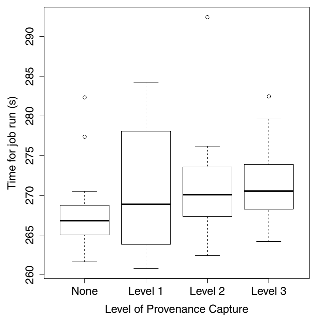
In this subsection, we discuss our results from the evaluation of three applications on two of NERSC HPC systems: Hopper and Carver. Each box plot shows the timings of the three levels



(a) GTC-Carver



(b) MILC-Carver

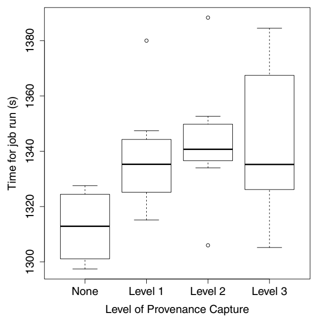


(c) PARATEC-Carver

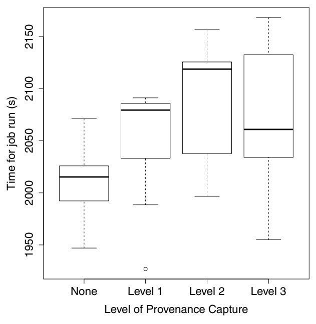
Fig. 3: Timings for base run and 3 different levels of provenance capture on Carver. Provenance capture does not yield significant overhead.

of provenance in comparison to the base case with the timings without any provenance capture. The box plots capture the median (dark bold lines in the middle of boxes), and also the quartile values (lower and upper boundary of boxes) The median is a more robust indicator over the mean and is less affected by outliers than the average.

Carver. For our experiments on Carver, we conclude that the overhead associated with GTC, MILC and PARATEC to be insignificant compared to the base case with no provenance capture. Based on the box plot for GTC (Figure 3a), we observe that the medians for all levels of provenance capture to be around 141–143 seconds. For MILC (Figure 3b), we observe the same trend with the median for no provenance capture being around 210 seconds and those with provenance capture ranging from 211–214 seconds. In the case of PARATEC (Figure 3c), the median without provenance capture is around 267 seconds and the ones with provenance capture having medians of 269–271 seconds. This shows us that the provenance capture overhead is within the normal variability (well within 2%) that these application already experience on these systems.



(a) GTC-Hopper



(b) MILC-Hopper

Fig. 4: Timings for base run and 3 different levels of provenance capture on Hopper. Provenance capture shows a slight overhead (2–3%) compared with the base case. Median timings for level 3 are lower than level 1 and 2 but have more spread.

Hopper. For GTC (Figure 4a), we note that the provenance capture accounts for about 22–28 seconds (overhead of slightly over 2%). Figure 4b shows the performance of MILC on Hopper with different provenance levels. The same trend occurs for MILC on Hopper with an overhead of about 46–104 seconds (2–5% overhead).

This falls within the variability that applications experience in these environments. This shows that the provenance capture mechanism minimally impacts timings of the original job runs.

Analysis of Overhead. We timed the execution of each individual script in our provenance capture module for a single job submission. A total of five scripts were timed. The qsub-prov shell script handles the pre-processing of the job script and invokes a qsub_pc.py python script. This python script handles the ingestion of provenance captured at this stage into MongoDB. Similarly, qsub_file_insert is a shell script that searches and inserts files into our provenance storage via a python script: file_insert.py. The other script that we timed was timestamp.py, which is responsible for capturing the begin and end timestamp entries. Table I details the breakdown of our timings for these scripts on Hopper and Carver respectively. The timings reflect the average taken from three instrumented runs on both systems. The provenance capture on Hopper uses level 2 while on Carver, the provenance capture used is level 3. From the results of both tables, we observe that our provenance module contributes very little to the overhead of the end-to-end execution of these applications. The Hopper overhead is a little higher than Carver since Carver is more directly connected to the file system server than Hopper.

TABLE I: Duration taken for individual scripts for provenance capture of a GTC job on Hopper (level 2) and Carver (level 3).

Script Name	**Duration (s)**	
	Hopper	**Carver**
qsub-prov	6.05	3.00
– qsub_pc.py[1]	(0.08)	(0.07)
timestamp.py (begin)	0.25	0.05
qsub_file_insert	7.77	0.30
– file_insert.py[2]	(0.43)	(0.03)
timestamp.py (end)	0.16	0.11

[1,2] Duration of qsub_prov.py and file_insert.py accounted within qsub-prov and qsub_file_insert respectively.

C. Evaluation of Provenance Query

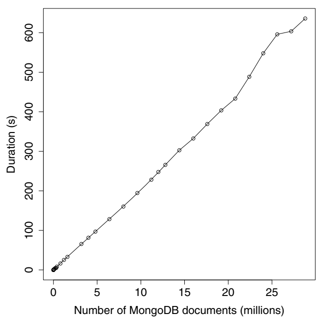


Fig. 5: Plot of query duration vs number of MongoDB documents returned. The return time for each query is generally linear with the number of results returned.

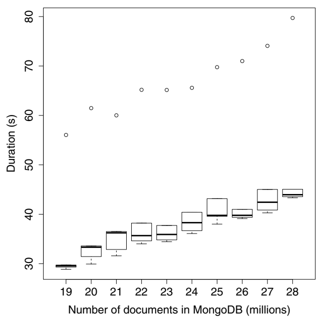


Fig. 6: Plot of query duration for a regular expression query of a single job ID vs number of MongoDB documents. Outliers reflect the timings of the first query issued. The query performance here is linear and the result size returned for each query is 16 documents.

We also evaluated the performance of querying the stored provenance through a few queries.

Regular Expression. The first case that we have evaluated is by observing the amount of time required to query for a regular expression job ID query by iterating through all available documents in MongoDB and returning all documents. We observe that the query performance generally increases linearly as the number of documents that are being stored in the database increases. This is depicted in Figure 5. We do notice that as the database scales up to approximately 10 million records, the time needed to iterate through the entire database is already 200 seconds, and for 25 million records it is 595 seconds. Thus, if there are queries that need to query all data in a database, they will benefit from a MapReduce framework that will allow scalability.

Limited set queries. The second case that we looked at was the evaluation for a regular expression query of a single job ID for a MongoDB database of different sizes. Figure 6 depicts the results of our evaluation. The outliers in this figure are the first queries issued against MongoDB. These first queries are retrieved and cached in memory, resulting in subsequent identical queries having less significant response times, approximately half the response times of the first queries.

Exact indexed queries. As a comparison, we also looked at exact queries for the same job ID. For these queries, the performance of the query is very fast with response times within 600–800 microseconds even for a large database with 24 million documents.

VI. Discussion

In this paper, we have described and evaluated the design and implementation of Milieu for semi-structured provenance collection and storage on HPC systems. In this section, we discuss provenance quality, operational challenges, usability, future storage and query scalability.

Uses of Provenance. Provenance captured in Milieu is a mixture of data and process provenance and can be used for a number of use cases. Our approach is useful in some cases and falls short in others. We discuss the pros and cons of our captured provenance in this section.

Data Management The captured provenance helps the user identify with ease where data objects are stored. Users can use Milieu to query the NoSQL data store using regular expressions to search previous job runs for a variety of information including data objects, status, etc.

Faults The provenance can be used to determine if the job script terminated properly. The provenance of a job script that terminates successfully contains a timestamp of when the job script terminates. Faults that affected single jobs or a group of jobs can be easily identified with queries and/or analyses. Issues with the underlying hardware or environment can be identified by examining the straces or the ipm log files.

Our current approach does not account for automatic identification of input data sources. Different scientific applications take input sources in different formats (e.g., fixed file names, on the command-line). This makes it difficult to automatically identify input sources. In future work, we plan to address this by prompting the scientists to indicate input data sources for the framework. Additionally, the captured provenance is not compatible with the OPM (Open Provenance Model) specification [15] since our goal is on efficient capture in close to raw format.

In future work, it will be important to assess the quality of provenance to ensure that the captured provenance can be used as intended [5]. The completeness and correctness of the provenance trace needs to be assessed such that captured provenance reflects unambiguous and factual events.

Flexibility. Milieu provides support for multiple levels of provenance. This allows the scientist or system administrator to pick the level that is most suitable for their use case. The support for user annotations allows users to capture notes and metadata that are often lost. It should be noted that Milieu does not require users to make any changes to their job scripts. The support for user annotations allows users to capture notes and metadata that are often lost but is not required to collect other provenance information. Provenance initiation is user controlled but instrumentation is automated.

Operational considerations. In our current implementation, all users are stored in a single collection. This optimizes the queries by system administrators performed across users. In future work, we plan to provide a start-time configuration to control per user collection if it is required in cases where the usage model might differ.

Additionally, we operate MongoDB in a single-server mode. MongoDB allows sharding that allows us to easily scale up as data grows. MongoDB sharding is well-documented and evaluated [8] and can be easily configured for MongoDB.

Scalable Storage. The "big data" movement has seen a number of NoSQL data stores for horizontal scalability distributed over many servers [4]. In this paper, we used Mongo database (MongoDB) [14], a document-oriented data store, since it was well suited for semi-structured provenance documents. MongoDB is typically well suited for write-once, read-many times pattern data access, as is the case with Milieu. The NoSQL systems are evolving rapidly and it is possible that some other data store might provide additional features and/or better performance. Our architecture and methodology is not tied to the use of MongoDB and thus it will be possible to use other data stores with Milieu in the future.

VII. Related Work

In recent years, multiple provenance collection systems have been developed in the context of workflow tools, e.g., VisTrails [3], Kepler [13], and Pegasus [12]. More recently, we have systems such as RAMP [17] that targets collecting provenance from MapReduce [7] workflows. Karma [18] is a workflow based system that supports provenance capture in semi-structured eScience environments and Provenance Aware Storage System (PASS) [16] is targeted at capturing provenance at the operating system and file system levels.

Gadelha Jr. et al. [10] described how the collection and querying of provenance can help in managing large-scale scientific computations in their respective environments. Jones et al. [11] detail a provenance enabled file system that automatically collections information flow provenance at the file system level with the goal of aiding scientific users in better organizing their data.

Our work captures and represents provenance from multiple layers of an application including the resources and environment on which an application is executed in the HPC environment. It also captures traces for data provenance, and provide the user with the ability to determine the level of fidelity of provenance that they want captured. The novelty of our work lies in our lightweight and minimally intrusive provenance capture that also allows user fidelity. In addition, our architecture design allows captured provenance to be streamlined for specific use cases.

VIII. Conclusion

In this paper, we present the design, implementation and evaluation of Milieu, a minimally-intrusive, light-weight, multi-level provenance collection, storage and query framework. Milieu supports the collection of semi-structured provenance data from jobs and user commands on HPC systems. Our evaluation on NERSC production systems, including a petascale machine shows that the collection overhead is minimal. Milieu makes it possible to build a multi-tiered provenance architecture where storage for collection and storage for optimized queries can be separated. A multi-tiered architecture will be able to support a wider range of provenance queries and analyses that is difficult if not impossible today.

IX. Acknowledgements

This work was funded by the Office of Science of the U.S. Department of Energy under Contract Number DE-AC02-05CH11231 (including use of resources at NERSC).

References

[1] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, and et al. Scientific Discovery at the Exascale: Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization. http://www.olcf.ornl.gov/wp-content/uploads/2011/01/Exascale-ASCR-Analysis.pdf, 2011.

[2] K. Antypas, J. Shalf, and H. Wasserman. NERSC-6 Workload Analysis and Benchmark Selection Process. Technical Report LBNL-1014E, Lawrence Berkeley National Laboratory, August 2008.

[3] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: visualization meets data management. In *Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of data*, SIGMOD '06, pages 745–747, New York, NY, USA, 2006.

[4] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.

[5] Y.-W. Cheah and B. Plale. Provenance analysis: Towards quality provenance. In *Proc. of the 2012 IEEE 8th Intl. Conf. on E-Science (e-Science)*, pages 1–8. IEEE, 2012.

[6] S. B. Davidson and J. Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of data*, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008.

[7] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[8] E. Dede, M. Govindaraju, D. Gunter, S. Canon, and L. Ramakrishnan. Semi-structure Data Analaysis using MongoDB and MapReduce: A Performance Evaluation. In *submission*.

[9] Django. https://www.djangoproject.com. Last accessed: April, 2013.

[10] L. M. R. Gadelha, Jr., M. Wilde, M. Mattoso, and I. Foster. Exploring Provenance in High Performance Scientific Computing. In *Proc. of the 1st annual workshop on High performance computing meets databases (HPCDB2011)*, 2011.

[11] S. N. Jones, C. R. Strong, A. Parker-Wood, A. Holloway, and D. D. E. Long. Easing the Burdens of HPC File Management. In *Proc. of the 6th workshop on Parallel Data Storage*, 2011.

[12] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the Wings/Pegasus system. *Concurrency and Computation: Practice and Experience*, 20(5):587–597, 2008.

[13] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, and et al. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[14] MongoDB. http://www.mongodb.org. Last accessed: April, 2013.

[15] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, and et al. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[16] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-Aware Storage Systems. In *Proc. of the Annual Conf. on USENIX '06 Annual Technical Conf.*, ATEC '06, Berkeley, CA, USA, 2006.

[17] H. Park, R. Ikeda, and J. Widom. RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows. In *37th Intl. Conf. on Very Large Data Bases (VLDB)*, August 2011.

[18] Y. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance Management for Data Driven Workflows. *Intl. Journal of Web Services Research*, 5(2):1–22, 2008.

[19] D. Skinner. Integrated Performance Monitoring: A portable profiling infrastructure for parallel applications. In *Proc. of ISC2005: Intl. Supercomputing Conf.*, 2005.