CloudTracker: Using Execution Provenance to Optimize the Cost of Cloud Use

Geoffrey Douglas, Brian Drawert, Chandra Krintz, and Rich Wolski

Computer Science Dept.
Univ. of California, Santa Barbara

Abstract. In this work, we investigate tools that enable dollar cost optimization of scientific simulations using commercial clouds. We present a framework, called CLOUDTRACKER, that transparently records information from a simulation that is executed in a commercial cloud so that it may be "replayed" exactly to reproduce its results. Using the automated CLOUDTRACKER provenance and replay facilities, scientists can choose either to store the results of a simulation or to reproduce it on-demand – whichever is more cost efficient in terms of the dollar cost charged for storage and computing by the commercial cloud provider. We present a prototype implementation of CLOUDTRACKER for the Amazon AWS commercial cloud and the StochSS stochastic simulation system. Using this prototype, we analyze the storage-versus-compute cost tradeoffs for different classes of StochSS simulations when deployed and executed in AWS.

Keywords: Cloud Computing, Provenance, Cost Estimation, Simulation, Replay.

1 Introduction

Easy and inexpensive access to vast compute and storage resources, in the form of cloud computing, along with the availability of prodigious amounts of digital information (financial, scientific, social) gathered via the Internet, have fueled the trend toward data-centric commercial application development. Data products originate from a variety of sources including mobile applications, streaming media, social networking, and large-scale analytics. Such data products in many cases require significant computational power for their generation and processing as well as substantial storage capacity for their preservation and collaborative sharing.

Scientific computing in general, and scientific simulation in particular, share many of these characteristics, but as yet, have failed to leverage the technological advantages offered by cloud computing. One reason for this lack of uptake is that the cost models associated with scientific computation are different than those associated with commercial enterprises. In a scientific context, the longevity of data (for the purposes of peer-actuated verification) is theoretically indefinite as is the need for reproducibility.

To aid enterprises with cost control, public cloud vendors make compute and storage resources separately available on an on-going, pay-per-use, rental basis. This "pay-as-you-go" model makes them attractive to businesses that experience transient fluctuations in computational needs, but can create infeasible long-term cost obligations for the storage of scientific results.

In this work, we investigate CLOUDTRACKER – a system for implementing the reproduction of scientific simulations in commercial cloud systems for the purposes of reproducibility and cost optimization. In many simulation contexts, the code that implements a simulation is significantly smaller than the results it produces. CLOUDTRACKER records the computational *provenance* (in a compact form) associated with a simulation so that it may be replayed exactly in a commercial cloud. Thus a scientist can choose either to store the results or to rerun the simulation when the results are needed – which ever yields the best cost-benefit relationship based on cloud storage and compute pricing. CLOUDTRACKER also facilitates data sharing (and verification) by making it possible for those, other than the progenitor, to regenerate a data set thereby aiding scientific reproducibility.

CLOUDTRACKER implements both automated provenance tracking and replay for applications that

- produce outputs deterministically from their inputs,
- use a "job manager" to automate application execution in a distributed setting, and
- can run in a cloud environment consisting of virtualized commodity and storage resources and yield the same numerical results.

Because clouds must automate the process of deploying an application (typically via a set of web services), it is possible to capture exactly the environment in which an application is executed in a cloud, including all of the operating system code, environment variables, and support library dependencies that are used. Moreover, commercial clouds operate at sufficient customer scale to make it difficult or impossible to deprecate "old" software. Operating system virtualization preserves the longevity of old releases so that they may be reconstituted years after they are first used under the same automated control. This longevity is essential for commercial adoption of clouds by enterprises as software lifecycle is a key cost control business parameter. CLOUDTRACKER leverages these properties to ensure that the results of a deterministic computation can either be stored or reproduced on demand at a later time.

In this way, CLOUDTRACKER complements previous work on data provenance for cloud systems [14, 1, 22, 9, 18]. In particular, CLOUDTRACKER records the minimal amount of meta-information necessary to re-execute an application (a.k.a a job) but it does not store the output data produced by the job itself. It captures this meta-information by observing the launch sequence of commands made by a job manager (that is responsible for running the job) and by interrogating the cloud platform for cloud configuration information associated with the job.

The work described in this paper focuses on a prototype implementation of CLOUDTRACKER for the Amazon Web Services (AWS) commercial cloud and the StochSS [20] cloud-based job manager for stochastic simulation. CLOUDTRACKER integrates with AWS and StochSS via a well-defined API that we believe generalizes to other systems with similar application properties. It also uses the Amazon Simple Storage Service (S3) – Amazon's inexpensive and most scalable storage service – to store the meta-data associated with each job run. Finally, CLOUDTRACKER exports a graphical interface (GUI) for scientists to use to track provenance, to initiate job replay, and to extract the actionable cost analysis that CLOUDTRACKER generates.

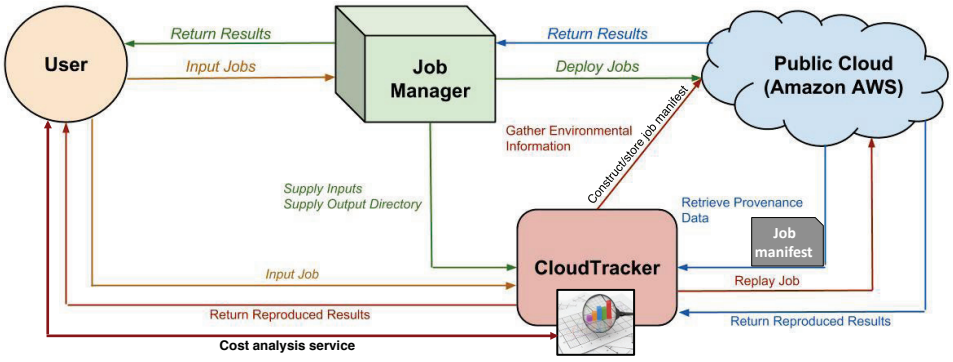

Fig. 1. CLOUDTRACKER System Overview. CLOUDTRACKER operates transparently alongside a job manager and with a public cloud, recording information about the execution environment of programs (jobs) into manifest files stored securely in cloud storage. At a later date, a user can use the CLOUDTRACKER web interface to specify a job to be replayed or to analyze the cost of job execution (production of the results) and result storage.

Our results indicate that at a sufficiently large scale (jobs that take multiple hours or days to complete and generate many gigabytes or terabytes worth of data), there are significant cost differences between storage and computation. For a given job, there is a specific point in time when storage becomes more expensive than computation. CLOUDTRACKER is able to present this information to users graphically. CLOUDTRACKER's reproduction engine is also able to replay a job on a wide range of AWS virtual machine configurations (each having a different cost) to allow the user to identify the most time and cost efficient combination. This information is also reported to users through the CLOUDTRACKER Web UI to give users insight into the cost of using different cloud resources.

In the sections that follow, we overview CLOUDTRACKER and describe its prototype implementation. We next present StochSS and show how we augment its job manager to support CLOUDTRACKER. We then evaluate CLOUDTRACKER for StochSS and present empirical results on using the system to trade off the costs of computation and storage. We follow this with a discussion of the limitations of the CLOUDTRACKER prototype, related work, and our conclusions.

2 CloudTracker

CLOUDTRACKER is an extensible software platform for tracking information about the execution of programs that are deployed using cloud infrastructures, such that they can be reproduced (i.e. replayed) at some later time. CLOUDTRACKER uses this tracking information to automate replay and to help users identify the best cost trade off between re-computing their data sets and storing them using cloud resources.

CLOUDTRACKER is designed to operate alongside an arbitrary, cloud-aware job management technology (e.g. tasking systems [7, 20, 12], batching systems [13, 21],

custom scripts, and others) as depicted in Figure 1. CLOUDTRACKER is not an interception architecture, requiring jobs to pass through before execution. Rather, it provides jobs managers with an API and library that they can use to request that CLOUDTRACKER record the execution provenance of a job. CLOUDTRACKER receives and records a unique identifier for each job to which it maps information about the execution environment of the job. Together, we refer to the map and provenance information that CLOUDTRACKER collects, stores, and manages, a *job manifest*. CLOUDTRACKER stores the manifest with read-only access rights in a cloud datastore that is universally accessible, persistent, and immutable.

CLOUDTRACKER also provides a web service with which users can replay a previously tracked execution by selecting the appropriate job ID and specifying their cloud credentials. The service retrieves the provenance information that it has associated with the job ID and uses it to replicate the state under which the original job ran. CLOUDTRACKER sets up a virtual environment in the cloud to replay the job using the original inputs and returns the results back to the user. CLOUDTRACKER can also store the results temporarily in cloud storage or forward them directly the requester. Users can then compare the results against those reported by the progenitor of the job.

The CLOUDTRACKER service also provides users with information about the cost of a job. This cost analysis service reports the cost for storing the results of the job in the persistent object store of the cloud (e.g. the Simple Storage Service (S3) in AWS or Cloud Storage in the Google Cloud Platform). The service reports the cost of the original execution on an hourly and per-minute basis and can replay the job to determine the cost under different scenarios (e.g. using different instance sizes). Finally, the cost analysis service estimates how long the results can be stored before the cost of doing so is outweighed by that of replaying the job and reproducing the job immediately prior to the time at which the results are to be used or shared.

2.1 CLOUDTRACKER Implementation

CLOUDTRACKER is written in the Python programming language so that it is portable and facilitates easy integration with popular public and private cloud APIs (AWS and Eucalyptus [8]), via the publicly available boto toolkit [6]. CLOUDTRACKER exports an application programming interface (API) with which job execution systems (or individual users and scientists) can interface to provide tracking, reproduction, and cost analysis support for their cloud-based applications. We overview this API in Section 2.3.

The CLOUDTRACKER platform is a distributed web application that can be integrated into any cloud infrastructure or platform-as-a-service (PaaS). The CLOUDTRACKER backend provides provenance tracking (the API implementation); the frontend provides users with access to job replay and cost analysis. The backend captures and records job information provided by authenticated job managers and metadata gathered from the cloud environment. CLOUDTRACKER manages this information in the form of a compact manifest which it stores in persistent, cloud storage using the credentials of the CLOUDTRACKER administrator. CLOUDTRACKER makes the recorded information (execution provenance) publicly readable but not modifiable to ensure trustworthy reproduction of results and datasets.

The frontend provides a web-based user interface (UI) for interacting with the CLOUDTRACKER platform. Users authenticate themselves and supply their cloud credentials (under which an application's execution is to be replayed) via the CLOUDTRACKER UI. They then select a job for which they want cost estimates or to reproduce output data. CLOUDTRACKER uses the securely stored information associated with the job ID to perform the service for the user and to return the result to the user via a web browser.

2.2 Leveraging Amazon Web Service Interfaces

The public cloud we use for our CLOUDTRACKER prototype is Amazon Web Services (AWS) [3] given its current position as the market leader in cloud infrastructure use [5]. We also selected it because the Eucalyptus private cloud infrastructure [15] is API-compatible with AWS and thus CLOUDTRACKER users can run CLOUDTRACKER on local cluster resources or in the public cloud without modification.

We leverage a number of recent advances in public cloud APIs from AWS to implement the CLOUDTRACKER platform. In our scenario, job managers use the Elastic Compute Cloud (EC2) service, and deploy jobs to EC2 instances. These are virtual machines, running Linux operating systems. Different machine types are available, offering different amounts of computational resources, each with their own usage price. Once configured with all of the necessary application software, libraries, and filesystem data, a snapshot of the instance can be saved as an immutable, reusable image called an Amazon Machine Image (AMI). Typically, job managers use the ID of a given AMI to launch EC2 instances to run applications in the cloud. CLOUDTRACKER records this AMI so that it can be used for the replay of a job. AMIs contain most of the relevant aspects of a job's execution environment and as such, CLOUDTRACKER need only store a small amount of information to capture this information.

CLOUDTRACKER stores job provenance information in AWS S3 using the AWS S3 manifest APIs [4]. Data in S3 are called objects and objects are grouped together into buckets. With S3, access permissions can be set at both the object and bucket level, giving users full control over who can view and edit their data.

2.3 Job Tracking

To perform job tracking, a job manager interoperates with the CLOUDTRACKER platform using CLOUDTRACKER API that we provide as an open source Python library. We summarize the API operations in Table 1. To initiate tracking for a new job, the job manager instantiates a new CLOUDTRACKER object. This operation sets up a communication channel with the name service and authenticates the job manager. The job manager provides CLOUDTRACKER with a unique identifier for the jobs; CLOUDTRACKER augments this with a unique identifier for the job manager (so that multiple concurrent job managers can be supported at once). CLOUDTRACKER refers to this unique identifier as the job's UUID and uses it to allocate a bucket in cloud storage and to create a manifest file in that bucket to hold the job's provenance data. CLOUDTRACKER then queries the EC2 metadata service to query details about the job's EC2 instance (given

API	Semantics
CloudTracker(uuid,instance_id)	Instantiates new CloudTracker instance Creates new directory in CloudTracker S3 bucket from the UUID Collects instance metadata and writes it to a manifest file
track_input(exec_string)	Writes executable name and input parameters to manifest Discovers input files and uploads them alongside manifest
track_output(output_dir)	Writes location of output directory to manifest
exec_state(time,size)	Computes execution time of job and size of output data Writes execution time and data set size to manifest

Table 1. CLOUDTRACKER API

the ID passed in from the job manager) including AMI-ID and instance type. CLOUDTRACKER writes all provenance data as key-value pairs in the manifest file.

To record the inputs to a job, the job manager uses the track_input() function to supply CLOUDTRACKER with the same execution string that the job manager uses to execute a job in an EC2 instance. This execution string contains the name of the executable and all of the command-line input parameters. Input files must be provided with a full file path in order to assure uniqueness and make them easier to locate. CLOUDTRACKER checks the filesystem for the existence of each input parameter, and those that do exist are assumed to be input files. Upon discovery, these files are copied and uploaded to the S3 bucket alongside the manifest. All of the other inputs are written to the manifest file itself.

Finally, the job manager uses the track_output() function to tell CLOUDTRACKER the location of the output directory it uses for job results (other than standard output and standard error, if any). This output directory location is also written to the manifest.

CLOUDTRACKER also requires that the job manager record metadata about the execution of the job so that it can use it to estimate future costs. To do so, the job manager times the execution of jobs and computes the size of the job output. CLOUDTRACKER records this information in the manifest file of the job. This interaction relies on the job manager returning the correct values to CLOUDTRACKER; if the job manager cannot be trusted, CLOUDTRACKER can instead execute the job itself using its reproduction engine. If the job manager participates in this way, it reduces the cost of reproduction and cost analysis for future users of the job and data.

2.4 Job Reproduction

With all of the provenance data previously recorded, the reproduction process is straightforward. The only input CLOUDTRACKER requires from the user is the UUID of a job to be replayed and the user's public cloud credentials. CLOUDTRACKER uses UUID to access the appropriate S3 bucket to obtain the manifest file and required input files.

CLOUDTRACKER launches an EC2 instance using the AMI ID and instance type using the user's cloud credentials. CLOUDTRACKER downloads the input files from S3 and stores them at the specified file paths. CLOUDTRACKER then reconstitutes the execution string and invokes the job. Finally, CLOUDTRACKER collects standard output,

standard error, and any results in the output directory and returns them to the user via links to S3 storage on a web page. CLOUDTRACKER also provides clean up utilities to garbage collect results no longer of interest and to reuse or terminate instances.

2.5 Cost Analysis

The other key component of the CLOUDTRACKER platform is the cost analysis service. The CLOUDTRACKER frontend allows users to specify or select the ID of a job for which they want to perform analyses. The platform retrieves the execution time and data set size in the job's manifest, and using pricing information provided by the cloud provider, estimates the cost of running that job and storing its data products in the cloud. The CLOUDTRACKER presents a graph that identifies the crossover point for computation and storage costs at which recomputing the result set via replay is less then continued storage. The analysis accounts for the time to produce the results themselves.

If the performance and storage data is not stored in the job manifest for the for one or more instance types, CLOUDTRACKER uses its reproduction engine to execute the jobs for the types of interest. Each instance type provides differing computational power at different costs to the user. Such runs are needed because execution time and program behavior are specific to both the application and instance type. Using this utility, CLOUDTRACKER automatically replays a job on one or more instance types in parallel. The default instance types in our EC2 prototype include seven different instance types ranging from t1.micro to c3.2xlarge. CLOUDTRACKER computes the cost of using each instance type given the execution profile of the job running within each. When the analysis is complete, CLOUDTRACKER presents the instance type that saves the most time and the most money (if different) to the user, and records this information in the job manifest for future use. CLOUDTRACKER uses reproduction if the time and size estimates for a job are not available (e.g. if not trusted, or not yet reproduced) or if requested by the user via the UI.

3 Prototype

To prototype CLOUDTRACKER, we target StochSS, a software platform that provides job management for simulations of discrete stochastic models [20]. Such models are useful for describing biological systems on the molecular scale (for which the number of species copies is small) and simulating their behavior accurately. StochSS "service-izes" the monte carlo based simulation engines that underly these technologies, including StochKit2 [17], ordinary differential equations, and others, by wrapping them with a web UI with which users can parameterize and customize their models, and a backend that deploys simulations on different deployment targets, including Amazon AWS.

We extend StochSS with the CLOUDTRACKER library that the job manager makes use of upon job deployment. The public StochSS VM image contains the simulation applications and their software dependencies. Upon job submission, StochSS uses the CLOUDTRACKER API to contact CLOUDTRACKER and request tracking of the job (i.e. instantiation, `track_input`, `track_output`, and `exec_state`). CLOUDTRACKER retrieves the AMI for the VM instance from the AWS metadata service and

the job's ID for each unique job deployed. CLOUDTRACKER appends the ID to a unique instance ID for the StochSS instance (so that it is able to correctly handle multiple StochSS job managers concurrently). Although the simulations are stochastic, StochSS is able to extract a known seed which it includes as part of the input string for each job. Once this communication completes, users are able to use the CLOUDTRACKER UI to replay or to analyze the cost of any previously executed StochSS job.

4 Empirical Evaluation

We next evaluate the overheads and efficacy of CLOUDTRACKER. We overview our experimental setting and then describe our results.

4.1 Methodology

We empirically evaluate CLOUDTRACKER using StochSS version 1.2 and t1.micro instances in AWS for provenance tracking. The simulation engine that we use in our experiments is an ordinary differential equation with sensitivity analysis on four parameters for a dimer decay model. This job requires one input XML file and eight input parameters.

We consider two simulation job *sizes*: large and small. The large job runs for approximately 500 million steps (10 measurements per unit) and generates 1TB of output data. The small job runs for 10,000 steps which generates 22MB of output data. The instance types that we use for CLOUDTRACKER cost analysis include t1.micro, m1.small, m3.medium, m3.large, c3.large, and c3.2xlarge.

4.2 Empirical Evaluation

We first evaluate the overhead of CLOUDTRACKER. Figure 2 shows the manifest file for an arbitrary StochSS job in its entirety. This example is 354 bytes in size and contains all of the necessary information to reproduce the results of its representative job. The only additional information needed is the one input file, the XML model file, which can vary in size based on the model parameters. The XML file for the dimer decay model is 2.47KB.

Of course the number of inputs and total input size (files include) is program specific and varies widely. However, CLOUDTRACKER does not add any additional storage overhead to inputs and is able to compress inputs to reduce this size if needed. Since CLOUDTRACKER uses S3 to store this information, the cost of S3 storage is on $0.03 per gigabyte per month. The CLOUDTRACKER prototype currently uses fewer than ten S3 PUT operations which cost $0.005 per 1,000 requests. To retrieve provenance information, the CLOUDTRACKER prototype currently uses fewer than 10 S3 GET requests which cost $0.005 per 1,000 requests. In summary, CLOUDTRACKER is able to collect, store, and access execution provenance data at very low cost.

Similarly, CLOUDTRACKER does not introduce overhead for replay beyond that required for original job execution. The process of replaying a job requires CLOUDTRACKER to launch a single EC2 instance which takes approximately the same amount of time as during the original run (on the order of a few seconds).

```
instance_type : c3.2xlarge
region : us-east-1
ami_id : ami-fcc53494
executable : /home/ubuntu/ode/stochkit_ode.py
--force :
-m : /mnt/input/9bcb7dbe-36b3-4b06-b680-b59cfc3a5658.xml
-i : 100000
--sensi :
-t : 10000.0
--parameters : c4 c2 c3 c1
--out-dir :  /mnt/output/result
exec_time : 0.999206
output_dir : /mnt/output/result
output_size : 23700332
```

Fig. 2. Example StochSS job manifest file - 354 bytes with a single input file.

4.3 Cost Analysis

We next use CLOUDTRACKER to perform a cost analysis using the large simulation job. Using the AWS m1.small instance type, job execution (and thus replay) takes 4.49 days; for m3.medium it requires 1.18 days; and for c3.large it requires 0.53 days. Figure 3 shows three graphs that exhibit the trade-off of storage and computational costs as presented by CLOUDTRACKER. The graphs assume that the original data set is produced on Day 0.

In all three graphs, the cost of storing the 1TB output is the same and increases over time (as determined by the AWS S3 pricing model). The computation cost data shows the constant cost to produce the output data once; each graph shows the cost for each of the three instance sizes we consider. In each graph, CLOUDTRACKER also plots the `Days to Recompute` to show the amount of time required to recompute the data via replay (relative to Day 0) if the data sets were deleted on Day 0.

For each instance type, there is a cross-over point for computation and storage. If the data set is not used prior to this point, it is less costly to recompute the data. If the data is used prior to this point, it is more cost efficient to store the data. In either case, scientists and collaborators must also account for the time required to recompute the data to ensure that they have the data ready when it is needed, if they choose not to store it and instead compute it on-demand. Via its UI, CLOUDTRACKER presents these plots to the user or reports the cross-over point and replay time so that users obtain actionable insights from the tool that help them optimize the cost of cloud use.

CLOUDTRACKER also reports cost anomalies to users through its UI as part of its replay and cost analysis service. Cost anomalies are non-intuitive cost trends that may not be obvious to users. An example of a cost anomaly is when an instance price is higher than another but a computation that uses it results in a lower overall cost. Such an anomaly is depicted in the graphs in the figures. Each instance type is more expensive than the previous: m1.large costs $0.044 per hour, m3.medium costs $0.07 per hour, and c3.large costs $0.105 per hour. However, in each graph, the total cost of the computation is reduced. This is because the application is able to take advantage of the additional available resources in each case to compute in less time.

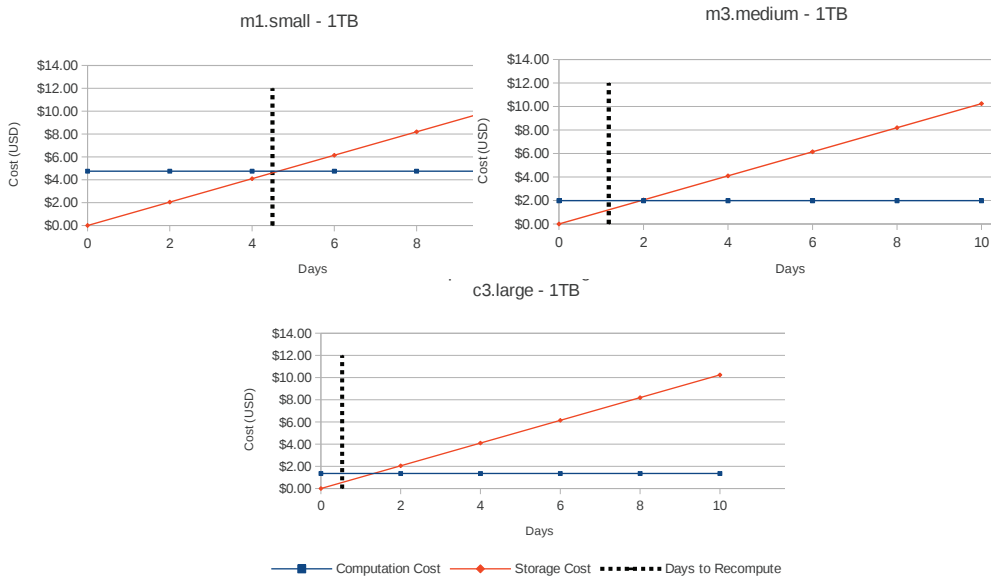

Fig. 3. Computation time versus storage cost for the large StochSS simulation job (1TB data set size) using m1.small, m3.medium, and c3.large instances sizes in AWS. Storage costs are the same for all three graphs because the data size is the same. The vertical dotted line represents the amount of time required to recompute the data set using CLOUDTRACKER replay on each machine.

Such anomalies are application specific and hard to determine ahead of time. The CLOUDTRACKER replay mechanism executes applications using different instances sizes to establish such ground truth for users. CLOUDTRACKER can be configured to run all or part of a program and to adjust its inputs accordingly to minimize the cost of performing replay across different instances sizes. Moreover, for collaborative systems, CLOUDTRACKER can cache the results across users to employ *crowd sourcing* to estimate these costs without direct replay. Finally, CLOUDTRACKER reuses instances across job types and job managers when it is able to do so (compatible AMIs) in an attempt to consume the full instance hour being charged for.

Figure 4 shows CLOUDTRACKER analysis for the small simulation job across all of the instance sizes CLOUDTRACKER currently considers as part of its replay. We also include a more detailed description of these results in Table 2; we consider average cost on a sub-hour basis in this table. The data shows that there is a significant time savings as we increase the size of the instance types (left to right). However, once the resources are fully utilized by the application, adding more compute resources will not decrease execution time and instead will only increase cost. For this job, CLOUDTRACKER identifies the c3.large as the best instance type which is over 8 times faster and provides a 37% reduction in cost over using t1.micro instances if the instances are used for a complete hour for this job type. CLOUDTRACKER is able to present this information to

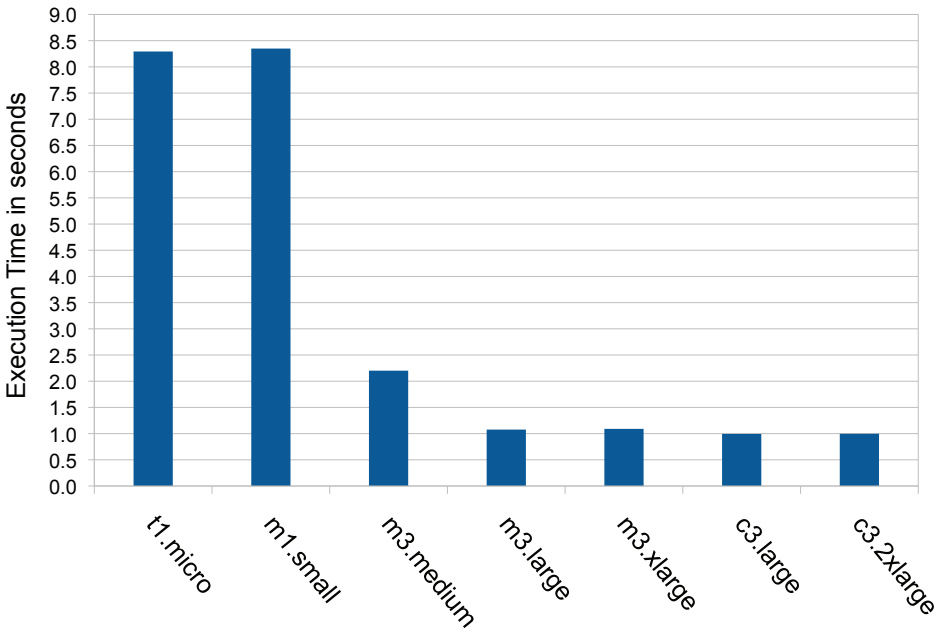

Fig. 4. CLOUDTRACKER execution time comparison for the small (22MB data set size) StochSS job, across the AWS instance sizes that the CLOUDTRACKER replay service considers.

users so that they can optimize their public cloud costs when computing, replaying, and storing large data sets.

5 Discussion

The CLOUDTRACKER design allows provenance tracking, job reproduction, and cost analysis to happen independently of and in concert with modern job management platforms used today to deploy arbitrary programs over cloud infrastructures. However, there are limitations with using CLOUDTRACKERthat we overview in this section and are considering as part of on-going work on this project.

First, CLOUDTRACKER supports only programs that can execute within a single virtual machine instance (i.e. distributed applications are not supported). Moreover, CLOUDTRACKER is unable to support applications for which the results generated depend on non-deterministic behavior (e.g. randomness, time-dependent inputs, external input from services not available during replay, etc.). With this work, we target scientific and data analytic programs for which the output (data product) is dependent only on program inputs (command line and files) and the VM execution environment.

Second, CLOUDTRACKER is able to interact with only those job managers capable of communicating job provenance data through the CLOUDTRACKER API. If, for example, the job manager is not capable of reporting the instance ID that is used for

	t1.micro	m1.small	m3.medium	c3.large	m3.large	m3.xlarge	c3.2xlarge
Time (sec)	8.29	8.35	2.20	0.99	1.08	1.09	1.00
Rate (USD/hr)	0.02	0.04	0.07	0.11	0.14	0.28	0.42
Avg Cost (USD)	$4.6x10^{-5}$	$1.0x10^{-4}$	$4.3x10^{-5}$	$2.9x10^{-5}$	$4.2x10^{-5}$	$8.5x10^{-5}$	$1.2x10^{-4}$
% Time improvement vs t1.micro	None	None	73.5%	88.0%	87.0%	86.9%	87.9%
% USD savings vs t1.micro	None	None	6.5%	37.0%	8.7%	None	None

Table 2. EC2 Instance Comparison for the small (22MB data set size) StochSS job.

job execution, CLOUDTRACKER will be unable to extract the execution provenance (e.g. AMI information from the cloud metadata server) necessary to replay the job or estimate cloud costs.

Third, CLOUDTRACKER use only makes sense for programs for which the cumulative size of the inputs is significantly smaller than that of the outputs. That is, there is only an opportunity to trade off computation for storage if the computation itself does not have significant storage requirements. For some cases in which the input size is significant but they are stored (which is paid for) by a third party (e.g. public data sets [19, 23, 2, 10, 11, 16] or a trusted collaborator), CLOUDTRACKER can be used to trade-off processing and storage costs.

Next, CLOUDTRACKER currently requires that users make their AMIs accessible to those with whom they wish to give reproducibility rights. That is, users can make their AMIs public for everyone to reproduce their results or to a subset of users. CLOUDTRACKER can facilitate this process through its web UI but the owner of the AMI is required to authenticate the change. Similarly, CLOUDTRACKER does not store the output data or the AMIs used for jobs under its credentials. It currently relies on the application progenitor or the job management system to do so. Such a model however requires that the AMI owners do not delete their AMIs. CLOUDTRACKER is able to make a copy of the AMI but will have to do so under the CLOUDTRACKER platform owner's credentials. As such, there is a question of who pays for and how to distribute cost across CLOUDTRACKER users. In our current prototype, we assume that AMIs are public and that they are not deleted (although CLOUDTRACKER reports any such issues to users upon reproduction/cost analysis).

6 Related Work

Although we are unaware of any work that employs execution provenance tracking to enable automatic and non-intrusive job replay and analysis of cloud cost trade-offs, other researchers have explored data provenance for cloud systems [14, 1, 22, 9, 18]. Muniswamy-Reddy, et al., claimed that provenance is a crucial feature currently missing from most cloud datastores, so they implemented a provenance protocol for cou-

pling data and provenance information together in cloud storage. In addition to verification, they identify faulty data propagation as another use case for data provenance, pointing to the fact that scientists wanting to build off of one anothers work have no means to verify that she is using data processed by the correct software [14].

Abaddi and Lyle, of the University of Oxford, move beyond the idea of data provenance for applications running in the cloud to discuss the need for provenance in clouds themselves. They show how provenance can be useful in the detection of bugs and security violations, and in the identification of their origins [1].

Finally, Zhang, et al., examine the different granularities of provenance required to describe data at different levels in the cloud computing model. These levels are the application, virtual machine, physical machine, cloud, and the Internet as whole. As the focus becomes broader, so does the type of provenance that is interesting to track [22]. While our work focuses on collecting provenance information at the application and virtual machine levels, there is definitely more information to be found at those higher levels.

7 Conclusions and Acknowledgments

The goal of our work is to investigate and develop new tools that bring the utility and potential of cloud computing to underserved communities such as those in the scientific computing community. A key reason behind the lack of uptake in cloud use by this community in particular is the difference in the cost models that underly scientific computation versus those for commercial enterprise applications. In particular, scientists require the ability to easily reproduce datasets (scientific results) for peer review, collaboration, and extension purposes and to make these datasets available to others for long periods of time.

In this paper, we investigate a tool called CLOUDTRACKER, that provides support for cost estimation for such data life cycles. To enable this, CLOUDTRACKER provides an easy to use cloud service (platform, client library, and UI) that extends the functionality of cloud-based job managers to facilitate automatic cost trade-off analysis between storing data and regenerating it on the fly. CLOUDTRACKER does so by tracking execution provenance so that it can reproduce the resulting data sets (for applications for which this is possible). With the ability to accurately reproduce datasets, CLOUDTRACKER is also able to estimate and report the best cost trade-off between storing data over time and reproducing it on-demand. We demonstrate the utility of CLOUDTRACKER for scientific simulations and find that it introduces negligible overhead (time and cost). We also find that for many applications there is a cross over point after which point it is more cost effective to regenerate the results rather than store them. In addition, CLOUDTRACKER is able to identify for users, opportunities for cost optimization across instance sizes offered by the cloud infrastructure.

We thank the reviewers for their valuable feedback on this paper. This work was funded in part by NSF (CNS-0905237 and CNS-1218808) and NIH (1R01EB014877-01).

References

1. Abbadi, I.M., Lyle, J.: Challenges for Provenance in Cloud Computing. In: USENIX Workshop on the Theory and Practice of Provenance (2011)
2. Amazon Public Datasets, https://aws.amazon.com/datasets [Online; acc 15-Jun-2014]
3. Amazon AWS, http://aws.amazon.com/ [Online; acc 15-Mar-2014]
4. Aws manifest file options – http://docs.aws.amazon.com/AWSImportExport/latest/DG/ManifestFileParameters.html
5. Aws market share – https://www.srgresearch.com/articles/amazon-continues-to-dominate-iaaspaas-despite-strong-push-from\-microsoft-ibm
6. Boto – http://boto.readthedocs.org/en/latest/
7. Celery, http://www.celeryproject.org/ [Online; acc 15-Mar-2014]
8. Eucalyptus – Open Source, AWS-Compatible Private Cloud Infrastructure, http://www.eucalyptus.com
9. Frew, J., Metzger, D., Slaughter, P.: Automatic capture and reconstruction of computational provenance. In: Concurrency and Computation: Practice and Experience (2008)
10. Google Public Datasets, https://www.google.com/publicdata/directory [Online; acc 15-Jun-2014]
11. HealthData.gov Public Datasets, http://healthdata.gov/dataset/search [Online; acc 15-Jun-2014]
12. Horuk, C., Douglas, G., Gupta, A., Krintz, C., Bales, B., Bellesia, G., Drawert, B., Wolski, R., Petzold, L., Hellander, A.: Automatic and Portable Cloud Deployment for Scientific Simulations. In: IEEE Conference on High Performance Computing and Simulation (HPCS) (2014)
13. Jette, M., Yoo, A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Job Scheduling Strategies for Parallel Processing (JSSPP) (2002)
14. Muniswamy-Reddy, K., Seltzer, M.: Provenance for the Cloud. In: USENIX Conference on File and Storage Technologies (2010)
15. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on. pp. 124–131. IEEE (2009)
16. ReadWriteWeb Open Data, http://readwrite.com/2008/04/09/where_to_find_open_data_on_the#awesm=~oHspy4ZUfG9lUr [Online; acc 15-Jun-2014]
17. Sanft, K., Wu, S., Roh, M., Fu, J., Lim, R.K., Petzold, L.: StochKit2: software for discrete stochastic simulation of biochemical systems with events. Bioinformatics 27(17), 2457–2458 (2011)
18. Simmhan, Y., Pale, B., Gannon, D.: A Survey of Data Provenance in e-Science. In: SIGMOD Record, 34(3) (2005)
19. Stanford Large Network Dataset Collection (SNAP), http://snap.stanford.edu/data/ [Online; acc 15-Jun-2014]
20. StochSS, http://www.stochss.org/ [Online; acc 20-Apr-2014]
21. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Practice and Experience 17(2-4) (2005)
22. Zhang, O., Kirchberg, M., Ko, R., Lee, B.: How to track your data: The case for cloud computing provenance. In: CloudCom (2011)
23. Zhao, B.: Social Network Datasets, http://current.cs.ucsb.edu/socialnets/#code [Online; acc 15-Jun-2014]