PROV-TE: A Provenance-Driven Diagnostic Framework for Task Eviction in Data Centers

Abdulaziz Albatli[1,2], David McKee[1], Paul Townend[1], Lydia Lau[1], Jie Xu[1]

[1] Distributed Systems and Services Research Group
School of Computing, University of Leeds, Leeds, UK
{sc11a2a, d.w.mckee, p.m.townend, l.m.s.lau, j.xu}@leeds.ac.uk

[2] Huraymila College of Science and Humanities
Computer Science Department, Shaqra University, Riyadh, Saudi Arabia
{albatlia}@su.edu.sa

Abstract—Cloud Computing allows users to control substantial computing power for complex data processing, generating huge and complex data. However, the virtual resources requested by users are rarely utilized to their full capacities. To mitigate this, providers often perform over-commitment to maximize profit, which can result in node overloading and consequent task eviction. This paper presents a novel framework that mines the huge and growing historical usage data generated by Cloud data centers to identify the causes of overloads. Provenance modelling is applied to add contextual meaning to the data, and the PROV-TE diagnostic framework provides algorithms to efficiently identify the causality of task eviction. Using simulation to reflect real world scenarios, our results demonstrate a precision and recall of the diagnostic algorithms of 83% and 90% respectively. This demonstrates a high level of accuracy of the identification of causes.

Keywords—Big Data; Data Centers; Cyberinfrastructure; Cloud Computing; Overcommitment; Overload; Provenance; PROV; Simulation; Distributed Systems;

I. Introduction

Infrastructure as a Service (IaaS) in Cloud Computing has introduced many new opportunities for businesses and individuals for extending accessibility and minimizing costs by providing users with access to remote resources [1]. However, as the Cloud Computing paradigm rapidly evolves, effective management of resource allocation to maintain a high level of overall system utilization becomes increasing important. Such management is typically addressed through the use of virtualization and over-commitment of resources to users; this requires data mining to be performed to quickly analyse the historical and current state of the relevant Cloud data centre. The cloud data centers that support IaaS process millions of tasks, and generate huge amounts of historical trace-log data. Mining this large, growing, and complex data is very challenging. This paper investigates the negative impact on users due to over-commitment resulting in task eviction, through a provenance-based big data analysis.

Cloud Computing is defined by the National Institute of Standards and Technology's (NIST) as “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [2]. It delivers virtualized, scalable and elastic resources (e.g. CPU, memory) over a network from data centers to enable users, including individuals, enterprises, and governments, to run complex operations requiring significant computational power.

Resource management in data centers is achieved through virtualization or containerization whereby tasks are executed within virtual machines (VM) which are scheduled on physical servers. In this way Cloud providers utilize over-commitment of physical resources in order to leverage unused capacity within their data centers and therefore maximize profits [3]. The over-commitment of resources often leads to an overload on the actual physical machines [3], which can lower the performance or lead to the failure of tasks due to lack of resources, i.e. CPU or RAM, and consequently lead to SLA violations. Over-commitment in IaaS is the practice of allocating more virtual resources on a physical machine than the actual physical capacities based on a predefined over-commitment ratio [4]–[6].

Currently, there are a number of different approaches to mitigate the overload, one of which is Task Eviction (TE) [7]. A provenance-driven diagnostic framework [8] has been developed, and is presented in this paper, using Google Cloud 29-Day dataset for learning [7], [9]. Its goal is to utilize provenance modelling to efficiently mine large-scale cloud trace-log data to identify the causes for task evictions. The framework extends the W3C PROV model [10] into PROV-TE which underpins a number of diagnostic algorithms for identifying evicted tasks due to specific causes. The records of an activity that led to a piece of data is the provenance of that of data [11]. Provenance describes the flow of data and processes across several heterogeneous layers and systems.

To verify the PROV-TE framework, a simulation of a generic over-committed data center - reflecting the configurations of Amazon EC2 and Google - is used. The Simulation EnvironmEnt Distributor (SEED) tool [12] has been used to systematically generate different Cloud datasets similar to real datasets, each with a different task eviction behavior. This generates large-scale simulated data consisting of several thousand servers.

This paper is organized in the following order. Section II presents the related work. Section III presents the development of the Provenance-Driven Diagnostic

Framework and its underpinning concepts and components. Section IV presents the process of using the simulation to generate test datasets that capture eviction behavior. Section V shows the framework's application and implementation on the simulated datasets. Section VI presents the analysis of the results and discussion on the framework's precision and recall and limitations. Finally, Section VII gives the overall conclusion and future work.

II. RELATED WORK

There have been attempts in solving the problem of resource over-commitment which is machines overload [3], [6], [7], [13]–[15] by introducing six different approaches (strategies) to mitigate the overload, which are Resource Stealing, VM Quiescing, Live Migration, Streaming Disks, Network Memory, and Task Eviction (TE). These studies look at mitigating the overload reactively but none have looked at approaches for solving the overload problem proactively. Overloads cause a deteriorating effect on the performance and availability of cloud services. Even though 88% of memory overloads are transient and last for less than 2 minutes [13], it is considered a massive drawback and can still violate the Service Level Agreements (SLAs) and Quality of Service (QoS), for which providers need to compensate the client.

Because data in cloud datacenters is widely used and shared, provenance plays an important role for both providers and users to audit the validity [16]. Provenance describes the flow of data and processes across several heterogeneous layers and systems. The process that led to a piece of data is the provenance of that data [10]. In [17], the authors have categorized the challenges of adopting provenance in cloud contexts into two categories. First, provenance-known issues such as object identification. Second, cloud related issues such as scalability, performance, and availability. For example, clouds are dynamically scalable, thus capturing and defining a provenance service is a complex task. A number of studies have considered using provenance in the clouds for different purposes [18], [19], [16], [20]–[23]. However, there were no attempts of using PROV model, which is a World Wide Web Consortium (W3C) standard that enables the exchange of the provenance information [24], for provenance but instead these studies have developed bespoke models as the PROV model was still being developed. Using standard models can help work undertaken by both research and industry communities to be easily understood and extended by building on them.

PROV model has recently started to gain attention in the cloud computing community. In [25], researchers applied PROV model in the cloud for security and trustworthiness purposes. One algorithm has been developed based on PROV model for controlling access to cloud data. It ensures the completeness of the causal dependencies between the data. Another study used PROV model as a basis for a provenance framework for gathering and storing cloud workflow provenance data for later analysis [26]. Even though these studies are notable, their aim and objectives are different than the ones of this paper. They do not look into the overload problem of physical machines. Li and Boucelma [27] used the open provenance model (OPM) and Colored Petri Net (CPN) for monitoring workflow and data provenance in the cloud. Their approach is similar to the approach conducted in paper. Also, they have used the simulation tool CPNTools to act as the diagnoser for their analysis. CPN used as the abstract model underpinning the diagnosis component which identifies the correct and faulty behaviors of the workflow, starting from the symptoms (faulty data or activities), and backward detecting the possible causes of the symptoms.

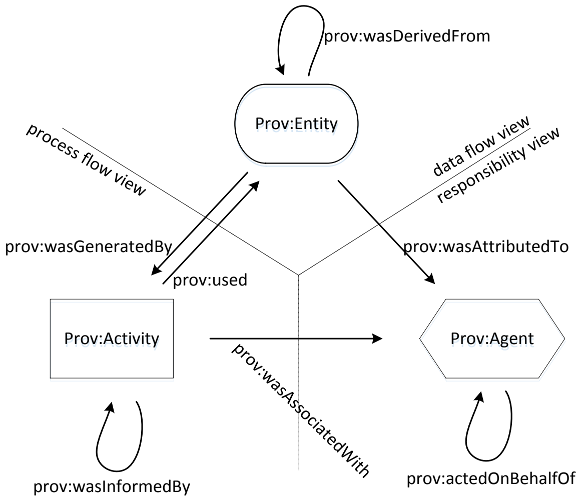


Figure 1. PROV Abstract Model [10]

III. PROVENANCE-DRIVEN DIAGNOSTIC FRAMEWORK

The purpose of this framework is for the identification of causes of task evictions from the log data of a cloud service provider. The process will help understand the components contributed to the overload, hence assist in prevention for the future.

A. Underpinning Concepts of the Framework

The reason for using provenance is because, first, it provides traceability of results. Second, reproducibility is ensured. Third, the schema facilitates the integration of diverse data sources. Analysis of provenance information of a given task would pave the way to extract knowledge from usage data that was not identified using the standard logging system.

PROV is W3C standard for provenance. As defined by W3C, "provenance is a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing" [28]. With regards to distributed systems, Moreau and Groth in [28] stated that provenance can relate to data, documents or resources since it is a record that computers have produced, processed, and exchanged. In addition, provenance is one essential dimension of process verification, reproducibility, reliability and trust in distributed systems [17]. PROV, shown in figure 1, is a model represents all types of tangible and untangle objects such as data, and allows the expression of links of causal relationships and dependencies between them through nodes and edges. The dependencies define the link between the effects and the cause in a backwards manner.

B. The PROV-TE Model

In order to extend PROV to address a specific overload problem – Task Eviction (TE), the PROV-TE model was

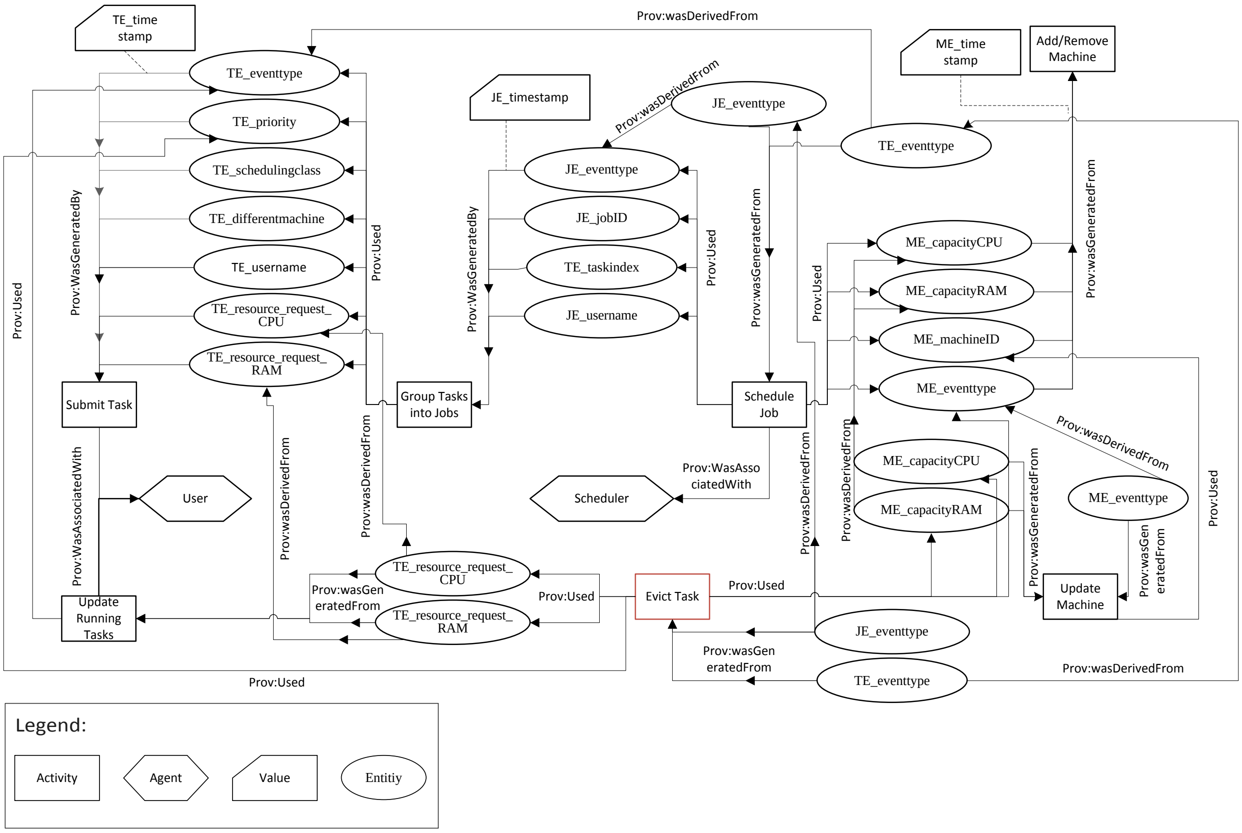


Figure 2. PROV-TE, a PROV Model for Task Eviction in Google Cloud.

developed and tested using the Google Cloud 29-Day Dataset [7], [29] for learning and exploration, Figure 2.

In PROV-TE, nodes can be one of the following: Entity: a digital, conceptual or physical thing of which we need to keep the provenance, such as TE_Priority; Activity: a process that occurs over a duration of time that act upon entities, such as Evict Task; and Agent: something/someone to which entities and activities are attributed or associated, such as Scheduler.

The relationships provided by the PROV model (i.e. the edges) are retained. These edges represent the dependencies between the nodes; for instance, prov:WasGeneratedBy, prov:WasDerivedFrom, prov:WasAssociatedWith, and prov:WasAttributedTo.

PROV-TE, shown in figure 2, is the second version of the extension [8]. The only difference between the two versions is that a correction and more entities were included in the second version. To illustrate, Agent: User is linked to Activity: UpdateRunningTasks and entities TE_username, TE_differentmachine, TE_schedulingclass were included. PROV-TE model guides the investigations into job/task behavior leading to the causes for TE as stated by Google [7]. These causes are: Higher priority tasks take over the lower priority ones, Increase in request of resources per task, Actual demand exceeds the machine capacity, Decrease in machine physical capacity, and Missing machines (failure or offline). PROV-TE underpin a number of diagnostic algorithms specific for each cause.

Algorithm 1a: Cause 1 Priority Identifier. Finding the priority of evicted tasks and isolating the tasks in a separate table called PriorityofEvictedTasks (PET) table.

```
FOR each task in Sc1Dataset table, until end of period
    IF Status = Killed
        STORE priority, timestamp of every distinct
        task in PET table.
    END IF
END FOR
```

C. Diagnostic Algorithms

Each cause is investigated by a number of diagnostic algorithms that are implemented using SQLite. For the scope of this paper, only three causes have been examined.

1) Arrival of Higher Priority Task

One of the causes of task eviction is due to higher priority tasks taking over the space of the lower priority ones. This trigger is due to the VMs' limited resources.

Algorithm 1b: Cause 1 Eviction Identifier. Identifying the number of evicted tasks from PET table within one-step interval from higher priority tasks being scheduled in the same VM.

```
FOR each task in PET (E), until end of period
    FOR each task in Sc1Dataset (S) table, until end of
    period
        IF (S.timestamp < E.timestamp <=
        (S.timestamp+ next time interval))
        AND (E.priority < S_priority)
        AND (E_machineID = S_machineID)
            STORE distinct E.Task in
            Cause1EvictedTasks (C1ET) table
    END FOR
END FOR
```

Algorithm 2a: Cause 2 Request Comparer. Comparing the resources' request of both CPU and MEM at the task's scheduling time with the new resources' request at running time, then identify the tasks with the increased update of resources' request and isolate them in Updated table (UT).

```
FOR each task in Sc2dataset table, until end of period
    IF (Status = scheduled (S)
    AND updated_while_running (U) = true)
    AND (TE_resource_request_CPU of U >
    TE_resource_request_CPU of S)
    OR (TE_resource_request_Mem of U >
    TE_resource_request_Mem of S))
        THEN STORE Task_timestamp, Task ID,
        machineID in UT
    END IF
END FOR
```

Algorithm 2b: Cause 2 Eviction Identifier. Looking within the lowest granularity interval of the trace, one-step interval, from the time of the task resources' request update in Updated Table (UT) to identify the tasks that have been evicted due to the increase in the update.

```
FOR each task in UT table, until end of period
    FOR each task in Sc2dataset table (ST) with an
    increase to their resources' request, until end of
    period
        IF (ST.Status = evict)
        AND (Task_timestamp (UT) < Task_timestamp
        (ST) <= (Task_timestamp (UT) + next time
        interval))
        AND Task priority (UT) > Task priority (ST)
        AND Task ID (ST) NOT IN C1ET table
            THEN display ST.Task_timestamp,
            ST.Task ID
        END IF
    END FOR
END FOR
```

Two Algorithms have been used to investigate this scenario. First, all evicted tasks in the log is captured and their priorities are ordered and stored (Algorithm 1a). The aim is to precisely identify the tasks the have been evicted only by Higher Priority Tasks being scheduled in the same Host (VM) and within one interval of higher priority task arrival timestamp (Algorithm 1b).

2) Increase in Resource Request

Another cause of task eviction is when users ask for more resources than they have initially requested while their tasks are running. Each task is scheduled in a specific VM with specific virtual resources (assigned resources according to their request). In case of over-commitment, when users request more resources, the scheduler neither can allocate more resources nor find an available virtual machine. A physical machine with fixed resource capacity would no longer be capable of hosting those tasks because the sum of the tasks' virtual resources' request is higher than the actual machine's capacity. So, lower priority tasks get evicted to avoid an overload in the machine (see Algorithms 2a-b).

3) Physical Overload

Algorithm 3a: Cause 3 Overload Calculator. Comparing the total physical capacities with the resources usage. Once the usage reaches threshold (80%), store physical machine ID with timestamp of overload in Overloaded Table (OT).

```
FOR each physical machine in Sc3dataset table, until
end of period
        Find total CPU/RAM usage in every interval
        IF CPU/RAM usage >  threshold level
            THEN Store PM ID, timestamp in
            Overloaded table (OT)
        END IF
END FOR
```

Algorithm 3b: Cause 3 Eviction Identifier. Per every overloaded physical machine in OT, find all evicted tasks within one interval of overload in the same machine.

```
FOR each physical machine in OT, until end of period
    FOR each task in Sc3dataset table (ST) hosted in
    an overloaded a physical machine that is in
    Overloaded table, until end of period
        IF (ST.Status = evict)
        AND (PM_timestamp < Task_timestamp  <=
        (PM_timestamp + next time interval))
        AND Task ID NOT IN C1ET table
            THEN display ST.Task_timestamp,
            ST.Task ID
        END IF
    END FOR
END FOR
```

Resources over-commitment causes overload [13], [3]. Providers set a usage threshold level where once it has been reached, an overload mitigating strategy, i.e. Task Eviction, is then triggered [13] (see Algorithms 3a-b).

Following is an illustration of how the PROV-TE, Figure 2, model can be used to trace the workflow of a task eviction due to the need to schedule a task with a higher priority. Normally, a user submits a task and specifies its scheduling priority. After a task is submitted (Activity: Submit Task), a number of Entities are generated, i.e. TE event type, TE priority, TE resource CPU/RAM, and all have a time stamp. Those entities are used by the Activity (Group Tasks into Job). Then a number of Entities are generated according to the grouping activity, i.e. JE jobID, JE event type, JE job name, TE task index, and a time stamp is recorded. The Activity: Schedule Job will use those entities and other entities related to the designated Machine, i.e. ME_MachineID, ME_eventtype, ME_capacity CPU/RAM) so that the task/job can be scheduled and hosted. One of the causes of TE is the submission of a new task with a higher priority, Algorithms 1a-b, to a VM that lacks resources, so the Evict Task Activity will react accordingly and processes the eviction of a task with the lowest priority (i.e. Entity: TE priority).

D. Instantiation of the Framework – The Auditor

The auditor, shown in Figure 3, consists of three components: Mapper, Database, and Query Handler. The Mapper takes the raw data from the log data as input and maps it to the PROV-TE model structure which then stored in the database. The Query Handler is the implementation of the diagnostic algorithms discussed earlier. It gets the structured dataset from the database as input and then runs

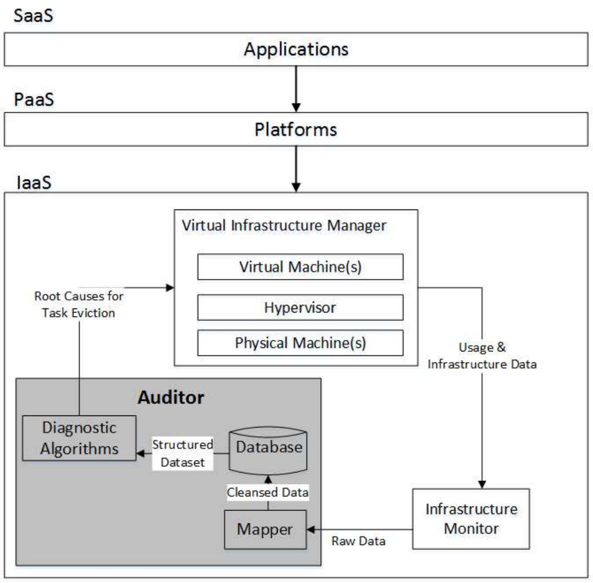


Figure 3. PROV-TE system model comprised of an Auditor, Infrastructure Manager, and Infrastructure Monitor

the algorithms using SQLite then informs the Virtual Infrastructure Manager with the causes of TE.

The potential use of the proposed framework is that following the process of framework development, the other five mitigating strategies could be modelled based on PROV and the relevant diagnostic algorithms could be developed. As a result, each mitigating strategy could have its own Auditor; e.g. Auditor for Live Migration causes, Auditor for VM Quiescing causes and so on.

IV. Using Simulation for Framework Evaluation

Both the model and the algorithms have gone through two iterations of development using Google 29-day dataset. Evicted tasks have been identified as well as the relevant causes based on metrics such as timestamp and shared physical machine. Due to the limited access to real Cloud datasets and in order to evaluate and assess the diagnostic framework, a simulation tool, SEED [12], has been used to generate test datasets according to three known Task Eviction behaviors.

Simulation in computer science domain is a vital systematic method for auditing and validating complex behaviors. There are a number of simulation tools that could have been used, such as CloudSim, GreenCloud, and MDCSim [30]–[32]. However they struggle to handle large-scale systems and require understanding of both the model domain as well as aspects relating to simulation synchronization. SEED facilitates the modelling of the domain based on graph notation and was designed specifically for modelling large-scale data centers with minimal user intervention and assumptions [12].

A. Purpose

The hypothesis of the research is that provenance adds value to the raw data by connecting the data in a way that provides additional meaning for further interpretation and analysis. Specifically, the analysis will provide the reasons and causes of an overload.

TABLE I. Design of Scenarios

	Scenario 1	Scenario 2	Scenario 3
Cause (C1)	•	•	•
Cause (C2)		•	
Cause (C3)			•

The aim of this evaluation is to test and evaluate the reasoning power of the proposed diagnostic algorithms and the underpinning PROV-TE model for the different overload scenarios. The simulation tool has been set up with a general data center configuration and has been used to generate 15 different datasets. Each data set comes with a log which includes details of the physical and virtual machines such as Host ID and CPU/MEM units, and tasks such as requested units of CPU/MEM and priority. Most importantly, it includes details about eviction of tasks such as relevant eviction cause and timestamp. These details will be used to validate the results of our framework by calculating the precision and recall of every algorithm.

B. Scope of Evalution

We focus on the following causes (behaviors):

Cause (C1). Arrival of higher priority tasks - higher priority tasks will always be scheduled no matter of how full the machines are; hence in a full capacity situation, lower priority tasks will be automatically evicted to make space.

Cause (C2). Request for increasing resources for current tasks - the machine has a fixed capacity, so at a specific point the lower priority tasks will be evicted to make space for the new requests of the higher priority tasks.

Cause (C3). The actual demand exceeds the machine capacity - once the maximum physical usage reached the threshold of the machine's capacity, the scheduler will automatically evict low priority tasks to avoid memory halts or breaching SLA's agreement, i.e. performance metrics.

These three causes will cover a reasonable range of typical patterns of behavior in resource management at the IaaS level of Cloud Computing.

C. Design of Evaluation

Three scenarios have been developed in SEED. Scenario 1 includes the behavior of C1. Scenario 2 includes the behavior of C1 and C2. Lastly, Scenario 3 includes the behavior of C1 and C3. Each scenario is run 5 times, resulting in a dataset similar to the log data from a data center. Having a dataset with more than one cause helps validate the accuracy of the diagnostic algorithms.

D. General Setup for the Simulation

The simulation environment has been configured to reflect a general data center setup and is shown in Figure 4. For every run, the tool creates 20 physical machines (PM) and 40 virtual machines (VM). Each PM has two VMs (1:2). The PMs' CPU and RAM sizes are fixed at 8 units and 15 GB, respectively. VM sizes are chosen randomly from a specified size list. Number of VM CPU can be 2, 4, or 8 units. VM RAM size can be 4, 6, or 8 GB. The sizes

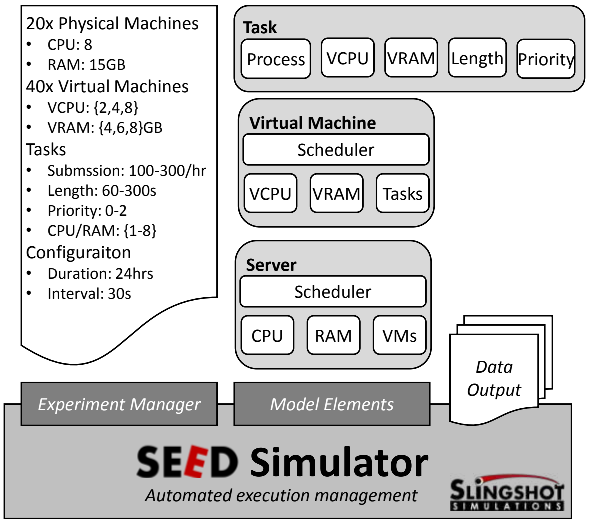


Figure 4. Configuration of the simulation environment using the SEED simulator

are a reflection of Amazon EC2 c3.2xlarge instance [33]. This particular instance allows over-commitment of resources. This hybrid configuration ensures that simulated data is similar to real data. The scale of the simulation can be generalized to larger environments with more PMs and VMs, generating huge volumes of data. This aims to demonstrate the feasibility of massive-scale simulation for implementing provenance-based techniques.

Tasks are then generated according to a random task submission rate (TSR). TSR is randomly chosen from 100-300 per hour. The simulation length is 24 hours. The method of task distribution is: send one task to one VM at a time, in equal distribution, then loop back again until all tasks are sent to be queued in every VM.

There are 4 variables assigned to each task. Firstly, a task's length is measured in steps. The task length is randomly chosen from 2 to 10. The length of the task is the number of steps needed to finish execution. Events are logged in a one-step interval. A step is a predefined interval of 30 seconds. Second, a priority is randomly assigned to each task. It is a number to define the privilege of a task - 0 (lowest), 1 and 2 (highest). Finally, requested resources, CPU and RAM, are assigned to the tasks. The resources are also chosen randomly from a predefined list (1, 2, 3, 4, 5, 6, 7, 8).

For every VM, there are three queues for tasks to be scheduled, once for each priority which ensures that every task get its fair time of waiting in the queue. In the scheduling method, there is a loop that goes around the three queues and dequeues one task from each queue at a time to be scheduled.

E. Scenario Specific Configration

Each scenario has additional configuration in order to generate the needed behavior in the dataset. Three algorithms are developed to mimic the behavior led by the three causes, namely – Task Evictor, Request Handler and Overload Manager. A brief description of these is detailed below.

1) Arrival of Higher Priority Tasks

For every VM, whenever there is a lack of RAM or CPU and there is a task waiting in the queue with higher priority than the ones running, lower priority tasks get evicted so the VM to be ready for the next task.

TABLE II. OUTPUT OF SIMULATION

	Scenario 1		Scenario 2			Scenario 3		
	Total Tasks	**Total Evicted Tasks**	**Total Tasks**	**Total Evicted Tasks**		**Total Tasks**	**Total Evicted Tasks**	
Run 1	4208	259	3735	C1	C2	3131	C1	C3
				904	20		326	54
Run 2	4530	266	4076	C1	C2	2341	C1	C3
				967	20		16	187
Run 3	4501	421	4328	C1	C2	2687	C1	C3
				1041	45		1	176
Run 4	4653	297	4035	C1	C2	3077	C1	C3
				1012	37		177	76
Run 5	4538	319	4049	C1	C2	2596	C1	C3
				994	43		96	157

Algorithm 4: Task Evictor: Whenever a task is to be scheduled, the scheduler has been configured to first check the available VM capacity, CPU and RAM. If there is enough space, then the task gets scheduled. Otherwise, if there lower priority tasks running in the VM, a list is then created to include all lower priority tasks ordered ascendingly by priority. From the top of the list, tasks get evicted until enough space becomes available. Then the task in question gets scheduled. In case there are no lower priority tasks, the to-be-scheduled task is to wait in the queue until free space becomes available.

2) Increase in Resourse Request

Scheduling a task on a specific VM depends on the task's requested capacities, in terms of CPU and RAM. Once hosted, a task can request a change in the requested resources. In case there is no free resource to accommodate this request and there are lower priority tasks on the same VM, the task eviction mechanism will be executed until the desired requested capacities become available.

Algorithm 5: Request Handler: While a task is running, the simulation tool has been configured to generte a new random request from a predefined list (1, 2, 3, 4, 5, 6, 7, 8). The request will get approved only if there is available space in the VM. Otherwise, the same lower priority task eviction method of Algorithm 4: Task Evictor is run until the requested space becomes available.

3) Physical Machine Overload

Over-commitment is a policy that is widely adopted in data centers to maximize resources' usage. Physical and virtual usage are managed by overload threshold levels [34]. The simulation tool has been configured with an 80% physical threshold level. Once physical usage exceeds it, eviction process is executed. Unlike the other scenarios, two policies are enforced when evicting tasks, lower priority task first and Last-In-First-Out (LIFO). It is more sensible to evict tasks that have just started than those near to finish.

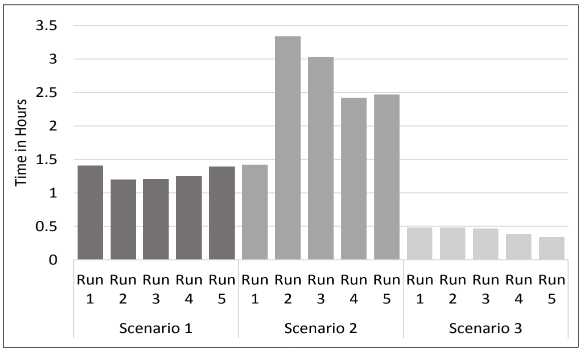


Figure 5. Execution time for every simulation run.

TABLE III. MEAN AND STANDARD DEVIATION OF EXECUTION TIMES

	Scenario 1	**Scenario 2**	**Scenario 3**
Mean	1.292	2.53	0.43
Standard Deviation	± 0.1	± 0.73	± 0.065

Algorithm 6: Overload Manager: For every physical machine, the total physical usage is calculated every one-step interval (30 seconds). If the total physical usage exceeds the predifented usage thresold limit, tasks get evicted following the same lower priority task eviction method of Algorithm 4: Task Evictor until normal usage behavior is restored. The list is ordered ascending by tasks priority and descending by time of hosting in the VM. The normal usage behavior means the total physical usage is less than the usage thresold limit.

F. Simulation Output

To show the randomness, variances and differences of the resulting generated behavior, each scenario has 5 simulation runs (i.e. 15 simulated datasets in total).

Table II summarizes the overall output of the 15 runs of the three scenarios. For each run, the total number of tasks as well as the total number of evicted tasks are shown. Also, because Scenario 2 and 3 have two causes each, the total number of evicted tasks related to each cause is also shown.

The aim for table II is that it will be used after applying the algorithms of the framework to calculate the precision and recall of the results, which will be explained in the next section.

Figure 5 shows the execution time of every run for every scenario. Table III illustrates the mean of the execution time of the five runs for every scenario and the standard deviation. Due to the small number of tasks of Scenario 3 which depends on the random TSR, the mean of Scenario 3 execution times is smaller compared to the means of Scenarios 1 and 2. Also, Scenario 2 execution times are relatively higher is because of the complexity of Algorithm: Request Handler. While tasks are running, their resources request are constantly and randomly changed. In order for every request to be approved or not, capacities comparison is undertaken. Every request in RAM or CPU must be less or equal the size of the VM. Also, if there is no VM space and there are lower priority tasks, the eviction process is triggered. Only then the request is approved.

TABLE IV. OUTPUT OF FRAMEWORK

	Scenario 1		**Scenario 2**			**Scenario 3**		
	Total Tasks Found	**Evicted Tasks Found**	**Total Tasks Found**	**Evicted Tasks Found**		**Total Tasks Found**	**Evicted Tasks Found**	
Run 1	4208	259	3735	C1	C2	3131	C1	C3
				900	13		307	58
Run 2	4530	266	4076	C1	C2	2341	C1	C3
				960	15		91	60
Run 3	4501	421	4328	C1	C2	2687	C1	C3
				1048	20		75	55
Run 4	4653	297	4035	C1	C2	3077	C1	C3
				1012	22		182	36
Run 5	4538	319	4049	C1	C2	2596	C1	C3
				997	23		159	55

V. TESTING THE DIAGNOSTIC FRAMEWORK

The following shows the results of the implementation of the framework, the Auditor component shown figure 3. For scenario 1, the auditor will only trigger C1 related algorithms. For scenario 2, the auditor will trigger C1 and C2 related algorithms. For scenario 3, the auditor will trigger C1 and C3 related algorithms, illustrated in table I.

The input to the Auditor is the 15 simulated test datasets. Then, the diagnostic algorithms are applied to find the causes of all eviction. The output of the Auditor is the identification of causes and relevant evicted tasks.

A. Output from the Auditor

Table IV summarizes the output of all diagnostic algorithms. In section VI, precision and recall statistical measures will be applied to evaluate the results.

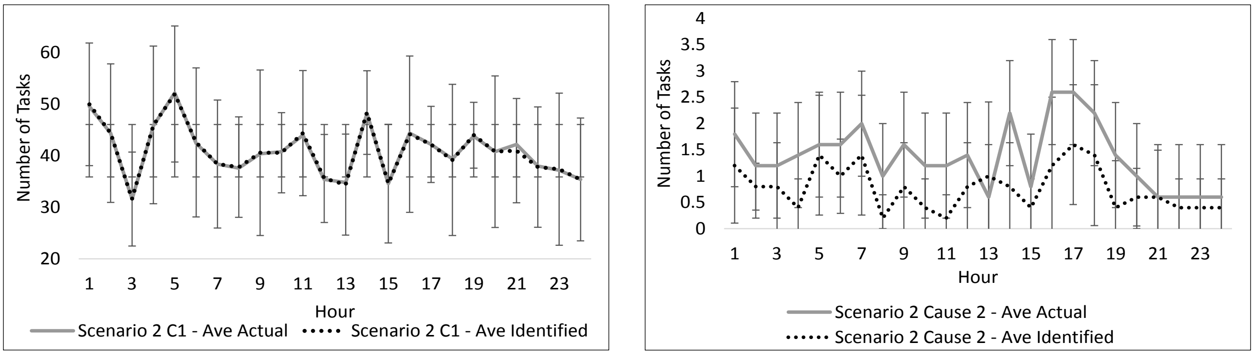
VI. ANALYSIS AND DISCUSSION

The simulation facilitated the generation of 15 cloud test datasets that captured specific behaviors for task eviction. The developed diagnostic algorithms make use of PROV-TE. This has proved to be helpful by both the ability of auditing the datasets and identifying evicted tasks and also distinguishing the relevant causes.

In order to evaluate the accuracy of the diagnostic algorithms, precision and recall statistical measures have been applied. Precision is a measure of the capability of the framework to only identify the relevant evicted task for each cause. It is a statistical measure of reliability of the framework. Recall is a measure of the capability of the framework to retrieve and identify the highest possible number of relevant evicted tasks for a specific cause. It is a statistical measure of the sensitivity of the framework.

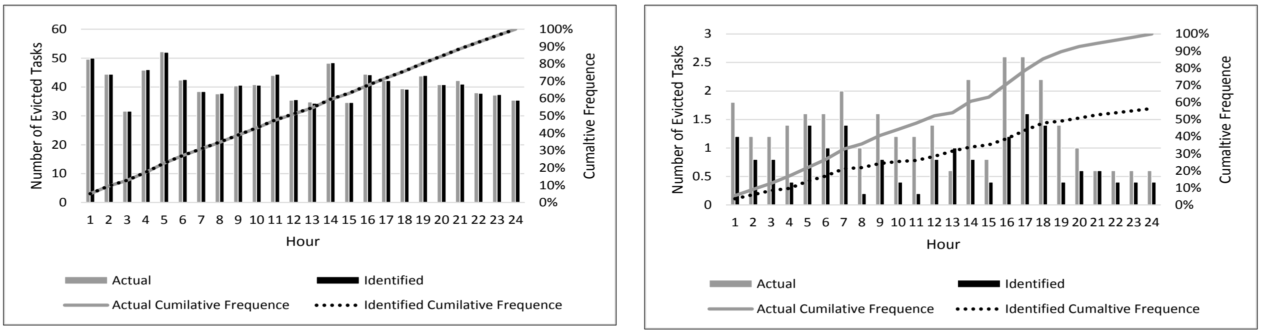
$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Positive(FP)}} \times 100 \qquad (1)$$

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Negetive (FN)}} \times 100 \qquad (2)$$



(a) Cause 1: Evictions due to higher priority tasks (b) Cause 2: Evictions due to increased resource request by higher priority tasks

Figure 6. Average of actual and identified evicted tasks per hour, showing the variance across all simulation runs.



(a) Cause 1 (b) Cause 2

Figure 7. Cumulative average task evictions over all simulations of Scenario 2.

TP, FP, and TN have been calculated by comparing the output of the simulation, Table II, with the output of the auditor, Table IV. For example, looking at the Tables II and IV, in Run 1 of Scenario 2, the Auditor has been able to identify all tasks, 3735, and also has been able to classify the evicted tasks based on the specific causes, C1 and C2. For C1, 900 (TP) evicted tasks out of 904 have been identified (FN = 4), so the precision is 100% and the recall is 99%. For C2, 13 evicted tasks out of 20 have been identified which makes precision 100% and recall 65%.

Since the output is overwhelming, the precision and recall have been calculated based on the mean TP, FP, and TN of the five runs for every cause in the three scenarios. The precision and recall of C1 related algorithms have been calculated in every scenario because all datasets capture C1 task eviction behavior whereas C2 task eviction behavior is captured in only Scenario 2 and C3 task eviction behavior is captured in only Scenario 3.

In table V, because the simulated datasets of scenario 1 have one behavior, the precision and recall are 100%. There is only one cause and the algorithms have identified all evicted tasks due to this cause, whereas the other scenarios have 2 behaviors so the algorithms precision and recall are not as high.

TABLE V. **SCENARIO 1:** MEAN PRECISION AND RECALL

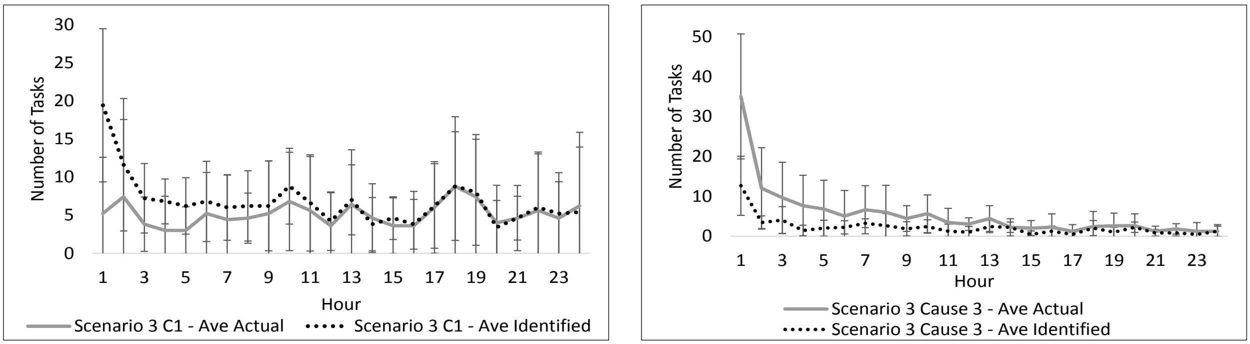
C1 Algorithms	**Relevant Tasks (Simulated)**	**Irrelevant Tasks (Simulated)**
Relevant Tasks (Framework)	TP = 312.4 Std Dev = ± 65	FP = 0 Std Dev = ± 0
Irrelevant Tasks (Framework)	FN = 0 Std Dev = ± 0	TN = 0 Std Dev = ± 0
Precision	100%	
Recall	100%	

Table VI summarizes the mean precision and recall of scenario 2 across all runs. It can be seen that the diagnostic algorithms of C1 are quite promising with 99% in both precision and recall, as seen in figures 6a and 7a. C2 diagnostic algorithms have returned precisely the relevant evicted tasks but failed to pick up 44% of the evicted tasks because of C2, as seen in figures 6b and 7b.

In table VII, C1 diagnostic algorithms of scenario 3 are able to identify relevant evicted tasks.

TABLE VI. **SCENARIO 2:** MEAN PRECISION AND RECALL

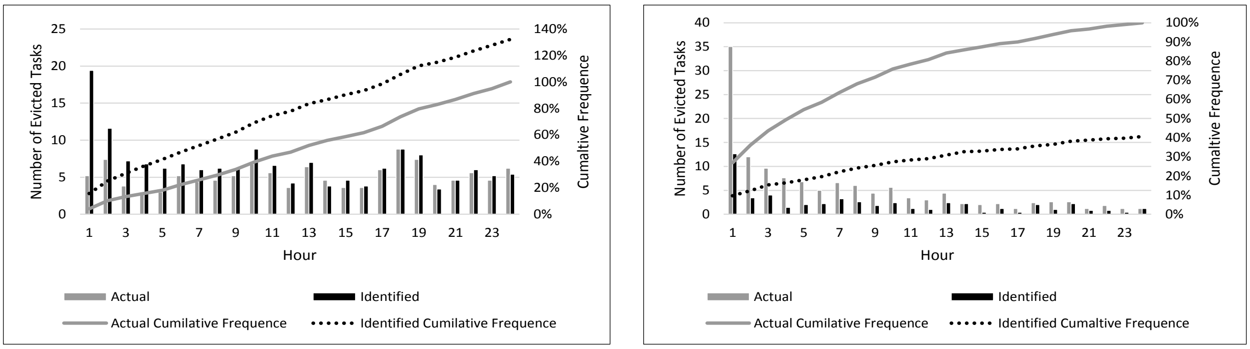
C1 Algorithms	**Relevant Tasks (Simulated)**	**Irrelevant Tasks (Simulated)**
Relevant Tasks (Framework)	TP = 981.4 Std Dev = ± 54	FP = 2 Std Dev = ± 3
Irrelevant Tasks (Framework)	FN = 2.2 Std Dev = ± 3.1	TN = 0 Std Dev = ± 0
Precision	99%	
Recall	99%	
C2 Algorithms	**Relevant Tasks (Simulated)**	**Irrelevant Tasks (Simulated)**
Relevant Tasks (Framework)	TP = 18.6 Std Dev = ± 4.3	FP = 0 Std Dev = ± 0
Irrelevant Tasks (Framework)	FN = 14.4 Std Dev = ± 8.4	TN = 0 Std Dev = ± 0
Precision	100%	
Recall	56%	



(a) Cause 1: Evictions due to higher priority tasks

(b) Cause 3: Evictions due to machine overload

Figure 8. Average of actual and identified evicted tasks per hour, showing the variance across all simulation runs.



(a) Cause 1

(b) Cause 3

Figure 9. Cumulative average task evictions over all simulations of Scenario 3

However, as shown in figures 8a and 9a there is probably an overlap in terms of the identified causes as 40% of which are irrelevant. The recall percentage of C3 diagnostic algorithms is high, 90%, which means it is capable of identifying the relevant evicted tasks. However, its precision measure is 40%, as seen in figure 9b. The precision and recall of C1 diagnostic algorithms are relatively high across all scenarios.

Figure 10 shows the power of C1 and C3 diagnostic algorithms combined. Almost 90% of all evicted tasks due to C1 and C3 have been identified. This could suggest that running a hybrid algorithm of two or more causes could return better results with higher precision and recall instead of auditing each cause separately.

TABLE VII. **SCENARIO 3:** MEAN PRECISION AND RECALL

C1 Algorithms	**Relevant Tasks (Simulated)**	**Irrelevant Tasks (Simulated)**
Relevant Tasks (Framework)	TP = 119.4 Std Dev = ± 126.2	FP = 43.4 Std Dev = ± 37.6
Irrelevant Tasks (Framework)	FN = 3.8 Std Dev = ± 8.4	TN = 0 Std Dev = ± 0
Precision	73%	
Recall	97%	
C3 Algorithms	**Relevant Tasks (Simulated)**	**Irrelevant Tasks (Simulated)**
Relevant Tasks (Framework)	TP = 52 Std Dev = ± 9.2	FP = 78 Std Dev = ± 55.5
Irrelevant Tasks (Framework)	FN = 0.8 Std Dev = ± 1.7	TN = 0 Std Dev = ± 0
Precision	40%	
Recall	98%	

For every run of every scenario, the Auditor can generate files for each cause which include the IDs of the evicted tasks and their physical and virtual host IDs which can be further investigated. Also, the Auditor can order the causes in terms of level of impact on the system. From tables IV, the most dominant cause is C1 which is the Arrival of Higher Priority Tasks and the least dominant cause is C2 which is Increase in Resource Request.

VII. CONCLUSION AND FUTURE WORK

In this paper we have discussed how cloud providers use resource over-commitment to leverage under-utilized capacity yet with a trade-off of introducing the problem of overload. A Provenance-Driven Diagnostic Framework has been presented and evaluated. The framework's aim is to efficiently mine big and growing data to find the causes of Task Eviction in a data center. Our main contribution of

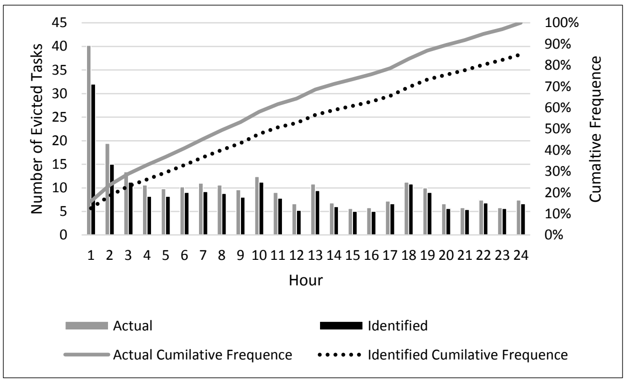


Figure 10. Cumulative average task evictions over all simulations of Scenario 3, combining C1 and C3.

this paper is the evaluation of the framework using a massive-scale simulation tool, SEED. SEED has been used to generate 15 different cloud test datasets with different task eviction behaviors. The Framework, PROV-TE and Diagnostic Algorithms, have been applied on these datasets and the found results have been compared with the simulation results. Finally, the results have been statistically analyzed using precision and recall measures to find the levels of sensitivity and reliability. The average precision and recall of the diagnostic algorithms are 83% and 90%, respectively. Although the diagnostic algorithms identify the causes of task eviction fairly precisely, there are limitations relating to the overlapping of identified causes for evicted tasks. This could explain the precision levels of Scenario 3 diagnostic algorithms.

For future work, we intend to run a more extensive simulation experiment with a larger environment. It will be of interest to apply the framework on a dataset that encompasses all task eviction behaviors. Also, this work is a first step for research that looks into mitigating TE in data centers, which is assumed to lead to a decrease in the number of overload instances. In addition, we will further improve the diagnostic algorithms and the approach of identifying Task Eviction root causes in order to increase the level of accuracy in terms of precision and recall.

ACKNOWLEDGMENT

The authors thank Slingshot Simulations Ltd. (http://www.slingshotsimulations.co.uk) for providing the use of the SEED simulator for this work.

REFERENCES

[1] Y. Amanatullah, C. Lim, H. P. Ipung, and A. Juliandri, “Toward cloud computing reference architecture: Cloud service management perspective,” in International Conference on ICT for Smart Society, 2013, pp. 1–4.

[2] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” 2011.

[3] S. Baset, L. Wang, and C. Tang, “Towards an Understanding of Oversubscription in Cloud,” in USENIX Hot-*ICE’12*, 2012.

[4] Y. Liu, “A Consolidation Strategy Supporting Resources Oversubscription in Cloud Computing,” in IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud), 2016, pp. 154–162.

[5] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, “Efficient datacenter resource utilization through cloud resource overcommitment,” in IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2015.

[6] T. Wo, Q. Sun, B. Li, and C. Hu, “Overbooking-Based Resource Allocation in Virtualized Data Center,” in 2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, 2012, pp. 142–149.

[7] C. Reiss, J. Wilkes, and J. Hellerstein, “Google cluster-usage traces : format + schema V2.1,” pp. 1–14, 2014.

[8] A. Albatli, L. Lau, and J. Xu, “Application of PROV Model for Modeling a VM Overload Mitigating Strategy: Task Eviction,” in Provenance Analytics Workshop, 2014.

[9] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale,” in Proceedings of the Third ACM Symposium on Cloud Computing, 2012, pp. 1–13.

[10] L. Moreau and P. Groth, “Provenance: An Introduction to PROV,” Synth. Lect. Semant. Web Theory Technol., vol. 3, no. 4, Sep. 2013.

[11] P. Townend, P. Groth, and J. Xu, “A provenance-aware weighted fault tolerance scheme for service-based applications,” in Proceedings - Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005, pp. 258–266.

[12] P. Garraghan, D. McKee, X. Ouyang, D. Webster, and J. Xu, “SEED: A Scalable Approach for Cyber-Physical System Simulation,” IEEE Trans. Serv. Comput., vol. 9, no. 2, 2016.

[13] D. Williams, H. Jamjoom, Y.-H. Liu, and H. Weatherspoon, “Overdriver: Handling Memory Overload in an Oversubscribed Cloud,” in Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2011.

[14] A. Stage and T. Setzer, “Network-aware migration control and scheduling of differentiated virtual machine workloads,” in ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009.

[15] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, “Dynamic load management of virtual machines in cloud architectures,” in Cloud Computing, vol. 34, Springer Berlin Heidelberg, 2010, pp. 201–214.

[16] K. Muniswamy-Reddy, P. Macko, and M. Seltzer, “Provenance for the Cloud,” in Proceedings of the 8th USENIX Conference on File and Storage Technologies, 2010.

[17] M. A. Sakka, B. Defude, and J. Tellez, Document provenance in the cloud: constraints and challenges, vol. 6164. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[18] I. Abbadi and J. Lyle, “Challenges for Provenance in Cloud Computing,” in 3rd USENIX Workshop on the Theory and Practice *of Provenance (TaPP’11)*, 2011.

[19] M. Imran and H. Hlavacs, “Provenance in the cloud: Why and how?,” Third Int. Conf. Cloud Comput. GRIDs, Virtualization, pp. 106–112, 2012.

[20] K.-K. Muniswamy-Reddy, M. Peter, and S. Margo, “Making a Cloud Provenance-Aware,” in 1st Workshop on the Theory and *Practice of Provenance (TaPP’09)*, 2009.

[21] S. M. S. Da Cruz, M. Manhaes, J. Zavaleta, and R. M. Costa, “Cirrus: Towards Business Provenance As-a-Service in the Cloud,” IEEE 19th Int. Conf. Web Serv., no. i, pp. 668–669, Jun. 2012.

[22] P. Macko, M. Chiarini, and M. Seltzer, “Collecting provenance via the Xen hypervisor,” in Proceedings of 3rd USENIX Workshop on *the Theory and Practice of Provenance (TaPP ’11)*, 2011.

[23] S. Townend, P; Venters, CC; Lau, L; Djemame, K; Dimitrova, V; Marshall, A; Xu, J; Dibsdale, C; Taylor, N; Austin, J; McAvoy, J; Fletcher, M; Hobson, “Trusted Digital Spaces through Timely Reliable and Personalised Provenance,” in 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, 2012, pp. 136–141.

[24] P. Groth and L. Moreau, “PROV-Overview: An Overview of the PROV Family of Documents,” 2013.

[25] J. Lacroix and O. Boucelma, “Trusting the cloud: A PROV + RBAC approach,” in IEEE International Conference on Cloud Computing, CLOUD, 2014, pp. 652–658.

[26] P. L. Rupasinghe, H. H. Weerasena, and I. Murray, “Trustworthy provenance framework for document workflow provenance,” in International Conference on Computational Techniques in Information and Communication Technologies, 2016, pp. 168–175.

[27] Y. Li and O. Boucelma, “Provenance Monitoring in the Cloud,” in IEEE Sixth International Conference on Cloud Computing, 2013.

[28] L. Moreau and P. Groth, “PROV-Overview: An Overview of the PROV Family of Documents,” W3C Note, no. April, pp. 1–9, 2013.

[29] I. Solis Moreno, P. Garraghan, P. Townend, and J. Xu, “Analysis, Modeling and Simulation of Workload Patterns in a Large-Scale Utility Cloud,” IEEE Trans. Cloud Comput., vol. PP, no. c, 2014.

[30] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim : A Toolkit for the Modeling and Simulation of Cloud Resource Management and Application Provisioning Techniques,” Softw. Pract. Exp., vol. 41, no. 1, pp. 23–50, 2011.

[31] S. Malekzai, D. Yildiz, and S. Karagol, “GreenCloud simulation QoSbox in cloud computing,” in 24th Signal Processing and Communication Application Conference (SIU), 2016.

[32] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, “MDCSim: A multi-tier data center simulation, platform,” in 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–9.

[33] Amazon Web Services Inc, “EC2 Instance Types – Amazon Web Services (AWS),” 2016. [Online]. Available: https://aws.amazon.com/ec2/instance-types/. [Accessed: 24-Oct-2016].

[34] R. Birke and L. Y. Chen, “Managing Data Center Tickets : Prediction and Active Sizing,” 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, Jun. 2016.