Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

HARDWARE-IN-THE-LOOP SIMULATION FOR AUTOMATED BENCHMARKING OF CLOUD INFRASTRUCTURES

Qi Liu
Marcio A. Silva
Michael R. Hines
Dilma Da Silva

IBM T. J. Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY 10598, USA

ABSTRACT

To address the challenge of automated performance benchmarking in virtualized cloud infrastructures, an extensible and adaptable framework called CloudBench has been developed to conduct scalable, controllable, and repeatable experiments in such environments. This paper presents the hardware-in-the-loop simulation technique used in CloudBench, which integrates an efficient discrete-event simulation with the cloud infrastructure under test in a closed feedback control loop. The technique supports the decomposition of complex resource usage patterns and provides a mechanism for statistically multiplexing application requests of varied characteristics to generate realistic and emergent behavior. It also exploits parallelism at multiple levels to improve simulation efficiency, while maintaining temporal and causal relationships with proper synchronization. Our experiments demonstrate that the proposed technique can synthesize complex resource usage behavior for effective cloud performance benchmarking.

1 INTRODUCTION

Cloud computing has emerged as an attractive paradigm for on-demand provisioning of computational resources to support a wide spectrum of applications (Armbrust et al. 2009). Virtualization is a key technology used in cloud-enabled data centers for elastic scaling, high availability, and functional isolation between applications consolidated on a shared physical platform (Vaquero et al. 2009). Three primary cloud service models are commonly adopted, namely *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS). Among them, IaaS clouds, such as Amazon EC2 (Amazon 2012) and IBM SmartCloud (IBM 2012a), allocate hardware resources in units of virtual machines (VMs), giving customers the illusion of having their own dedicated servers while allowing providers to improve infrastructure efficiency at reduced cost. However, the intrinsic interplay between volatile resource demand, changing application behavior, and adaptive resource management strategies poses a significant challenge to evaluating system and application performance (Iyer at al. 2009; Jayasinghe et al. 2011). Such evaluation is crucial not only for cloud providers who attempt to optimize resource utilization in the absence of application specific information, but also for cloud customers who are interested in comparing the service offerings from different providers (Li et al. 2010).

One approach to addressing this challenge is the use of simplified analytical models to estimate the average performance measures based on the steady-state arrival rate of requests and their resource requirements (Chen et al. 2007; Jung et al. 2008; Ghosh et al. 2010). Nevertheless, using analytical models is often inadequate to study transient system behavior under unexpected conditions. Another approach is through experimentation in real clouds, leading to the development of a few cloud benchmarking tools

(Yigitbasi et al. 2009; Ye et al. 2010; Cooper et al. 2010). Despite their success, most of these tools lack the capability of adaptive experimental control and statistical resource demand aggregation for complex testing scenarios. In contrast to direct experimentation, full system simulation has also been employed to evaluate performance in simulated clouds, which can be controlled precisely to reproduce test results (Calheiros et al. 2011; Sriram and Cliff 2010; Ostermann, Plankensteiner, and Prodan 2011; Nunez et al. 2011). Whereas this approach is useful when the cloud infrastructure is not readily available, simulating the cloud service architecture at full scale is an enormous (if not infeasible) undertaking, requiring difficult tradeoffs between precision and speed to obtain results in a reasonable time.

Hardware-in-the-loop (HIL) simulation is a methodology for hybrid system synthesis where selected hardware and software components are immersed in a closed-loop virtual simulation environment (Ledin 1999). It provides a middle ground between physical prototyping and virtual simulation, combining the advantages of both approaches. Using the HIL simulation methodology, CloudBench is built as an extensible and adaptable framework at IBM Research for automated, scalable, controllable, and repeatable benchmarking of cloud infrastructures. This paper presents the HIL simulation technique that underlies the CloudBench capabilities. Specifically, the technique integrates an efficient discrete-event simulation with real cloud infrastructures in a unified testing framework. A generic method is proposed to facilitate the modeling of highly dynamic resource usage behavior. The requests from different types of applications are multiplexed statistically to emulate realistic resource demand fluctuation in IaaS clouds. The technique also exploits the parallelism between and within application instances to improve execution efficiency and experiment scalability, while preserving temporal and causal relationships with proper synchronization. In addition, a closed feedback loop is used to enhance experimental controllability and adaptability. Our preliminary results show that the technique can effectively assist cloud performance benchmarking by synthesizing complex resource usage patterns from straightforward configurations.

In the rest of the paper, Section 2 reviews related work. Section 3 introduces the CloudBench framework. The HIL simulation technique is covered in Section 4, and the experimental results are discussed in Section 5. Section 6 concludes the paper with suggested future research directions.

2 RELATED WORK

To evaluate performance in IaaS clouds, different analytical methods have been attempted, such as queueing models (Chen et al. 2007), Markov chains (Ghosh et al. 2010), and layered queueing networks (Jung et al. 2008). Although these methods can be used to estimate the *average* performance and resource utilization, they are incapable of describing short-lived, transient behavior when system conditions change drastically (e.g., a sharp surge in application workloads or a sudden change in data access patterns). Moreover, a substantial level of expertise is required to construct and solve sophisticated analytical models (Woodside, Franks, and Petriu 2007), especially for large scale cloud infrastructures.

Workload characterization and synthesis play an important role in capacity planning and stress testing of virtualized data centers. For instance, a statistics-based approach was taken to generate workloads for MapReduce applications by sampling empirical distributions extracted from production traces (Chen et al. 2010; Ganapathi et al. 2010). In (Bodik et al. 2010), the authors analyzed workload and data spikes in real Web servers and used a closed-loop generator to synthesize such spiky loads. These studies offer useful insight that can help system optimization. However, many of them focus only on certain applications, without paying much attention to the volatility of resource usage as VMs come and go in IaaS clouds.

Direct experimentation in a real cloud environment (or a miniature prototype of it) is still the widely accepted method for accurate performance assessment, as exemplified in various lines of research (see, e.g., Jackson et al. 2010). As traditional benchmarks do not fit well the dynamic nature of elastic cloud services (Binnig et al. 2009), a few cloud benchmarking tools were developed to achieve automated experiment configuration, application workload generation, and on-line performance monitoring (Yigitbasi et al. 2009; Ye et al. 2010; Cooper et al. 2010). Yet some tools restrict themselves to a specific aspect of system performance, while others fall short of adaptive control and a reliable way of ensuring experiment repeatability. These limitations hinder their usability in real test cases.

Several full system simulators have been developed for cloud infrastructures, including CloudSim (Calheiros et al. 2011), SPECI (Sriram and Cliff 2010), GroudSim (Ostermann, Plankensteiner, and Prodan 2011), and iCanCloud (Nunez et al. 2011). Nonetheless, a cloud infrastructure typically includes multiple geographically distributed data centers connected via wide area networks. Each data center can have thousands of servers and storage systems networked together. This vast pool of physical resources is managed by layers of interacting software components. Modeling such a large-scale and diverse distributed system in detail is an exceedingly complex task, which is compounded further by the fact that the design and implementation of most commercial clouds are opaque to outside modelers, making it difficult to validate the models. Moreover, a full infrastructure simulation would suffer from an explosion of parameters (Paxson and Floyd 1997), requiring a systematic sensitivity analysis to avoid misleading results (Mills, Filliben, and Dabrowski 2011). To obtain performance data within a reasonable time, the simulation would need to trade precision for speed, even with advanced parallel and distributed simulation techniques. On the other hand, modeling and simulation (M&S) has proven to be an invaluable tool for studying various aspects in cloud computing, from horizontal scaling (Idziorek 2010) to resource scheduling (Assuncao, Costanzo, and Buyya 2009), just to name a few.

Traditionally used as a prototyping and validation technique for embedded systems, HIL simulation has evolved as a methodology for synergistic system integration and optimization (Schludermann, Kirchmair, and Vorderwinkler 2000; Papp, Dorrepaal, and Verburg 2003; Hosking and Sahin 2009). A HIL simulation is a *control system* that combines a physical system under test (SUT) and a virtual simulation environment within a bidirectional closed loop (Fathy et al. 2006). The simulated environment monitors the state of the SUT via sensor signals and injects synthetically generated actuator commands into the SUT to trigger operations at appropriate times (Ledin 1999). The use of synthetic command generation and closed-loop feedback allows for the automation of experiments in a controllable, adaptable, and repeatable way. In this paper, we apply the principles of HIL simulation to cloud performance benchmarking, using a discrete-event simulation (with plug-in statistical models) to drive experiments in the clouds.

3 THE CLOUDBENCH FRAMEWORK

This section briefly introduces the CloudBench framework, providing the necessary background for the proposed HIL simulation technique. From a high-level view as shown in Figure 1, CloudBench consists of two main components: a *front-end* and a *back-end*, which interact with each other in a closed feedback loop. The former is built around a discrete-event simulation engine that generates synthetic resource usage patterns, whereas the latter carries out application benchmarking in a chosen cloud infrastructure.

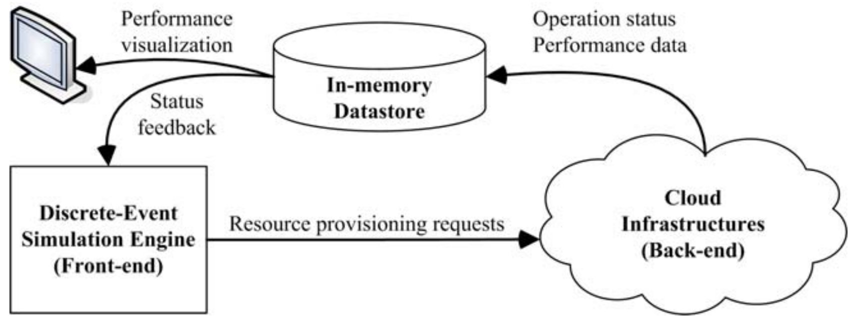
Performance visualization
Operation status Performance data
In-memory Datastore
Status feedback
Discrete-Event Simulation Engine (Front-end)
Resource provisioning requests
Cloud Infrastructures (Back-end)

Figure 1: CloudBench overview

The front-end models and simulates the resource usage behavior in a cloud by submitting a sequence of requests through the cloud's public interface. Each *request* is represented as an *event* composed of a set of ingredient *operations* for creating or destroying a specific type of *application instance* with one or more constituent VMs. For example, an instance of the DayTrader online stock trading benchmark application has three VMs (DayTrader 2012): a *workload driver* that mimics a variable number of clients by issuing concurrent stock purchase and sell orders to a WebSphere *application server* (IBM 2012b), which processes the client orders using a DB2 *database server* (IBM 2012c). Note that the requests from the front-end are actually served by *actuators* operating in the back-end; and the complete stack of application VMs is allocated in the cloud. This enables the front-end to execute in a cloud- and application-agnostic fashion, without being overwhelmed by the hardware and application configurations. Furthermore, as the clients of the application benchmarks are also hosted in the cloud, this approach enhances experiment scalability by taking advantage of the scalability of the cloud infrastructure itself.

In response to a request, the back-end actuators perform the VM provisioning operations. Once the VMs are up and running, the actuators trigger the necessary configuration operations, defined as scripts in the VM image, to establish the connection between the VMs based on the application topology. The actuators then start the application benchmark and monitor its performance in the cloud. At the end of each operation, the execution status is sent back to the front-end using a publish/subscribe mechanism through an in-memory datastore, such as Redis (Redis 2012). The status feedback is used by the front-end for a variety of adaptive experimental controls, as will be discussed in Section 4. In the meantime, application performance data (as well as system and guest VM performance measures, if available) are streamed into the datastore for visualization in real time.

CloudBench supports the definition of *resource pools*, each with one or more *VM containers*. A VM container can represent a single physical server or an EC2 availability zone with multiple data centers. Different placement strategies, such as round-robin and random, can be applied when submitting VM provisioning requests to the pools and containers. In this way, the benchmark application clients can run at remote locations different from that of the application servers, enabling truly distributed benchmarking experiments. Furthermore, it allows for the investigation of VM migration and load balancing algorithms across VM containers in a prototype cloud testbed. The framework already includes a collection of macro and micro application benchmarks, such as DayTrader (DayTrader 2012), LAMP (Linux/Apache/MySQL/PHP stack), Hadoop (Apache 2012), Windows desktop applications, CoreMark (EEMBC 2012), IOzone (Norcott and Capps 2006), and Netperf (Jones 2012). It also provides an interface to incorporate additional benchmarks. In the next section, we present the HIL simulation technique used in CloudBench.

4 HIL SIMULATION

4.1 Requirements

To drive large-scale cloud benchmarking experiments in real time, the front-end discrete-event simulation needs to fulfill the following requirements.

- **Synchronous advance of simulated time**. Unlike a typical discrete-event simulation wherein the (simulated) virtual time jumps immediately to the time stamp of the next event after processing the current event, the front-end must pace the advance of virtual time in synchrony with the advance of real wallclock time. This is required so that the resource usage patterns appear realistic to the physical cloud infrastructure under test.
- **Asynchronous event processing**. Provisioning a VM can take up to tens of minutes, while the execution of an application benchmark can last hours to days to measure performance variations under changing loads and across different time periods. Hence, the events should be processed asynchronously in a parallel and distributed manner to avoid unnecessary blocking.
- **Adaptive control**. Individual VMs may fail during their lifetime, which could render an application instance unusable. Moreover, a user may want to limit the number of active VMs to contain experiment cost or to avoid running out of available resources. Based on the feedback from the back-end, the front-end should be able to respond to such situations appropriately in order to, for example, control event dispatch rate or clean up a failed application instance.
- **Experiment replay**. It is often desirable to regenerate exactly the same resource usage behavior as observed in a previous experiment for evaluating alternative algorithms or comparing different cloud configurations. Therefore, the front-end should preserve the events (and their attributes) in an experiment to allow for deterministic replay later.

4.2 Architecture

Figure 2 gives an architectural view of the HIL simulation, showing the main logical processes (LPs) and the interactions between them. The front-end employs three types of LPs: a set of *event generators* (EGs), an *event dispatcher* (ED), and a set of *operation dispatchers* (ODs). On the other hand, the back-end defines one type of LPs (or *actuators*) for each type of benchmarking operation performed in a cloud infrastructure under test (e.g., create or destroy a VM; define, execute, or undefine an application, and so forth). While the ED runs continuously in an experiment, the other LPs can be spawned and ended dynamically as needed. Depending on the scalability requirement, the LPs can be implemented as physical processes, threads, or a mix of them, which can be hosted on one or more physical or virtual machines.

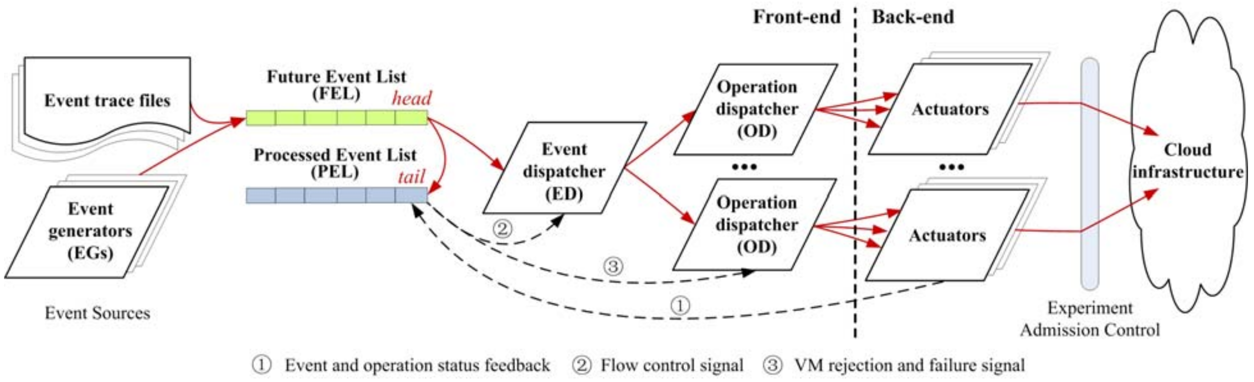

Figure 2: HIL simulation architecture

At the front-end, events are loaded from trace files (*static sources*), or generated by EGs (*dynamic sources*), or a combination of both. A static source may be derived from a cloud production trace with arbitrary empirical distributions (e.g., event inter-arrival time and lifetime), while a dynamic source generates events using predefined parametric distributions that model the dynamics of a certain type of application. The events within a source are kept in non-decreasing time stamp order, with a special stop event at the end of the stream. Multiple streams of events from different sources are multiplexed into a centrally managed future event list (FEL), which serves as a producer-consumer buffer between the EGs and the ED. This multi-source statistical event multiplexing offers two advantages. First, it allows for the decomposition of complex resource usage patterns, which would not be easily described using a monolithic approach, into more manageable and mathematically tractable components. Each component can be modeled and validated separately, reducing the input modeling effort and promoting the reuse of validated models. Secondly, the aggregation of individual components with varied characteristics facilitates the study of emergent overall behavior, bringing the experiment condition closer to the reality in IaaS clouds.

During an experiment, the ED retrieves events scheduled for the same time from the FEL and creates a set of ODs (one per event) to process them concurrently, exploiting *event-level parallelism* between simultaneous application requests. An OD in turn creates a set of actuators (one per operation) to execute the event's internal operations in parallel, exploring *operation-level parallelism* between independent activities while maintaining the correct ordering of causally related operations. All events executed are appended into the processed event list (PEL), which is dumped incrementally into a trace file that can be used for deterministic replay. The status feedback from the actuators is stored along with the events and operations in the PEL, allowing for adaptive experimental control as will be discussed in Section 4.5.

4.3 Event Generation

An EG generates events using statistical models that describe the resource usage characteristics of a specific type of application. Although out of the scope of this paper, these statistical models can be constructed by mining the data collected from a production cloud, which is an active research area (e.g., Ganapathi et al. 2010). Alternatively, they can be provided by users to create synthetic test conditions.

a. Creation Event Template

$\mathbf{T_a}$ @ event {<u>evAnno</u>}
0 @ create VM *\$name$_1$ \$type \$size \$container \$placement* {<u>opAnno</u>}
…
0 @ create VM *\$name$_N$ \$type \$size \$container \$placement* {<u>opAnno</u>}
1 @ define APP *\$appName \$appType \$name$_1$,…,name$_N$* {<u>opAnno</u>}
2 @ execute APP *\$appName \$loadLevel \$duration* {<u>opAnno</u>}

b. Destruction Event Template

$\mathbf{T_d}$ @ event {<u>evAnno</u>}
0 @ undefine APP *\$appName* {<u>opAnno</u>}
1 @ destroy VM *\$name$_1$* {<u>opAnno</u>}
…
1 @ destroy VM *\$name$_N$* {<u>opAnno</u>}

Figure 3: Event templates

As shown in Figure 3, an event has a *composite structure* with several internal operations needed to create or destroy an application instance in the cloud. For each application instance, a pair of *creation* and *destruction* events is generated using the event templates and subsequently inserted into the FEL. Based on given distributions, the event time stamps represent the instance arrival time (T_a) and departure time (T_d) respectively, relative to the experiment start time. The internal operations of an event are grouped by *sequence numbers* (shown as bold fields), which serve as a tie-breaking mechanism that indicates the causal dependency between groups of operations within that event. Operations with the same sequence number are independent and can be executed in parallel, whereas operations with a larger sequence number must wait until the preceding group is completed. Taken together, the time stamps and sequence numbers unambiguously determine the temporal and causal relations between the events and operations.

An operation is specified by a set of parameters (shown as italic fields), which are passed to back-end actuators. For example, a VM creation operation uses five parameters: name, type, size, container, and placement of the VM. An application instance is defined by its name and type, along with the names of its constituent VMs. The load level and duration parameters tell an actuator how to vary application workload periodically during the execution. The VM names assigned in a creation event are used in the corresponding destruction event to destroy the application instance properly. These template parameters are filled in by an EG using predefined values or random numbers sampled from statistical distributions. Besides, an event or operation is associated with an annotation (shown as underscored fields), which is a key-value map for storing the status feedback updated by the actuators.

Figure 4 depicts the stream of events generated by an EG. The lifespan of an EG is configured by setting its activation and end times (called *activation by schedule*). Alternatively, an EG can be spawned and terminated dynamically according to specific conditions such as when a certain level of resource overcommitment is reached in the cloud (called *activation by condition*). Instead of generating and inserting events into the FEL well in advance, the EGs do so *just in time* when the time stamp of a creation event becomes imminent, for several reasons. First, it allows the event insertions to be scattered naturally following the inter-arrival distributions, alleviating the contention when multiple EGs access the FEL at the same time. Secondly, the FEL is kept relatively short as events are extracted by the ED shortly after insertion, with lower overhead for event queue operations. Finally, it accelerates the deletion of any unprocessed future events originated from an EG when the EG is terminated prematurely before its end time.

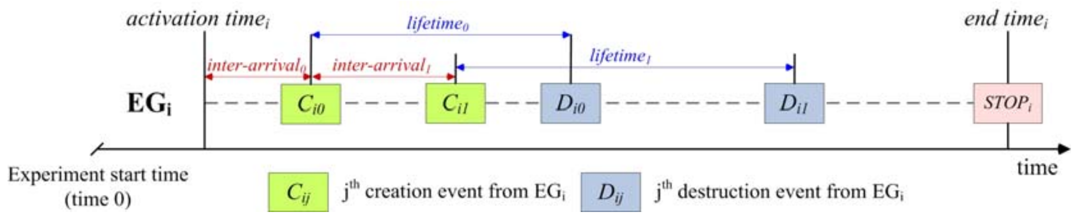

Figure 4: Event generation

4.4 Event Dispatching

In the simulation main loop given in Figure 5a, the ED checks the FEL periodically (a configurable *polling interval Δ*) to extract events with time stamps smaller than or equal to the current execution time, thus synchronizing virtual event time with real wallclock time. An event with time stamp t is guaranteed to be processed by the ED no later than wallclock time (*t* + *Δ*). This polling interval should be chosen to strike a balance: a small value improves the timeliness of event retrieval, but wasting ED cycles that could otherwise be spent on event dispatching; whereas a large value increases execution efficiency, but at the cost of reduced responsiveness. In practice, this interval is usually set to be a couple of seconds.

a. ED – Simulation Main Loop

```
Input: Δ (FEL polling interval)
1.  while termination_flag == false do
2.     current_time = time elapsed since experiment start time
3.     publish current_time in datastore
4.     next_time = minimum event time stamp in FEL
5.     while current_time < next_time do
6.        sleep(Δ)                  //wait for next future events
7.        current_time = current_time + Δ
8.        publish current_time in datastore
9.     end while
10.    current_events = extract all current events from FEL
11.    if current_events != Ø then
12.       append current_events to the end of PEL
13.       sg = divide current_events into simultaneous groups
14.       for each group in sg do
15.          for each event in group do  //parallel dispatch
16.             spawn an OD to process event asynchronously
17.          end for each
18.       end for each
19.       set termination_flag if the stop condition is met
20.    end if
21. end while
```

b. OD – Operation Dispatch Flowchart

```
Input: evIdx (event index into PEL)
1.  event = PEL[evIdx]          //retrieve event from PEL
2.  opg = divide operations into groups of same sequence
3.  for each group in opg do
4.     for each op in group do  //parallel dispatch
5.        spawn an actuator to execute op asynchronously
6.        if op is of type "execute APP" then
7.           barrier = false
8.        else
9.           barrier = true
10.       end if
11.    end for each
12.    if barrier == true then
13.       wait for group completion   //inter-group barrier
14.    end if
15. end for each
```

Figure 5: A skeleton of event and operation dispatch algorithms

In addition, the ED's current execution time, called *ED time*, is published to the datastore periodically (line 3, 8). The ED time defines a moving lower bound on the time stamps of events that can be safely inserted into the FEL without causing causality errors during event processing. On the other side, an EG ensures that the future events to be inserted are always scheduled after the present ED time, regenerating the events if necessary. This timing service is especially useful in large-scale distributed experiments where the ED and EGs are hosted on different machines without a shared clock. The current events extracted from the FEL are appended to the PEL and divided into *simultaneous* groups (line 12-13). For each group, a set of ODs is spawned to process the simultaneous events in parallel (line 16). The simulation is terminated when all the stop events are executed or when a user-specified stop time is reached (line 19).

As shown in Figure 5b, an OD retrieves the event scheduled for it from the PEL, dispatching groups of operations with the same sequence number to concurrent actuators (line 5). Note that the actuator created to execute/monitor an application instance is completely detached from the OD and continues to run in the back-end until the instance is destroyed at departure time, while the actuators created for other types of operations are joined by the OD when the operations are done, using a barrier to ensure causal consistency between consecutive operation groups (line 12-14).

4.5 Experimental Control

The HIL simulation includes several mechanisms for adaptive experimental control, as follows.

- **Parallelism control**. The simulation achieves event-level parallelism by processing *simultaneous* events concurrently. In a statistical sense, the number of simultaneous events increases with the number of event sources, as multiple streams of events are merged in the FEL. To increase event parallelism further, the front-end allows those events scheduled close enough with each other to be considered as simultaneous, as long as the difference between their time stamps is below a predefined *time tolerance threshold*, exploiting the temporal uncertainty in the arrival and departure of application instances to improve simulation efficiency. This approach can find analogy in other works (e.g., Fujimoto 1999; Loper and Fujimoto 2004). As the application instances have a much longer lifetime than the threshold, the resulting perturbation is negligible. Furthermore, an event is annotated with its actual dispatch time, which can be used as the effective time stamp of the event in replay mode to ensure experiment repeatability. On the contrary, the front-end can also put an upper bound on the number of simultaneous events processed at any time, enabling one to effectively control the maximum degree of event parallelism in an experiment. This parallelism control provides the adaptability needed to conduct experiments in clouds with restrictions on the number of VMs that can be created in parallel at a time.

- **Flow control**. Before an actuator creates a VM, the request is filtered by an admission control module at the back-end, as illustrated in Figure 2. This module keeps track of the number of VMs currently active in the overall experiment and on a per VM container basis. The ED is informed when these numbers reach predefined limits. Consequently, the ED starts to drop the creation and destruction events of newly arrived application instances, while allowing the destruction events to pass through for those instances that already existed. The dispatching of creation events is resumed once some of the existing VMs depart the cloud. This flow control mechanism can help contain cost for experiments in commercial clouds and prevent running out of resources in prototype cloud testbeds that do not have internal admission control capabilities.
- **Rejection and Failure response**. If an application instance fails in the cloud due to the rejection or failure of some of its VMs, the event/operation status is updated based on feedback from the actuators. Accordingly, the OD schedules a *transient event* to clean up any dangling VMs in that rejected/failed application instance and deletes the original destruction event in the FEL. This has the effect of moving the original destruction event to the present execution time. Purging the dangling VMs allows resources to be released quickly, making room for new application instances and reducing the experiment cost incurred.
- **Replay control**. Deterministic replay is achieved by loading the events processed in a previous experiment directly from a static source *without* regenerating the random numbers. If a positive tolerance threshold was used in the original experiment, it is reset to zero during replay to eliminate the uncertainty in event time stamps. Based on the status of the events and their operations, a replay can be controlled to create the exact conditions observed previously. Depending on the purpose of the replay, for example, the events could be scheduled at their original time stamps or at the recorded dispatch times. Those events dropped in the previous experiment would also be dropped in the replay. Moreover, the front-end could inject specially synthesized events into the cloud to reproduce the effect of previous VM rejections and failures.

5 EXPERIMENTAL RESULTS

The simulation engine was implemented in Python using multiprocessing. To demonstrate the front-end capability of synthesizing resource usage patterns, experiments were conducted over a period of 10 hours in a cloud testbed with three IBM BladeCenter HX5 servers running Red Hat Enterprise Linux 6.2 KVM hypervisors. The front-end and the Redis datastore were hosted on another server in the same chassis. Each server has two Intel Xeon® E7-4800 processors, a 50GB solid state disk, and different amount of memory (from 141GB to 283GB). These servers are connected by a 1GB dedicated Ethernet network.

The test case involved three types of synthetic applications of varied sizes (number of VMs per instance), inter-arrival and lifetime distributions, and workload characteristics. All application instances were created from a common VM image (1 virtual CPU, 192MB virtual memory, and 2GB virtual storage), executing CoreMark and IOzone periodically in an infinite loop throughout its lifetime. Different types of applications were instructed to pass different parameters to the benchmarks to maintain different levels of CPU and disk I/O workload intensity. The experiment was first run in *parallel event processing mode*, using one EG to generate instances for each type of application. Based on the event trace file obtained, the experiment was then replayed using the original event time stamps in *sequential event processing mode* with only one OD. Both runs exploited the operation-level parallelism inherent within each application instance. Table 1 summarizes the EG configuration used in the parallel processing mode.

Table 1: EG configuration (time unit: minute)

EG Name	Instance Size	Activation Time	End Time	Inter-arrival Distribution	Lifetime Distribution	Application Workload
EG1	3 VMs	0	600	Gamma (α=0.12, β=2.08)	Gamma (α=0.36, β=166.67)	Intensive CPU / low disk I/O
EG2	3 VMs	0	600	Gamma (α=0.42, β=2.38)	Gaussian (μ=45, σ=5)	Moderate CPU / moderate disk I/O
EG3	1 VM	0	600	Exponential (λ=0.6)	Uniform (a=30, b=60)	Intensive CPU / intensive disk I/O

Figure 6 shows the observed patterns, where the y-axis represents the number of active VMs and the number of each type of application instances attained in the cloud testbed over time. During the ten-hour period, there were 1088 VM creations and 980 VM destructions in the parallel run, whereas the sequential run managed to create and destroy only 528 and 372 VMs respectively (or replaying the events scheduled in the first 247 minutes in the trace file). It is clear that, all other conditions being equal, the parallel event processing mode is able to achieve a much higher throughput than the sequential one (by a factor of 2.3 in this test case), making it more suitable for larger-scale experiments. Moreover, the parallel mode can generate more complex and dynamic resource usage behavior with irregular spikes and valleys, as commonly experienced in IaaS clouds. A detailed validation of resource usage patterns in production cloud environments is beyond the scope of this paper and will be addressed elsewhere.

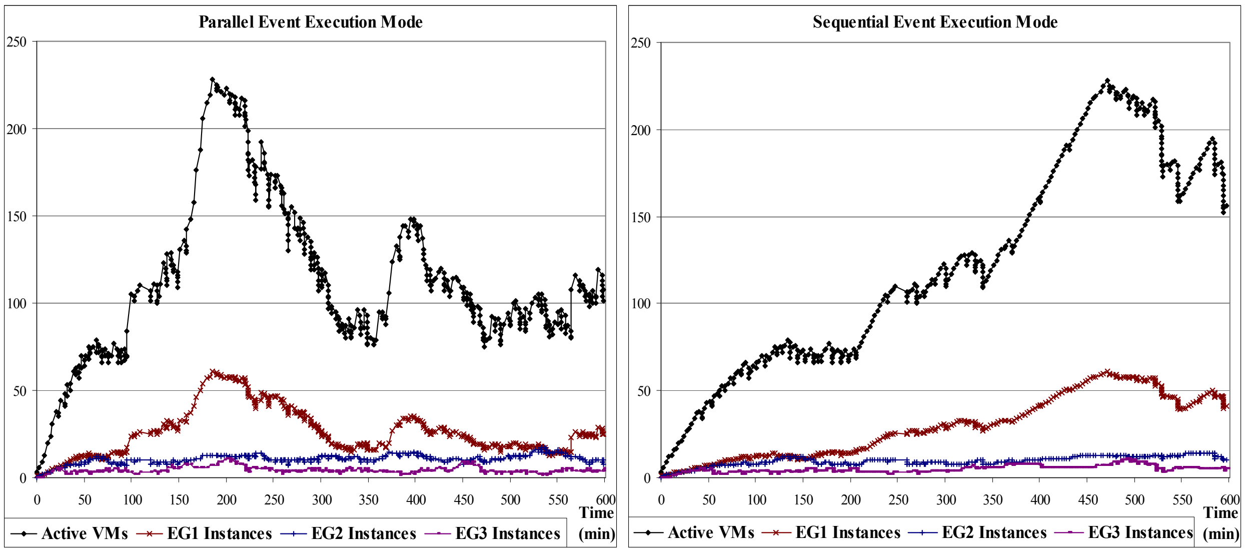

Figure 6. Patterns obtained in parallel and sequential event execution modes

The length of the FEL was monitored regularly during the parallel run. Thanks to the just-in-time event insertion, the FEL was kept relatively short throughout the experiment with a maximum length of 155 and a time-weighted average length of 75.9 events. In terms of the aggregate memory usage across all front-end processes, the sequential run consumed a maximum of 1.37GB, while the parallel run required a maximum of 1.44GB memory (or a slight increase of 4.73%).

6 CONCLUSION AND FUTURE WORK

With the increasing adoption of the IaaS cloud service model, accurate evaluation of application and system performance on such virtualized infrastructures has attracted considerable interest. This paper presented the HIL simulation technique used in the CloudBench framework for automated benchmarking of IaaS clouds in a scalable, controllable, and repeatable way. By using generic event templates, the technique integrates a cloud- and application-agnostic discrete-event simulation with the cloud infrastructure under test within a closed feedback control loop. It supports the decomposition of complex resource usage patterns into manageable components and provides a mechanism for statistical multiplexing of requests from different types of applications to synthesize realistic and emergent behavior in an experiment. It also exploits parallelism at multiple levels to improve simulation efficiency, while maintaining temporal and causal relationships with proper synchronization. The experiments demonstrated that the technique can synthesize complex resource usage patterns for effective cloud performance benchmarking.

As part of our future research, we plan to extend the technique to use fully distributed event management for higher scalability and to model other resource management activities in IaaS clouds, such as VM migration and image capturing. We also aim to explore the CloudBench capabilities beyond the performance benchmarking arena to orchestrate application deployment within and across cloud infrastructures and to facilitate decision making in the design and implementation of cloud service architectures.

REFERENCES

Amazon Corporation. 2012. Elastic Compute Cloud. Accessed March 19. http://aws.amazon.com/ec2/.

Apache Software Foundation. 2012. Hadoop. Accessed March 19. http://hadoop.apache.org/

Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. 2009. "Above the Clouds: A Berkeley View of Cloud Computing." Technical Report No. UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, CA.

Assuncao, M. D., A. Costanzo, and R. Buyya. 2009. "Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters." In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, 141-150. Munich, Germany.

Binnig, C., D. Kossmann, T. Kraska, and S. Loesing. 2009. "How is the Weather Tomorrow? Towards a Benchmark for the Cloud." In *Proceedings of the 2nd International Workshop on Testing Database Systems*. Providence, RI.

Bodik, P., A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. 2010. "Characterizing, Modeling, and Generating Workload Spikes for Stateful Services." In *Proceedings of the 1st ACM Symposium on Cloud Computing*, 241-252. Indianapolis, IN.

Calheiros, R. N., R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. 2011. "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms." *Software: Practice and Experience* 41(1):23-50.

Cooper, B. F., A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears. 2010. "Benchmarking Cloud Serving Systems with YCSB." In *Proceedings of the 1st ACM Symposium on Cloud Computing*, 143-154. Indianapolis, IN.

Chen, Y., S. Iyer, X. Liu, D. Milojicic, and A. Sahai. 2007. "SLA Decomposition: Translating Service Level Objectives to System Level Thresholds." In *Proceedings of the 4th IEEE International Conference on Autonomic Computing*, Jacksonville, FL.

Chen, Y., A. Ganapathi, R. Griffith, and R. H. Katz. 2010. "Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay." Technical Report No. UCB/EECS-2010-81, Electrical Engineering and Computer Sciences, University of California at Berkeley, CA.

DayTrader. 2012. "DayTrader – A More Complex Application." Accessed March 19. https://cwiki.apache.org/GMOxDOC22/daytrader-a-more-complex-application.html.

EEMBC. 2012. "CoreMark Benchmark." Accessed March 19. http://coremark.org/.

Fathy, H. K., Z. S. Filipi, J. Hagena, and J. L. Stein. 2006. "Review of Hardware-in-the-Loop Simulation and Its Prospects in the Automotive Area." In *Proceedings of SPIE – Modeling and Simulation for Military Applications* 6228:6228E. Orlando, FL.

Fujimoto, R. M. 1999. "Exploiting Temporal Uncertainty in Parallel and Distributed Simulations." In *Proceedings of the 13th IEEE Workshop on Parallel and Distributed Simulation*, 46-53. Atlanta, GA.

Ganapathi, A., Y. Chen, A. Fox, R. Katz, and D. Patterson. 2010. "Statistics-Driven Workload Modeling for the Cloud." In *Proceedings of the 26th IEEE International Conference on Data Engineering*, 87-92. Long Beach, CA.

Ghosh, R., K. S. Trivedi, V. K. Naik, and D. S. Kim. 2010. "End-to-End Performability Analysis for Infrastructure-as-a-Service Cloud: An Interacting Stochastic Models Approach." In *Proceedings of the 16th IEEE Pacific Rim International Symposium on Dependable Computing*, 125-132. Tokyo, Japan.

Hosking, M., and F. Sahin. 2009. "A Discrete Event XML based System of Systems Hardware-in-the-Loop Simulation for Robust Threat Detection." ." In *Proceedings of the 4th IEEE International Conference on System of Systems Engineering*. Albuquerque, NM.

IBM Corporation. 2012a. SmartCloud. Accessed March 19. http://www.ibm.com/cloud-computing/.

IBM Corporation. 2012b. WebSphere Application Server. Accessed March 19. http://www.ibm.com/software/webservers/appserv/was/.

IBM Corporation. 2012c. DB2 Database Server. Accessed March 19. http://www.ibm.com/software/data/db2/.

Idziorek, J. 2010. "Discrete Event Simulation Model for Analysis of Horizontal Scaling in the Cloud Computing Model." In *Proceedings of the 2010 Winter Simulation Conference*, Edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yucesan, 3004-3014. Baltimore, MD.

Iyer, R., R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. 2009. "VM^3: Measuring, Modeling and Managing VM Shared Resources." *Computer Networks* 53:2873-2887.

Jackson, K. R., L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. 2010. "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud." In *Proceedings of the 2011 IEEE 2nd International Conference on Cloud Computing Technology and Science*, 159-168. Indianapolis, IN.

Jayasinghe, D., S. Malkowski, Q. Wang, J. Li, P. Xiong, and C. Pu. 2011. "Variations in Performance and Scalability when Migrating n-Tier Applications to Different Clouds." In *Proceedings of the 2011 IEEE International Conference on Cloud Computing*, 73-80. Washington, DC.

Jones, R. 2012. "Netperf Benchmark." Accessed March 19. http:// www.netperf.org/.

Jung, G., K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. 2008. "Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments." In *Proceedings of the 5th IEEE International Conference on Autonomic Computing*, 23-32. Chicago, IL.

Ledin, J. A. 1999. "Hardware-in-the-Loop Simulation." *Embedded Systems Programming* 12(2):42-60.

Li, A., X. Yang, S. Kandula, and M. Zhang. 2010. "CloudCmp: Comparing Public Cloud Providers." In *Proceedings of the 10th Annual Conference on Internet Measurement*, Melbourne, Australia.

Loper, M. L., and R. M. Fujimoto. 2004. "Exploiting Temporal Uncertainty in Process-Oriented Distributed Simulations." In *Proceedings of the 2004 Winter Simulation Conference*, Edited by R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 395-401. Washington, DC.

Mills, K., J. Filliben, and C. Dabrowski. 2011. "An Efficient Sensitivity Analysis Method for Large Cloud Simulations." In *Proceedings of the 4th IEEE International Conference on Cloud Computing*, 724-731. Washington, DC.

Norcott, W. D. and D. Capps. 2006. "IOzone File system Benchmark." Accessed March 19. http://www.iozone.org/.

Nunez, A., J. L. Vazquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente. 2011. "Design of a New Cloud Computing Simulation Platform." In *Proceedings of the 2011 International Conference on Computational Science and Its Applications*, LNCS 6784:582-593. Santander, Spain.

Ostermann, S., K. Plankensteiner, and R. Prodan. 2011. "Using a New Event-Based Simulation Framework for Investigating Resource Provisioning in Clouds." *Scientific Programming* 19:161-178.

Papp, Z., M. Dorrepaal, D. J. Verburg. 2003. "Distributed Hardware-in-the-Loop Simulator for Autonomous Continuous Dynamical Systems with Spatially Constrained Interactions." In *Proceedings of the*

International Parallel and Distributed Processing Symposium. Nice, France.

Paxson, V., and S. Floyd. 1997. "Why We Don't Know How to Simulate the Internet." In *Proceedings of the 1997 Winter Simulation Conference*, Edited by S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, 1037-1044. Atlanta, GA.

Redis. 2012. Redis Key-Value Store. Accessed March 19. http://redis.io/.

Schludermann, H., T. Kirchmair, M. Vorderwinkler. 2000. "Soft-Commissioning: Hardware-in-the-Loop-based Verification of Controller Software." In *Proceedings of the 2000 Winter Simulation Conference*, Edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 893-899. Orlando, FL.

Sriram, I., and D. Cliff. 2010. "Effects of Component-Subscription Network Topology on Large-Scale Data Centre Performance Scaling." In *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems,* 72-81. Oxford, U.K.

Vaquero, L. M., L. R. Merino, J. Caceres, and M. Lindner. 2009. "A Break in the Clouds: Towards a Cloud Definition." *ACM SIGCOMM Computer Communication Review* 39(1):50-55.

Woodside, M., G. Franks, and D. C. Petriu. 2007. "The Future of Software Performance Engineering." In *Proceedings of the IEEE Workshop on the Future of Software Engineering*. Minneapolis, MN.

Ye, K., J. Che, X. Jiang, J. Chen, and X. Li. 2010. "vTestkit: A Performance Benchmarking Framework for Virtualization Environments." In *Proceedings of the 5th Annual ChinaGrid Conference*, 130-136. Guangzhou, China.

Yigitbasi, N., A. Iosup, D. Epema, and S. Ostermann. 2009. "C-Meter: A Framework for Performance Analysis of Computing Clouds." In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 472-477. Shanghai, China.

ACKNOWLEDGEMENT

This research is partially supported by the Industrial Development Agency (IDA) of the Irish Government. We thank our colleagues Yoonseo Choi, Kyung Ryu, and Livio Soares for their contributions to the overall CloudBench project.

AUTHOR BIOGRAPHIES

QI LIU is a postdoctoral researcher at the IBM T. J. Watson Research Center in NY, USA. He received his doctorate in Electrical and Computer Engineering from Carleton University, Ottawa, Canada. His research interests include modeling and simulation methodologies and tools, parallel and distributed simulation, exascale systems, and cloud computing. His email address is liuqi@us.ibm.com. More information at researcher.ibm.com/person/us-liuqi.

MARCIO A. SILVA is a postdoctoral researcher at the IBM T. J. Watson Research Center in NY, USA. He received his doctorate in Computer Engineering from University of São Paulo, Brazil. His research interests include performance modeling, virtualization, storage systems and cloud computing. His e-mail address is marcios@us.ibm.com.

MICHAEL R. HINES is a research staff member at the IBM T. J. Watson Research Center in NY, USA. He received his doctorate in Computer Science from S.U.N.Y Binghamton University in New York, USA. His research interests include virtualization, operation systems, cloud computing and distributed systems. His e-mail address is mrhines@us.ibm.com.

DILMA DA SILVA is a researcher and manager at the IBM T. J. Watson Research Center in NY, USA. She is an ACM Distinguished Scientist and ACM Distinguished Speaker. She received her PhD from Georgia Tech, USA. Her research interests include operating systems, distributed computing, cloud computing, and high end computing. Her e-mail address is dilmasilva@us.ibm.com. More information at www.research.ibm.com/people/d/dilma.