CloudBench

Experiment Automation for Cloud Environments

Marcio Silva, Michael R. Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, Dilma da Silva
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598-0218
{marcios, mrhines, dsgallo, liuqi, kryu, dilmasilva}@us.ibm.com

***Abstract*—The growth in the adoption of cloud computing is driven by distinct and clear benefits for both cloud customers and cloud providers. However, the increase in the number of cloud providers as well as in the variety of offerings from each provider has made it harder for customers to choose. At the same time, the number of options to build a cloud infrastructure, from cloud management platforms to different interconnection and storage technologies, also poses a challenge for cloud providers. In this context, cloud experiments are as necessary as they are labor intensive.**

CloudBench [1] **is an open-source framework that automates cloud-scale evaluation and benchmarking through the running of controlled experiments, where complex applications are automatically deployed. Experiments are described through experiment plans, containing directives with enough descriptive power to make the experiment descriptions brief while allowing for customizable multi-parameter variation. Experiments can be executed in multiple clouds using a single interface. CloudBench is capable of managing experiments spread across multiple regions and for long periods of time. The modular approach adopted allows it to be easily extended to accommodate new cloud infrastructure APIs and benchmark applications, directly by external users. A built-in data collection system collects, aggregates and stores metrics for cloud management activities (such as VM provisioning and VM image capture) and application runtime information.**

Experiments can be conducted in a highly controllable fashion, in order to assess the stability, scalability and reliability of multiple cloud configurations. We demonstrate CloudBench's main characteristics through the evaluation of an OpenStack installation, including experiments with approximately 1200 simultaneous VMs at an arrival rate of up to 400 VMs/hour.

Keywords- Cloud computing; Benchmark; Virtual Machines

I. INTRODUCTION

The cloud computing paradigm presents several benefits for both customers and providers. According to the Infrastructure as a Service (IaaS) [2] model, customers can acquire computational resources from one or more cloud providers, in the form of Virtual Machines (VMs). These are employed for application execution in a flexible and incremental manner. For cloud providers, the decoupling between supplied resource (VM) and supplier equipment (Hosts) means leveraging several aspects of economies of scale, from bulk discounts with both server manufacturers and public utility companies [3] to server utilization maximization through dense VM packing and resource over commitment [4].

Cloud customers have to select among many different providers and numerous different offerings (VM types and configurations) from each provider. Cloud providers are no less burdened, having to select a combination of technologies and architectures to strike a good balance between cost and performance while building their infrastructures. The number of options can be narrowed down through heuristics and design principles, but a large set of options that require actual experimental analysis still remain to be considered.

IaaS clouds are best evaluated by exercising them in the same way that actual customers would do: by running applications inside acquired VMs. For providers, this may involve emulating the behavior of a large set of applications from multiple prospective customers, across multiple architectures, in order to decide on a configuration for their cloud management stack. For customers, this may involve running their applications (or some emulated equivalent) on multiple cloud infrastructures with multiple combinations of parameters in order to select the most appropriate environment. These are time and labor-intensive tasks, and appropriate automation can greatly mitigate the effort required to carry out these evaluations.

In Section 2, the most relevant aspects and challenges to cloud experimentation and benchmarking are briefly discussed. The challenges are then addressed through the description of our framework for automated cloud experimentation and benchmarking, CloudBench, in Section 3. Section 4 describes its implementation, and presents experimental results. Section 5 presents related work, while Section 6 discusses future work and presents conclusions.

II. MOTIVATION

A. Need for Cloud Experimentation

Cloud providers, on their quest to remain competitive and grow their service portfolios and revenues, are required to continuously re-engineer their applications and systems, experimenting with design options and methods to tune service components, increasing resiliency and profitability.

Such experimentation has distinctive challenges. Not only is there a large parameter space to explore, but both customers and providers experience a similar conflict: either they spend a long time preparing a large number of different experiments or they save this preparation time by restricting themselves to a much smaller number of arbitrarily pre-selected parameter combinations. Clouds are massively

distributed computing infrastructures, requiring experiments that span a large number of elements, for a long time, with multiple repetitions to yield statistically meaningful results. The systematic exploration of this parameter space requires scalable experiment automation.

A second challenge is the experiment itself. Because customers need to run their own applications, we observe that the smallest unit of cloud experimentation is in fact not individual VMs, but rather an application. This means that in fact there are orthogonal skill sets required for carrying out experiments: expertise in the setup of an application with representative resource demands, expertise in the *generation of load* for that application, and expertise comprehending the mapping between applications and the underlying cloud infrastructure. Having all three skill sets in a customer's staff is rare. In this context, being able to lessen the need for these experts by abstracting application handling through a uniform method is of paramount importance.

A third distinctive challenge in cloud experimentation is performance data collection and aggregation. Given the necessary scale in time and space for an experiment to be representative, a large mass of data from multiple sources will be generated. A proper data collection method is needed to establish a history that can then be used to compare not only between cloud providers, but also to track regressions between changes in infrastructure design and architectures.

A more subtle challenge comes from the nature of a cloud infrastructure as a distributed system. Due to the large number of "moving parts" on a cloud, automating experimentation requires dealing with constant failures during application deployment and execution. Reliable error handling is essential, because it would be unacceptable to require human intervention during an experiment.

The main argument advanced here is not the uniqueness of the challenges described: *systematic parameter space exploration*, *lack of uniformity across multi-cloud and multi-application deployments*, *data collection and processing,* and *deployment error detection and recovery*. Rather, it is their conspicuousness in the face of the need for constant experimentation, which makes clear the need to address these challenges in an automated fashion.

B. Cloud Applications

We evaluate clouds by executing experiments at an appropriate scale. A consumer searching for a provider to outsource the execution of its own internal applications might opt to deploy them on each candidate cloud. However, certain types of applications are not suitable for such exploratory deployment. In many cases, benchmarks become necessary substitutes for producing evaluation metrics.

The provider faces a similar problem. Since the currently running customer applications cannot be duplicated for experimental execution, the running of benchmark applications is also an attractive alternative. By having a mix of application types and parameters with appropriate variety, an experiment could apply a load to the provider's infrastructure that emulates a large number of unrelated customers with different applications.

Thus, the discussion of some useful applications categories is appropriate. The first category is comprised by *transactional* applications, representing typical commercial multi-tier applications. These are characterized by having large numbers of small transactions, being I/O bound (normally because of the database tier) and limited horizontal scalability. Examples include LAMP stacks or middleware-applications like DayTrader [5], a tiered (application server and database) online stock trading enterprise application. The second category consists of *data-intensive distributed* applications. Large transactions – normally data transformations over large amounts of pre-produced data – and very good horizontal scalability are the characteristics of this class. An example of these would be Hadoop (workloads) [6]. The third category is composed of *High Performance Computing* (HPC) applications. These are also characterized by having large transactions and variable horizontal scalability (typically only loosely-coupled HPC applications are horizontally scalable). An example of this category would be the HPC Challenge benchmark suite (HPCC) [7]. Finally, the fourth and last category comprises *synthetic benchmark a*pplications - simple pieces of code that generate a heavy demand on a particular resource (e.g., processing, memory, networking, storage). An example of this category is the coremark [8] benchmark (processor-intensive only).

C. Cloud Metrics and Figures of Merit

Here, we itemize the most relevant metrics and figures of merit for a cloud that serve as an initial guide for evaluation of customer/provider cloud benchmarks. Through the observation of dimensional variations in cloud metrics, figures of merit are assessed. One metric is the *provisioning latency*, indicating how much time elapses between the arrival of a request to create a new VM and its availability for use. Another metric is *provisioning throughput*, indicating how many simultaneous provisioning requests the cloud can handle. Afterwards, once applications are started, we also include the metric application *runtime performance*, measured by *latency*, *throughput*, and *bandwidth*.

The aforementioned metrics can then be used to compose three figures of merit: scalability, stability, and reliability. Scalability indicates how the metrics vary with increases in the number of running VMs and the load intensity submitted to the applications. Stability indicates how these metrics vary over time. Reliability indicates how likely provisioning requests or running applications are to fail while assessing the cloud's scalability and stability. (Note that failures during the brief period of time which CloudBench spends configuring applications is treated as a hard failure, for which the tool knows how to log, scrap, and restart.)

D. Costs and Alternative Scoring Methods

Benchmark scoring is highly dependent on baselines. As an example, the "cost" of a resource can vary greatly based on supply and demand. We've made the explicit decision in the implementation of CloudBench to outsource the production of any score to components running outside of our framework. This can be done easily, in an online manner,

by making API calls to our framework during runtime to apply a scoring function to any of the data reported by the framework (and subsequently alter the behavior of the benchmark if necessary).

III. CLOUDBENCH

Through the use of controlled experiments, an experiment is composed of a series of declarative statements, describing the types and number of applications to be deployed into a cloud, and their behavior over time (e.g., lifetimes). The applications used in the experiments are pre-defined benchmarks that fall into one of the categories previously discussed, making CloudBench a meta-benchmark (or benchmark harness).

Every step of application deployment is automated in a straightforward manner: VM creation, application configuration, controlled execution, data collection and VM termination. If the application is not suitable for benchmark automation without human intervention, then it *must be modified* to do. Since each different cloud has its own API for VM creation and termination, and each application has its own methods for configuration and controlled execution, CloudBench is organized as an extensible framework. In order to make it flexible and extensible, each cloud infrastructure management platform (henceforth designated *cloud manager*) and application has a self-contained *adapter*. The framework is responsible for providing basic common services, such as distributed state management, structured data sharing and message passing.

A. Mode of Operation

To use CloudBench, a provider or customer writes an *experiment plan*. Submitted as a list of directives in the form <object> <operation> <parameters>, the *experiment plan* is processed and translated to cloud manager-specific actions. Normally an experiment starts with the setup of connections to a cloud manager, followed by the the startup of the data collection services. From this point on, operations are performed against experiment objects. The basic operations are *attach* and *detach*, which will add and remove objects from an ongoing experiment.

Experiment objects are defined within CloudBench, and used by it to control the effective deployment and execution of benchmark applications. Objects are of two classes. The *concrete* objects are managed and tracked by both CloudBench and the cloud manager. The *abstract objects* are the ones whose meaning and state are tracked only by CloudBench representing a logical aggregation of the multiple instances of concrete objects. Abstract objects can represent either a single individual instance of a benchmark application deployed on a cloud, or a group of inter-related instances. It is through the specification of abstract objects that an experiment assumes a truly dynamic behavior.

Concrete objects are of four flavors: *Clouds*, *Regions*, *Hosts* (exposed by some Clouds) and *Virtual Machines* (VMs). The Cloud object represents the cloud manager, and includes in its description all information required to establish a connection to it, including access and authentication credentials. By having a whole cloud as an object, CloudBench allows one direct an individual experiment plan at multiple clouds to compare them against each other. The Region can be described as the smallest point of access where a VM is instantiated. While the meaning of Region is invariant, its scope is very specific to each particular cloud. It can range from a single host (in a virtualized environment with the libvirt [9]/KVM duo) to a whole geographic region with multiple "availability zones" (in the case of Amazon's Elastic Compute Cloud [10]). The definition of the Region as a distinct object is useful, allowing CloudBench to exploit intra-cloud parallelism (e.g., in a geographically distributed cloud). The last concrete object, VM, is the execution point for the applications. Through the logical combination of one or more VMs, CloudBench creates and manages the most important abstract object: *Virtual Applications* (or VApps).

A VApp is deployed in two phases: First, its abstract object specification is translated into concrete objects (VM creation requests) to be submitted to a given cloud manager (through a cloud-specific API adapter). If the cloud manager supports exposing Host information to the user, then CloudBench can pass along any placement recommendations eventually specified by the user. CloudBench then waits for the VMs to become available, starting the second phase of deployment: we upload a list of configuration files and execute specific commands within each VM, culminating with applications ready to process submitted load. CloudBench keeps track of the mapping between concrete and abstract objects, and between hierarchies of abstract objects in an *Object Store*.

During an experiment, relevant metrics are automatically collected: provisioning latency and throughput, and application runtime performance. Lastly, the experiment is terminated after the conditions specified in the experiment plan are met (e.g., a certain amount of time has passed, or a number of parameter variations have happened). All the VMs are terminated in order to prevent run up costs, a salient point in clouds with usage billing by unit of time. Figure 1 presents CloudBench's main components.

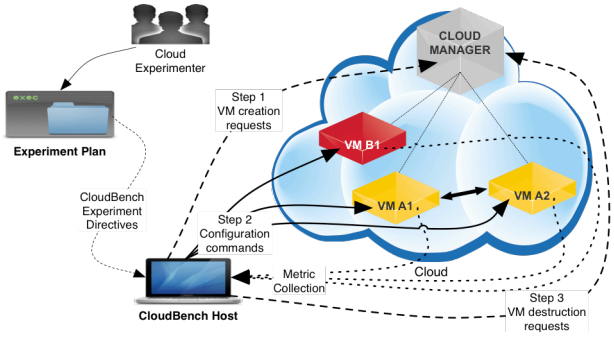

Figure 1 – Interactions with users, cloud manager and VMs.

B. The Virtual Application (VApp) Abstraction

CloudBench relies on abstract objects to achieve a balance between the need for synthesis and descriptiveness of experiment plans. The previously mentioned abstract object VApp represents a group of VMs logically grouped to

execute a specific benchmark. A VApp is defined according to four fundamental attributes: *type*, *topology*, *configuration steps* and *load behavior*. The first, type, represents the actual benchmark application to be executed. The topology attribute defines how many VMs will be needed for the application, as well as their roles. A role is associated with a VM image that contains specific binaries, data, and interconnection rules. The configuration steps attribute defines which scripts/binaries should be run inside each VM role, in order to fully setup and start the benchmark. Finally, the fourth attribute, load behavior, describes the dynamic variation of load applied to the VApp.

In addition to a set of VMs housing the VApp's individual benchmark components (e.g., a web server and database in the case of a transactional application), each VApp has a set of VMs to run *Load Generator* (LG) processes. The Load Generator is specified as an attribute in the VApp definition, submitting load to participating VMs, generating the resource demand pattern. In Figure 2, where VApp examples are shown, the LG VM for DayTrader runs the iwlengine [11] multi-client simulator, while the LG VM for Hadoop just submits a request for a long running job.

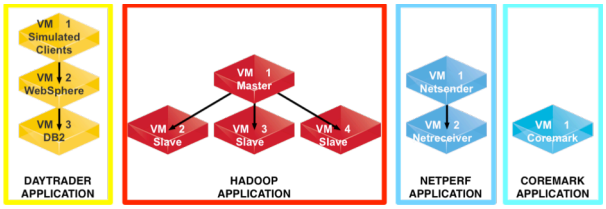

Figure 2 – Examples of CloudBench Virtual Application types.

A VApp is defined through templates, textual descriptions (in the form <attribute, value> pairs). Spawning a deployment daemon on the local CloudBench host carries out the deployment of a new instance of a VApp. This daemon performs the two-step process described previously: it submits the appropriate creation requests, waits for the VMs to boot, contacts each VM using SSH and starts the VApp's, components, as shown in Figure 3.

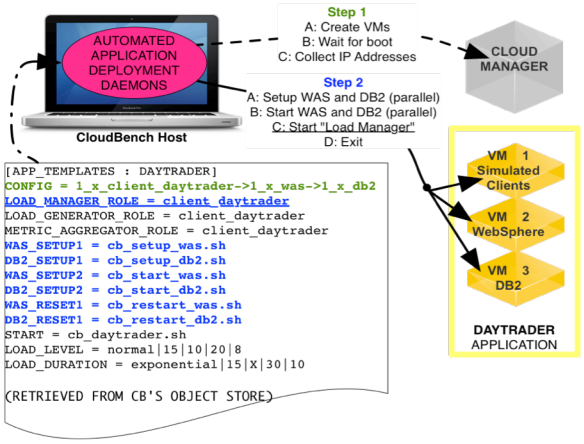

Figure 3 – Deployment of the Virtual Application type DayTrader.

None of the aforementioned steps can, when done at scale, tolerate any human intervention. We designed for failure recovery from the beginning. In case of any failure during the VApp deployment, the daemon will automatically terminate all VMs that belong to it. The reason for this is the integral need for VApps to effectively generate load on the cloud. Figure 3 depicts the deployment steps we have described.

Once a VApp is deployed, a controlling daemon – *Load Manager* (LM) – is instantiated in one of the participating VMs to take control of the execution. Running in the VM selected in the VApp definition, the LM will be in charge of managing the load behavior, which is described by two parameters: *load level* and *load duration*. The load level is very specific to each VApp type and can have a variety of meanings, including "number of concurrent clients issuing requests" for a transactional VApp, or the "size of dataset to be analyzed" for a data-intensive distributed VApp. The load duration represents the time interval in which the VApp is executed with a fixed load level.

The steady-state operation of a deployed VApp is: after the startup of a Load Manager daemon in one of the VMs, it obtains the parameters for load level and load duration based on two random probability distributions (of the user's choosing), rolls the proverbial dice, and invokes the application binary or script specified in the VApp template to begin generating load. This binary or script is the actual Load Generator, run inside the VM selected from the VApp's description. The Load Manager daemon then waits for the LG to end its execution. Afterwards, optional application-specific "reset" scripts are executed. Figure 4 displays the temporal behavior of a DayTrader application.

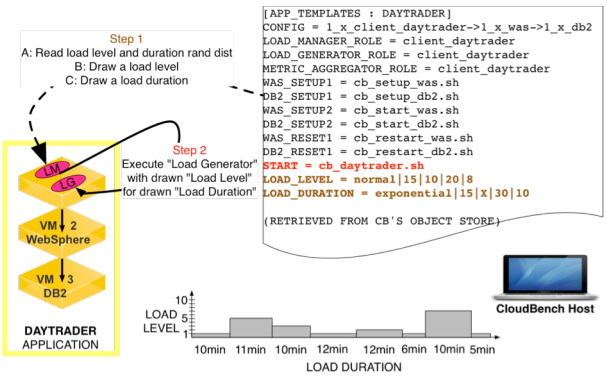

Figure 4 – Running of the Virtual Application type DayTrader.

The Daytrader VApp execution is managed by the Load Manager daemon that runs in VM 1, in a manner totally independent of the CloudBench host.

The adopted design, with a per-VApp Load Manager implementing the specified load behavior, has at least two desirable aspects. First, it allows great flexibility in the coverage of different parameters under different experimental scenarios. A customer willing to assess cloud stability might create an experiment plan where a large fixed number of applications is deployed, each having a fixed load level and a very long load duration. On the other hand, a

provider that wants to assess its infrastructure's scalability, by measuring its capacity to cope with sudden resource demand peaks might choose to create a plan with a small number of applications using a highly variable load level and small load duration. The second desirable aspect refers to the scalability of the CloudBench framework itself. Since each deployed VApp operates as an entirely independent entity, executing it with the specified load behavior in a fully parallel manner without any need for centralized control, the increase in the number of VApps does not pose a bottleneck for experiment execution.

C. *The Virtual Application Submitter Abstraction*

As useful as the VApp abstraction might be, it still lacks the descriptive power to emulate the load imposed on a cloud for a crucial use case: customers arriving and leaving the cloud. Providers need to be able to assess VApp churn. As in the networking world, VApp churn happens when there is both a continuous flow of VMs being provisioned/terminated while customers run their applications. Customers also need to be able to assess a cloud's provisioning throughput, by verifying how quickly they can increase the number of VMs in case of need.

In order to experimentally assess these situations, another abstract object, the *Virtual Application Submitter* (VAS or "Submitter") is defined. A Submitter represents a group of VApps whose deployment is spread over time. It is defined according to three attributes: the VApp's *type*, *inter-arrival time* and *lifetime*. Submitters are a composite abstraction, employing VApps to create churn. Each Submitter is implemented by a long running daemon on the CloudBench host, which continuously deploys new VApps at each inter-arrival interval (determined by a user-specified probability distribution), and removes expired ones when they reach the end of their lifetimes. Each deployed VApp has the individual load behavior as described previously. The main difference between an explicitly deployed VApp and one that is generated by a Submitter is that the latter gets automatically removed at the end of its lifetime.

This design, where a Virtual Application Submitter daemon operates like a scheduler, instantiating VApps at regular intervals, has two advantages: the first is that by implementing its own self-contained scheduler, each Submitter emulates the realistic behavior that is expected on a cloud (multiple VApps leave and arrive in an uncoordinated manner). The second is scalability, since the absence of a central scheduler or event queue removes a bottleneck.

A Submitter is also defined through templates. Both the inter-arrival time and lifetime for each individual VApp generated can be described as fixed values, as monotonically increasing/decreasing sequences, or as random distributions. In fact, in addition to these attributes, a Submitter can be specified even with a per-VApp load behavior that overrides what was defined on the VApp's template. With that, it is possible to reuse the same VApp definition to implement experiments that emulate vastly different behaviors for the same VApp type. For instance, the same type could be used to define two different Submitters, one "slow but heavy" (i.e., the inter-arrival time is large, but the average load level for each VApp is high), or another that is "fast but light" (i.e., small inter-arrival time, low average load level). Figure 5 depicts an experiment where one Submitter deploys DayTrader VApps, while another deploys Hadoop VApps.

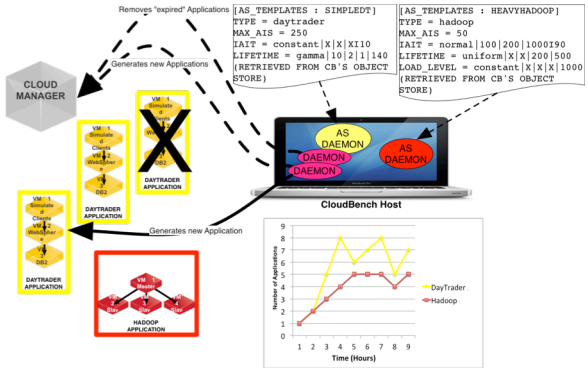

Figure 5 – *Application Submitters*.

D. *Operations and Additional Abstract Objects*

Experiment plans are written as a list of directives in the form <object> <operation> <parameters>. Having introduced both the concrete objects (Clouds, Regions, Hosts, and Virtual Machines) and the two main abstract objects (VApps and Application Submitters), operations can now be elaborated. The possible operations performed against each object are divided in two groups. The *back-end operations* are performed through the cloud manager, directly affecting a concrete object. The two basic back-end operations are attach/detach, which creates/removes objects in an ongoing experiment. Each attach or detach operation is translated to cloud manager-specific commands, hence the name back-end operations.

Additional object-specific back-end operations are required by CloudBench to provide the descriptive power for meaningful exploration of cloud infrastructures. In addition to attach and detach, the *capture* operation is also applicable to VMs. If a customer spends time configuring a group of VMs, they will likely opt to preserve their state in order to turn them into a set of base VM images to be provisioned. In order to provide proper automation in this scenario, capture requests can be submitted explicitly (i.e., interactively) or implicitly, through a Virtual Machine Capture Submitter (VMCS). A VMCS, another abstract object, keeps generating capture requests, directed to VMs that are part of a deployed VApp. This object is defined by two main attributes, the *inter-arrival time* (between *capture* requests), and the *minimum runtime* of a deployed VApp before a capture request can be issued. The main goal of the latter attribute is prevent the capture of VMs that were just created.

In addition to full support for the determination of the metrics required for the assessment of the scalability, stability and reliability figures of merit, limited support for the assessment of agility and resiliency is also available. Agility can be assessed by how quickly a VApp can scale-out (in fact, increase and decrease service capacity). CloudBench provides a *resize* operation, which increases or

decreases the number of VMs belonging to a VApp. This operation is only useful for VApps designed to leverage adding or removal of worker nodes (e.g.: data-intensive distributed or HPC).

The next figure of merit, resiliency, can be assessed through the execution of the *fail* and *repair* operations. Since the most important metric here is the overall failure rate in face of induced failures in specific elements of the system, the fail (and repair) operations are applicable to Regions or Hosts that are part of a given Region. It is possible to execute the fail/repair operation over individual Hosts, while observing the global behavior of the cloud infrastructure. The fail operation is internally implemented as an ungraceful shutdown of the cloud services running on a Host. In addition to the fail/repair operation applicable to Regions/Hosts, the operations suspend and resume, which will stop and re-start a VM, can be used to simulate failures of individual VMs.

The second group of operations is represented by the *front-end operations*. These are operations performed only through CloudBench, without direct translation into native cloud manager commands. The purpose of these operations is to either display or alter configuration or state information about a given object inside the Object Store (i.e., the "front end"). An experimenter can interactively use the *list* operation to obtain a list of objects of a given type currently attached to an experiment. The *show* operation is used to inspect the attributes of an individual object, while the *alter* operation is used to change an attribute of such object.

Despite the fact that front-end operations are not translated into native commands to the cloud manager, they are extremely relevant to modify the dynamic behavior of VApps and Submitters in an experiment. For instance, the alter operation can be performed over a deployed VApp in order to change its load level to a much higher value. An experimenter could use the alter operation to introduce coordinated load peaks during an experiment, in a controlled and predictable manner.

Some of the most unique and important front-end operations are the experiment-centric operations *waitfor*, *waituntil*, and *waiton*, which give the experimenter the ability to automatically stop and alter the flow of operation execution according to specific conditions. It is important to differentiate between the flow of operation execution and the execution of the VApps. The waitfor operation pauses the execution of new directives for a specific period of time. The waituntil command will do so until a certain condition is met, such as a certain number of VMs provisioned. Finally the waiton operation pauses the execution until the CloudBench host receives a message.

E. Data Collection

CloudBench experimental capabilities are just as good as the framework's ability to collect data during the execution of experiment plans. Metric data samples are collected, processed and stored on a *Metric Store,* a data repository accessible by both VMs and the CloudBench host. Metrics are divided in two groups: *management* and *runtime*. Management metric data samples are reported directly by the cloud manager, and are collected, processed and stored by the CloudBench host. Each VM has only a small group of management metric data samples associated to it, reported in few key moments of the VM's lifecycle: creation time, boot time, application startup and termination time. Runtime metric data samples are reported directly by the VMs, and include both performance samples reported by the application (e.g., throughput and latency) and VM OS, such as processor, memory, network and storage utilization. While application runtime metrics are collected directly from the output of the Load Generation process, the OS runtime metrics are collected through the Ganglia [12] distributed system monitor. In cloud infrastructures where the Hosts are visible and accessible, Ganglia is also employed to collect OS runtime metrics directly from them.

All metric groups are time stamped according to a common time base, which allows for correlation between events like provisioning or VApp load level change and (VM or Host) resource usage. By making each VApp independent in both load management and metric data samples collection, the increase in the number of VApp instances is not critical to CloudBench's scalability.

IV. Experimental Results

A. Code Description

CloudBench was implemented in the Python language, comprising approximately 20K lines of code. Currently, it has the following benchmarks available as different VApp types: DayTrader [5], SPECWeb [13], SPECjbb [14], Hadoop [6], HPCC [7], LOST (a Linux-Apache-MySQL-PHP stack used internally), coremark [8], netperf [15], and filebench [16]. A new VApp type is typically implemented in approximately 150 lines of shell script code: the only function of the scripts is to tell CloudBench how to setup and start (e.g., run a binary) the VApp's components. No changes in CloudBench's core code are required. This allows VApp specialists to extended it incrementally.

The supported cloud providers currently available are Amazon Elastic Compute Cloud (EC2) [10], IBM's Smart Cloud Provisioning (SCP) [17], OpenStack [18] and direct libvirt (for "cloudless" deployments). A new cloud adapter can be implemented in a single file with approximately 600 lines of python code, allowing cloud specialists to also extend CloudBench incrementally.

CloudBench's Object Store is implemented using Redis [19], an in-memory NoSQL data store with data types somewhat richer than the usual key-value store (e.g., sets and lists), support for publish/subscribe communication and remote accessibility through drivers in several languages (including Python and shell script). These characteristics make it a good platform for configuration and state sharing between the CloudBench host and the VMs. The Metric Store was implemented using MongoDB [20], a persistent, scale-out NoSQL data store with "schema-less" tables.

B. Interfaces and Interaction

An experimenter can interact with CloudBench through a Command Line Interface (CLI), an XML-RPC server and a

GUI. Through the CLI, experiment directives (i.e., commands) are submitted and immediately executed, usually run in blocking mode. For instance, the deployment of a new application (vappattach command) will cause the CLI to block until the deployment is completed. By adding the keyword async to an experiment directive, CloudBench executes it in non-blocking mode, spawning a new daemon to perform the activity. The instantiation of a new Virtual Application Submitter (vasattach command) is always non-blocking, since it will start a long running daemon that keeps deploying and removing VApp instances.

The CLI can be used for programmatic experiment execution, through the use of experiment plans. These plans are supplied as a text files containing a sequence of experiment directives that are read and executed sequentially.

The second option is the interaction via its XML-RPC server. CloudBench will start an XML-RPC server capable of receiving and executing individual experiment directives submitted by a client running the programming language of the user's choosing, returning the result of the execution back to it. This second interaction option allows external applications to leverage the automation capabilities of CloudBench in a transparent manner and develop complex policy engines or control loops in tandem with the benchmark.

Finally, all experiment directives can also be submitted through a Web GUI. In addition to interactive experiment directive submission, the GUI features a dashboard that summarizes the current state of an experiment, including the detailed object listing and the most recent performance data samples for each object. Interfacing with CloudBench through XML-RPC, the Web GUI is a prime example of the use of external "client applications" with CloudBench.

C. Experimental Evaluation

The CloudBench framework contributes to cloud experimentation and benchmarking with three useful characteristics: flexible description of boundaries for parameter space coverage, uniform interface to automatically deploy an application against different cloud infrastructures, and integrated data collection. Equally important, it displays these characteristics in a scalable fashion, being able to handle thousands of individual VMs during an experiment. We demonstrate the framework's characteristics by going through a series of scenarios where experimental needs are presented, and then a solution is achieved through the execution of an experiment plan.

All experiments were performed on a 16-node OpenStack installation (Essex release). Each node has 6 Intel Xeon cores operating at 2.93 GHz, 128 GB of DDR3 DRAM operating at 1333 MHZ, 1 TB 7200 rpm SATA HDD, two Gigabit Ethernet and one 10 Gigabit Ethernet NICs. The OpenStack *nova-scheduler* and *keystone* services were configured on one of the nodes, alongside the CloudBench code. Another node housed the *glance* image repository. The rest of the nodes (14) were configured as compute nodes. All nodes were installed with Red Hat Enterprise Linux 6.2.

For the first case, consider a scenario where a potential customer wants to investigate the application performance profile of a cloud. To this end, the customer deploys a small set of different VApp types on the cloud, using a uniform distribution for load level, and a small fixed value for the load duration. The application performance can be profiled as a function of the load level. The experiment plan for this scenario is shown on Figure 6.

```
1 cldattach osk TESTCLOUD
2 clddefault TESTCLOUD
3 regattach all
4 monattach
5 apiattach
6 cldalter ai_templates daytrader_load_level=uniform|X|X|1|30
7 cldalter ai_templates daytrader_load_duration=90
8 cldalter ai_templates hadoop_load_level=uniform|X|X|1|9
9 cldalter ai_templates hadoop_load_duration=90
10 vappattach daytrader async=12:2
11 vappattach hadoop async=8:1
12 waitfor 10h
13 vappdetach all
14 apidetach
15 mondetach
16 monextract HOST os
17 monextract VM os
18 monextract VM app
19 exit
```

Figure 6 – *Experiment plan* for first scenario.

The experiment directives (commands) 1 through 5 are required for initial experiment setup, being mandatory for almost every experiment plan. Initially, a new cloud has to be added to this experiment. In this case, the OpenStack cloud (i.e., osk) was added (i.e. attached) to the experiment, identified as TESTCLOUD. After that, this cloud is set as the "default" cloud. This is useful to avoid the need for typing (or writing) the cloud name on each experiment directive, as CloudBench can experiment with more than one cloud attached simultaneously. After that, all Regions in the TESTCLOUD are attached to the experiment. Then, the data collection infrastructure is activated by the *monattach* command. The 5th directive activates the XML-RPC server, allowing the Web GUI to be used for visual inspection of the experiment's progress.

The next few directives (6-13) synthesize the core of the experiment. Here, the VApp definitions (stored as templates in the Object Store) of two types (DayTrader and Hadoop) have their load level and load duration parameters changed, in order to provide wider coverage. For DayTrader, the boundaries are set between 1 and 30 simultaneous clients, while for Hadoop they are set between 1 (1000 100-Byte records) and 3 (5000 100-Byte records). After the parameter change, 12 DayTrader and 8 Hadoop VApp instances are deployed in parallel and non-blocking mode. Each application is composed of three (DayTrader) or four (Hadoop) VM images of 29 GB in size, containing a Red Hat Enterprise Linux 6.1 installation. The VMs are configured with 2 vCPUs and 2 GB of RAM. After being deployed, all applications are left to run for 10 hours (waitfor 10h command), while the load level continues to vary on 90-second intervals. Finally, the applications are removed from the experiment (*vappdetach all* command).

The last few experiment directives (14 - 19) perform the experiment wrap-up, deactivating the XML-RPC server and data collection, and then extracting all runtime performance data samples to comma-separated text files. The file produced by the 16th directive presents all per-Host OS

runtime metric data samples (e.g., processor, memory). The 17th directive does the same for the VMs. The file produced by the *monextract VM app* command presents all per-VM application runtime metric samples.

Figure 7 illustrates the application throughput after running of experiment.

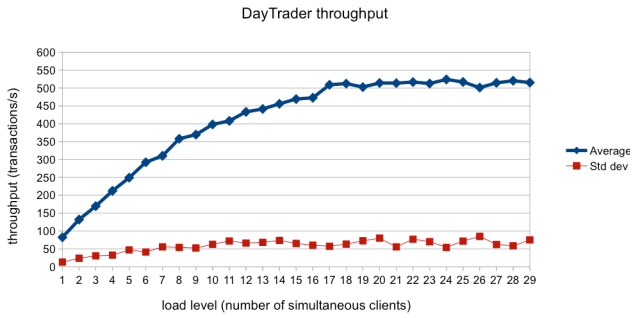

Figure 7 – Average throughput for the DayTrader *Application*.

The graph was produced directly from the data contained in the per-VM application runtime metric samples file. The DayTrader performance is presented as a function of the load level, averaged over thousands of samples from all 12 individual DayTrader VApp instances. In this scenario, each instance contributed hundreds of samples for each load level. The aggregation of data in this form is useful for a "static" performance analysis (i.e., not taking in account variation over time). For this particular cloud, the relatively uniform distribution on the per-load level average throughput can be credited to the absence of other customers using this cloud and competing for resources during the experiment.

Another illustrative example of the analysis made possible by the data made available by CloudBench collection facilities is presented in Figure 8.

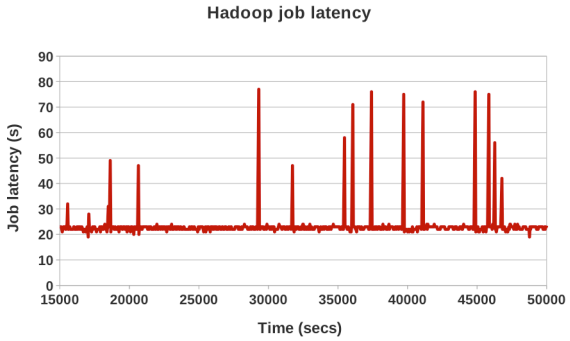

Figure 8 – Hadoop job latency in one of the *Virtual Applications*.

This first experiment highlights how a prospective customer could use CloudBench for the extraction of VApp runtime performance metrics. It is important to note that, in addition to providing a convenient method for automated VApp deployment and parameter space variation, the experiment is cloud-agnostic. Also noteworthy, CloudBench's aggregate data collection capabilities allowed for the determination of application performance profiles in a simple and convenient manner.

The second case is built around a scenario where a provider wants to profile the provisioning scalability of its cloud infrastructure. To this end, VApps are continuously deployed, under different inter-arrival times. Since the objective is the profiling of VM provisioning baseline performance as a function of the arrival rate, a simple "null workload" VApp type is used. This is composed of a single VM, which executes a long running process who just sleeps for 2 minutes, and outputs an artificially generated runtime performance data sample. Being defined with a single small VM image (with only 2 GB on disk) containing an Ubuntu 11.10 installation, this application is also useful for experiments with large scale.

This experiment uses a special Virtual Application Submitter, designated "simplenw". For each selected inter-arrival time (arrival rate), a new Submitter is instantiated (vasattach), left running for one hour (waitfor), and removed alongside all the deployed VApps (vasdetach and vappdeatach all). The arrival rates are fixed values ranging from 10 to 400 VM/h. Before attaching a new Submitter, the value of the inter-arrival time is changed through the front-end operation alter.

The provisioning runtime performance metric samples file, extracted through the monextract VM management, contains a per-VM detailed profile of the VM creation and termination processes. During VM creation, individual cloud-specific steps like VM image creation time and VM instance boot time are explicitly recorded. VM provisioning failures are also recorded in this file, alongside the last known state of the VM before failure. An illustrative graph is presented in Figure 9.

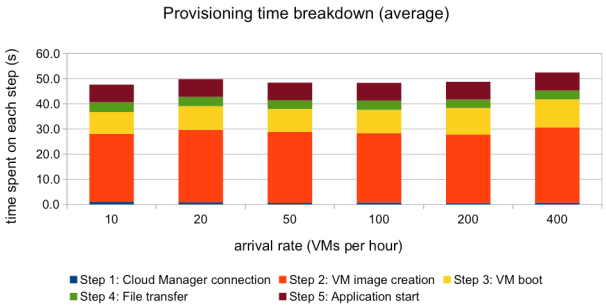

Figure 9 – Provisioning time breakdown collected by CloudBench.

This second experiment highlights how a prospective provider could use CloudBench for the extraction of the VM provisioning latency of its cloud. The provisioning profile presented in Figure 12 is useful to diagnose the problem areas in its provisioning process. In this particular scenario, the use of a small VM image proved to be convenient. The average total provisioning time is small enough to allow the cloud to be submitted with the intended high arrival rates without suffering secondary effects from an excessive number of outstanding simultaneous provisioning processes.

Finally, the third case is presented to illustrate CloudBench's scalability. Here, the goal is to be able to collect performance data in a timely and reliable manner. This will be illustrated in two steps. First, to demonstrate the framework's core scalability, an experiment on a simulated cloud is executed.

A simulated cloud represents a cloud where abstract objects are created only in CloudBench's Object Store. To this end, two Virtual Application Submitters, with an aggregated arrival rate of 7,200 VM/h, was run for over 4 hours, creating approximately 37,500 simulated VMs. With this experiment, we demonstrate that 1) CloudBench itself is capable of submitting VM creation requests at a very high arrival rate, at least one order of magnitude of what is achievable on a real cloud and 2) that the memory footprint required to manage a large scale experiment is well within the limits of a moderately sized desktop computer. To register and manage roughly 50,000 experiment objects (12,500 DayTrader VApps with 3 VMs each), the total memory usage by the Object Store (Redis) was approximately 1 GB, while the Metric Store (MongoDB) required 4 GB of disk space. It is important to note, however, that simulated VMs do not write runtime performance samples back to the Metric Store.

After verifying the upper limits in resource consumption by CloudBench for a very large experiment, we proceeded to attempt a similar experiment on a real cloud. We employed the previously described "simplenw" Submitters, with fixed inter-arrival times between 20 and 30 seconds, yielding an aggregate arrival rate between 500 and 600 VMs/h. After 2 hours, the Submitter created approximately 1,200 VApps on the 14-node OpenStack cloud. Given the fact that the "null workload" is a VApp type with a single VM, we reached a steady state at approximately 1,200 VMs simultaneously provisioned on the cloud, continuously reporting data samples.

While the design of both the Load Manager daemon and of the scheduling mechanism for the Virtual Application Submitter are intrinsically capable of scaling-out, it is important to verify the same for data collection capabilities. Each time a VApp completes an execution at a given load level for a given load duration, it produces a set of application metrics and tags it with a load identifier. Monotonically increasing each time a new load level is used, this identifier allows the measurement of the rate of progress made by the Load Manager daemon running in a VM. By comparing the number of the highest recorded load identifier for each Application instance with the number of samples recorded in the Metric Store, the number of samples that were lost during the collection can be determined. In Figure 13, a per-VApp application performance metric sample loss rate is presented for this experiment.

While the vast majority of all 1,200 VMs run for a very similar period, the number of load identifiers differed greatly, indicating that different VMs progressed at wildly different paces. Given the fact that the VMs that arrived late had a significantly lower number of samples submitted, with their Load Manager daemons progressing at a very low pace, our initial suspicion was that the resource bottlenecks for the number of physical processors and disk spindles was strengthened. We observed that some of the latest VMs did not have any progress in their execution at all! With 128 GB of memory and 10 Gigabit Ethernet network cards, each node could accommodate approximately 600 VMs. However, the application performance – measured in this case as the progression of the load identifier increase – started to degrade even before reaching 100 VMs per node.

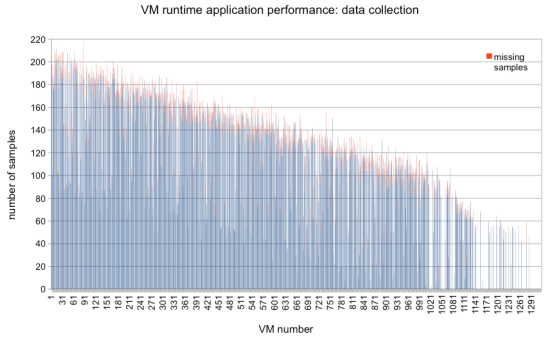

Figure 10 – Management and Runtime metrics collection.

Despite this noticeable performance drop as the number of VMs increased, we note that for the effectively submitted application performance metric samples, an average only 4.2% were lost. With this experiment, we could demonstrate that CloudBench's data collection infrastructure is capable of handling experiments with moderate to large scales.

V. Related Work

The need for cloud experimentation and benchmarking is clear for both research and operational purposes. The variety of recent or active research projects with goals similar to ours is a good indicator of this fact. Works that most closely match the goals of CloudBench include CloudCmp [21], C-Meter [22], Cloud-Gauge [23], CloudHarmony [24], CloudSuite [25], and the Yahoo YSCB [26].

CloudGauge provides low-level ability to interface with private cloud deployments and attempts to provide abstractions at the workload-level by providing a template-based framework to include more complex workload interactions. C-Meter [22] provides an EC2-compatible extension to the GrenchMark grid benchmarking system, but is quite limited since benchmarks that only employ sequential grid-style or MPI applications are not well suited to demonstrate the complex interactions that exist in service-oriented architectures.

CloudCmp [21] provides a multi-cloud comparison framework, with micro-benchmark-level profile of multiple public cloud providers. CloudHarmony [24] provides a form of "benchmarking as a service" offering an online interface to generate reports on types of resource usage across public clouds. It does not provide a generic way to model cloud behavior or build new workloads. The Yahoo YSCB [26] is a data-serving-focused benchmark that attempts to provide a way to explore new data-serving paradigms quite dissimilar from traditional OLTP applications designs. CloudSuite [25] also addresses the micro-architectural inefficiencies in the running of scale-out workloads in the cloud.

Our work is orthogonal to these projects because we show that a missing benchmarking dimension – automation of datacenter-scale, end-to-end cloud performance – is very difficult to do without human intervention. This level of automation for representative, complex workloads is non-trivial and very application specific. Also, the prohibitive

cost restraints on the use of public cloud services mandated that we make CloudBench suitable for use in a private cloud. We designed it from the ground up to scale up to thousands of VMs, making visible much larger implications of heavy customer/provider interactions. CloudBench goes to great strides to expand the scale and scope of the cloud benchmarking problem by providing realistic abstractions that can stress different aspects of a cloud, such as placement, resource and network management. CloudBench is unique in its per-application dynamic load variation, failure injection, and extensibility.

Other related efforts for cloud evaluation, while not directly focused on automation, scaling abilities, or abstractions across multiple cloud APIs include the Intel Benchmark and Test Tool (BITT) [27], and multiple analyses on Amazon EC2 (e.g., [28]).

VI. Conclusions and Future Work

CloudBench is open-source and available for download today [1]. We have used it to perform a wide variety of evaluations, ranging from different forms of application performance modeling, resource over-commitment, customer/provider bottleneck analysis, and high-availability testing. In each of these cases, the time-to-start for the experimenter was nearly zero and complex experiments running for days were carried out without any human interaction. CloudBench allows us to represent and benchmark almost every observable interaction of a cloud, including applications, load generation, cloud management stack communication, hypervisor resource usage, failure behavior and more. In addition to that, through powerful abstractions, CloudBench provides a method for the experimenter to combine the testing of these interactions in an expressive way. The experimenter can apply probability distributions to almost every interaction of interest (e.g., application load, VM arrival rate) to leverage the full automation abilities enabled by our framework.

References

[1] M. Silva and M. Hines, "Cloud Rapid Experimentation and Analysis Toolkit." [Online]. Available: http://github.com/ibmcb/cbtool.

[2] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Proceedings of the Grid Computing Environments Workshop - GCE'08*, 2008, pp. 1–10.

[3] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," Berkeley, CA, USA, 2009.

[4] M. R. Hines, A. Gordon, M. A. Silva, D. Da Silva, K. D. Ryu, and M. Ben-Yehuda, "Applications Know Best: Performance-Driven Memory Overcommit With Ginkgo," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science - CloudCom'11*, 2011, pp. 8-16.

[5] "DayTrader - A more complex application." [Online]. Available: https://cwiki.apache.org/GMOxDOC22/daytrader-a-more-complex-application.html.

[6] "Hadoop." [Online]. Available: http://hadoop.apache.org/.

[7] P. Luszczek and D. Koester, "HPC Challenge v1.x Benchmark Suite," 2005.

[8] "Coremark." [Online]. Available: http://www.coremark.org/home.php.

[9] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive Virtualization Management using libvirt," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition - DATE'10*, 2010, pp. 574-579.

[10] "Amazon Elastic Compute Cloud (EC2)." [Online]. Available: http://aws.amazon.com/ec2/.

[11] "WebSphere Studio Workload Simulator Programming Reference." [Online]. Available: http://publibfp.boulder.ibm.com/epubs/pdf/c3163082.pdf.

[12] "Ganglia Monitoring System." [Online]. Available: http://ganglia.sourceforge.net/.

[13] "SPECweb2009." [Online]. Available: http://www.spec.org/web2009/.

[14] "SPECjbb2005." [Online]. Available: http://www.spec.org/jbb2005/.

[15] "Netperf." [Online]. Available: http://www.netperf.org/netperf/.

[16] "Filebench." [Online]. Available: http://sourceforge.net/projects/filebench/.

[17] "IBM Smart Cloud Provisioning." [Online]. Available: http://www-01.ibm.com/software/tivoli/products/smartcloud-provisioning/.

[18] "Openstack." [Online]. Available: http://www.openstack.org/.

[19] "Redis." [Online]. Available: http://redis.io/.

[20] "MongoDB." [Online]. Available: http://www.mongodb.org/.

[21] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp : Comparing Public Cloud Providers," in *Proceedings of the 10th annual conference on Internet Measurement - IMC'10*, 2010, pp. 1-14.

[22] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, "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 - CCGRID'09*, 2009, no. Section IV, pp. 472-477.

[23] M. A. El-Refaey and M. A. Rizkaa, "CloudGauge: A Dynamic Cloud and Virtualization Benchmarking Suite," in *Proceedings of 19th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises - WETICE'10*, 2010, pp. 66-75.

[24] L. Gillam, B. Li, J. O'Loughlin, and A. P. S. Tomar, "Fair Benchmarking for Cloud Computing Systems," 2012.

[25] M. Ferdman et al., "Clearing the Clouds: A Study of Emerging Workloads on Modern Hardware," in *Proceedings of the 17th Conference on Architectural Support for Programming Languages and Operating Systems -ASPLOS 2012*, 2011, no. Asplos, pp. 1-11.

[26] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 143.

[27] "Intel ® Benchmark Install and Test Tool (BITT)," 2011.

[28] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," in *Proceedings of the First International Conference on Cloud Computing - CLOUDCOMP '09*, 2009, pp. 115-131.