Position Paper: Cloud System Deployment and Performance Evaluation Tools for Distributed Databases

Markus Klems, Hoàng Anh Lê
Karlsruhe Institute of Technology (KIT)
76131 Karlsruhe, Germany
klems@kit.edu, hoang.le@student.kit.edu

ABSTRACT

Creating system setups for controlled performance evaluation experiments of distributed systems is time-consuming and expensive. Re-creating experiment setups and reproducing experimental results that have been published by other researchers is even more challenging. In this paper, we present an experiment automation approach for evaluating distributed systems in compute cloud environments. We propose three concepts which should guide the design of experiment automation tools: (1) capture experiment plans in software modules, (2) run experiments in a publicly accessible cloud-based Elastic Lab, and (3) collaborate on experiments in an open, distributed collaboration system. We developed two tools which implement these basic concepts and discuss challenges and lessons learned during our implementation. An initial exemplary use case with Apache Cassandra on top of Amazon EC2 provides a first insight into the types of performance and scalability experiments enabled by our tools.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation*

Keywords

cloud services; experiment automation; cloud testbeds; NoSQL databases; Cassandra

1. INTRODUCTION

Complex experiment setups on top of compute clusters are widely used in science and industry. Controlled experiments can provide evidence for both scientific reasoning and industrial decision making. Experiment-driven tuning, performance evaluation, and benchmarking of distributed databases and data processing frameworks have become hot topics in cloud computing. Several benchmarking tools have been developed for evaluating the performance and scalability of cloud databases, like Cassandra, HBase, MongoDB, etc. Unfortunately, controlled experiments with complex, distributed systems are still challenging due to the difficulty to deploy and properly configure the systems (table 1). This work argues to make better use of cloud services as an open environment for reproducible experiments with distributed systems, in particular for experiments for evaluating cloud database systems.

We suggest a generic framework, described in section 2, and in more detail in a previous publication [13], which automates the setup and execution of experiments with distributed database systems on top of compute cloud infrastructure. The approach enables experiment creation, variation, and re-creation on an open, collaborative, and elastically scalable environment. A major contribution of our work is the experiment tool design which abstracts from Infrastructure-as-a-Service providers and operating system resources, and furthermore allows deployment and benchmarking of a variety of cloud database systems.

Section 3.1 provides a brief overview of the cloud database benchmarking approach and tool used in our implementations. Sections 3.2 and 3.3 present two experiment toolkits that implement the concepts laid out in section 2. We discuss lessons learned from the initial tool design 3.2, and how it influenced the subsequent tool implementation 3.3. In section 4 we demonstrate an initial use case of our experiment tools, i.e., benchmarking the distributed database system Apache Cassandra on top of Amazon EC2, and discuss ongoing development work.

HotTopiCS'13, April 20–21, 2013, Prague, Czech Republic.

2. EXPERIMENT FRAMEWORK

In this section, we present the basic concepts of our experiment approach and framework: experiment modules (2.1) which can be executed in an Elastic Lab (2.2), accompanied by experiment collaboration and version control mechanisms (2.3).

2.1 Experiment Module

The experiment plan which encodes the sequence of experiments according to an experiment design is captured in a shareable *experiment module*. An experiment module consists of a directory structure that contains files which encode one or more *experimental units*, *treatments*, and *observational units*. The objective of experimentation is to measure the effect of a treatment on an experimental unit by observing the observational unit.

Challenge	Solution
The **effort** involved in a complex experiment setup, requiring different skills for system, software, and networking configuration, can quickly overwhelm a single experimenter.	**Fully automate** the experiment setup process with deployment & configuration management tools. Build the setup on top of virtual appliances.
Working on a continuously evolving shared experiment setup and experiment plan requires diligent **management** and **coordination**.	Experiment setups and plans are packaged as **experiment modules** which can be managed and shared with a collaborative version control system.
It can be difficult to obtain **hardware resources** for experiments, particularly if they do not need to be run regularly.	Allocate and de-allocate virtualized experiment resources **on-demand** (and pay-per-use) in a public compute cloud or storage cloud.

Table 1: Experimentation challenges and proposed solutions.

2.1.1 Definitions

The following definitions are based on Design of Experiment textbook definitions [2] which we have adapted to cloud-based experiments.

- **Experimental Unit.** The smallest unit to which a treatment can be applied. There are three types of experimental units: (a) cloud, (b) cluster, and (c) machine experimental unit.
- **Treatment.** A treatment is the entire description of what can be applied to an experimental unit. Treatments can be applied as (a) cloud, (b) cluster, and (c) machine treatment.
- **Observational Unit.** An observational unit is the smallest unit of the cluster system on which a response can be measured. There are three types of observational units: (a) cloud, (b) cluster, and (c) machine observational units.

2.1.2 Experiment Scaffolding

The experiment module itself can be fairly complex and consist of a number of files which reference to one another. Typos in the file names or in the file content can lead to experiment failures which are difficult to track down. We therefore propose the technique *experiment scaffolding* which helps experimenters to quickly and reliably generate syntactically and semantically correct experiment modules. The experimenter uses the command line or a graphical user interface to create ("scaffold") the directory structure filled with stub files that follow the conventions of an experiment module. Scaffolding is a technique that has been popularized by the agile Web development framework Ruby on Rails [18].

2.2 Elastic Lab

The experiments are created and executed on an *Elastic Lab* which is provisioned on-demand on top of virtual machines of a compute cloud. The resource capacity of the Elastic Lab can thus be adjusted to the experiment.

The following features are essential to the Elastic Lab:

- `launch` - Provision a cluster or individual machine in the compute cloud in a given configuration.
- `destroy` - Destroy the cluster or individual machine by terminating the virtual machine(s).
- `apply-treatment` - Apply the experiment treatment to one of {cloud, cluster, machine}.
- `prepare-experiment` - Configure the observer servers and the experimental unit servers.
- `run-experiment` - Run an experiment and save data in the local data repository.

We thought of two ways to apply a cluster treatment. One can either re-launch a new cluster (or a new machine) for each experiment, i.e., first apply a `destroy` action, followed by a `launch` action with the new treatment configuration. Alternatively, one can change cloud, cluster, or machine configurations, respectively, during experiment runtime via the command `apply-treatment`. The second treatment approach intentionally creates side-effects of a previous experiment to the next experiment.

Applying a treatment requires the re-configuration of the cluster which can affect multiple cluster services. We distinguish two types of cluster services: experimental unit services and observer services. The experimental unit is the system under test; the observer is the system that collects observations. For example, when scaling a database cluster that is observed with a monitoring system, we must add a new node to the cluster that runs both a database service and a monitoring service. The new node must be integrated in both the database cluster and the monitoring cluster.

The command `prepare-experiment` configures experimental unit servers and observer servers. For example, workloads are generated from workload files and database tables are set up for a performance evaluation experiment. The command `run-experiment` executes the experiment, e.g., benchmarking clients execute a workload.

2.2.1 System Automation

Automation is key to our experiment approach as is relieves the experimenter from time-consuming manual tasks. Machines, clusters, and the compute cloud must be set up in a desired configuration state. Since the compute cloud offers a homogeneous programming interface, automation is not too difficult to achieve on this level. Automating the configuration of machines and clusters, however, is less straightforward. The desired state of a machine, as well as the desired state of a cluster must be captured in an executable model. The model can then be instantiated by the experiment workflow (`prepare-experiment`, `apply-treatment`).

2.3 Experiment Collaboration

The purpose of an experiment collaboration system is to facilitate collaboration on a shared experiment workspace, including features well known from source code management, such as branching and merging. In a public repository

of experiment modules, experimenters can find and copy experiment modules, and commit their experiment module changes. Experimenters can also share experimental data and thus more easily retrieve and compare experimental results.

3. EXPERIMENT TOOLS

3.1 Cloud Database Benchmarking

Cloud database systems are supposed to adapt quickly to changing workloads, such as flash crowds [8] causing temporary hotspots [4]. We are therefore particularly interested in evaluating the performance and scalability qualities of distributed database systems, given challenging workloads. Performance is measured in terms of throughput and upper-percentile request latency. Scalability is characterized by the change in throughput and latency if hardware resources are added to the cluster, or removed from the cluster, respectively.

3.1.1 YCSB and YCSB++

YCSB [6] is an extensible, modular benchmarking tool with adapters to a variety of NoSQL database systems. It can be used for performance, scalability, and elasticity evaluation experiments. YCSB++ [15] extends YCSB with features, such as multi-client workload execution, multi-phase workloads, and system monitoring.

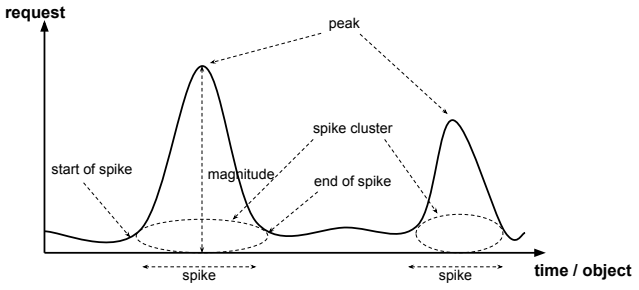

Figure 1: Hotspot Workload.

3.1.2 Hotspot Workloads

Bodik et al. [4] have introduced a statistical workload model for hotspot workloads. A hotspot workload is characterized by two types of spikes. A *volume spike* is the temporary increase of user requests. A *data spike* is the temporary shift in popularity towards a small number of hot objects which are requested. The overlap of a volume spike and a data spike is a *hotspot*. Figure 1 illustrates a hotspot workload with parameters "peak", "magnitude", "start of spike", "end of spike", and "spike cluster".

Based on the statistical workload model in [4], we implement a similar hotspot workload model for YCSB. The three parameters N, V and α allow generating a workload with data spikes. N specifies the number of hotspots, V the variance which determines the entropy of the popularity distribution, and α the spacial locality of hotspots. Generating a hotspot distribution is done in two steps. First, the popularity distribution of all hotspots is computed using a Dirichlet distribution. Second, the spatial locality of all hotspots is generated with a Chinese Restaurant Process [9].

3.2 Implementation #1: Whirr Experiment Tool

We implement an experiment tool on the basis of Apache Whirr[1]. Whirr can be used for cluster service deployment on a number of compute clouds, using the Java library jclouds[2], a multi-cloud software abstraction. The cluster services can thus be deployed on different supported cloud providers, such as Amazon EC2 and Rackspace. The deployment and configuration of services is remotely executed as shell scripts which are set up on the machines in the cluster. We extend Whirr with several features and services, including a performance benchmarking service.

3.2.1 Whirr Experiment Module

The experiment module for benchmarking distributed database systems consists of a *cluster* directory with cluster specification files, a *workload* directory with workload model files. Furthermore, there is a *template* directory which contains *dbcluster* and *workload* template files. User can scaffold an experiment module with a basic directory structure, a *run.sh* shell script, as well as *dbcluster* and *workload* properties files which are based on templates and customized by scaffolding options given on the command line.

3.2.2 Whirr Elastic Lab Architecture

Figure 2 shows a high-level overview of the Elastic Lab architecture that we implemented with Whirr and the connection to an experiment module version control system as well as an observation data store. The experiment executable *run.sh* encodes a sequence of commands, for example the following sequence, depicted in figure 2: (1) launch clusters with cassandra database service, ganglia monitoring service, zookeeper, and ycsb benchmarking service, (2.b/c) clone or pull the remote modules repository from github and push the current module (if not yet existing), (2.d) run benchmarking load phase, (3.a) collect benchmarking data from load phase, (2.e) run benchmarking transaction phase, (3.b) collect benchmarking data from transaction phase, (3.c) push benchmarking data to Amazon S3, (4) destroy clusters. Whirr already provides a number of the required features described in 2.2. We add additional features that enhance Whirr to enable the missing Elastic Lab features for cluster experiments. The extensions are listed and shortly described in table 2.

3.2.3 Collaboration via Github, EC2 and S3

There is a plurality of cloud-based services that can be used to implement the basic experiment collaboration features mentioned in section 2.3. For example, a Subversion service, e.g., Google Code, Assembla.com, or Unfuddle.com could be used as a basis for the experiment module collaboration system.

We use the distributed version control system git[3] with the cloud service github.com[4]. Moreover, the features provided by git are a good fit to requirements stated in section 2.3.

The cluster services are bootstrapped from basic Ubuntu Linux virtual machine images. Whirr allows specifying the image id as a parameter in the cluster specification file. The parameter is passed down to the jclouds multi-cloud ab-

[1]http://whirr.apache.org
[2]http://www.jclouds.org
[3]http://git-scm.com
[4]https://github.com

Elastic Lab Features	
Command	**Description**
`prepare`	Apply basic experiment setup configurations.
`run`	Run a YCSB benchmarking experiment and save data in the local repository. Option parameters are: `workload-phase` and `workload-file`.
`push`	Push measurement data from the observer servers to the remote observation data store.
`converge`	Apply a new cluster specification. Launch missing instances, and/or destroy supernumerary instances, respectively.

Table 2: Whirr Elastic Lab API.

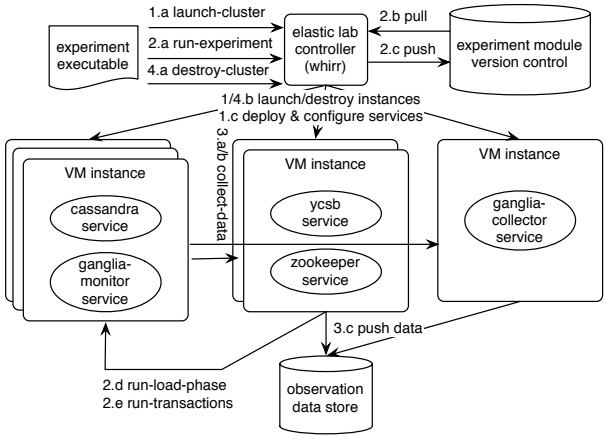

Figure 2: Elastic Lab Architecture.

straction layer and mapped to the specific cloud provider implementation. One can also prepare pre-packaged virtual appliances [20]. Amazon EC2 offers access control features to share access to images with other EC2 users or give public access to images. Virtual appliances can encode a time-consuming experiment treatment and thus enable more rapid bootstrapping of experiment setups.

We decide to use Amazon S3 as central observation data repository. Each observer server sends its' observation measurements to an Amazon S3 bucket with the name of the experiment. Amazon S3 offers access control features that allow implementing a variety of file sharing features. As an alternative to S3, one could also use Dropbox or other file sharing services.

3.2.4 Lessons Learned

The extensible, modular nature of both Whirr and the benchmarking tool YCSB (Yahoo! Cloud Serving Benchmark [6]) allow for a wide variety of distributed system benchmarking experiments on a variety of compute clouds by different providers. Whirr is an extensible software project with good unit and integration test coverage. Whirr offers deployment features for a wide spectrum of services, such as Zookeeper, Ganglia, and multiple database and data processing services. However, the implementation is not perfect. In the following, we discuss one design flaw and one missing feature of the Whirr experiment tool.

Use configuration management software: We successfully used the Whirr experiment tool for scalability evaluation experiments. We then tried to use it for automated parameter testing. However, Whirr is not a good choice for these types of experiments for two reasons. First, the software installation and configuration statements which are applied during the bootstrapping and configuration phases of a cluster launch must be implemented with shell scripts. Second, the main Whirr project is implemented in Java. We experienced that these two factors combined make it difficult to efficiently develop and debug additional, complex service configuration features. Whirr could be extended in this direction by using configuration management solutions, such as Chef and Puppet, which are already available as Whirr services. However, we have decided to implement a new Ruby-based experiment tool because such a tool integrates natively with State-of-the-Art configuration management software.

Faster bulk-loading: Performance benchmarking experiments with a heavily loaded database system deliver different results than those with lightly loaded databases. YCSB++ [15] offers an extension to YCSB that allows rapid bulk-loading of the target database using Hadoop MapReduce. However, Hadoop adds additional complexity to the experiment software stack. We therefore prefer a simpler bulk-loading mechanism using database backup-and-restore features.

3.3 Implementation #2: CSDE Experiment Tool

We built the Cloud System Deployment and Experiment Tool (CSDE)[5] for automated performance evaluation of distributed database systems. CSDE implements sound design choices of the Whirr-based experiment tool described in the previous section, however, takes a different approach to automating system configuration and adds better bulk-loading capabilities. We decide to implement CSDE in Ruby as it more naturally integrates with the configuration management software.

3.3.1 CSDE Experiment Module

Listing 3.3.2 shows an excerpt of an experiment profile. A *profile* node describes the compute cloud infrastructure where the distributed system is rolled out. The child node *provider* specifies the cloud provider. The child node *region* declares the machine setup for database cluster and workload generator in the region. If more than one region is declared, CSDE will deploy the database cluster in *multiple-regions*.

A *service* node specifies distributed system services. For now, CSDE implements only two services, namely the cassandra database and ycsb benchmarking services. The *service attribute* node with *key* and *value* node describes all needed information of this service.

3.3.2 CSDE Elastic Lab

The high-level architecture of CSDE is introduced in this section. CSDE is a Ruby on Rails Web application composed of loosely coupled components, Ruby modules, each of which is responsible for a particular task: Profile Parser, Profile Generator, Infrastructure Management, Deployment

[5]https://github.com/myownthemepark/csde

and Configuration Management, Workload Executor, and Bulk-Loader.

Profile Parser. In a first step, CSDE must parse the experiment profile which is provided as input by a user. The parser transforms the declarative profile document into Ruby objects upon which CSDE's business logic operates.

Profile Generator. The *profile generator* can be used to automatically generate experiment profiles that follow a factorial experiment design [2]. The design ensures that the number of experiments necessary for producing statistically useful results is minimized. Figure 3 shows the Web form where users can enter experiment parameters, e.g., "machine type", "heap size", and "cache". Each parameter can take two or three values (low, default, and high).

Listing 1: Experiment Profile Example

```
service1:
    name: cassandra
    attributes:
        replication_factor: 1

profile1:
    provider: aws
    regions:
        region1:
            name: us-east-1
            machine type: xlarge
            template: 3 cassandra
```

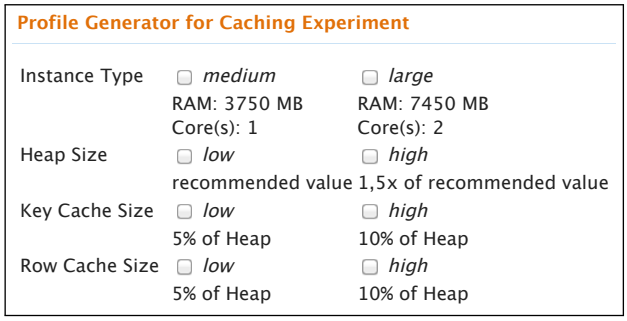

Figure 3: Profile Generator.

Infrastructure Management. Infrastructure management is responsible for launch and termination of virtual machines in the compute cloud. The database cluster as well as the benchmarking services are installed on top of these on-demand machines. Just like Whirr's jclouds, CSDE uses a multi-cloud abstraction, the `fog`[6] Ruby library, which provides a homogeneous programming model for different compute clouds.

Deployment and Configuration Management. After the infrastructure management component has launched virtual machine instances, the database cluster and benchmarking services are automatically set up using the deployment and configuration manager. This component uses Chef Server[7], a popular open source configuration management software. Chef "cookbooks", contain collections of configuration scripts, or "recipes". CSDE uses Chef recipes for setup and configuration of database and benchmarking services.

[6]http://fog.io

[7]http://community.opscode.com

Workload Executor. Once the database cluster and the benchmarking services are deployed and configured, the benchmarking service invokes workload generation and runs the workloads on the database cluster. CSDE saves the benchmarking results, uploads them to Amazon S3 and informs users via email.

Bulk-Loader.

Preparation of the database cluster for experiments requires bulk-loading the cluster with synthetical data. CSDE uses a backup-and-restore mechanism to more rapidly load the cluster. CSDE implements two options: backup-and-restore with S3 or with EC2 AMIs.

Backup-and-Restore using S3.

In the *backup* phase, a workload is executed which sends INSERT statements to the database cluster and thereby loads the cluster. When the INSERT-only workload has finished, each database node makes a snapshot of its data. All snapshot files are compressed in a tarball using `pbzip2`[8], a parallel version of the Unix compression tool `bzip2`. Each database node in the cluster uploads its tarball to Amazon S3.

Restore is the inverse operation of the backup step and is executed once for each experiment. At first, tarballs from Amazon S3 are downloaded to the machines running in Amazon EC2. There is a fast connection of ca. 30 MBytes per second between Amazon S3 and Amazon EC2. Then, the tarballs are decompressed, again using `pbzip2`. Finally, each database node imports the extracted files. Note that, each Cassandra node has to download the corresponding tarball, because it is set up with the dedicated data range.

Backup-and-Restore using EC2 AMIs. With this mechanism the two disk types "EBS disk" and "ephemeral disk" are used for each Cassandra machine. Two ephemeral disks are configured to build a Linux software RAID0 array and are used for Cassandra data and log storage. RAID0 block striping improves Cassandra's performance if disk I/O is a bottleneck. Ephemeral disks are attached to a virtual machine instance and are lost once the machine is terminated or re-boots. Amazon EBS (Elastic Block Storage) offers services similar to a NetApp filer. We use EBS volumes for backup and restore of prepared data dumps.

Backup: data from the RAID0 array is transferred to an EBS volume which is attached to the EC2 instance. Then we create a snapshot of the EBS volume. *Restore:* the launch configuration of a machine instance is provided with an EBS volume created from a prepared snapshot.

3.3.3 Lessons Learned

Although our experience with CSDE suggests that the second implementation approach has achieved some of the desired improvements over the first implementation, there are still very difficult open problems. A professional configuration management solution, in the case of CSDE we use Chef, provides more reliable service than configuration with shell scripts. However, the capabilities of Chef to automate system configuration rely on the quality of the configuration scripts given to Chef by the CSDE tool user. The user must still have sufficient knowledge of the system under test to provide correct Chef scripts and debug the scripts in case of configuration failures. Moreover, CSDE has grown into a complex system itself and requires some effort to set up. A

[8]http://compression.ca/pbzip2

more lightweight solution for experiment automation would certainly be desirable.

4. CASE STUDY EXPERIMENTS WITH THE CASSANDRA DATABASE SYSTEM

4.1 Whirr Experiments

We use the Whirr Experiment tool to run experiments with the distributed database system Apache Cassandra 1.0 [14]. As cloud provider we choose Amazon EC2 in the US East Region Virginia zone (us-east-1) and test a variety of different cluster sizes and different instance types: small, medium, and large[9].

We conduct scalability experiments with a Cassandra cluster using a YCSB workload with an 80/20 read/update ratio and a Zipfian request distribution. The cluster is only lightly loaded with 100MB per instance; 100K operations × the cluster size are performed per benchmarking phase. In our scalability tests the database cluster is re-launched for every experiment (first destroy, then launch with increased cluster size). The results indicate that Cassandra does not scale well when using small server instances. For clusters with 3, 4, and 5 small instances we measure approximately the same average maximum throughput of about 1000 ops/s. The 6-instance cluster shows only slightly better performance with 1,200 ops/s. The medium and large instance clusters offer better scalability. The database latency is volatile because of either disk contention or network contention. We can improve disk I/O considerably by setup of a RAID0 array. In ongoing experiments we address the problem of network bandwidth variability by pre-testing to determine a conservative throughput target, given a latency upper bound.

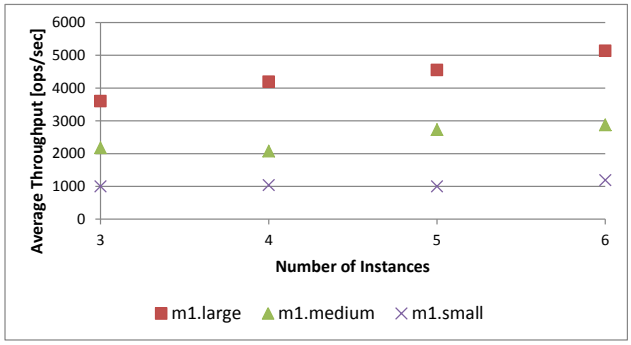

Figure 4: Cassandra Scalability Measurements.

The Whirr experiment tool is well-suited for running scalability tests. However, we discovered that our system settings were not optimal. For more versatile performance evaluation, including automated parameter testing, we developed CSDE. Some exemplary experiments with CSDE are presented in the next section.

[9]EC2 m1.small instance type: 1 EC2 Compute Unit (ECU), 1.7 GB memory, 8 GB of RAID 0 EBS storage, 64-bit. EC2 m1.medium instance type: 2 ECUs, 3.75 GB memory, 8 GB of RAID 0 EBS storage, 64-bit. EC2 m1.large instance type: 4 ECUs, 7.5 GB memory, 8 GB of RAID 0 EBS storage, 64-bit. ECU: 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

4.2 CSDE Experiments and Ongoing Work

Initial experiments with CSDE demonstrate its value for assessing the performance impact of configuration changes given a broad variety of synthetical workloads. We conducted several parameter testing experiments, including cache tuning. We also compared the system performance against well-known Zipfian workloads as described in [6], and the hotspot workload model as described in [4] To our surprise, the hotspot workload, which was intended to stress the Cassandra database performed much better than the Zipfian workloads. This is the case because the "hot objects" are served from Cassandra's cache which resides in the operating system's memory. In ongoing work, we run experiments on different operating systems and on a different cloud infrastructure to evaluate the multi-system, multi-cloud capabilities of our implementation. We plan to extend our experiments to other database systems, in particular, with HBase.

5. RELATED WORK

Cloud and Grid computing research testbeds, e.g., OpenCirrus [1] and PlanetLab [5], provide researchers access to compute and storage resources in globally federated data centers. The testbeds provide a great service to experiment-driven research, however, require additional tools for experiments with distributed systems.

Test automation in Grid computing testbeds. A number of distributed system testing tools have been developed for Grid computing environments [19]. The NMI Test & Build Laboratory [16], for instance, enables continuous integration of heterogeneous distributed computing systems by automating configuration, build processes, and unit testing. Automating functional testing, however, does not provide features for software and system quality evaluation. Most related to our work are experiment automation tools used in Grid testbeds for performance and scalability evaluation of distributed systems. DiPerF [7] is an automated performance evaluation framework for Grid services which has been used for experiments with client-server systems, such as the metadata and directory service of the Globus Toolkit. The Weevil framework [21] is most related to our work. Weevil combines a simulation-based workload generator with a model-based configuration management and system automation approach for experiments with distributed systems on PlanetLab. The framework has been evaluated with different types of distributed systems, however, not including distributed database systems. The automation approach of Weevil is similar to our initial implementation, the Whirr-based experiment toolkit. Using shell script templates for system automation is a mechanism that has several drawbacks, as discussed in section 3.

Test automation in cloud computing testbeds. A recent study reviews the state of research in cloud-based testing techniques [17]. The study lists 23 cloud-based testing frameworks in different categories, such as model based testing, performance testing, fault injection testing, etc. In the performance testing category, the study lists five references, including performance testing frameworks for virtualized applications [10] and for Network Management Systems [11]. The Virtualized-Aware Automated Test Service (VATS) [10] is most related to our work. Similar to our Whirr-based experiment tool and CSDE, VATS integrates

automated deployment of the system under test with automated benchmarking. The capabilities of VATS are demonstrated with a remarkable case study, the performance evaluation of a Xen-virtualized SAP R/3 system. Different to our approach, VATS seems to be very focused on evaluating SAP products in a specific virtualized environment. The design of VATS does not indicate that it can be run on a public compute cloud, such as Amazon EC2 and Rackspace Cloud, neither is it suitable for testing a wide array of distributed systems.

The most significant difference we see with regard to the related work is that our approach and tools focus on evaluating distributed database systems which are deployed in cloud computing environments. The use of public cloud services makes experimentation accessible to a larger audience, which is in contrast to closed Grid computing environments or private virtualized data centers. Furthermore, our tool design is modular and extensible with respect to workloads and systems under test. This is enabled by our focus on a specific type of distributed systems, i.e., open source distributed database systems, like Cassandra, HBase, etc. The approach and tools shown in this paper are particularly useful for developing advanced cloud database performance, scalability, and fault-tolerance measurement and optimization systems, as discussed in [3, 13]. Our tool design builds on top of a multi-cloud software layer to make experiments executable on different compute cloud infrastructures. This design decision, which is similar to the multi-Grid approach in [7], in combination with using Chef as a configuration management solution, similar to the use of cfengine in [16], provide a good starting point for future work on evaluating the dependence of experimental results on the cloud environment, a general problem known from related work on experimentation in virtualized computing testbeds and compute clouds [12, 21].

6. CONCLUSION

We present a new approach to experimentation with distributed database systems which is characterized by three basic concepts. First, experiment modules capture experiment plans in an declarative way. Second, an Elastic Lab can load and execute experiment modules. The lab allocates and de-allocates experiment resources rapidly on-demand in a public compute cloud. Third, collaborating on experiment modules and sharing observation data is supported by cloud-based collaboration services. We implement the concepts and evaluate our tools with an initial case study by benchmarking the performance and scalability of the distributed database system Apache Cassandra on top of Amazon EC2. In our experience, the approach of experiment automation is promising, however, requires a solid solution for configuration management of complex distributed systems. We are working towards offering a set of out-of-the-box reproducible experiments which can be executed on heterogeneous operating systems and compute clouds.

7. ACKNOWLEDGEMENTS

The work presented in this paper was performed in the context of the Software-Cluster project EMERGENT (software-cluster.org). It was funded by the German Federal Ministry of Education and Research (BMBF) under grant number "01IC10S01". The authors assume responsibility for the content. We thank the reviewers for their valuable feedback and suggestions which have helped to improve the quality of this paper.

8. REFERENCES

[1] A. Avetisyan, R. Campbell, I. Gupta, M. Heath, S. Ko, G. Ganger, M. Kozuch, D. O'Hallaron, M. Kunze, T. Kwan, K. Lai, M. Lyons, D. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J.-Y. Luke, and H. Namgoong. Open Cirrus: A Global Cloud Computing Testbed. *Computer*, 43(4):35 –43, 2010.

[2] R. A. Bailey. *Design of Comparative Experiments*. Cambridge Series in Statistical and Probabilistic Mathematics. University of London, 2nd edition, 2008.

[3] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing. How is the Weather tomorrow? Towards a Benchmark for the Cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, DBTest '09, pages 9:1–9:6, New York, NY, USA, 2009. ACM.

[4] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 241–252, New York, NY, USA, 2010. ACM.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[6] 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, pages 143–154, New York, NY, USA, 2010. ACM.

[7] C. Dumitrescu, I. Raicu, M. Ripeanu, and I. Foster. Diperf: An automated distributed performance testing framework. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 289–296, Washington, DC, USA, 2004. IEEE Computer Society.

[8] J. Elson and J. Howell. Handling flash crowds from your garage. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 171–184, Berkeley, CA, USA, 2008. USENIX Association.

[9] B. A. Frigyik, A. Kapila, and M. R. Gupta. Introduction to the Dirichlet Distribution and Related Processes. Technical Report 206, Uni Washington, 2010.

[10] S. Gaisbauer, J. Kirschnick, N. Edwards, and J. Rolia. Vats: Virtualized-aware automated test service. In *QEST*, pages 93–102, 2008.

[11] Z. Ganon and I. Zilbershtein. Cloud-based performance testing of network management systems. In *Computer Aided Modeling and Design of Communication Links and Networks, 2009. CAMAD '09. IEEE 14th International Workshop on*, pages 1 –6, june 2009.

[12] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Cluster, Cloud and Grid Computing (CCGrid),*

2011 11th IEEE/ACM International Symposium on, pages 104 –113, may 2011.

[13] M. Klems, D. Bermbach, and R. Weinert. A runtime quality measurement framework for cloud database service systems. In *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology*, pages 38–46. IEEE, Conference Publishing Services (CPS), September 2012.

[14] A. Lakshman and P. Malik. Cassandra: structured storage system on a P2P network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, PODC '09, pages 5–5, New York, NY, USA, 2009. ACM.

[15] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SoCC '11, New York, NY, USA, 2011.

[16] A. Pavlo, P. Couvares, R. Gietzel, A. Karp, I. D. Alderman, M. Livny, and C. Bacon. The nmi build & test laboratory: Continuous integration framework for distributed computing software. In *LISA*, pages 263–273, 2006.

[17] Priyanka, I. Chana, and A. Rana. Empirical evaluation of cloud-based testing techniques: a systematic review. *SIGSOFT Softw. Eng. Notes*, 37(3):1–9, May 2012.

[18] S. Ruby, D. Thomas, and D. H. Hansson. *Agile Web Development with Rails*. The Pragmatic Programmers. The Facets of Ruby Series. Pragmatic Bookshelf, Raleigh, NC, 3. ed. edition, 2009.

[19] C. C. Ruiz Sanabria, O. Richard, B. Videau, and I. Oleg. Managing Large Scale Experiments in Distributed Testbeds. Research Report RR-8106, INRIA, Oct. 2012.

[20] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual Appliances for Deploying and Maintaining Software. In *Proceedings of the Seventeenth Large Installation Systems Administration Conference*, October 2003.

[21] Y. Wang, M. J. Rutherford, A. Carzaniga, and A. L. Wolf. Automating experimentation on distributed testbeds. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 164–173, New York, NY, USA, 2005. ACM.