Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds

Deepal Jayasinghe, Galen Swint, Simon Malkowski, Jack Li, Qingyang Wang, Junhee Park and Calton Pu.
Center for Experimental Research in Computer Systems, Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765, USA.
{*deepal, swintgs, zmon, jack.li, qywang, jhpark, calton*}*@cc.gatech.edu*

***Abstract*—Cloud computing is an emerging technology paradigm that revolutionizes the computing landscape by providing on-demand delivery of software, platform, and infrastructure over the Internet. Yet, architecting, deploying, and configuring enterprise applications to run well on modern clouds remains a challenge due to associated complexities and non-trivial implications. The natural and presumably unbiased approach to these questions is thorough testing before moving applications to production settings. However, thorough testing of enterprise applications on modern clouds is cumbersome and error-prone due to a large number of relevant scenarios and difficulties in testing process. We address some of these challenges through Expertus—a flexible code generation framework for automated performance testing of distributed applications in Infrastructure as a Service (IaaS) clouds. Expertus uses a multi-pass compiler approach and leverages template-driven code generation to modularly incorporate different software applications on IaaS clouds. Expertus automatically handles complex configuration dependencies of software applications and significantly reduces human errors associated with manual approaches for software configuration and testing. To date, Expertus has been used to study three distributed applications on five IaaS clouds with over 10,000 different hardware, software, and virtualization configurations. The flexibility and extensibility of Expertus and our own experience on using it shows that new clouds, applications, and software packages can easily be incorporated.**

***Keywords*-Aspect, Automation, Clouds, Code Generation, Datacenter, EC2, Emulab, IaaS, Multi-Tier, Open Cirrus, Performance, Scalability, Testing, Template.**

I. Introduction

Exhaustive application configuration testing on Infrastructure as a Service (IaaS) clouds is motivated by three key observations. First, and perhaps least surprising, the performance of two identical configurations can be vastly different on two different platforms [27]. In fact, we have observed that a particular best practice (i.e., highest performance) configuration can become the worst-performing configuration by virtue of nothing more than a change in platform (e.g., migrating the RUBBoS application from Emulab [11] to Amazon EC2 [16]). Second, the same applications can exhibit non-trivial performance characteristics due to a combination of complicating factors even in a single platform [28]. Third, application performance can vary unexpectedly with changes to a relatively small set of factors. Such factors may include garbage collection, network driver overhead, and context-switching overhead and that are often subtle and not directly controllable by users.

While manual and *ad hoc* (i.e., partially scripted) approaches for application testing are being used heavily in traditional data centers, modern clouds require adoption of automated testing strategies to mitigate cloud complexities. Automation removes the error prone and cumbersome involvement of human testers, reduces the burden of configuring and testing distributed applications, and accelerates the process of reliable applications testing. One way to build an automation framework is through code generation, yet building a flexible and extensible code generator for distributed systems remains a significant research challenge. Environmental and design changes pressure the input language to change and evolve. Similarly, the generated code (output) often needs customization to a range of software, cloud and operating systems, also typically due to unyielding market and technology evolution. This constant evolutionary pressure of input and output formats has so far limited the practical life span of code generators developed for distributed system software.

In this paper we discuss Expertus, a code generation framework that we have developed to automate the testing of large scale distributed applications in IaaS cloud. More precisely, Expertus is designed to automate performance and scalability testings in cloud platforms. Expertus was created by extending our previous work on code generation for Infopipes [21]. Expertus takes an experiment specification (application parameters, target cloud, and test scenarios) as an input and creates the resource (e.g., scripts) to fully automate the testing process (i.e., deployment, configuration, execution and data collection). Expertus is designed as a multi-pass compiler which leverages template-driven code generation to modularly incorporate different software applications and clouds. As documented by our results, the flexibility and extensibility of Expertus have enabled cloud experiments at a scale that is beyond manual test execution.

Expertus makes significant strides towards realizing flexible and scalable application testing for today's complex cloud environments. For example, we have used Expertus in over 500 different hardware configurations and over 10,000 different software configurations. Concretely, Expertus was used to test software applications deployed on over 100,000 computing nodes in five different cloud environments (i.e., Emulab, Amazon EC2, Open Cirrus [14], Georgia Tech university cluster, and Wipro [15] private cloud) using three representative applications (i.e., RUBBoS [12], RUBiS [13], and Cloudstone [7]). As more cloud offerings become available, we expect the number and variety of test scenarios to increase by another order of magnitude with our current Expertus version.

Expertus is designed for flexibility and extensibility. A template developer can extend Expertus to support new clouds, applications, and software packages with only a few tem-

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.98

plate line changes. For example, with 8.21% of template line changes, we were able to support Amazon EC2 (here after EC2) once we had support for the Emulab cloud. This 8.21% caused a 25.35% change in the generated code for an application with 18 compute nodes. Similarly, 9.33% template lines required to support Open Cirrus cloud, but, the changes to the generated code were 26.91%. Switching from the RUBBoS to the RUBiS required only a 5.66% template change.

Expertus is very efficient when carrying out many repetitive or slightly different experiments. A user can change the specification from one experiments to another by switching platform settings, software settings, and computing clouds with minimum effort. For example, with only one line change (i.e., `<param name=''platform'' value=''EC2''/>`) and by changing the IP address of the Emulab's experiment specification, it is possible to generate scripts to automate the same Emulab experiment on EC2. Similarly, automated scripts can be generated for a new RUBiS experiment from a preexisting RUBBoS experiment specification with less than 20 line modifications.

The remainder of this paper is structured as follows. In Section II we discuss distributed software testing and challenges. We present the code generation framework in Section III. We quantify the significance of our approach and illustrate its feasibility in Section IV. Finally, we provide a discussion of related state of the art approaches in Section V, and we conclude the paper with Section VI.

II. Distributed Testing and Challenges

Experiment challenges: Performance testing for enterprise applications consists of multiple closely related scenarios by varying a few configurable (preferably one) parameters at a time. For example, to find the maximum achievable throughput for an e-commerce application on EC2 may consist of multiple experiments where the difference between any two would be the number of concurrent client workloads (e.g., 1000 users vs. 2000 users). In fact, two different software or hardware configurations may produce the same throughput [27], [28], thus increasing the testable configuration space. Also, components of a distributed application can interact in complex ways, yielding a virtually infinite number of possible states. In reality, evaluating all possible configurations is practically infeasible for a vast majority of systems. Thus, the key challenge is to maximize the amount of configurations that can be tested with limited time and human resource constraints.

Application challenges: In distributed software testing the applications should start efficiently and in a provably correct order by simultaneously enforcing serialization constraints and leveraging the distributed system's inherent parallelism. This process may be trivial for applications with a fewer nodes; in contrast, for applications with many nodes and multiple application components (e.g., multi-tier applications), the process becomes challenging mainly due to complex dependencies (e.g., database URL, load balancers). For example, a web application with Apache as the load balancer and Tomcat as the application server requires modification to the Apache configuration whenever application servers are added or removed. Similarly, changing the DBMS causes to change configurations in application server (e.g., JDBC, URL).

Cloud challenges: Testing software systems in today's cloud environments introduces many new challenges; first, selecting the most appropriate cloud from many cloud offerings is a non-trivial task; second, migrating an application between two clouds is a complex, time consuming and error prone task; third, communication, coordination, synchronization, monitoring and complete management challenges; lastly, the dynamic nature of the cloud introduces extra complexity. For example in a traditional data center, the IP address of a given server is assumed to be constant indefinitely; hence, once an application is deployed with pre-configured IP addresses then that configuration can be used for multiple test cases; however, in computing clouds, whenever we rent new computing instances, they come with a fresh set of IP addresses. Thus, if we want to run multiple test cases, we would need to modify the configuration scripts after each test case to account for the new IPs which is a non-trivial task for large applications.

III. Expertus

Expertus is designed to automate large scale distributed experiment studies in IaaS clouds with the goal of addressing three challenges discussed in Section II. We tackle experiment challenges by automating complete experiment process i.e., application deployment and configuration, experiment execution, data collection, and data analysis. The cornerstone of experiment automation is the code generation, which generates all the necessary resources for automation process. Addressing application challenges, and to handle complex application dependencies; a clear separation of different levels (e.g., application vs. platform) and types (e.g., deployment vs. runtime) of dependencies is needed. Our solution to this problem is multi-stage code generation, where one type/level of dependency is handled at one stage. To address platform challenges and other cross cutting requirements (e.g., monitoring), the generated code needs to be modular i.e., a developer should be able to make controlled changes to the code. The modularity is achieved by using aspect oriented programming (AOP) techniques inside the code generator.

A. Flexibility and Extensibility through XML and XSLT

Expertus is architectured to operate on top of XML and XSLT. Use of XML provides the code generator a high degree of extensibility. This stems from XML's simple, well-defined syntax requirement and ability to accept arbitrary new tags thereby bypassing the overhead encountered when managing both XSLT templates and AOP. For example a template can add an arbitrary element to the intermediate XML; however, unless the processing code is written to process that new tag, the newly added tag remains untouched.

XSLT transformation is the process of converting an XML document into another document through the use of XSL. Typically, XSLT converts an XML document into another XML

document (e.g., HTML) or any other type of document. Expertus consists of two types of templates namely `Resource templates` and `Aspect templates`. Former is used to generate application/platform independent part of a resource and latter is used to modify (weave) the generated resource for the target application/platform (e.g., Emulab vs. EC2).

B. Multi-Stage Code Generation

The code generator adopts a compiler approach of multiple serial transformation stages. The intuition is to have better extensibility and flexibility by dividing the large problem into smaller pieces and processing them one at a time. For example, it processes the application related transformations at one stage and cloud related transformations at another stage. As shown in Figure 1, at each stage Expertus takes an XML document and produces another XML document through XSLT transformation. Expertus treats the first and last stage differently compared to the rest; in the first stage, it takes the experiment specification as the input and in the final stage it generates automation resources to the file system (in lieu of an intermediate XML).

The number of stages in the transformation phases is determined by the experiment, application, software stack, operating system, and cloud. For an example, if RUBBoS application is configured with Apache, Tomcat, and MySQL running on Fedora in EC2, then the transformation phase consists of 8 stages. The first stage is to transform the specification to a domain specific document. Then, there is a unique transformation stage for each software component, application, operating system, and cloud (i.e., Apache, Tomcat, MySQL, Fedora and EC2). The order of different stages is determined by dependencies between different components (Tomcat stage appears before Apache, if latter depends on former). The final stage is to transform intermediate XML to generated code.

At each stage Expertus uses the intermediate XML document created from the previous stage as the input to the current stage. It uses the intermediate XML file to retrieve the names of the templates that are needed for the current stage and transforms the intermediate XML document, which then creates another XML document (Transformed XML in Figure 2). If needed, AOP `pointcuts` will be added to the intermediate XML during the transformation phase. Consequently, Aspect Weaver is used to weave such `pointcuts` . Aspect Weaver processes the `pointcuts` through `Aspect templates` and creates the woven XML (Figure 2). Finally, the woven file is written to the file system through the file writer. More precisely, the automation scripts are written to the file system during the final stage of the pipeline and at each other stage it generates an intermediate XML and calls the next stage in the pipeline. The complete process in a single stage is shown in Figure 2.

Figure 3 illustrates how code generation works in practice and shows code snippets from an XSLT template, an intermediate XML and a generated script. The highlighted lines in the XSLT template indicate parameter names it should

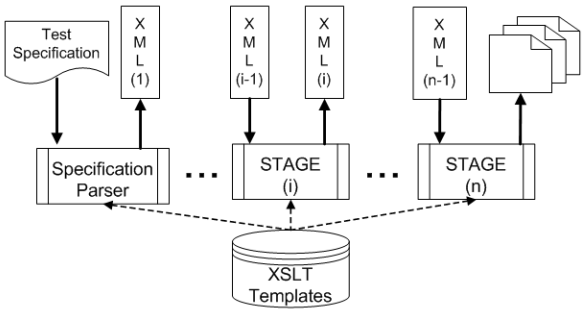


Fig. 1. Multi-Stage Code Generation Process

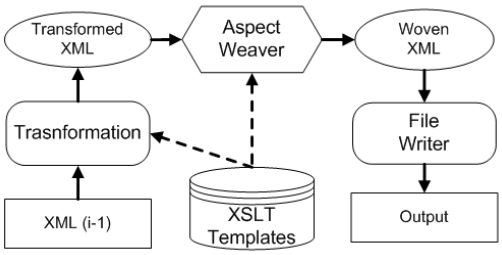


Fig. 2. Transformations Steps within a Single Stage

use from the XML document to create the output. The XML file is highlighted with the corresponding parameter values, and those parameters can either come from user inputs (specification) or generated from a previous stage by another XSLT template. Finally, the generated script is also highlighted with parameter values. As illustrated in the figure, the XSLT template has extracted the values from intermediate XML and created the output XML file. We use this simple logic to generate resources to automate complex test scenarios.

C. Modularity

One of the key challenges in experiment automation is to handle the variances and continuous evolution in applications, software packages, operating systems, and clouds. For example, in Emulab, a user can communicate between two nodes by typing `SSH node2`; in contrast, in EC2 it has to be `SSH -i gsg-keypair node2`. Similarly, for older Apache versions `mod_jk` has to be placed in `jk/native2`, however newer versions require it to be in `jk/`. Likewise, when doing performance testing, crosscutting features like monitoring and logging are important factors and may need to change from one experiment to another. Thus, to handle variances and crosscutting features, Expertus itself has to provide better modularization. Expertus achieves this modularization by leveraging AOP techniques.

The `Resource templates` are created by identifying output variances (e.g., SSH communication) and crosscutting features. Each identified variance is embedded into `Resource templates` as `pointcuts` (namespace qualified XML elements). `Aspect templates` are created by embedding the `advices` (code that is run at the joint points) which are needed to support the `pointcuts`. At the transformation phase `Resource templates` add `joint points` (well defined points in the execution flow) to

XSLT Template

```
.....
cd <xsl:value-of select="//params/env/param[@name='OUTPUT_HOME']/@value"/>
source set_elba_env.sh
echo "  INSTALLING RUBBOS on $HOSTNAME"

mkdir -p <xsl:value-of
select="//params/env/param[@name='RUBBOS_TOP']/@value"/>
# install RUBBoS
tar xzf <xsl:value-of
select="//params/env/param[@name='SOFTWARE_HOME']/@value"/>/<xsl:value-of
select="//params/env/param[@name='RUBBOS_TARBALL']/@value"/> --
directory <xsl:value-of
select="//params/env/param[@name='RUBBOS_TOP']/@value"/>

.....
make
sudo make install
echo "  DONE INSTALLING RUBBOS on $HOSTNAME"
```

Intermediate XML

```
<xtbl name="Rubbos" version="0.1">
 <instances><params><env>
   <param name="OUTPUT_HOME" value="/opt/rubbos/output"/>
   <param name="SOFTWARE_HOME" value="/opt/softwares"/>
   <param name="RUBBOS_TOP" value="/mnt/rubbos"/>
   <param name="RUBBOS_TARBALL " value="RUBBoS-servlets.tar.gz"/>
   <param name="MYSQL_PORT" value="3313"/>
   .....
<env><params>
<tomcat-conf>
   <param name="maxThreads" value="330"/>
   .....
</tomcat-conf>

<instance name="BENCHMARK" type="benchmark_server">
    .....
</instance>
```

Generated Script

```
.....
cd /opt/rubbos/output
source set_elba_env.sh
echo "  INSTALLING RUBBOS on $HOSTNAME"

mkdir -p /mnt/rubbos
# install RUBBoS
tar xzf /opt/softwares/RUBBoS-servlets.tar.gz --directory=/mnt/rubbos
.....
make
sudo make install
echo "  DONE INSTALLING RUBBOS on $HOSTNAME"
```

Fig. 3. Generating Output/Intermediate XML with XSLT Transformation

the intermediate XML and at the weaving phase `Aspect templates` transform intermediate XML into woven XML by applying correct `advices`. If new `pointcuts` are needed we can easily modify the `Resource templates` and add (or change) the `Aspect templates` to have the corresponding logic to process the new crosscutting features.

A simple aspect weaving scenario is illustrated in Figure 4, where a cloud specific aspect template is used to generate a script to have different `SSH` communication for EC2 and Emulab. In the figure, the XSLT template is highlighted with the `pointcuts`, intermediate XML is highlighted to illustrate the `joint points`, and finally woven code with applied advice is highlighted for EC2 and Emulab.

D. Experiment Automation

Here we provide a brief overview to experiment automation with Expertus. The automation process is illustrated in Figure 5, and brief description about each item is given below:

1) Create experiment specification with the application, software packages, cloud and experiments. This includes complex system dependencies and semantics of the application (components, # of nodes, parameters), cloud (e.g., URLs, user name, password, key-pair, node types), software (e.g., thread pool, maxClients), and experiments (e.g., type, load configurations).
2) Use Expertus and generate scripts. To start the experiments the user remotely connects via (`SSH`) to the control node and executes the `Main` script. The `Main` script uses other scripts and drives the complete automation which in fact consists of multiple stages.
3) Platform configuration sets up the target cloud this may include starting and configuring computing instances, mounting the file systems, creating users, setting user permissions and installing the operating system libraries.
4) Application deployment is to deploy the target application on the configured cloud (e.g., downloading/copying the installation packages).
5) Enterprise systems are inherently complex due to component dependencies, so application configuration is needed to correctly configure them.
6) `Main` script runs the test plan (6), which in fact consists

XSLT Template

```
.....
tar xzf <xsl:value-of
select="//params/env/param[@name='SOFTWARE_HOME']/@value"/>/<xsl:value-of
select="//params/env/param[@name='JAVA_TARBALL']/@value"/> --
directory=<xsl:value-of select="//params/env/param[@name=' RUBBOS_TOP
']/@value"/>
<esl:add-aspect name="ssh" xmlns:esl="http://expertus.cc.gatech.edu" />
```

Intermediate XML

```
....
tar xvf /opt/software/jdk1.5.tar.gz -directory=/mnt/rubbos
<esl:add-aspect name="ssh" xmlns:esl="http://expertus.cc.gatech.edu" />
```

Woven XML For EC2

```
.....
tar xvf /opt/software/jdk1.5.tar.gz -directory=/mnt/rubbos
ssh -i gsg-keypair ec2-184-73-86-137.compute-1.amazonaws.com
```

Woven XML For Emulab

```
.....
tar xvf /opt/software/jdk1.5.tar.gz -directory=/mnt/rubbos
ssh node2.expertus.emulab.net
```

Fig. 4. Handling Cloud Variances through Aspect weaving

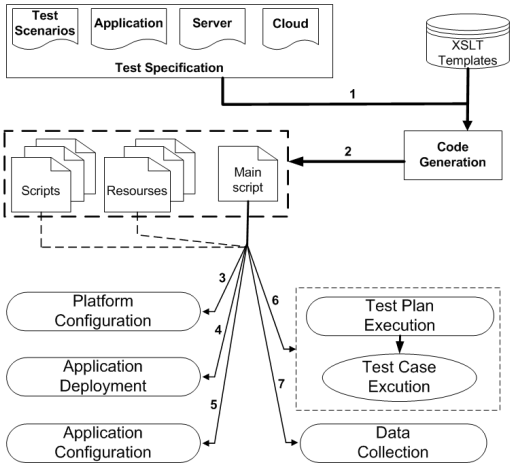


Fig. 5. Automated Testing Process with Expertus

of multiple iterations. In order to produce consistent and reproducible results, all the dirty data from previous test cases have to be removed before starting a new one.

7) Upload the resource monitoring and performance data to the data warehouse.

IV. Experiences

We have been using Expertus extensively and actively to perform a large number of experiments with various configurations; through these, we have thoroughly tested our framework and improved it to have better usability, extensibility and flexibility. In this section we demonstrate our experiences of using Expertus on experiment automation, supporting new clouds, new application and new software packages.

A. Usability of the Tool

Here we look at how quickly a user can change an existing specification to run the same experiment with different settings (e.g., MySQL Cluster [10] vs. C-JDBC [22]), the same experiment on different clouds (e.g., Emulab vs. EC2), the same experiment with different numbers of nodes (e.g., two vs. four app servers), or entirely different applications (e.g., RUBBoS vs. Cloudstone).

In our analysis, we created a specification (say `a.xml`) to run the RUBBoS application on Emulab with a total of 16 nodes and generated automated resources through Expertus. We then changed `a.xml` to generate automated scripts for EC2 which required only a single line change (i.e., `<param name=''platform'' value=''EC2''/>`) in `a.xml` and a modification of the IP address; even though the lines changed in the configuration file is small, the changes to the generated code is non-trivial and significantly larger. Our results are illustrated in Figure 6(a). We followed the same procedure and modified `a.xml` to change the database middleware from C-JDBC to MySQL Cluster. This change required only 36 lines (mostly MySQL Cluster specific settings), but, as shown in Figure 6(a) the differences in generated code were huge. Similarly, with 4 lines changes to `a.xml`, we were able to move from 2 to 8 Application servers. Furthermore, with only 52 template line changes we were able to extend the support from RUBBoS to Cloudstone.

B. Generated Script Types and Magnitude

The biggest advantage of our approach becomes visible when automating experiments for complex applications. The amount of resources being generated depends on the application (e.g., RUBBoS, RuBiS), software packages (e.g., Tomcat, JBOSS), deployment platform (e.g., Emulab, EC2), number of experiments, number of servers, and number of configuration parameters. To show the significance of the generated code, six different hardware configurations (on Emulab) were selected and counted the number of lines generated for each configuration. Our results are shown in Figure 6(b); as shown in the figure, when the number of nodes becomes larger the size of the generated code becomes very significant as does the difference between the generated code. Magnitude of generated code implies two factors: first, it shows the effectiveness of our approach, and second, it shows the difficulties of manual approaches. For example, to perform an experiment with 43 nodes requires around 15K lines of shell scripts and writing all of those manually is a non-trivial task.

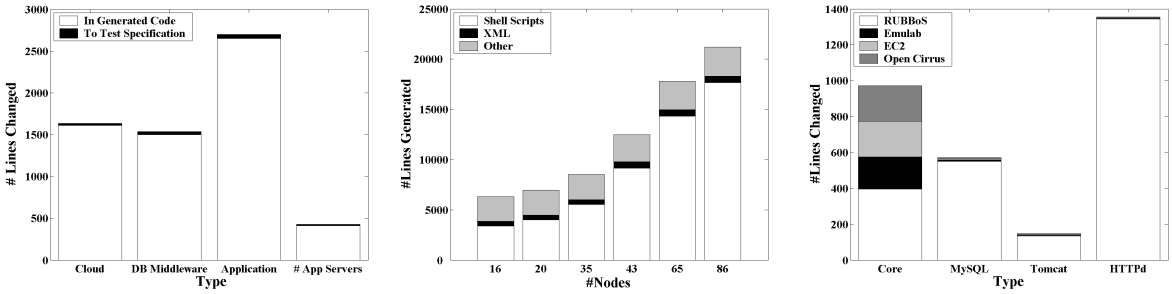
C. Richness of the Tool

Richness is considered in two dimensions: 1) magnitude of completed experiments; 2) amount of different software packages, clouds, and applications it supports. Expertus has been used over three years to conduct a large number of experiments spanning five clouds (Emulab, EC2, Open Cirrus, Wipro, and Elba), three applications (RUBBoS, RUBiS, and Cloudstone), five database management systems (C-JDBC, MySQL Cluster, MySQL, PostgreSQL, Oracle), various resource monitoring tools (dstat, sar, vmstat), and different types and numbers of nodes. A high level summary of supported software packages, platforms, and applications are shown in Table I with the number of template lines for each item.

TABLE I
Supported Software packages, Clouds and Applications

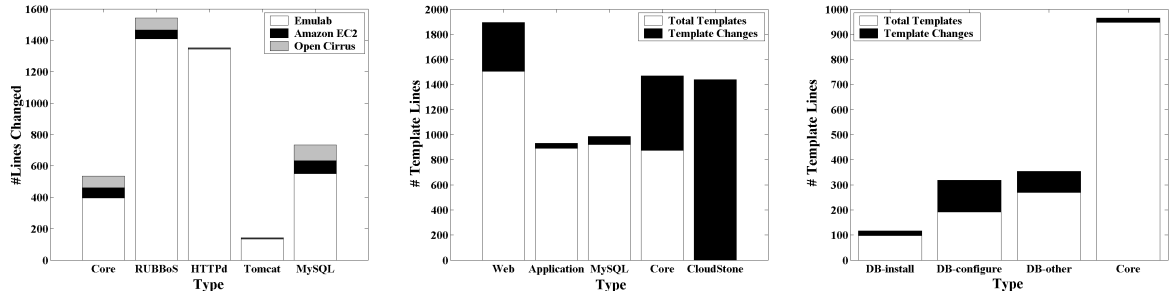
	Platform					Application		
Type	EC2	Emulab	Open Cirrus	Elba	Wipro	RUBBoS	RUBiS	Cloudstone
Apache HTTPd	√	√	√	√	√	√	√	√
Tomcat	√	√	√	√	√	√	√	√
MySQL	√	√	-	√	-	√	√	√
MySQL Cluster	√	√	√	-	√	√	-	√
C-JDBC	√	√	-	√	-	√	√	-
Postgres	-	√	-	√	-	√	√	-
Sequoia	-	√	-	√	-	√	-	-
PHP	√	√	√	√	√	-	-	√
JOnAS	-	√	-	-	-	-	√	-
Core	√	√	√	√	√	√	√	√

Table II provides a high level summary (only the results we used for our publications) of different experiments performed using RUBBoS, RUBiS, and Cloudstone applications. An experiment refers to an execution of a particular workload against a combination of hardware and software configurations. In the table, nodes refer to the total number of machines we have used during our experiments. We calculated the number of nodes by multiplying number of experiments by the number of nodes for each experiment. Configurations mean the number of different software and hardware configurations that have been studied. Experiments per week is to give an idea of average number of experiments executed in one week. Typically, an experiment execution takes one hour which is the aggregated duration of: reset time, start time, sleeping time, ramp-up time, running time, ramp-down time, stop time, and data copy time. Hence, as an example in Eumlab we have conducted approximately 132 hours of experiments per week. Finally, human efforts refers to the actual human time spent on experiment configuration (i.e., creating or modifying experiment configuration file). Hence, as shown in the table, our approach significantly reduces the required human efforts and helps to explore configuration space quickly.



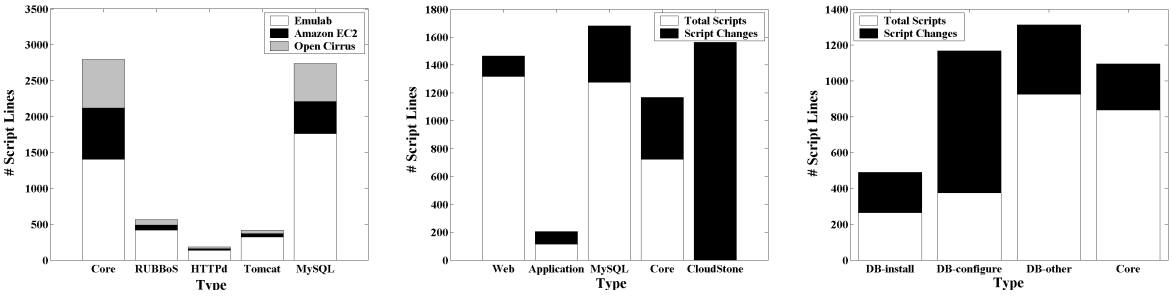
(a) # Changes: Specification vs. Generated (b) #Nodes vs. # Lines generated (c) RUBBoS To RUBiS

Fig. 6. (a) # Lines changed in experiment specification vs. generated code, when changing cloud, database, application and # app servers; (b) Magnitude of generated code when increasing the # nodes in the experiment; (c) #Lines (template) changed for adding a new application(RUBBoS to RUBiS)



(a) Adding New Clouds (b) Adding New Applications (c) Adding New Database

Fig. 7. Template Line Changes: (a) Moving from Emulab to EC2 and Open Cirrus (figure shows total # lines for Emulab); (b) Supporting CloudStone from RUBBoS; (c) Moving from C-JDBC to MySQL cluster



(a) Adding New Clouds (b) Adding New Applications (c) Adding New Database

Fig. 8. Changes to Generated Code: (a) Emulab to EC2 and OpenCirrus; (b) RUBBoS and CloudStone; (c) C-JDBC and MySQL Cluster

TABLE II
NUMBER OF TEST SCENARIOS PERFORMED WITH EXPERTUS

Type	Emulab	EC2	Open Cirrus	Elba	Wipro
Experiments	9215	1436	480	3567	120
Nodes	102687	25848	4987	9863	430
Configurations	392	86	28	163	8
Experiments Per Week	132	34	120	77	14
Human Effort (in minutes)	87	110	44	47	51

D. Extensibility and Flexibility

1) Supporting New Clouds: Our approach is fully realized when trying to support new clouds; we can use existing templates and only implement the absolutely necessary pieces, which is usually a quick and easy process. Traditionally, using manual approaches to support new clouds means writing a set of scripts to deploy, configure, execute the test cases, and finally to collect the data, which can be a cumbersome and tedious task. Here, we show the extensibility of Expertus by counting the amount of changes we performed to support RUBBoS application on EC2 and Open Cirrus once the tool already has support for RUBBoS on Emulab. Not surprisingly, we observed many differences among the three clouds that make migrating applications from one cloud to another a non-trivial task.

To get a good understanding of how easy it is to extend Expertus to support new clouds and the significance of the generated code, we show our own experiences on migrating RUBBoS application from Emulab to EC2 and then to OpenCirrus. Initially we developed XSLT templates for the Emulab cloud and then used the same templates to extend the applications to new clouds. We first calculated the total number of templates line in Emulab and then counted the total number of templates lines modified (i.e., added, removed, and updated). Our results are shown in Figure 7(a), as illustrated in the figure once we have implemented templates for Emulab,

the amount of changes required to support EC2 was very small (less than a 10% template change). To better understand the differences we have divided the changes into five categories: 1) Changes to core templates (cloud independent resources), 2) Web server (e.g., Apache HTTPd), 3) Application server, 4) Database server and, 5) RUBBoS specific templates. As shown in the figure we have made very little changes to HTTPd and Apache Tomcat; in contrast, we changed 10% of the total lines in the core templates, which was mainly due to the differences in EC2. Likewise, moving from Emulab to Open Cirrus involved a similar amount of changes, and our results are shown in the same figure (Figure 7(a)).

To show the significance of template changes on generated code, we created an experiment specification (say `a.xml`) with 16 nodes and generated code for Emulab. Next, we modified that for EC2 and generated the code and calculated the differences between the two sets of codes. To highlight the differences we divided the generated code into the same five categories as in templates. Our results are illustrated in Figure 8(a). As shown in the figure, a small change in template makes a huge difference in the generated code. Likewise we modified the `a.xml` to support Open Cirrus and generated the code and analyzed them. As shown in Figure 8(a), the difference between Emulab and Open Cirrus is similar to that of EC2.

2) Supporting New Applications: Here we show the amount of template changes required to support two different applications (RUBiS and Cloudstone) once we have support for RUBBoS. Both RUBBoS and RUBiS follow the same application architecture and deployment procedure whereas Cloudstone requires a slightly different deployment topology.

Supporting RUBiS concurred of following modifications to the templates: 1) changed the web server load balancing URL, 2) built the RUBiS web application (in lieu of RUBBoS), 3) specified the dataset location, and 4) changed the workload executor. Supporting RUBiS required creating three new templates for the client side. The amount of change to the template on three different clouds (Emulab, EC2 and Open Cirrus) is shown in Figure 6(c). As shown in the figure we made only a 30% change to the core template (the original RUBBoS template) to integrate RuBiS into Expertus.

Supporting Cloudstone required more changes compared to that of RUBiS. In Cloudstone Web server became the key entity whereas in RUBBoS/RUBiS it was just a load balancer. Importantly, we were able to reuse most of the templates for the application and database tier. In Figure 7(b) we have given the number of template lines we changed for this task, in fact we have divided the template changes into to five categories to highlight the changes. Due to the major difference in RUBBoS and CloudStone we had to create a set of new templates for CloudStone that aggregates around 1600 templates lines. Nevertheless, more than 95% of those lines are contributed from new XML and PHP configurations needed for CloudStone and rest we reused from the existing templates. Next, to show the significance of templates changes on generated code, we created a specification with the same number of nodes for both RUBBoS and CloudStone. We generated the resources for Emulab and measured the differences, as shown in Figure 8(b) we observed a significanct difference in Web, App and Database tiers.

3) Supporting a New DB Middleware: To illustrate the flexibility of Expertus, we show the amount of template changed to support MySQL Cluster [10] database given that Expertus already has support for C-JDBC [22] (with MySQL). Concretely, we calculated the number of templates lines changed during migration from C-JDBC to MySQL Cluster. This involved a few template changes and the addition of a few new templates to handle three node types in MySQL Cluster.

MySQL Cluster is implemented by a combination of three types of nodes (Management, SQL and Data nodes) as compared to a single type in C-JDBC. In addition, with C-JDBC, it is easy to scale-out (i.e., to add more database nodes). It is just a matter of deploying a new database server and copying the data dump. In contrast, scaling-out MySQL Cluster requires a completely different procedure (i.e., database partitioning) and needs to repopulate the database using SQL statements. Additionally, C-JDBC has only one database URL (i.e., URL of C-JDBC), but MySQL Cluster contains multiple SQL nodes which requires that each application server to be configured to communicate with each SQL node. As a result of all these changes; moving from C-JDBC to MySQL Cluster involves around a 24% change to database templates. Nevertheless, once the template is created, we can easily scale MySQL Cluster up to any level with simple configuration changes.

Figure 7(c) illustrates the total number of template lines, lines changed, and lines changed in generated code. To better understand, we have categorized the changes into four types: DB install, DB-configure, DB-other (start, stop, reset), and core changes. We then changed the specification for C-JDBC with 8 database servers and generated resources for Emulab cloud. We analyzed the generated code in the same four categories. Next, we modified the experiment specification for MySQL Cluster and generated code for Emulab. As expected, we observed huge differences in the generated code and this difference become non-trivial when increasing the number of nodes. Our results are shown in Figure 8(c), as shown in Figure 7(c) and Figure 8(c) a small template change caused a huge difference in the generated code.

V. Related Work

Experimentation is an essential approach used in both academia and industry to gain an understanding of system behavior, formation and testing of hypotheses, system configuration and tuning, obtaining solution information, and resolving performance bottlenecks. There are many open source and commercial tools for configuration management and small scale application deployment [17]–[20]. Yet, there have been relatively few efforts aimed at building software tools to reduce complexity associated with large-scale testing of distributed applications [1]–[6].

Steafan et al. [29] introduced Cloud9, a platform for automated testing of software by using scalable parallelization

of symbolic execution on clusters of commodity hardware, to help cope with path explosion. Same authors extend their works and introduced the concept of testing as a service (TaaS) [31]. The authors claimed that the combination of recent advances in test automation and the availability of compute clouds can offer unprecedented levels of testing quality. In our approach, we go beyond and provide software tools to empirically evaluate complex distributed system, but, both approaches share the same goal of making experiment evaluation easier.

Our project parallels several others using XML and XSLT for code generation. For example, the SoftArch/MTE [25] and Argo/MTE teams have also had favorable experiences using XML + XSLT generators to "glue" off-the-shelf applications together [23], [25]. Likewise, XML+XSLT is advocated for code generation tasks in industry as well [24]. One of the closest to our approach is Weevil [8], which also focuses on workload generation and script creation. In fact, later they observed some of the limitations in their approach and proposed four enhancements to explore richer scenarios and to obtain results with greater confidence [5]. A DSL based approach for software development and deployment is presented by Krzysztof et al. [30]. To our knowledge, these efforts have not explored the issues of extensibility, flexibility, or modularity of test automation that is presented here in this paper.

VI. Conclusion

Expertus, our experiment automation framework for distributed software systems in IaaS clouds, has been developed to minimize human errors and maximize efficiency when conducting large scale distributed experiments. More importantly, it helps move forward with the rapid evolving cloud computing paradigm by automating exhaustive software testing in a timely manner. We have applied our own experiences and extended research work to improve the usability of the framework. In this paper we illustrated how our approach can be used to deploy and test several representative applications on heterogeneous cloud platforms. More generally, our evaluation results show the feasibility, extensibility and usefulness of our approach in cloud testing. We have also identified several limitations of our tool and continue our research work to address them. Our final goal is to make Expertus available for public use. Our tool can significantly reduce the deployment and configuration cost of running distributed test scenarios in today's production cloud environments, which indicates great promise for the future.

Acknowledgment

This research has been partially funded by National Science Foundation by IUCRC/FRP (1127904), CISE/CNS (1138666), RAPID (1138666), CISE/CRI (0855180), NetSE (0905493) programs,and gifts, grants, or contracts from DARPA/I2O, Singapore Government, Fujitsu Labs, Wipro Applied Research, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

References

[1] Y Ioannidis, M Shivani, G Ponnekanti. ZOO: A Desktop Experiment Management Environment. In *Proceedings of the 22nd VLDB Conference*, Mumbai(Bombay), India, 1996.

[2] K L. Karavanic, B P. Miller. Experiment management support for performance tuning. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, Mumbai(Bombay), India, 1996.

[3] R Prodan, T Fahringer. ZEN: A Directive-based Language for Automatic Experiment Management of Distributed and Parallel Programs. In *ICPP 2002*, Vancouver, Canada.

[4] R Prodan, T Fahringer. ZENTURIO: An Experiment Management System for Cluster and Grid Computing. In *Cluster 2002*.

[5] Y Wang, A Carzaniga, A L. Wolf. Four Enhancements to Automated Distributed System Experimentation Methods. In *ICSE 2008*.

[6] S Babu, N Borisov, S Duan, H Herodotou, V Thummala. Automated Experiment-Driven Management of (Database) Systems. In HotOS 2009, Monte Verita, Switzeland.

[7] A Fox, W Sobel, H Wong, J Nguyen, S Subramanyam, A Sucharitakul, S Patil, D Patterson. Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement tools for Web 2.0. In *CCA 2008*.

[8] Y. Wang, M.J. Rutherford, A. Carzaniga, and A. L. Wolf. Automating Experimentation on Distributed Testbeds. In *ASE 2005*.

[9] Cooper, B.F, Silberstein, A, Tam, E, Ramakrishnan, R, Sears R. Benchmarking Cloud Serving Systems with YCSB. In *SOCC*, 2010.

[10] MySQL Cluster. http://www.mysql.com/products/database/cluster/.

[11] Emulab - Network Emulation Testbed. http://www.emulab.net.

[12] RUBBoS: Bulletin board benchmark. http://jmob.objectweb.org/rubbos.html.

[13] RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[14] Open Cirrus. https://opencirrus.org/.

[15] Wipro Technologies. www.wipro.com/.

[16] Amazon Elastic Compute Cloud. http://aws.amazon.com.

[17] Distributed IT Configuration Management. http://cfengine.com.

[18] Puppet Data Center Automation Solution. http://www.puppetlabs.com/.

[19] Smart Framework for Object Groups. http://www.smartfrog.org/.

[20] Software Testing Automation Framework. http://staf.sourceforge.net.

[21] G Swint, C Pu, C Consel, G Jung, A Sahai, W Yan, Y Koh, Q Wu. Clearwater - Extensible, Flexible, Modular Code Generation. In *Automated Software Engineering (ASE 2005)*.

[22] E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-jdbc: Flexible database clustering middleware. In *Proceedings of the USENIX 2004 Annual Technical Conference*, 2004.

[23] Cai, Y., Grundy, J., and Hosking, J. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool. In *ASE 2004*.

[24] Sarkar, S. Model driven programming using XSLT: an approach to rapid development of domain-specific program generators In *www.XML-JOURNAL.com. August 2002*.

[25] Grundy, J., Cai, Y., and Liu, A. SoftArch/MTE: generating distributed system test-beds from high-level software architecture descriptions. In *ASE 2001*.

[26] Malkowski, S., Hedwig, M., and Pu, C. Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. In *IISWC 2009*, Austin, TX, USA.

[27] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., and Pu, C. Variations in performance and scalability when migrating n-tier applications to different clouds. In *CLOUD 2011*.

[28] Wang, Q., Malkowski, S., Jayasinghe, D., Xiong, P., Pu, C., Kanemasa, Y., Kawaba, M., and Harada, L. Impact of soft resource allocation on n-tier application scalability. In *IPDPS 2011*.

[29] S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel Symbolic Execution for Automated RealWorld Software Testing. In *EuroSys 2011*.

[30] K. Sledziewski, B. Bordbar and R. Anane. A DSL-based Approach to Software Development and Deployment on Cloud. In *2010 24th IEEE* International Conference on Advanced Information Networking and Applications.

[31] G. Candea, S. Bucur, and C. Zamfir. Automated Software Testing as a Service. In *SOCC 2011*.