Automated Scalability Testing of Software as a Service

Paulo Moura
Department of Computer Science
University of São Paulo - Brazil
pbmoura@ime.usp.br

Fabio Kon
Department of Computer Science
University of São Paulo - Brazil
fabio.kon@ime.usp.br

***Abstract*—Scalability has been studied in several areas of Computer Science and scalability testing and evaluation of contemporary software systems is an active topic. However, most of the times, these activities are still performed in a predominantly *ad hoc* fashion. There are a few tools to automate this process, but they present several restrictions about what systems can be tested and how to evaluate scalability. In this paper, we introduce a flexible and extensible framework for automated scalability testing of software offered as a service and propose to evaluate the scalability using hypothesis tests. Additionally, we argue that, instead of stating if a system is scalable or not, we should find out how it could scale better.**

I. Introduction

Scalability and performance testing and evaluation have been an active research topic [1] Even though we can find some practitioners yet manually testing and evaluating the scalability of systems [2], [3] or using tools or scripts in part of the process [4], [5], [6], there are efforts in automating scalability tests [7], [8], [9], [10], [11]. These tools, generally, can be used to define scalability tests, perform them, and collect metrics. Some include support for analysis by plotting graphs or calculating scalability ratio metrics.

Although scalability has been studied in parallel computing, distributed computing, databases, and service-oriented architecture (SOA) [12], still today there are many researchers using vague definitions that simply state scalability concerns with the relation between system load and performance [12], [13], [11], [14], [6]. Also, these definitions usually limit scalability to a linear relation between those characteristics. Duboc et al. [15] states that limiting scalability to performance and linear relations are misconceptions and states that "scalability is the ability of a system to satisfy quality goals to levels that are acceptable to its stakeholders when characteristics of its application domain and design vary over operational ranges". It is also important to note that when considering the scalability of computer systems, not only the software must be taken under consideration, but also the infrastructure where it runs on [16], [17].

However, these definitions do not precisely quantify scalability; a better way to understand how to assess the scalability of a system could be considering scalability metrics. A set of them is listed by Zheng Li et al. [18]. From those metrics, we highlight the aggregate performance comparison, speedup, and degradation. The first consists of comparing the aggregated performance of the system when receiving different workloads, usually while resources are also scaled; the second is about the relative performance gain when more infrastructural resources are added to the system, comparing to a fixed workload; and the third concerns the performance loss for a given system configuration, when the workload increases. Additionally, other metrics have been proposed to compare the scalability of two different configurations of a system [8], [7].

In this work, we introduce the Scalability Explorer, which consists of a framework for automated scalability testing. Although it is flexible and extensible, currently our focus is on scalability tests of software as a service, more specifically, software implemented as web service compositions, as part of the CHOReOS[1] project.

In the remainder of this paper, we discuss other frameworks for automatic testing of distributed systems in Section II; introduce our framework in Section III and explain the facilities it provides for scalability analysis in Section IV; then we show a proof of concept where this framework was used to perform scalability tests in a distributed matrices multiplication system in Section V and discuss the results in Section VI; finally, Section VII brings the final remarks and discusses future work.

II. Related Work

This session contains a description of other frameworks for automated scalability testing and summarize some of their main characteristics in Table I.

STAS (Scalability Testing and Analysis System) [7] is a system that provides scalability analysis of algorithms and systems. The first step, when using STAS, is to run a speed measurement module that collects computing power of the nodes. Then a pre-analysis component takes the system source code and performs a workload analysis, based on user hints. After, node sets are constructed based on node computing power, doubling the computing power of the set at each iteration. STAS can run the tests, collecting and storing the execution time. The workload size is obtained from the previous analysis. Finally, the isospeed-e scalability metric is calculated, to verify whether the system is scalable. This metric considers the computing power of the nodes and the execution time and compares two configurations, giving their

[1] www.choreos.eu

978-1-4673-6161-3/13 © 2013 IEEE

AST 2013, San Francisco, CA, USA

scalability ratio, where 1 means perfect scaling. This system can be used to analyze the scalability of MPI programs and is said to be easily configurable to support PVM or HPF. As can be seen, this framework has some limitations in how tests can be performed since it only allows the number of nodes to be scaled by a fixed ratio. Also there is only one possibility to analyze the measurement results, the isospeed-e metric, and, although this metric is a generalization of a previous metric to consider heterogeneous environments, it is still only suitable for homogeneous systems.

Jogalekar and Woodside [19], [8] presents a scalability framework that is based on a scaling strategy for scaling up or down a system, controlled by a scale factor that rules the value of a set of variables. They introduce a metric based on throughput, performance, and resource usage. As with the isospeed-e, this metric gives a scalability ratio relating two configurations. The main drawback of this kind of metric is that, in practice, it is very hard to achieve the value 1, which is considered a perfect scalability. So it is common to take on an arbitrary threshold to say if a system is scalable. Also, as they only work for a pair of configurations, for a set with more than two configurations, the metric can attest the system is scalable when comparing each neighboring configuration but gives a bad value when comparing further ones. It means that, for a system with linear degradation, the choice of the configurations used to evaluate its scalability influences the analysis.

ASTORIA [9] is a framework for automated performance and scalability testing for Rich Internet Applications. This framework scales load by using virtual machines that simulate user interaction with the system through a GUI-less browser and logs the response time of each performed action. Throughput is also calculated. It assumes the provisioning of the testing resources is handled automatically. The interpretation of the collected metrics is left outside the system.

Almeida et al. [10] presents a methodology and a framework for testing P2P applications combining functional and non-functional tests due to the volatility and scale variability of P2P. The tests are written in Java similarly to jUnit tests, using annotations and assertions. The framework applies scalability testing by changing the scales of the application and validating their functionalities. However, each execution scenario must be specified, which can make the test specification cumbersome.

Klems et al. introduced a framework specific to test distributed databases deployed on clouds [11]. It includes mechanisms for set up and decommission of cloud database service systems, to perform horizontal and vertical scaling, and some database specific resources, such as tuning configuration. It also allows deployment of distributed workload executors. This framework can plot graphs to support scalability analysis, but does not include any specific scalability metrics.

TABLE I
COMPARISON OF SCALABILITY TEST FRAMEWORKS

Framework	Supported systems	Horizontal scalability	Vertical scalability	Analysis
STAS	MPI	doubling computer power	no	isospeed-e metric
Jogalekar's	not specified	according to the scale factor	according to the scale factor	Jogalekar's metric
ASTORIA	RIA	no	no	not specified
Almeida's	P2P	defining scenarios with different number of peers	no	assertions
Klems'	databases distributed on clouds	specified by user	specified by user	graphs

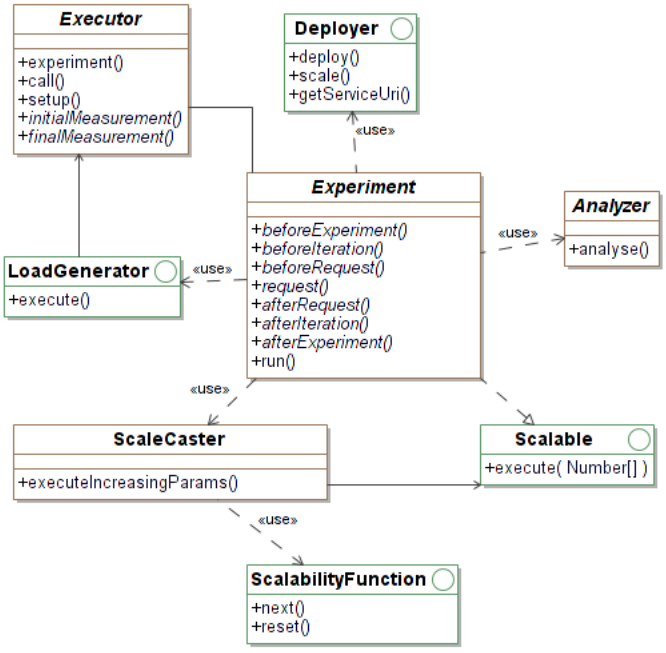

Fig. 1. Scalability Explorer Structure

III. SCALABILITY EXPLORER

Our Scalability Explorer is a Java framework conceived to be flexible and contain common facilities and methods, so that one can write a scalability test with very little code specific to the interaction with a particular system. Its structure is shown in Figure 1. `Experiment` is a template for test implementation. It implements the `Scalable` interface and, thus, the `ScaleCaster` can repeatedly invoke it, varying the values of some of its parameters, as defined by a `ScalabilityFunction`.

The framework also includes other classes and interfaces with which the `Experiment` interacts. A `LoadGenerator` defines the frequency and distribution of the requests over time and interacts with an `Executor`, responsible for triggering the requests and collecting metrics. A `Deployer` can be used to set up the services that compose the evaluated system. We provide a `Deployer` implementation that is a client of

the Enactment Engine [20], which manages nodes in a cloud and deploys services on them. Also, an `Analyzer` can be provided to evaluate the experiment.

An experiment consists of a sequence of iterations in which a number of requests is sent to the system under test. The `ScaleCaster` increases the scaled parameters at each iteration and the requests are performed according to the `LoadGenerator` distribution. The `Deployer` is called at the beginning of the experiment to enact the system and before each iteration to scale the system. After the last iteration, the `Analyzer` receives the data collected during the experiment. This flow can be seen in Figure 2. The `Experiment` class provides methods invoked before and after (1) the whole experiment, (2) each iteration, and (3) each request, as well as the method that performs the request whose scalability will be evaluated. All these methods can be overridden by subclasses so that the framework is customized to specific experiments.

Three sample subclasses of `Experiment` are already provided. `ArchitectureExperiment` scales the parameter that is passed to `Deploy.scale()`, allowing to update the system architecture. It could be used for horizontal or vertical scalability, depending on the method implementation. Another extension, the `WorkloadExperiment`, scales the frequency of requests. Lastly, the `ScalabilityExperiment` scales both parameters.

Finally, to support the easy interaction with software systems provided as a service on the Web, our Scalability Explorer was built as part of the Rehearsal [21] testing framework, which provides facilities to simplify tests of web services. It includes a dynamic web service client, a proxy to intercept messages for evaluation and a web service mocking component. Nevertheless, since the interaction with the system under test is implemented programmatically in Java, the Scalability Explorer can be used with any system that can be invoked from a Java application.

IV. Scalability Analysis

As seen in the last section, this Scalability Explorer provides a hotspot to include a component to analyse the experiments. Further, some analyzers are included in the framework. We begin this section showing how some common techniques of scalability analysis are available in the framework. Then we present an implementation that performs a statistical analysis of the experiment data.

One common evaluation performed in scalability tests is the system speedup [22], [23]. It consists of the evaluation of the relative performance the system delivers while the number of nodes running the system grows. The degradation/slowdown [23], [24], [11], another common evaluation technique, is obtained by increasing the workload for a fixed system architecture. Speedup and Degradation analysis can be performed with the same analyzer: `RelativePerformance`. The difference is in how the experiment is performed. For instance, using this analyzer in a `WorkloadExperiment` would show a degradation chart, while using it in an `ArchitectureExperiment` would show speedup.

It is also common to evaluate scalability through the aggregate performance [2], [25], plotting a chart with the aggregated value of a performance metric. This kind of analysis can be made using a `SimpleChart` analyzer, which can be used with an aggregation function such as arithmetic mean or percentile.

Although useful, these techniques lack precision to state whether a system properly scaled. Thus, another analyzer is provided to address that through a statistical test. The procedure, known as Analysis of Variance or ANOVA, is one of the most widely used statistical techniques. In spite of its name, this technique is not concerned with analyzing variances, but rather with analyzing variation in means. ANOVA is used to test the hypotheses, when we have several populations, with no difference in the populations means. But, to assure the reliability of the test, it is required that the sample size be meaningful [26].

We provide two analyzers to properly address these tests. The `SampleSizeEstimation` can be used to calculate the appropriate sample size for an experiment, and the `ANOVATest` to perform the Analysis of Variance. Thus, the user can execute a pilot experiment to estimate the sample size before running the experiment with ANOVA analysis, assuring its reliability. The last experiment, then, will state that the system scaled if it had maintained equivalent performance in all the iterations.

V. Evaluation

We illustrate the use of the Scalability Explorer with an experiment in which we evaluate a web-service-based distributed matrices multiplication system deployed on the Amazon EC2 cloud computing platform. The system consists of a coordinator and a set of multiplication services. The coordinator splits the matrices across the services, which, in their turn, return the multiplication of their parts to be reunited by the coordinator. The goal of this experiment is to identify the ratio between the matrices size and the number of multiplication services to keep the system performance constant. We executed tests consisting of a sequence of requests to multiply two square matrices and measured the response time in milliseconds. We scaled the workload (matrices size) and the architecture (number of multiplication services running on different cloud nodes). All the tests were executed in five iterations, with one request per second, in Amazon m1.small instances, which have 1.7GiB of memory and 1 EC2 Compute Unit power (equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor)[2].

The first step was to verify the degradation and speedup of the system. For degradation, we started with 200x200 matrices and increased the dimensions in 200 for each iteration. Figure 3 depicts the performance loss graph of the system as generated automatically by the framework. The speedup was

[2]http://aws.amazon.com/ec2/instance-types

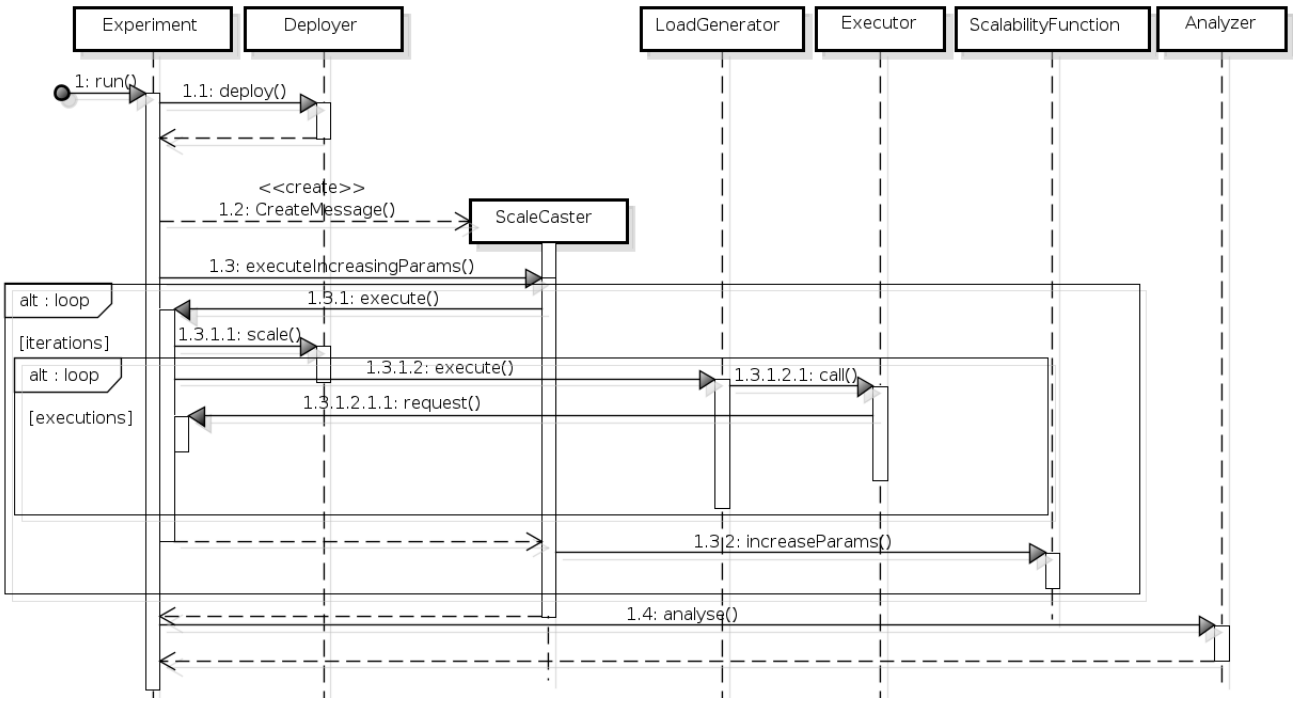

Fig. 2. `Experiment` execution flow

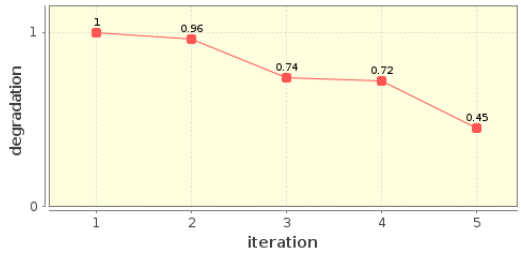

Fig. 3. Distributed matrices multiplication degradation

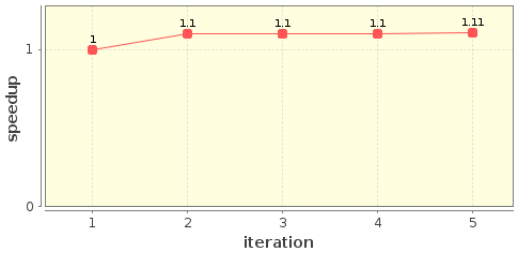

Fig. 4. Distributed matrices multiplication speedup

measured with 1000x1000 matrices, beginning with one node and adding one more per iteration. Figure 4 shows the result. As can be seen, the system performance was more affected by the variation in workload than the variation in the number of nodes. These experiments performed 30 requests per iteration and also calculated the appropriate sample size for a 95% confidence level in ANOVA tests. As we obtained a maximum value of 27, we remained performing 30 requests per iteration for the rest of the experiments.

Those tests give an idea of how the system behaves with different workloads or architectures but, to have a better view of the relation between these dimensions, we conducted other tests varying both together. The speedup and degradation results suggest that the system would not scale properly with the chosen load and architectural facets varying in the same proportion, because the speedup rate was smaller than the degradation rate. As we are dealing with two dimensional matrices, a linear growth in their dimension results in a quadratic growth in the input size. Hence, we ran a test linearly scaling the matrices size (test1). For so, while the number of nodes was linearly increased from 1 to 5 (by 1 per iteration), the matrices dimensions was increased in a lower rate so that its square (the matrix size) was linearly increased. The aggregate performance of this test can be seen in Figure 5. With this chart we could consider that, although not constant, the variation in performance was acceptable, but we aim to avoid this kind of subjective analysis. Instead, we also submitted the measured performances to ANOVA analyzer, which considered the performances of each iteration not equivalent.

The issue is that even though the data size and computational power were growing in the same proportion, the computation complexity was not. A matrix multiplication is an operation of cubic complexity. On the other hand, the computation is split among a number of nodes. Considering this, another test (test2) was performed with the matrices dimension growing according to the function $dimension = \sqrt[3]{S * n}$ where S was fixed to 8,000,000, that is the number of operations executed

TABLE II
MATRICES DIMENSIONS

iteration	test1	test2
1	200	200
2	283	252
3	346	288
4	400	317
5	447	341

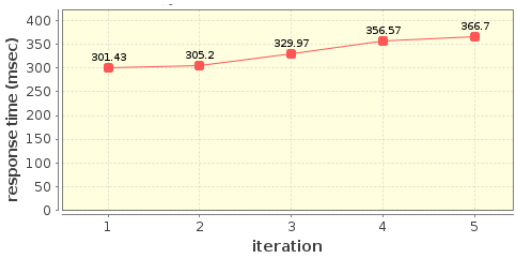

Fig. 5. test1 - Aggregate performance

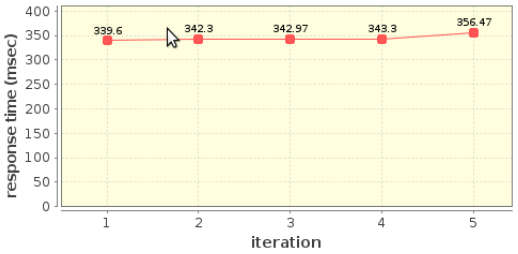

Fig. 6. test2 - Aggregate performance

in the multiplication of 200x200 matrices, and n is the number of nodes. The aggregate performance of these scalability tests is shown in Figure 6 and the performances were considered equivalet by ANOVA.

Table II shows the matrices dimensions used in each iteration of *test1* and *test2*.

Figure 7 shows sample code of a scalability test written for our framework: the speedup test performed during this evaluation. It extends `ArchitectureExperiment` to promote the nodes scaling. The method `beforeExperiment` is executed once when the test starts to create a web service client to interact with the system coordinator. Before each iteration, the framework invokes the deployer to perform the scaling. Just after each scaling, and before performing the requests, `beforeIteration` is executed and it registers the new services in the coordinator. The `request` method invokes a multiplication. In the `main` method, the test is configured and executed.

VI. DISCUSSION

Considering that the scalability is not necessarily linear, it might be important to identify at which rate a system scales to properly allocate resources as needed. In the previous section, we have shown one way it could be done, supported by the framework we are developing. In our example, we performed two scalability tests with linear architectural scaling: the first

```
package experiments;

import eu.choreos.vv.aggregations.Mean;

public class MMArchExperiment extends ArchitectureExperiment {

    private static final int NUMBER_OF_REQUEST_PER_STEP = 30;
    private static final int NUMBER_OF_REQUEST_PER_MINUTE = 60;
    private static final int NUMBER_OF_STEPS = 5;
    private static final long MATRIX_SIZE = 1000;
    private static final int INITIAL_RESOURCE_QUANTITY = 1;

    private static final String
        DEPLOYER_URI = "http://choreos.ime.usp.br:9102/clouddeployer/";

    private int step;

    WSClient client;

    @Override
    public void beforeExperiment() throws Exception {
        String url = this.getDeployer()
                .getServiceUri("multiplyMatrices");
        url = url.substring(0, url.length() - 1);
        client = new WSClient(url + "?wsdl");
        step = 0;
    }

    @Override
    public void beforeIteration() throws Exception {
        step++;
        for (int i = INITIAL_RESOURCE_QUANTITY; i > 0; i--) {
            String name = "MultiplyMatrixService"
                    + (step * INITIAL_RESOURCE_QUANTITY - i);
            String uri = this.getDeployer().getServiceUri(name);
            uri = uri.substring(0, uri.length() - 1);
            System.out.println("adding " + uri);
            client.request("addServer", uri);
        }
    }

    @Override
    public void request() throws Exception {
        client.request("multiplyMatrices", "" + MATRIX_SIZE, "false");
    }

    public static void main(String[] args) throws Exception {
        MMArchExperiment s = new MMArchExperiment();

        s.setAnalyser(new ComposedAnalysis(new SampleSizeEstimation(.95),
                new AggregatePerformance("distributed matrix multipl.",
                        new Mean())));
        s.setDeployer(new MMDeployer(0, DEPLOYER_URI));
        s.setInititalResoucesQuantity(INITIAL_RESOURCE_QUANTITY);
        s.setRequestsPerMinute(NUMBER_OF_REQUEST_PER_MINUTE);
        s.setNumberOfSteps(NUMBER_OF_STEPS);
        s.setNumberOfRequestsPerStep(NUMBER_OF_REQUEST_PER_STEP);

        s.run("speedup");
    }

}
```

Fig. 7. Speedup test example

increased the matrices dimensions in a way that kept the data size growth linear and the second increased matrices size so that the computational load grew linearly. The obtained results have shown that the system performance was more susceptible to processing capacity. But it is important to note that the system is not limited to its software, but also includes the infrastructure where it runs on, and the analyses are restricted to the tested scenario.

Law [16] states that, as it is difficult to quantify all the facets that affect scalability, only a subset or single facet is considered for architecture and for the workload, and also that, in practice, there might be a limit in the architecture/workload ratio beyond which the system no longer scales. This limit could, though, be related to the evaluated facets and the system could still scale if another facet is scaled. As an example, one could find a scaling relation between workload and bandwidth

that only works up to a limit. After that limit, the bottleneck might have changed to CPU or memory capacity.

For more accurate analysis, measuring resources usage, such as processor, memory, and network traffic is important to identify bottlenecks, especially in heterogeneous systems. We are currently working on this for a future version of our Scalability Explorer.

Notwithstanding, our goal in this work is to prevent subjectivity in scalability analysis, introducing the hypotheses tests. For a simple comparison, the isospeed-e [7] calculated for the second and fourth iteration, comparing with the first, gave ratios of 2.1 and 3.4 respectively. First, the values are too far from the desired because this metric assumes linear scalability. Second, these values are very different, highlighting that the metric is very sensitive to the scaling performed in the configuration.

At last, roughly, we can say that the matrices multiplication system achieved a linear scalability between the number of nodes and the system computational load. The issue is that it might be not always practical to find out or to consider a pair of workload and architecture facets that scale linearly.

VII. Conclusion

Although an active research topic, scalability tests are not yet well standardized, as well as scalability itself. A deeper investigation on quantification of scalability is required to allow such standardization. In this work, we considered a broader definition [15], which is appealing because it repeals the requirement that scalability must be linear, which is not leverageable for many systems due to their complexity. According to that, a scalability test should not be concerned with whether a system is scalable or not, but how it scales. Yet, this definition lacks a quantification.

Our next step in the evolution of this Scalability Explorer is to provide a better evaluation of the relation between workload and architecture. For that, we are going to automate the collection of resource usage from the nodes where the system runs and conduct statistical analysis over that data. Moreover, with the growing trend for Cloud Computing [27], we also intend to provide metrics related to economics [18], reliability, and availability.

This framework is available as open source software under the Mozilla Public License at http://github.com/choreos/choreos_v-v .

Acknowledgment

This research has received funding from HP Brasil under the Baile Project and from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet).

References

[1] W.-T. Tsai, Y. Huang, and Q. Shao, "Testing the scalability of SaaS applications," *IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 1–4, Dec. 2011.

[2] Y. Luo, S. Luo, J. Guan, and S. Zhou, "A RAMCloud storage system based on HDFS: Architecture, implementation and evaluation," *Journal of Systems and Software*, vol. 86, no. 3, pp. 744 – 750, 2013.

[3] M. Andreolini, M. Colajanni, and M. Pietri, "A scalable architecture for real-time monitoring of large information systems," in *IEEE Second Symposium on Network Cloud Computing and Applications*, 2012.

[4] J. Mizher, M. Dunham, L. Lu, and Y. Xiao, "Scalability of oat," in *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, 2005, p. 48.

[5] Z. Kenan, Y. Baolin, C. Xi, and C. Li, "A highly scalable evaluation system with service-based architecture," in *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, vol. 1, march 2010, pp. 230 –233.

[6] M. Kamra and R. Manna, "Performance of cloud-based scalability and load with an automation testing tool in virtual world," in *IEEE Eighth World Congress on Services (SERVICES)*, june 2012, pp. 57 –64.

[7] Y. Chen and X.-H. Sun, "Stas: A scalability testing and analysis system," in *Cluster Computing, 2006 IEEE International Conference on*, sept. 2006, pp. 1 –10.

[8] P. Jogalekar and M. Woodside, "Evaluating the scalability of distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 6, pp. 589 –603, jun 2000.

[9] N. Snellman, A. Ashraf, and I. Porres, "Towards automatic performance and scalability testing of rich internet applications in the cloud," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 30 2011-sept. 2 2011, pp. 161 –169.

[10] E. de Almeida, G. Sunye, Y. Le Traon, and P. Valduriez, "A framework for testing peer-to-peer systems," in *19th International Symposium on Software Reliability Engineering*, nov. 2008, pp. 167 –176.

[11] 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*. IEEE, September 2012, Inproceedings, pp. 38–46.

[12] W.-T. Tsai, Y. Huang, X. Bai, and J. Gao, "Scalable Architectures for SaaS," *IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pp. 112–117, Apr. 2012.

[13] 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*, ser. DBTest '09. ACM, 2009, pp. 9:1–9:6.

[14] X. Guerin, W. Tan, Y. Liu, S. Seelam, and P. Dube, "Evaluation of Multi-core Scalability Bottlenecks in Enterprise Java Workloads," *IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 308–317, Aug. 2012.

[15] L. Duboc, E. Letier, and D. S. Rosenblum, "Death, Taxes, & Scalability," *IEEE Software*, vol. 27, no. 4, pp. 20–21, Jul. 2010.

[16] D. R. Law, "Scalable means more than more: a unifying definition of simulation scalability," in *Winter Simulation Conference Proceedings*, vol. 1, dec 1998, pp. 781 –788 vol.1.

[17] D. Jayasinghe, G. Swint, S. Malkowski, J. Li, Q. Wang, J. Park, and C. Pu, "Expertus: A generator approach to automate performance testing in IaaS clouds," in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, june 2012, pp. 115 –122.

[18] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a catalogue of metrics for evaluating commercial cloud services," in *ACM/IEEE 13th International Conference on Grid Computing (GRID)*, sept. 2012, pp. 164 –173.

[19] P. Jogalekar and M. Woodside, "Evaluating the scalability of distributed systems," in *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, vol. 7, jan 1998, pp. 524 –531 vol.7.

[20] F. Kon *et al.*, "D3.2.2 : CHOReOS middleware implementation," The CHOReOS Consortium, Tech. Rep., 2012.

[21] F. Besson, P. Moura, F. Kon, and D. Milogicic, "Rehearsal: a framework for automated testing of web service choreographies," in *Tool session of the 3rd Brazilian Conference on Software: Theory and Practice*, sep 2012.

[22] J. Rehr, F. Vila, J. Gardner, L. Svec, and M. Prange, "Scientific computing in the cloud," *Computing in Science Engineering*, vol. PP, no. 99, p. 1, 2011.

[23] B. Cornea, J. Bourgeois, T. Nguyen, and D. El-Baz, "Scalable performance predictions of distributed peer-to-peer applications," in *IEEE 14th International Conference on High Performance Computing and Communication, IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, june 2012, pp. 193 –201.

[24] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of Windows Azure," *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, p. 367, 2010.

[25] H. Liu and S. Wee, "Web server farm in the cloud: Performance evaluation and dynamic architecture," in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 369–380.

[26] G. Casella and R. Berger, *Statistical Inference*, 2nd ed., ser. The Wadsworth & Brooks/Cole Statistics/Probability series. Wadsworth & Brooks/Cole Advanced Books & Software, 2002.

[27] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009.