SABER: An Infrastructure-Agnostic Benchmark Tool for Elasticity Evaluation on Cloud Foundry Based Platforms

Wagner Marques, Felipe Rubin, Paulo de Souza, Rômulo de Oliveira, and Tiago Ferreto
Graduate Program in Computer Science
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
Email: wagner.marques@edu.pucrs.br

***Abstract*—Elasticity can be considered as one of the main features of cloud computing. It can be described as the ability to dynamically provision (or de-provision) resources to meet an application's demand. Cloud-native platforms support this feature, allowing applications to be vertically or horizontally scaled. However, evaluating the performance of elasticity in cloud-native platforms is not a trivial task. These platforms tend to focus on providing higher abstractions to their customers, which typically results in the absence of low-level metrics that are required for measuring and evaluating elasticity performance. Therefore, this paper presents SABER, an infrastructure-agnostic benchmark tool for elasticity evaluation of Cloud Foundry based platforms. SABER collects low-level metrics regarding elasticity typical procedures (e.g., allocation and de-allocation of resources) and can be used to evaluate elasticity on different cloud infrastructures. Furthermore, we present an evaluation of SABER running on Pivotal Application Service, a Cloud Foundry based platform provided by Pivotal Web Services.**

***Keywords*—Elasticity, cloud-native platforms, Pivotal Cloud Foundry, Cloud Computing and Benchmarking.**

I. Introduction

Cloud computing has emerged with popularity and providing many benefits, such as elasticity and resiliency. Cloud computing is defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources, including networks, servers, storage, applications, and services that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. As such, virtualization is presented as a key component of cloud computing, enabling the pooling and allocation of hardware resources dynamically [2].

The virtualization technology is used to unify and manage all the available resources according to users' demand [3]. Currently, instance replication is the most widely used method for providing elasticity in cloud applications [4]. Elasticity is also considered an essential characteristic of cloud computing [1], and it is defined as the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [5].

Due to these characteristics and benefits, many organizations are using cloud computing [6].

The way we develop software is changing to meet emerging cloud computing technologies. Cloud native is a new approach that has emerged as a solution to extract the maximum potential from cloud computing [7] [8] [9]. Contrary to a monolithic application, it is decomposed into smaller modules called microservices, and each microservice runs in a container in order to optimize resource utilization [10] [11]. Cloud-native applications are designed to run on cloud-native platforms, which runs on top of a cloud computing infrastructure.

Many elasticity mechanisms for cloud computing environments have been proposed in the literature [4]. However, monitoring performance and collecting metrics related to elasticity from cloud-native platforms is not a trivial task. Platforms like OpenShift [12], Heroku [13] and Cloud Foundry [14] usually do not provide data related to the elasticity (e.g., the provisioning time of instances). Besides, Cloud Foundry is infrastructure-agnostic, which means it can run on top of different public and private cloud infrastructures, providing different elasticity results. To the best of our knowledge, there are still no services or tools aimed at evaluating the elasticity of Cloud Foundry based platforms running over different infrastructures, such as Google Cloud Platform, Amazon Web Services, Microsoft Azure or VMware vSphere. **This paper presents SABER, an infrastructure-agnostic benchmark tool for elasticity evaluation of Cloud Foundry based platforms. As such, we can also highlight the following contributions:**

- SABER allows operators to get insights on how changing the underlying infrastructure affects the elasticity delivered by Cloud Foundry platforms.
- SABER takes advantage of Cloud Foundry's built-in mechanisms to provide a set of metrics related to elasticity without requiring any extra configuration.

The remainder of this paper is organized as follows: Section II provides an overview of elasticity, and cloud-native applications and platforms. Section III presents SABER architecture design and features, along with the metrics and

outputs description. Section IV presents an evaluation scenario used to validate SABER. This section also presents the results and a discussion where we provide insights on the evaluation results. Related work is provided in Section V. Conclusions and future work are presented in SectionVI.

II. Background

There are several definitions in the literature for elasticity. Al-Dhuraibi et al. [15] define elasticity as a system's ability to add or remove computing resources of an application to adapt to real-time load variation, such as CPU cores, memory, virtual machines, and container instances. Similarly, Li et al. [16] define elasticity as the ability of a system to adapt to workload changes that may occur in a short period of time. Aisopos, Tserpes, and Varvarigou [17] also define elasticity as the ability of a provider to allocate the amount of memory, CPU, and disk space required for a specific job dynamically, and therefore, its performance and capabilities may vary based on the demand for an application's computational resources. There are two types of elasticity:

- **Horizontal elasticity**, also known as scale-out, which consists of adding or removing instances associated with an application [18].
- **Vertical elasticity**, also known as scale-up, which consists of adjusting the computing resources available for an application based on the demand [15].

Although most of modern cloud platforms adopt several elasticity mechanisms, the effectiveness of these mechanisms can be directly affected by specific infrastructure characteristics. For instance, in a scenario where two infrastructure providers have the same amount of resources for applications, one may deliver elasticity more effectively than the another by using more powerful hardware components such as processors that operate in higher frequencies. Therefore, there is a concern around elasticity benchmark tools that provide means for evaluating the elasticity of cloud-native platforms based on specific infrastructure-level characteristics.

In Cloud Foundry applications, the main component for delivering elasticity is called App-Autoscaler [19]. This component orchestrates monitoring and provisioning mechanisms available for Cloud Foundry in order to adjust applications resources according to prior specified CPU, memory, throughput, and response time requirements.

III. SABER Elasticity Benchmarking Tool

SABER aims to provide a set of metrics related to the elasticity of applications running on cloud-native platforms based on Cloud Foundry, which is one of the most prominent cloud-native platforms nowadays. Figure 1 illustrates the SABER architecture design and process flow. The overall structure of the architecture design is based on a client-server architecture: The **client** is a local machine running SABER benchmark, and the **server** refers to cloud plataform, where a sample application is deployed. In the server-side, SABER provides a sample application implemented in Ruby on Rails that is deployed in the Cloud-native platform for evaluating it.

On the client-side, SABER uses an input file, where the user can define execution parameters. SABER internally uses the Siege Benchmark to stress the application deployed in the cloud by increasing its resources demands and generate the elasticity metrics. Siege Benchmark is an open-source tool developed to stress an application by simulating concurrent users and gathering metrics such as throughput and latency. While in execution, SABER monitors application's resource demands using the Cloud Foundry API, obtaining detailed information.

Before running SABER, the user must have an environment correctly setup. So, the user must first run "Deploy Script" to deploy the sample application and configure App Autoscaler instance, which then binds this instance to the sample application one, and finally, creates a memory rule to scale the application. This script requires the user to fill out his platform credentials. Right after setting up the testing environment, the user must provide the information (input data) needed to run SABER correctly, which includes the application route, the environment specification, and testing cycles definition, as described in III-A.

After this, the Siege benchmark begins stressing the application and getting the metrics. When Siege execution is completed, SABER displays the results and also generates a CSV (Comma-separated values) file with them. The CSV file can be used to get the resource demands and the resource supply in each instant analyzed by the benchmark tool, making it possible to create different charts and use it in data analysis. In order to provide an overview of all the results, the benchmark tool also automatically generates a chart reporting the resources demand and the resources supply.

A. Input File

SABER configuration is based on a JSON-based input file called *input.json*. This file contains necessary environment information, such as the name of the application (parameter *app_name*), the location where the application is installed (parameter *app_env*), and the route to access the application (parameter *app_route*).

The input file is also used to define how the benchmark will stress the application. SABER defines a set of demand cycles, where the user can set the number of simulated concurrent users that will make requests during a period of time for each cycle. Each cycle has the following parameters: *time*, which defines for how long time the concurrent users of this cycle will perform requests (cycle duration in seconds); *concurrent_users*, which is the number of users that will be simulated by the Siege benchmark during this cycle. SABER requires at least one cycle, but the user can create multiple cycles.

Figure 2 presents an example of the *input.json* file, where two cycles are defined. The first cycle is defined with 1000 concurrent users for 60 seconds and the second cycle is defined with 0 concurrent users for 60 seconds as well. Therefore, the first cycle increases the resource demand, while the second cycle decreases the resource demand and makes it possible to evaluate the scaling down behaviour on the application.

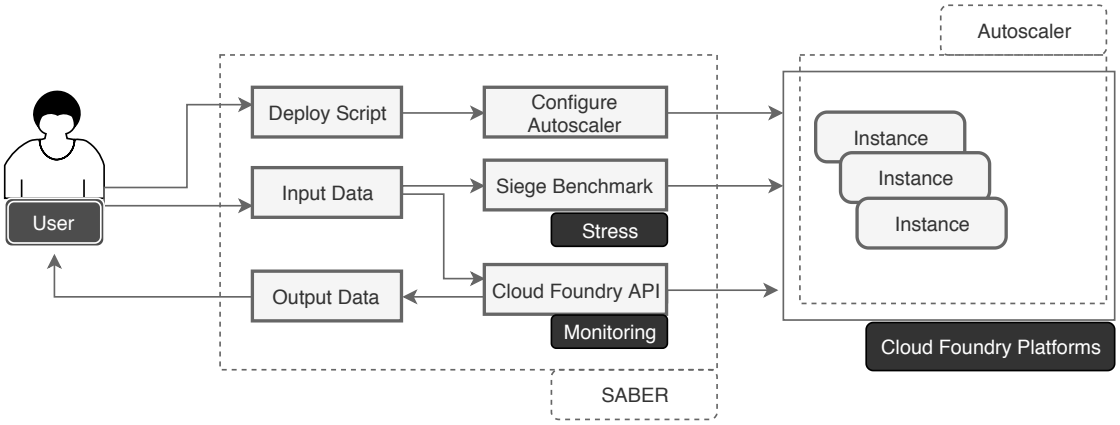

Fig. 1. SABER architecture design.

```
{
  "app_name": "grin",
  "app_env": "api.run.pivotal.io",
  "app_route": "grin.cfapps.io",
  "cycles": [
      {
          "time":"60",
          "concurrent_users":"1000"
      },
      {
          "time":"60",
          "concurrent_users":"0"
      },
      ...
  ]
```

Fig. 2. Example of input.json that is used to define the application name, environment name, application route, and all the test cycles.

B. Output Metrics

The following output metrics provided by SABER:

- **Provisioning time**: The time that an application requires to attend the workload demand. It is important to emphasize that it is not only the time to provide a new instance but all the time needed to provide an instance with the application running and receiving requests from the load balancer.
- **Deprovisioning time**: The time that the platform needs to reduce the number of resources provided to the application when the demand for computational resources decreases. This metric provides an insight into how long the platform needs to detect the resource decrease.
- **Average amount of provisioned resources**: The average number of overprovisioned resources. It can be used to verify if the amount of resources defined to each instance is adequate in order to attend the current demand.
- **Accumulated quantity of provisioned resources**: The accumulative number of provisioned resources, allowing administrators to verify the amount of idle computing resources during a period.
- **Number of concurrent users**: This metric is not directly related to elasticity. Nevertheless, the number of concurrent users in each cycle can be used to verify the impact that different amounts of users can generate regarding the resources demand of applications.
- **Failed requests**: The number of failed requests when the quantity of concurrent users increases in a short time.

IV. Evaluation

In order to validate SABER, we performed an experiment for evaluating the elasticity of Pivotal Application Service [20], which is a Cloud Foundry based platform that focuses on providing a complete environment to run cloud-native applications.

Nowadays, Pivotal Cloud Foundry is a master brand that consists of three main platform services: Pivotal Application Services (PAS), Pivotal Container Service (PKS) [21] and, Pivotal Function Service (PFS) [22]. PAS allows customers to deploy cloud-native applications, while PKS provides an abstraction for containers that can be used to execute containers, ISV applications, ElasticSearch and Apache Spark on the PCF platform. Lastly, PFS is a Serverless service that provides an environment for executing functions in the cloud through various types of events. In this study, we use a Pivotal Application Service distribution provided on Pivotal Web Services, which runs on EC2 servers.

A. Proof-of-Concept

As previously mentioned, SABER provides a sample application for evaluating the Cloud Foundry platform. However, the user can define its own application for playing this role. For this proof-of-concept experiment, we used the sample

application offered by SABER, which requires 128MB of RAM. As the free account on Pivotal Web Service provides a maximum of 2GB of memory, it is possible to have 16 instances of our application running in parallel. Different cycles were evaluated in the experiments. The specification of each scenario is presented in Table I. Each configuration scenario (A, B, C, and D) was executed ten times, and we considered the average of all the collected results.

App Autoscaler was set to scale up when the memory usage was higher than 80% and scale down when the memory demand was less than 70%. We adopted this configuration since the application starts with the usage of 62% of the memory supplied. The tests were defined in this way in order to verify the scale-up, and the down of this application, where the first cycle has more concurrent users to increase the resources demand, while the latter has less concurrent users in order to decrease the resources demand.

The values of the Accumulated Amount of Overprovisioned Resource increased during the test per different scenario. These results were expected, since this metric collects the amount of overprovisioned resources on each application instance, in each verification (5 seconds), of our benchmark tool. Therefore, the longer the benchmark tool executes, the higher will be the number of resources accumulated. The complete result is illustrated in Figure 3. Also, according to the results on the metric Average Amount of Overprovisioned resources, during the execution of the three configurations, the obtained values were similar.

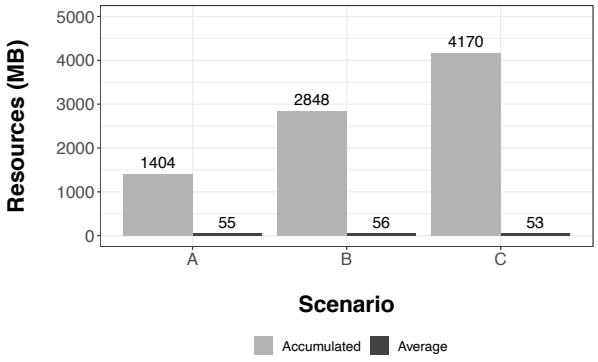

Fig. 3. Accumulated amount of the overprovisioned resources and the average amount of overprovisioned computing resources in MegaBytes.

Besides, when the number of concurrent users increases, more instances will be allocated to supply the resource demand, and more resources might be accumulated. Such event can be seen in scenario B, where there are more concurrent users. The time to provision resources was also evaluated during the tests by the developed benchmark tool. As already emphasized, the App Autoscaler service performs just horizontal elasticity since the application needs to be restarted when the vertical elasticity is performed. The time to create a new application instance, configure routes, and configure a load balancer is higher than the time needed to delete an instance, which is expected. The provisioning time of instances are presented in Figure 4.

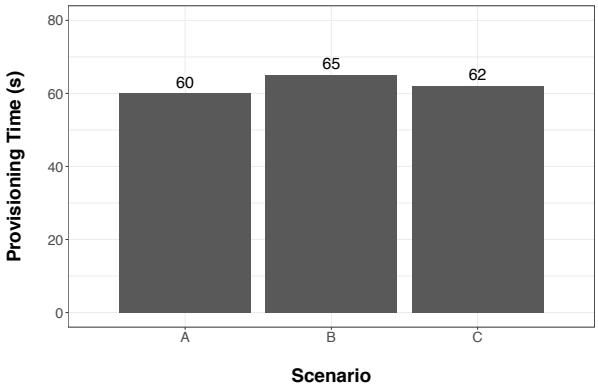

Fig. 4. Provisioning time.

During each benchmark verification, if the resource demand becomes higher than the limit, a new instance is provisioned, and the verification process stops for 30 seconds. If the resource demand still increasing during this period, the amount of time the platform will need to provision more resources will be higher.

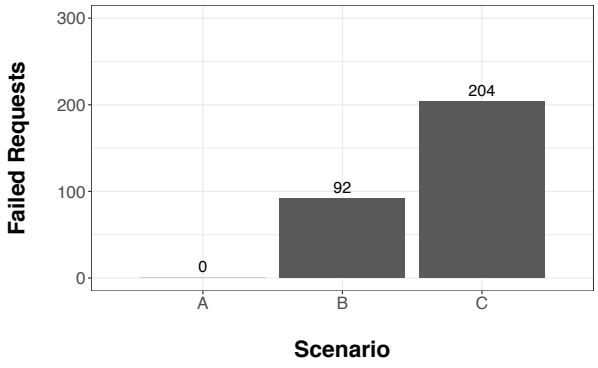

Fig. 5. Number of failed requests for each evaluation scenario.

In our evaluation, we can also note that the number of failed requests increases as the number of concurrent users accessing the application grows (Figure 5). In scenario A, we have no failed requests since the tests were performed with a smaller number of users. However, in scenario B, several failed requests were identified because the tests were performed during more time (120 seconds). So, the application needs to scale, as such, as each instance needs a amount of time to boot and some requests fail during this period. Scenario C presented the highest number of failed requests as it involves stressing the application for a longer period, while a higher number of users are accessing it.

V. Related Work

Several studies have presented benchmark tools in order to evaluate applications and services running on the cloud.

TABLE I. TESTBED SPECIFICATION.

Scenario	Number of Cycles	Execution Time	Concurrent Users
A	2	60	10
			0
B		120	10
			0
C		240	30
		480	5

Cooper et. al [23] created a standard benchmark tool and benchmark framework to assist in the evaluation of different cloud systems. The authors emphasize their focus on serving systems. Two benchmark tiers were adopted for evaluating the performance and the scalability of cloud serving system. The performance tier focuses on the latency of requests, while the scaling tier aims to examine the impact in the performance when more machines are added to the system. In order to evaluate the benchmark tool, a set of tests was defined including different database systems, such as Cassandra, HBase, and PNUTS.

In the same scenario, Li et. al [24] describes CloudCmp, a benchmark suite developed to systematically compare the performance and cost of cloud providers along dimensions that matter to customers. Regarding elasticity, the authors divide latency into two segments: a provisioning latency and a booting latency. The authors also present the comparison results among the following four providers: Amazon Web Services, Microsoft Azure, Google App Engine, and Rackspace CloudServers. Besides, this benchmark suite provides several metrics, including scaling latency, response time, throughput, time to consistency, and cost per operation.

Besides, there are several open source benchmark tools that aim to assess the performance of systems running in the cloud relative to scalability, such as the Benchmark Rally [25]. This tool provides a framework for performance evaluation and benchmarking of each OpenStack component, as well as provides an evaluation of a complete production OpenStack cloud implementation. Rally automates and unifies the deployment of multiple OpenStack nodes, cloud verification, and testing and profiling. Using this tool, the user can check how OpenStack would work on a larger scale. Rally can be used through a lightweight and portable CLI application or as a set of tools that present the web user interface.

Cloud Suite [26] is also an open source cloud benchmark tool for cloud computing and it was developed in order to provide real-world software stacks and represent real-world setups evaluation. The third release consists of several applications. It includes benchmarks that represent massive data manipulation with tight latency constraints, such as in-memory data analytics using Apache Spark. The tools selected in this suite are used to characterize the inefficiencies in the microarchitecture of modern server CPUs used in a cloud computing environment. Among the tasks included in this set of benchmarks, we can cite data serving, MapReduce, media streaming, SAT solving, web hosting, and web search. These components are open source applications as well. In the same context, this suite was developed to provide an easy deployment into private and public cloud platforms since it offers Docker containers images for all CloudSuite benchmarks, such as web search, data caching or data analytics benchmark.

Similar to Cloud Suite, HiBench is also an open source benchmark suite for Hadoop. This tool consists of a set of Hadoop programs, including both synthetic micro-benchmarks and real-world applications [27]. The application currently has several workloads classified into four categories: Microbenchmarks, Web Search, Machine Learning, and Data Compression. Microbenchmarks provide several algorithms, like Sort, WordCount, Sleep, enhanced DFSIO, and TeraSort programs contained in the Hadoop distribution. These benchmarks are used to represent a significant subset of real-world MapReduce jobs. Web Search uses PageRank and Nutch indexing benchmark tools. These benchmark tools are adopted because the large-scale search indexing is one of the most significant uses of MapReduce. It is important to emphasize that the HiBench also provides several other machine learning benchmark alternatives, like Linear Regression and Gradient Boosting Trees.

To the best of our knowledge, SABER complements the existing benchmark tools as it allows operators to evaluate how good Cloud Foundry's elasticity mechanisms behave on different infrastructures. SABER provides several elasticity metrics, including provisioning and deprovisioning time, average amount of provisioned resources, and accumulated amount of provisioned resources. Therefore, considering that Cloud Foundry based platforms are meant to run on any infrastructure, SABER stands out by providing means to evaluate which infrastructure would best satisfy customers' elasticity requirements.

VI. CONCLUSION AND FUTURE WORK

Elasticity is a vital characteristic for cloud-native applications as it allows wisely provisioning resources according to the demand.

In this paper, we present SABER, an elasticity benchmark tool that allows operators to analyze how good Cloud Foundry based platforms behave on different infrastructures. SABER provides a set of metrics for evaluating elasticity, including provisioning and deprovisioning time, the average amount of provisioned resources, and the accumulated amount of provisioned resources.

As future work, we are looking forward to extending SABER to support other metrics, including CPU usage, which may provide specific insights for infrastructure operators.

VII. Aknowledgements

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248/91).

References

[1] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[2] U. Gurav and R. Shaikh, "Virtualization: a key feature of cloud computing," in *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*. ACM, 2010, pp. 227–229.

[3] C. Vazquez, R. Krishnan, and E. John, "Cloud computing benchmarking: A survey," in *Proceedings of the International Conference on Grid Computing and Applications (GCA)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2014, p. 1.

[4] G. Galante and L. C. E. d. Bona, "A survey on cloud computing elasticity," in *Proceedings of the 2012 ieee/acm fifth international conference on utility and cloud computing*. IEEE Computer Society, 2012, pp. 263–270.

[5] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*, 2013, pp. 23–27.

[6] A. Gajbhiye and K. M. P. Shrivastva, "Cloud computing: Need, enabling technology, architecture, advantages and challenges," in *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*. IEEE, 2014, pp. 1–7.

[7] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. IEEE, 2016, pp. 1–6.

[8] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[9] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The making of cloud applications: An empirical study on software development for the cloud," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 393–403.

[10] P. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and A. A. Franklin, "A cloud native solution for dynamic auto scaling of mme in lte," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017 IEEE 28th Annual International Symposium on*. IEEE, 2017, pp. 1–7.

[11] S. Brunner, M. Blöchlinger, G. Toffetti, J. Spillner, and T. M. Bohnert, "Experimental evaluation of the cloud-native application design," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 488–493.

[12] Red Hat OpenShift, 2019. [Online]. Available: https://www.openshift.com

[13] Heroku, 2019. [Online]. Available: https://www.heroku.com/

[14] Cloud Foundry, 2019. [Online]. Available: https://www.cloudfoundry.org/

[15] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018.

[16] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei, "A scalable and elastic publish/subscribe service," in *2011 IEEE International Parallel Distributed Processing Symposium*, May 2011, pp. 1254–1265.

[17] F. Aisopos, K. Tserpes, and T. Varvarigou, "Resource management in software as a service using the knapsack problem model," *International Journal of Production Economics*, vol. 141, no. 2, pp. 465–477, 2013.

[18] S. Jangiti, V. S. Sriram, and R. Logesh, "The role of cloud computing infrastructure elasticity in energy efficient management of datacenters," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*. IEEE, 2017, pp. 758–763.

[19] Cloud Foundry App-Autoscaler, 2019. [Online]. Available: https://github.com/cloudfoundry/app-autoscaler

[20] Pivotal Application Service, 2019. [Online]. Available: https://pivotal.io/platform/pivotal-application-service

[21] Pivotal Container Service, 2019. [Online]. Available: https://pivotal.io/platform/pivotal-container-service

[22] Pivotal Function Service, 2019. [Online]. Available: https://pivotal.io/platform/pivotal-function-service

[23] 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*. ACM, 2010, pp. 143–154.

[24] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 1–14.

[25] Rally, "Rally benchmark," Available at: https://github.com/openstack/rally. Accessed: 2019-04-14, 2019.

[26] C. Suite, "Cloud suite 3.0," Available at: https://github.com/openstack/rally. Accessed: 2019-04-10, 2019.

[27] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai, "Hibench: A representative and comprehensive hadoop benchmark suite," in *Proc. ICDE Workshops*, 2010.