EasyCloud: Multi-clouds made easy

EasyCloud: Multi-clouds made easy

Cosimo Anglano, Massimo Canonico and Marco Guazzone
{cosimo.anglano,massimo.canonico,marco.guazzone}@uniupo.it
Computer Science Institute, DiSIT, University of Piemonte Orientale, Italy

***Abstract*—Interoperability between different cloud platforms is a critical requirement for letting users to smoothly switch between different cloud providers and combine their services. However, the lack of standard interfaces to access these cloud platforms may result in the vendor lock-in situation, whereby users are locked into a specific cloud provider. In this paper, we present EasyCloud, a toolkit able to effectively support the creation and usage of Multi-cloud Systems (MSs) by providing interoperability, platform independence, effective resource provisioning, and ease of use. We describe its architecture and implementation, and experimentally assess the performance of EasyCloud, and compare it to existing alternative MS toolkits that are representative of the state-of-the-art. Our results clearly show that EasyCloud is highly scalable, quite efficient, and outperforms the other alternative toolkits.**

***Index Terms*—Multi-clouds, Toolkit, Resource provisioning**

I. Introduction

In the last few years, *cloud computing* [1] has become a very popular and effective solution for enterprises to provide their services in a "pay-per-use" basis. In particular, with the *Infrastructure-as-a-Service* model, users can rent virtualized resources to run their services inside virtual execution environments like *Virtual Machine* (VM) instances and *containers.* [1]

Cloud providers have developed their own platforms featuring proprietary web, command-line, and programming interfaces, by means of which customers can manage their VMs and analyze the performance of their applications. Unfortunately, the lack of standard interfaces to access these platforms makes it difficult for customers to easily switch between different providers. This situation often leads to the so called *vendor lock-in* [2]. To overcome these limitations, *Multi-cloud Systems* (MS) [3], i.e. cloud infrastructures composed of resources drawn from different cloud platforms, have been recently proposed as a way to allow customers to combine services or resources of different cloud platforms by means of a unified interface. That way, customers can exploit the advantages of different cloud providers (e.g., in terms of service cost, quality of service, and performance [4]–[6]) without being locked into a single one.

To create an MS, a suitable toolkit, providing the necessary "glue" among different cloud infrastructures and the appropriate level of abstraction, is required. In particular, such a toolkit should provide the following features: (1) *interoperability* (the ability to support multiple cloud platforms, so as to avoid vendor lock-in), (2) *platform independence* (the provision of a unified, platform-agnostic user interface that hides the API heterogeneity of these platforms), (3) *effective resource provisioning* (the ability of ensuring that the VMs delivering a given cloud service will be sufficiently resourced in a timely manner as load increases, so as to achieve desired levels of performance, efficiency, and availability), and (4) *ease of use* (the provision of suitable mechanisms enabling users to interact with ease and effectiveness with the MS).

[1]Unless otherwise stated, we refer to VM instances and containers as VMs.

Various toolkits have been proposed in the literature to allow smooth cloud interoperability and to harness multi-cloud heterogeneous resources but, unfortunately, they provide only a subset of the features mentioned above [3]. To fill this gap, in this paper, we propose *EasyCloud*, an MS toolkit able to effectively support the creation and usage of MSs by providing all the features discussed above.

In particular, EasyCloud provides *interoperability* and *platform-independence* by means of an extensible cloud interfacing subsystem. EasyCloud also provides *effective resource provisioning* by coupling two separate mechanisms, namely: *VM metrics collection and dispatching* (that collects in (near) real-time user-defined metric data – e.g., CPU and memory load – from VMs and dispatches them to multiple "sinks" for storage as well as for further analysis and processing, thus enabling users to analyze and understand how their applications and services are performing), and *VM monitoring and provisioning* (that exploits collected metric data to monitor the performance of VMs in real time and to trigger management actions according to user-defined policies, e.g., to implement load balancing and autoscaling). Finally, EasyCloud provides *ease of use* by means of a unified API that frees users from learning the different proprietary APIs exposed by the various cloud platforms it supports, and of an interactive and intuitive user interface that allows even inexperienced users to easily manage their VMs.

EasyCloud is written in Python (thus ensuring portability and fast development) and its source code is publicly available [7] (thus fostering research and providing reproducibility).

In this paper, we describe a new and extended version of EasyCloud, whose design and implementation have been revised and greatly enhanced with respect to the version presented in [8]. In particular, this paper provides the following contributions:

- We present the architecture of the new version of EasyCloud, whose design and implementation have been considerably improved and extended compared with the prior version presented in [8]. Specifically, we extended the monitoring functionality of EasyCloud with the ability to dispatch metric data gathered from VMs (e.g., memory and CPU utilization) to multiple “sinks” for storage, analysis and processing, so as users can understand how their applications and services are performing. Also, for performance reasons, we partially rewrote some components of EasyCloud to replace the use of *Apache Libcloud* [9] with direct calls to the native APIs provided by the supported cloud platforms (e.g., to interact with OpenStack we use directly the OpenStack SDK).

- We present an extensive experimental evaluation where (1) we show the ability of EasyCloud to scale with respect to the number of VMs to manage, (2) we assess the impact of the monitoring functionality on its performance, and (3) we compare its performance with the one achieved by both the version of EasyCloud proposed in [8], and by a representative set of state-of-the-art alternatives.

II. Related Work

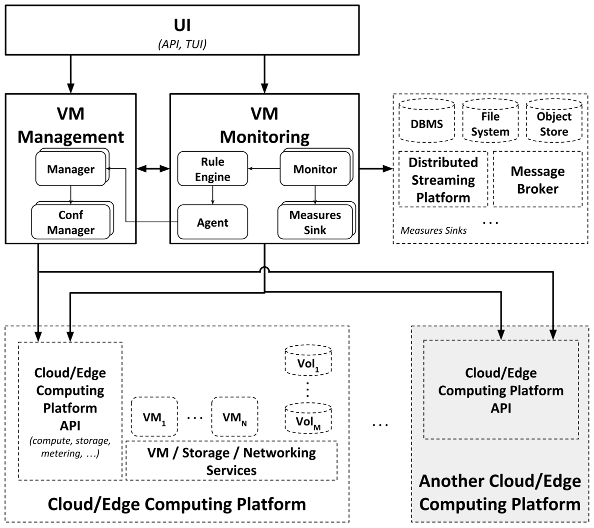
Various MS toolkits, similar in spirit and purposes to EasyCloud, have been proposed in the literature.

Apache JClouds [10] is an MS toolkit for the Java platform that facilitates developing applications for a wide range of cloud platforms. JClouds supports many cloud providers and software stacks, including AWS, OpenStack and Google Cloud. Despite its many features, JClouds does not provide any effective resource provisioning mechanisms (e.g., it is not possible to check the health of VMs), and provides ease of use to a limited extent, as it requires Java programming skills.

Apache Libcloud [9] is an MS toolkit written in Python that supports several popular cloud providers through a unified API able to hide differences between the APIs of different cloud providers. As JClouds, Libcloud does not provide any effective resource provisioning mechanisms (e.g., monitoring and gathering performance metrics), and provides ease of use to a limited extent, as it requires Python programming skills.

Cloudmesh [11] is another MS toolkit able to provide access to various cloud platforms, such as AWS, Azure, Google Cloud and OpenStack. It has a variety of repositories that add features to Cloudmesh based on needs by the user. Unlike JClouds and Libcloud, Cloudmesh is easier to use, but like them it does not provide any effective resource provisioning mechanism.

Finally, *Terraform* [12] is an open source tool that allows to define an infrastructure as code using a declarative language and to deploy and manage that infrastructure across a variety of public cloud providers (e.g., AWS, GCP, MS Azure) and private cloud (e.g. OpenStack) using a few commands. Terraform provides only partially the effective resource provisioning since, for instance, it has not a unique interface to describe an auto-scaling policy for the various cloud platforms. In particular, it provides a specific auto-scaling library for each cloud platform supported; this means that the user has to implement different code to apply the same auto-scaling policy.


UI
(API, TUI)
VM Management
Manager
Conf Manager
VM Monitoring
Rule Engine
Monitor
Agent
Measures Sink
DBMS
File System
Object Store
Distributed Streaming Platform
Message Broker
...
Measures Sinks
Cloud/Edge Computing Platform API
(compute, storage, metering, ...)
VM_1
...
VM_N
Vol_1
Vol_M
VM / Storage / Networking Services
Cloud/Edge Computing Platform
...
Cloud/Edge Computing Platform API
Another Cloud/Edge Computing Platform
Fig. 1: The architecture of EasyCloud.

With respect to the above works, EasyCloud is the only MS toolkit that provides interoperability, platform independence, effective resource provisioning, and easy of use, while the other toolkits fail to provide one or more of them.

III. Architecture

EasyCloud has been designed with the following key objectives in mind: *interoperability* (to avoid vendor lock-in), *platform-independence* (to abstract the API heterogeneity of the various cloud platforms), *flexibility* (to enable users to exploit platform-specific features), *effective resource provisioning* (to enable users to monitor their VM instances in (near) real-time and to trigger actions in response to specific events), and *ease of use* (to enable ease and fast management of VM instances).

The resulting system architecture is shown in Figure 1, where solid boxes denote the components of EasyCloud (and stacked boxes represent components that may be instantiated multiple times), dashed shapes represent external components (e.g., components of a cloud platform) outside of EasyCloud with whom EasyCloud interacts with, and arrows denote interactions between components (e.g., an arrow from component C_i to component C_j means that C_i uses the functionality of C_j).

As shown in the figure, the components of EasyCloud are grouped into three subsystems, namely *VM Management*, *VM Monitoring* and *UI*, that present a unified interface (independent by any cloud platform) that allows users to transparently interact with different cloud platforms at the same time, without worrying about the high heterogeneity between the APIs provided by the various cloud platforms. At the same time, these subsystems give users full control to use, if needed, platform-specific features (e.g., to access instance information available only on a given cloud platform) by also presenting a low-level interface specific to a particular cloud platform.

Currently, EasyCloud supports the following cloud platforms: *Amazon AWS*, *Chameleon Cloud* [13], *Google Cloud*, and *OpenStack*. Furthermore, thanks to its modular architecture, it can be easily extended to support new cloud platforms by just providing the concrete implementation of its interface specific for those new platforms.

The design and implementation of the *VM Management* and the *VM Monitoring* subsystems have been presented in detail in [8]. In this section we thus focus on the new components that have been added to the version of EasyCloud presented in [8], namely (a) the *Measure Sink* component of the *VM Monitoring* subsystem, and (b) the *UI* subsystem; due to space limits, we instead omit new implementation features and optimizations.

A. The Measure Sink component

The *Measure Sink* component provides support for the analysis of the metrics collected by the *Monitor* component of the *VM Monitoring* subsystem. In particular, it receives a collection of metrics from the *Monitor* and forwards them to a "sink", that is to a third-party component external to EasyCloud (e.g., a database server, or a message broker), for further processing or storage. Since there can be multiple instances of the *Measures Sink* component at the same time (e.g., one connected to a database server, and another one connected to a message broker), these instances will receive from the *Monitor* component the same collection of metrics. Currently, EasyCloud supports the following sinks: *Apache ActiveMQ*, *Apache Cassandra*, *Apache Kafka*, *CSV files*, *MongoDB*, *Prometheus*, *RabbitMQ*, and *Redis*. Furthermore, thanks to its modular architecture, it can be easily extended to support new sinks by just adding new *Measures Sink* components specific for such sinks.

B. The UI subsystem

The *UI* subsystem provides an *API* and a *Text-based User Interface* (TUI), built on top of the API, to access the functionality provided by the *VM Management* and *VM Monitoring* subsystems. Specifically, at the lowest level, it exposes an object-oriented API in Python that allows developers to use EasyCloud as a framework to interact with the services of different cloud providers in a programmatic and platform-agnostic way. This API provides a high-level (platform-independent) interface, to transparently interact with the various cloud platforms without caring of the heterogeneity of the API provided by such platforms. Furthermore, to achieve flexibility, the API also provides a low-level (platform-dependent) interface to fully exploit platform-specific features (i.e., to access platform-specific information about a given VM instance that is not provided by the higher-level interface). Moreover, on top of the platform-independent API, the *UI* subsystem provides a user-friendly TUI through which users can use EasyCloud interactively, via a text-based interface.

IV. Experimental Evaluation

In order to assess the scalability and performance of EasyCloud, we perform an experimental evaluation in which (1) we evaluate the ability of EasyCloud to scale with respect to the number of managed instances, (2) we assess the overhead caused by monitoring, and (3) we compare its performance against that attained by alternative state-of-the-art MS toolkits.

Our results clearly indicate that EasyCloud is able (1) to perform the above operations very quickly, (2) to scale very well with respect to the number of managed VMs, even when monitoring is activated (although, as shown below, it adds some overhead), and (3) to outperform the state-of-the-art alternatives considered in our experiments.

In the rest of this section, after describing the experimental settings (Section IV-A), we discuss the results concerning the scalability of EasyCloud (Section IV-B), and the performance comparison against state-of-the-art MS toolkits (Section IV-C).

A. Experimental settings

Our experiments feature a set M of VM instances, hosted on a given public cloud infrastructure, that are managed either by using EasyCloud (in the scalability experiments) or one of the alternative MS toolkits (in the performance comparison experiments).

In each experiment, we measure the *CPU time* (both in kernel space and in user space) taken by the MS toolkit under consideration to both retrieve the detailed information of the M VM instances and to start just $N \leq M$ of them. Note that, by considering the CPU time instead of the wall-clock time, we are sure that our results are not influenced by other external and non-controllable factors, like the scheduling policy of the operating system running on the same host where the MS toolkit components are run, unpredictable network load conditions, and API rate limiting policies specific to a given cloud provider (e.g., [14]).

We use as performance index the *average CPU time* that we compute by averaging the CPU time obtained in the various runs of an experiment, until the relative error of its 95% confidence interval is of at most 4% [15]. Therefore, the number of runs of a single experiment is computed online, and may change from one experiment to another.

Our experiments are performed by using a physical testbed composed of:

- a front-end system, consisting of two Fujitsu Server PRIMERGY RX300 S7 (equipped with two 2.4 GHz Intel Xeon E5-2665 processors with 8 cores and 96 GiB of RAM, and running the Linux kernel version 5.9.13) located in Italy, running both the components of each MS toolkit and the server-side components of the EasyCloud's measures sinks (see later);
- a back-end system, consisting in a public cloud infrastructure hosting the set of M VMs that was managed either with EasyCloud, or with one of the alternative MS toolkits considered for comparison purposes.

To understand whether and how much the specific back-end cloud infrastructure impacts on the performance attained by EasyCloud, in the experiments we consider three different public cloud infrastructures, that is AWS EC2 (region "us-east-2", located in Virginia, USA, and instance type "t2.micro"), Chameleon Cloud (site "KVM@TACC", located in Texas, USA, and instance flavor "m1.tiny") and Google Compute Engine (hereafter, GCE; regions "us-central1" and "us-west1", located in Iowa and Oregon, USA, respectively, and machine type "e2-medium"), where we hosted $M = 45$ Linux instances. [2]

B. Scalability of EasyCloud

As anticipated, to asses the scalability of EasyCloud we run a set of experiment in which we progressively increase the number N of VM instances started on the back-end, until the maximum value M is reached, and we measure the average CPU time taken by this operation.

Specifically, we first assess the scalability of EasyCloud with the monitoring functionality disabled (see Section IV-B1). Then, we repeat the same experiments by enabling the monitoring functionality of EasyCloud, and by activating different number and type of measures sinks (see Section IV-B2).

1) Scalability of EasyCloud without monitoring: In Figure 2, we present the results of the experiments with the monitoring functionality disabled, both for AWS EC2 (see Figure 2a), Chameleon Cloud (see Figure 2b) and GCE (Figure 2c) cloud platforms. In these figures, each filled point represents the average CPU time (in seconds) taken by EasyCloud to perform the operations in each run for a given number of instances N (with N ranging from 5 to 45), and the associated error bar denotes its 95% confidence interval.

Also, for comparison purposes, we show the performance of both the current version of EasyCloud (simply labeled as "EasyCloud") and its previous version [8] (labeled as "EasyCloud (old)"), where we use *Apache Libcloud* [9] (version 3.2.0) as the abstraction layer to interact with different cloud infrastructures.

These figures show that both versions of EasyCloud are able to scale well with respect to the number of instances to manage, as the average CPU time just grows linearly as a function of the number of managed instances. In particular, while for AWS EC2 both versions of EasyCloud do not show any relevant performance difference, for Chameleon Cloud the current version significantly outperforms the older one, thanks to directly using the native APIs of the cloud infrastructure that avoids the overhead due to the additional abstraction layer provided by Apache Libcloud.

2) Scalability of EasyCloud with monitoring: To investigate the impact of the monitoring functionality on the scalability of EasyCloud, we repeat the above experiments with the monitoring functionality enabled and by activating an increasing number of measures sinks. In particular, we incrementally activate the following sinks: "file" (that writes data to CSV files), "activemq" (that sends data to an *Apache ActiveMQ* server), "kafka" (that sends data to an *Apache Kafka* server), "mongodb" (that stores data into a *MongoDB* server), and "redis" (that sends data to a *Redis* server). Therefore, we carry out a set of experiments first with only the "file" sink enabled; then, we repeat the same experiments with both "activemq" and "file" enabled; and so on, until the case with all sinks enabled.

To avoid cluttering, here we show only the results related to the Chameleon Cloud infrastructure. The results obtained for the other cloud infrastructures, however, do not significantly differ as they show very similar patterns.

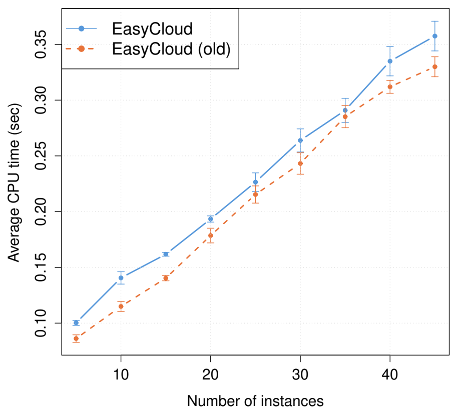
The results of such experiments are shown in Figure 3, where each point denotes the average CPU time (in seconds) taken by EasyCloud to manage a given number of instances N (with N ranging from 5 to 45), the associated error bar denotes its 95% confidence interval, and each line labeled as "EasyCloud with monitoring (sinks: X, Y, ...)" represents the scenario where the monitoring functionality is enabled and only the sinks "X", "Y" and so on are active. The figure reports also the results for two other cases, namely "EasyCloud with monitoring (sinks: none)" and "EasyCloud," where the former denotes the case where we carry out the above experiments with the monitoring functionality enabled (included the interaction with the telemetry services of the cloud platform) but without any sink activated (so as to assess the impact on the performance of the monitoring framework, regardless the type and the number of sinks activated), while the latter (which is the same one showed in Figure 2b) denotes the case with the monitoring functionality disabled and is used as a baseline for comparison purpose.

From the figure, we note that, with respect to the baseline case, the monitoring functionality adds some overhead. For instance, in the case "EasyCloud with monitoring (sinks: none)", the average CPU time increases by 50% on average, with a maximum of 81.6% at 45 instances. This is mainly due to the fact that the monitoring framework spends some CPU time to periodically gather metric data from the managed instances.

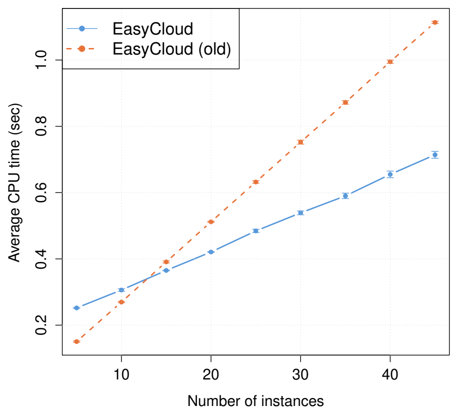
However, we also note that, even with the monitoring functionality enabled and regardless the number and the type of sinks activated, EasyCloud is able to scale well at least until 40 instances. Conversely, starting from 45 instances, the CPU time shows a steeper growth. This can be ascribed to the way the Python interpreter handles multi-threaded execution (more specifically, the so-called *Global Interpreter Lock* [16], a mechanism that assures that only one thread executes Python bytecode at a time) that may prevent programs to take fully advantage of running in multiprocessor systems [17].

Finally, we note that among the number and the type of sinks activated, the factor that has a stronger negative impact on CPU time is the latter, as the interaction protocol of a particular sink with its server-side components (e.g., the interaction of the "kafka" sink with the *Apache Kafka* server) may be quite complex. For instance, the figure shows that the impact of activating "kafka" or "mongodb" on the CPU time is larger than the one of "redis".

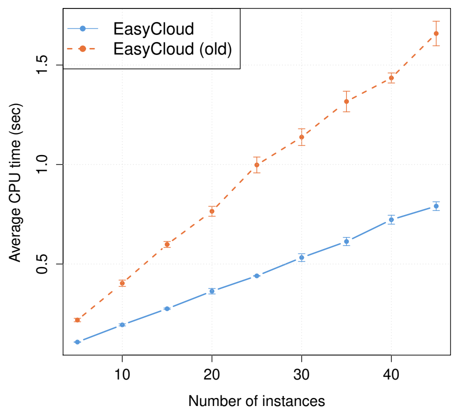
[2]For GCE, due to its default per-region quota limits, we placed the first 24 instances in region "us-central1" and the remaining 21 instances in region "us-west1".



(a) AWS EC2.



(b) Chameleon Cloud.



(c) GCE.

Fig. 2: Performance of EasyCloud on different cloud infrastructures for increasing numbers of managed instances. Each filled point denotes the CPU time in seconds (y-axis) that EasyCloud spent, on average, to manage a particular number of instances (x-axis) hosted on a specific cloud platform, and the associated error bar represents its 95% confidence interval.

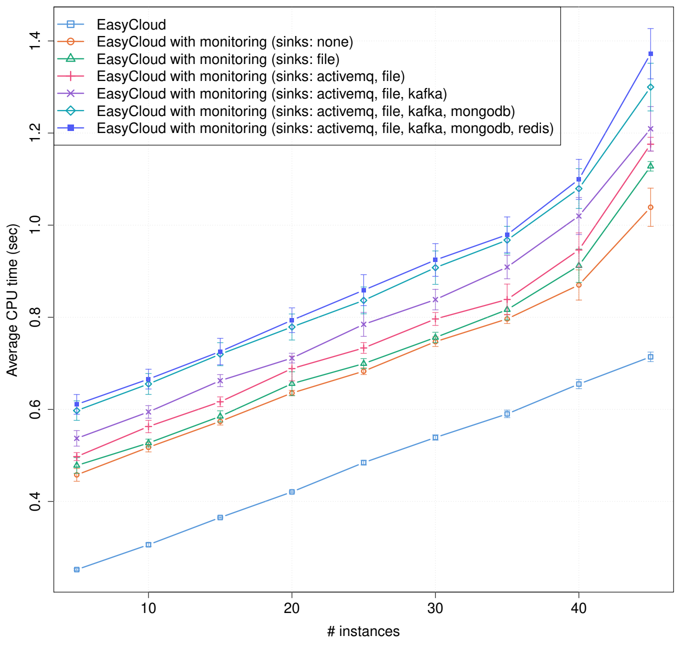


Fig. 3: Impact of the monitoring functionality on the performance of EasyCloud. Each point represents the CPU time in seconds (y-axis) that EasyCloud spent, on average, to manage a particular number of instances (x-axis) hosted on a specific cloud platform, and the associated error bar represents its 95% confidence interval.

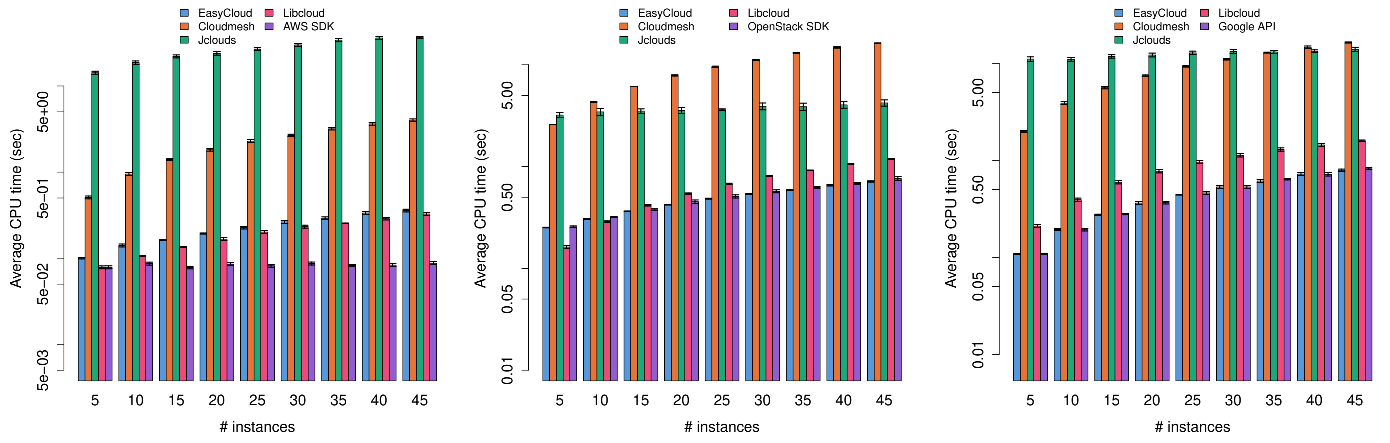
C. *Comparison agains state-of-the-art alternative MS toolkits*

To compare the performance of EasyCloud with the state of the art, we repeat the above experiments by using the following MS toolkits to manage the VMs running on the back-end system: *Cloudmesh* [11] (ver. 4), *Apache Libcloud* [9] (ver. 3.2.0), and *Apache Jclouds* [10] (ver. 2.2.1) . For completeness, we also use the native APIs provided by the various cloud infrastructure through their Python clients, namely *AWS SDK for Python* for AWS (ver. 1.16), *Google API Client for Python* for Google Cloud (ver. 1.12.8), and *OpenStack SDK* for Chameleon Cloud (ver. 0.52), that are used internally by EasyCloud and hence can be employed as a baseline.

In Figure 4, we show the results of such experiments, both for AWS EC2 (see Figure 4a), Chameleon Cloud (see Figure 4b) and GCE (Figure 4c), where each filled bar represents the CPU time (in seconds and in log scale) spent by a specific library to manage a given number of instances N (with N ranging from 5 to 45) and the associated error bar denotes its 95% confidence interval.

These results show that, except for the native APIs (which clearly provide the best performance), EasyCloud and Libcloud always take the lowest CPU time. In particular, they outperform the alternative MS toolkits (namely, CloudMesh and Apache JClouds) we consider for the comparison.

In particular, with respect to the native APIs, EasyCloud adds a little overhead (especially with respect to AWS SDK, as shown in Figure 4a) due to the abstraction layer built on top of them so as to provide a unified interface. Also, with respect to Libcloud, EasyCloud shows similar (see results for AWS EC2) or better performance. Finally, with respect to the other competitors, EasyCloud always shows better performance.



(a) AWS EC2. (b) Chameleon Cloud. (c) GCE.

Fig. 4: Comparison of EasyCloud with state of the art alternatives on different cloud infrastructures and for increasing numbers of managed instances. Each filled bar denotes the CPU time in seconds (y-axis, in log scale) that a given library took, on average, to manage a particular number of instances (x-axis) hosted on a specific cloud platforms, and the associated error bar represents its 95% confidence interval.

V. Conclusion

In this paper, we presented EasyCloud, a toolkit able to effectively support the creation and usage of MSs by providing interoperability, platform independence, effective resource provisioning, and ease of use.

We have experimentally assessed the performance of EasyCloud, and compared it to existing alternative MSs toolkits that are representative of the state-of-the-art. Our results clearly show that EasyCloud is highly scalable (even with monitoring enabled), quite efficient (it adds little overhead to the native APIs of cloud platforms when monitoring is not used), and outperforms the other alternative toolkits (with the exception of Libcloud, that in some cases delivers similar performance).

As future work, we plan to reduce the overhead added by the abstraction layer of EasyCloud; in particular, the significant overhead for the AWS platform that we noted in the experimental evaluation leaves room for improvement. Moreover, we want to extend EasyCloud to support additional cloud platforms, like *Microsoft Azure* and *Kurbernetes*, and more measures sinks, like *Fluentd* [18] and intelligent fault-detection systems [19], [20]. Finally, we plan to integrate EasyCloud with our prior works concerning the management [5], [21], [22] and experimentation [23] of cloud/edge infrastructures.

Acknowledgment

This work was partly supported by Università del Piemonte Orientale and INdAM – GNCS Project 2020. Results presented in this paper were in part obtained using the Chameleon testbed supported by the National Science Foundation.

References

[1] P. M. Mell and T. Grance, "The NIST definition of cloud computing," NIST, Tech. Rep. SP 800-145, 2011.
[2] D. G. Kogias, M. G. Xevgenis, and C. Z. Patrikakis, "Cloud federation and the evolution of cloud computing," *Computer*, vol. 49, no. 11, pp. 96–99, 2016.
[3] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Comput. Surv.*, vol. 47, no. 1, 2014.
[4] C. Anglano, M. Canonico, P. Castagno, M. Guazzone, and M. Sereno, "A game-theoretic approach to coalition formation in fog provider federations," in *Proc. of the 3rd International Conference on Fog and Mobile Edge Computing (FMEC)*, 2018, pp. 123–130.
[5] C. Anglano, M. Canonico, and M. Guazzone, "Profit-aware resource management for edge computing systems," in *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys)*, 2018, pp. 25–30.
[6] C. Anglano, M. Canonico, P. Castagno, M. Guazzone, and M. Sereno, "Profit-aware coalition formation in fog computing providers: A game-theoretic approach," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, 2020.
[7] C. Anglano, M. Canonico, and M. Guazzone. (2021, Jan.) EasyCloud repository. [Online]. Available: https://gitlab.di.unipmn.it/DCS/easycloud
[8] ——, "EasyCloud: a rule based toolkit for multi-platform cloud/edge service management," in *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, April 2020, pp. 188–195.
[9] Apache Software Foundation. (2021, Jan.) Libcloud. [Online]. Available: https://libcloud.apache.org
[10] E. Toews and D. Advocate, "Introduction to apache jclouds," *Apr*, vol. 7, p. 23, 2014.
[11] G. von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, "Accessing multiple clouds with cloudmesh," in *Proc. of the 2014 ACM International Workshop on Software-Defined Ecosystems (BigSystem)*, 2014, pp. 21–28.
[12] Y. Brikman, *Terraform: Up & Running: Writing Infrastructure as Code*. O'Reilly Media, 2019.
[13] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth, *Chameleon: A Scalable Production Testbed for Computer Science Research*. CRC Press, 2019, ch. 5.
[14] Google. (2021, Jan.) Google Compute Engine: API Rate Limit. [Online]. Available: https://cloud.google.com/compute/docs/api-rate-limits
[15] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.
[16] Python. (2021, Jan.) Python documentation. [Online]. Available: https://docs.python.org/3/glossary.html#term-global-interpreter-lock

[17] D. Beazley, "A tale of two concurrencies (part 1)," *;login:*, vol. 40, no. 3, 2015.

[18] Fluentd Project. (2021, Jan.) Fluentd: an open source data collector for unified logging layer. [Online]. Available: https://www.fluentd.org/

[19] S. Montani and C. Anglano, "Case-based reasoning for autonomous service failure diagnosis and remediation in software systems," in *Advances in Case-Based Reasoning*, T. R. Roth-Berghofer, M. H. Göker, and H. A. Güvenir, Eds. Springer Berlin Heidelberg, 2006, pp. 489–503.

[20] C. Anglano and M. Botta, "Now g-net: Learning classification programs on networks of workstations," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 463–480, 2002.

[21] C. Anglano, M. Canonico, and M. Guazzone, "FCMS: A fuzzy controller for CPU and memory consolidation under SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[22] C. Anglano, M. Guazzone, and M. Sereno, "Maximizing profit in green cellular networks through collaborative games," *Computer Networks*, vol. 75, Part A, pp. 260–275, 2014.

[23] C. Anglano, M. Canonico, and M. Guazzone, "Prometheus: A flexible toolkit for the experimentation with virtualized infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 11, 2018.