Wukong: A Heuristic-based Framework for Generating Generic-API for JointCloud

Yuanjia XU
Institute of Software
Chinese Academy of Sciences
Beijing, China
xuyuanjia2017@otcaix.iscas.ac.cn

Heng WU
Institute of Software
Chinese Academy of Sciences
Beijing, China
wuheng@iscas.ac.cn

Haijun LI
Information Technology Department
China Mobile E-commerce Co.,Ltd
Changsha, China
lihaijun@hn.chinamobile.com

Yuewen WU
Institute of Software
Chinese Academy of Sciences
Beijing, China
wuyuewen@otcaix.iscas.ac.cn

Shijun QIN
Institute of Software
Chinese Academy of Sciences
Beijing, China
qinshijun16@otcaix.iscas.ac.cn

Tianze HUANG
School of Computer Science
Beijing University of Post and Telecommunications
Beijing, China
huangtianze@bupt.edu.cn

***Abstract*—JointCloud gives developers the freedom to create applications those are portable across multiple clouds (e.g., Amazon EC2, Aliyun ECS). However, any modification in a vendor-lock API will make JointCloud unusable due to the problem of API synchronization. We assume that adapter flows should be decoupled from the manners in which they are executed. To verify this idea, we built Wukong, a framework that can dynamically map front-end adapter flow descriptions to a broad range of back-end clouds by the generic APIs. Generic APIs for VM services have been proposed to evaluate our Wukong framework: (1) the overhead of reflection is acceptable when WuKong generates the generic APIs, (2) the feasibility of Wukong has been verified by some cloud vendor-lock APIs, (3) Wukong with heuristic rules can accelerate the generation of APIs.**

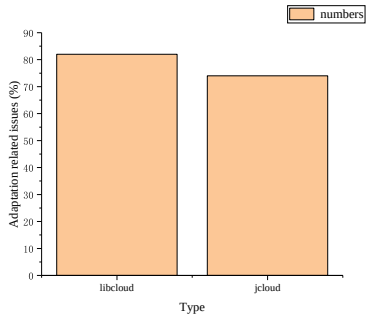
***Index Terms*—API synchronization, Generic APIs, Heuristic**

I. Introduction

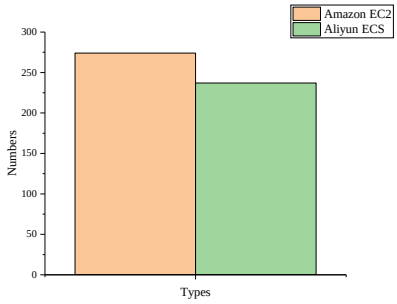
JointCloud aims at empowering the cooperation among multiple clouds to provide efficient cross-cloud services [1]. In this context, each cloud service (e.g., Amazon EC [2], Windows Aure [3], and Aliyun ECS [4]) has its own vendor-lock APIs for managing cloud applications. Although a lot of efforts have tried to provide the generic APIs for cloud developers by various adapter methods, any modification in a vendor-lock API will make JointCloud unusable. The problem of API synchronization is the major obstacle that hinders the cloud adaption.

Recently, several research efforts have been proposed to address the problem of applications' portability [5–10] by providing the generic APIs, but all these APIs are manually implemented. When the parameters of a specific method in a specific API change, all the references to this method need to be manually updated in the adapter code. Authors in [11, 12] have solved the API synchronization problem by providing a semantic-based and white-box method. It means that we should modify varied cloud service APIs. Thus, such methods are unpractical in production.

Corresponding author: wuheng@iscas.ac.cn



(a) Issues of two famous JointCloud implementations



(b) VM API numbers

Fig. 1: Some statistical information for clouds

As shown in Figure 1, (1) nearly 80% and 74% of issues of libcloud and Jcloud, which are the two famous JointCloud implementations, are caused by the API synchronization problem; (2) the API versions of major cloud vendors are updated more than 40 times per year. If we use the white-box methods described in [11, 12], more than 200 APIs of each cloud would

```
AmazonEC2Client aws = getAWSClient();
CreateVpcRequest req = new CreateVpcRequest();
req.setCidrBlock("192.168.1.1");
req.setInstanceTenancy("china");
CreateVpcResult res = aws.createVpc(req );
res.getVpc();
```

(a) Java coding samples for Amazon

```
DefaultAcsClient aliyun = getAliyunClient();
CreateVpcRequest req = new CreateVpcRequest();
req.setCidrBlock("192.168.1.1");
req.setRegionId("china");
CreateVpcResponse res = aliyun
                        .getAcsResponse(req);
res.getVpcId();
```

(b) Java coding samples for Aliyun

Fig. 2: Java coding samples

be maintained manually. So we argue that adapter flows should be decoupled from the manners in which they are executed by using a black box method to generate adapters. The decouple principle can minimize the time-consuming and error-prone manual work.

In this paper, we present Wukong, a proof-of-concept adapter manager which verifies the decouple principle. To decouple adapter flow descriptions from their executions, we rely on the fact that users prefer to use the high-level and generic APIs. Those APIs abstract the low-level details of heterogeneous APIs in clouds. Wukong breaks the tight bound between adapter flow descriptions and execution engines, and our contributions include: (1) We analyse the VM service APIs of seven clouds both in China and US, and extract the API usage patterns from them. (2) We adopt a heuristic-based method to generate the adapter codes with few manual works even when some cloud APIs change. (3) We choose varied VM service APIs to evaluate our method, the results illustrate the feasibility of Wukong framework, although generating APIs would introduce some performance overheads.

The rest of this paper is organized as follows. Section II presents the empirical study to show the API usage patterns. The auxiliary components of the Wukong framework are elaborated in Section III. An example for code generation of the VM case study is shown in Section IV. The evaluation of Wukong framework is discussed in Section V. Section VI provides the related work and Section VII concludes the Wukong framework.

II. Empirical Study

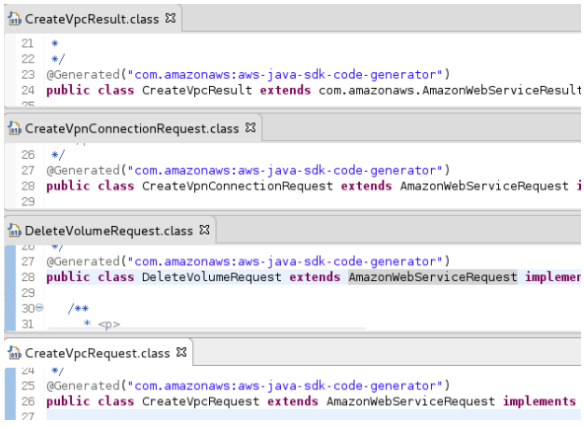
We take the Java APIs of Amazon EC2 and Aliyun ECS as examples. Firstly, we extract the API usage patterns. Then, we use these patterns to generate adapter codes by the heuristic algorithm in Section III.C.

Finding 1: these APIs are in the style of nested objects, and can be expressed by a tree-based data structure.

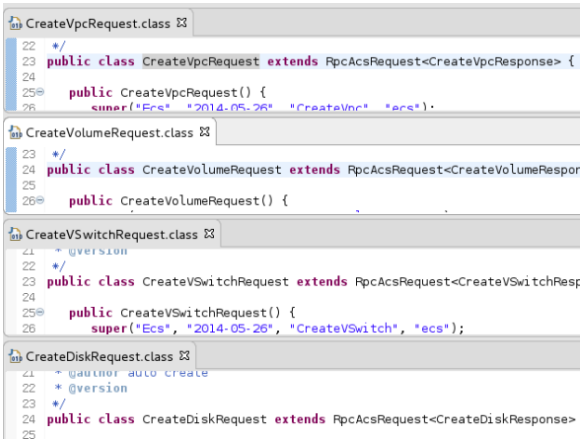
As shown in Figure 2, the root of the tree can express the client object, which is used to connect to the specified cloud. The second level of the tree means the specific operation of varied cloud providers, such as $CreateInstance$, $CreateDisk$ of Aliyun ECS. Moreover, other levels of the tree are parameters, which are primitive types or complex structures of an operation. A complex structure implies that the operation is using a Java getter/setter programming model.

Finding 2: all operations implement the same interface.

From a Java class's perspective, we can extract cloud operations because there are a lot of methods in the client objects. Fortunately, we found that if the return object of a method implements a specific interface, it is a cloud operation other than an auxiliary method, as shown in Figure 3.



(a) all Amazon operations should implement the $AmazonWebServiceRequest$ interface



(b) all Aliyun operations should implement the $RpcAcsRequest$ interface

Fig. 3: Cloud operation's characteristics

These findings will serve as heuristic rules for our subsequent code generation.

III. WuKong framework

This section presents the details of our Wukong framework. we provide the architecture first and then describe two requirements for using it. At last we depict the code generation algorithm.

A. Architecture

As shown in Figure 4, Wukong framework includes three components.

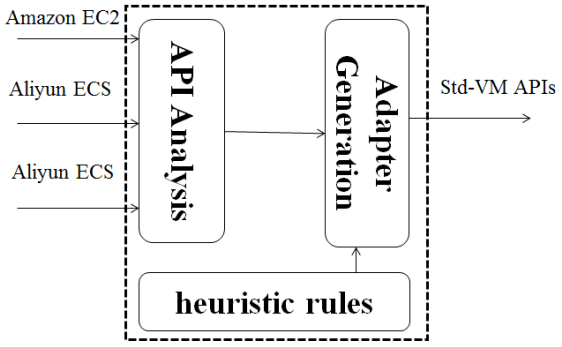


Fig. 4: Wukong architecture

(1) API Analysis Layer: API Analysis Layer (AAL) can automatically analyze a set of VM APIs based on the above heuristic rules, it uses the reflection mechanism to parse a VM specific-API. Some parts of specific-APIs are manually implemented. We will explain the algorithm in Section III.C.

(2) Heuristic rules manager: The heuristic rules of Wukong have three parts: (i) it describes how to find the operations of VM-APIs when Heuristic Rules Manager (HRM) automatically parses binary codes based on the reflection mechanism; (ii) it explains the way to find out all the related parameters of an operation; (iii) it illustrates the mapping relationships between the names of Std-VM APIs and the names of varied cloud APIs.

(3) Adapter generation layer: Adapter Generation (AG) is used to generate the adapters based on the Std-VM API templates and the heuristic rules. It works mainly in two cases: (i) when a new cloud platform is added to the Wukong framework, it can generate a specific adapter and (ii) when an existing cloud specific-API changes, it can generate a new method.

B. Requirements

Although Wukong can semi-automatically generate the adapters based on heuristic rules, it involves two requirements: (1) The proposed Std-VM APIs are based on Java. Only Java applications can currently benefit from the Wukong framework. But the framework can cope with other programming languages if they provide a mechanism similar to the Java reflection (e.g., reflection in .NET and introspection in Ruby). (2) Our framework assumes that the semantic descriptions of all object-oriented methods are available as inputs. For example, $CreateVM$ and $CreateInstance$ from Amazon EC2 and Aliyun ECS are the same operation.

Algorithm 1 Adapters generating algorithm

Input:
heuristic rules: $rules$
Cloud client i: $client_i$
Std-VM API templates: ct
Output:
The adapter for the specified Cloud.

```
1: init queue q
2: for each returnname rn in clientmethods do
3:    if rn.getInterfaces() in rules then
4:       sort(q)
5:    else
6:       q.addLast(rn.getInterfaces())
7:    end if
8: end for
9: init openration tag tag = q.enqueue()
10: init openration set os
11: for each returnname rn in clientmethods do
12:    if rn implements tag then
13:       os.add(rn)
14:    end if
15: end for
16: for each methodname mn in os do
17:    if mn.paramters.number != 1 then
18:       continue
19:    end if
20:    if mn's parameter is simple type then
21:       os.CunrrentgetOperation().add(mn)
22:    else if mn's parameter is complex type then
23:       os.CunrrentgetOperation().add(mn)
24:       goto 21
25:    end if
26: end for
27: for op in os do
28:    Std-API.name = rules.get(op)
29:    for param in op do
30:       Std-API.paramters = rules.get(param)
31:    end for
32:    Generate Std-API
33: end for
```

C. Algorithm

So far we have described how to generate adapters by using heuristic rules. Here, we implemented an algorithm to generate varied adapters, as shown in Algorithm 1. Lines 1-15 show how to find the operations of VM-APIs. Lines 16-26 explain the way to find out all the parameters of an operation. Lines 27-33 illustrate how to generate the adapter by using heuristic rules.

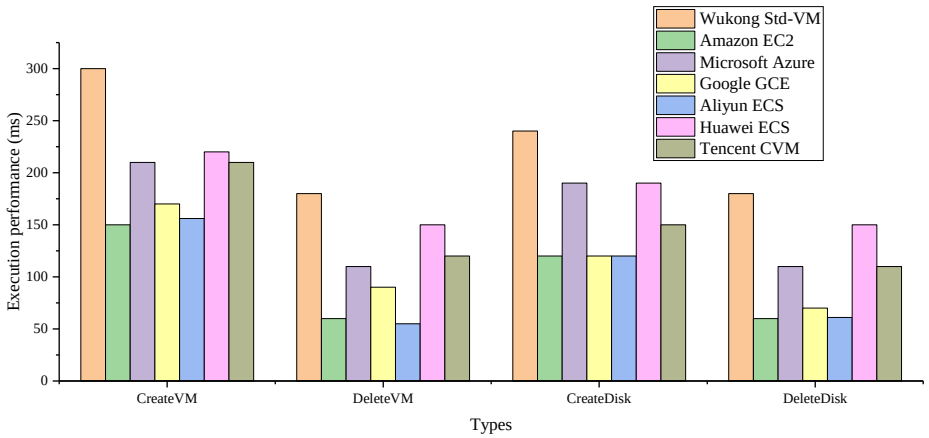


Fig. 5: The performance of Std-VM APIs

IV. Evaluation

Now we evaluate the effectiveness of Wukong's work with three aspects: (1) can Wukong find out all the operations based on heuristic rules for major cloud vendors? (2) comparing with varied cloud APIs, can Std-VM APIs get a better performance? (3) is the code generation time-consuming?

A. Heuristic rules

Metrics. We analyze Amazon EC2, Microsoft Azure, Google GCE, Aliyun ECS, Huawei ECS and Tencent CVM, which are the popular cloud vendors in China and the US. We compare the operation counts by gathering the official API documents for each cloud. If the value is equal, we consider the heuristic rules work effectively.

As shown in Table 1, the heuristic rules work effectively except for Aliyun ECS and Huawei ECS. For Aliyun ECS, we find 40 more APIs which are not described in the official ECS API documents, but we find these APIs are in the official VPC API documents. For Huawei ECS, we cannot find almost 200 APIs which are described in the official ECS API documents. The reason is that Openstack4j API has a low quality, which is extended by Huawei ECS. Figure 5 shows that some operations are implementing the same interface, but the others are not. So our heuristic rule 2 (all operations are implementing the same interface) does not work.

TABLE I: APIs number from two sources

Providers	Heuristic rules	Official document
Amazon EC2	274	274
Microsoft Azure	242	242
Google GCE	256	256
Aliyun ECS	237	187
Huawei ECS	212	421
Tencent CVM	168	168

B. The performance of Std-VM APIs

Metrics. By selecting 4 APIs randomly, we evaluate the execution time of cloud APIs compared with the related Std-API. We consider that a better Std-VM API takes a less execution time.

As shown in Figure 6, the performance of the Std-APIs' execution time is about 2× slower compared with the official APIs. Besides, we also find that the API execution performance is different from each official API, the maximum performance difference is nearly 1.5×. The API execution performance of Amazon EC2 and Aliyun ECS outperform the others. One possible reason is that the two cloud vendors are well in Java optimizations.

```
public interface ComputeService extends RestService {

    public interface ComputeImageService extends RestService {

        public interface ServerService {
```

Fig. 6: Huawei ECS API samples(we can see $ServerService$ is missing its parent, so we have to ignore it)

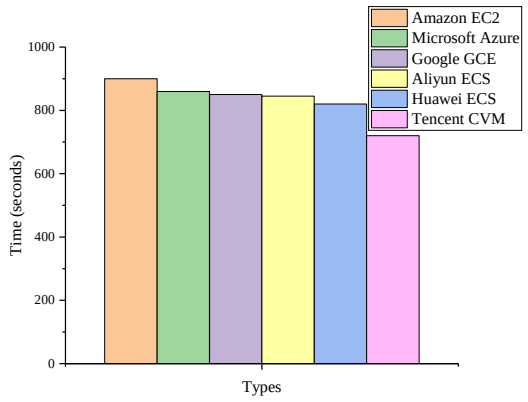
C. The cost of the code generation

Metrics. The cost of the code generation includes two parts: (1) the automatic analysis for cloud APIs by using heuristic rules and (2) The generation of the adapter.

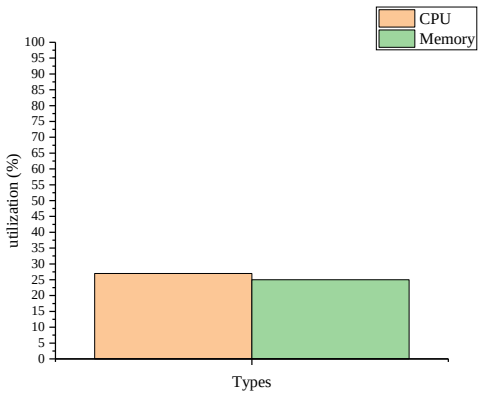
We deploy the operations on a machine with 4 Cores (i5-3210M CPU @2.50GHZ), 8 GB RAM and CentOS 7.5. As shown in Figure 7, the generating time of Wukong for an adapter cost is about 15 minutes. And it is much longer if the cloud has more APIs. Practically, running Wukong only cost about 1CPU and 2G memory and the cost of Wukong is acceptable.

V. Related work

Std-PaaS APIs: Std-PaaS APIs [12] aim at to solve the application portability problem and many methods have been



(a) Generating performance for different Clouds of Wukong



(b) Resource utilization of Wukong

Fig. 7: Wukong framework performance

discussed to evaluate the various kinds of applications' portability [6, 7, 10]. A development framework is designed in [11] to alleviate the heterogeneity among clouds. Minimal Protocol Adaptors [13] only concentrate on the messages that cause services mismatch and gain the efficiency of adaption in the design and running period. A PaaS recommending algorithm is proposed in [14] by APIs matchmaking and ranking.

Storage-SaaS-APIs: CoCloud [15] supports file collaborations among four different storage services by being aware of nearby proxies and the similarity of different versions in a file. Unidrive [16] and MetaSync [17] employ distributed consistency protocols and complex cooperative structures to guarantee data security and consistency among various consumer cloud storage (CCS) services. Cdport [8] proposes a common data model to transfer data or software among popular NoSql systems.

Unified interface: an unified interface is proposed in [18]. But in production environments, it still faces a huge gap to fulfil due to the private cloud vendors' interests. As the definition of the artifacts has been raised in [19], COAPS API [20, 21] is designed to deploy applications on some clouds like Cloud Foundry (CF) and Google AppEngine (GAE) with isolating API managements. However, those methods do not concern much about the API modification.

Semantics-based APIs: STAGER [22] also brings out the idea of semantic adapters of manual interventions under the Source Code Ontology Population (SCOP). But the rules of constructing SCOP are not mentioned and fewer cloud providers are compared, which makes it less practical and available in general situations. Other semantics-based frameworks like [1, 5] take API synchronization problems into consideration and solve them by less-invasive measures (e.g., annotations, SOA).

VI. Conclusion

We presented Wukong, a framework which can dynamically map front-end adapter flow descriptions by the generic APIs to a broad range of back-end cloud vendors. Although there exists some overheads for generating the generic APIs, the evaluation results prove the feasibility of our Wukong framework. In the future, we will extend Wukong to support more general production environments like data storage with less manual interventions.

Acknowledgment

This work was supported by National Key Research and Development Program of China (2016YFB100103).

References

[1] Eman Hossny, Sherif Khattab, Fatma A Omara, and Hesham Hassan. Semantic-based generation of generic-api adapters for portable cloud applications. In *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms*, page 1. ACM, 2016.

[2] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[3] Borko Furht and Armando Escalante. *Handbook of Cloud Computing*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[4] Dimitris Zeginis, Francesco D'andria, Stefano Bocconi, Jesus Gorronogoitia Cruz, Oriol Collell Martin, Panagiotis Gouvas, Giannis Ledakis, and Konstantinos A Tarabanis. A user-centric multi-paas application management solution for hybrid multi-cloud scenarios. *Scalable Computing: Practice and Experience*, 14(1):17–32, 2013.

[5] Francesco DAndria, Stefano Bocconi, Jesus Gorronogoitia Cruz, James Ahtes, and Dimitris Zeginis. Cloud4soa: multi-cloud application management across paas offerings. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 407–414. IEEE, 2012.

[6] Stefan Kolb and Guido Wirtz. Towards application portability in platform as a service. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)*, pages 218–229. IEEE, 2014.

[7] Ajith Ranabahu, E Michael Maximilien, Amit Sheth, and Krishnaprasad Thirunarayan. Application portability in cloud computing: an abstraction-driven perspective. *IEEE Transactions on Services Computing*, 8(6):945–957, 2015.

[8] Ebtesam Alomari, Ahmed Barnawi, and Sherif Sakr. Cdport: A framework of data portability in cloud platforms. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, pages 126–133. ACM, 2014.

[9] Luís A Bastião Silva, Carlos Costa, and José Luís Oliveira. A common api for delivering services over multi-vendor cloud resources. *Journal of Systems and Software*, 86(9):2309–2317, 2013.

[10] Sami Yangui, Roch H Glitho, and Constant Wette. Approaches to end-user applications portability in the cloud: A survey. *IEEE Communications Magazine*, 54(7):138–145, 2016.

[11] Fotis Gonidis, Iraklis Paraskakis, and Anthony JH Simons. Leveraging platform basic services in cloud application platforms for the development of cloud applications. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 751–754. IEEE, 2014.

[12] Eman Hossny, Sherif Khattab, Fatma Omara, and Hesham Hassan. Towards standard paas implementation apis. *International Journal of Cloud Computing*, 6(4):306–324, 2017.

[13] Rik Eshuis, Ricardo Seguel, and Paul Grefen. Synthesizing minimal protocol adaptors for asynchronously interacting services. *IEEE Transactions on Services Computing*, (3):461–474, 2017.

[14] Nick Bassiliades, Moisis Symeonidis, Georgios Meditskos, Efstratios Kontopoulos, Panagiotis Gouvas, and Ioannis Vlahavas. A semantic recommendation algorithm for the paasport platform-as-a-service marketplace. *Expert Systems with Applications*, 67:203–227, 2017.

[15] E Jinlong, Yong Cui, Peng Wang, Zhenhua Li, and Chaokun Zhang. Cocloud: Enabling efficient cross-cloud file collaboration based on inefficient web apis. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):56–69, 2018.

[16] Haowen Tang, Fangming Liu, Guobin Shen, Yuchen Jin, and Chuanxiong Guo. Unidrive: Synergize multiple consumer cloud storage services. In *Proceedings of the 16th Annual Middleware Conference*, pages 137–148. ACM, 2015.

[17] Seungyeop Han, Haichen Shen, Taesoo Kim, Arvind Krishnamurthy, Thomas Anderson, and David Wetherall. Metasync: File synchronization across multiple untrusted storage services. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 83–95, 2015.

[18] Stefan Kolb and Cedric Röck. Unified cloud application management. In *Services (SERVICES), 2016 IEEE World Congress on*, pages 1–8. IEEE, 2016.

[19] Mark Carlson, Martin Chapman, Alex Heneveld, Scott Hinkelman, Duncan Johnston-Watt, Anish Karmarkar, Tobias Kunze, Ashok Malhotra, Jeff Mischkinsky, Adrian Otto, et al. Cloud application management for platforms. *OASIS, http://cloudspecs. org/camp/CAMP-v1. 0. pdf, Tech. Rep*, 2012.

[20] Mohamed Sellami, Sami Yangui, Mohamed Mohamed, and Samir Tata. Paas-independent provisioning and management of applications in the cloud. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 693–700. IEEE, 2013.

[21] Eman Hossny, Sherif Khattab, Fatma A Omara, and Hesham A Hassan. Implementing generic paas deployment api: repackaging and deploying applications on heterogeneous paas platforms. *International Journal of Big Data Intelligence*, 3(4):257–269, 2016.

[22] Sherif Khattab, Fatma A Omara, Hisham Hassan, et al. Stager: Semantic-based framework for generating adapters of service-based generic-api for portable cloud applications. *IEEE Transactions on Services Computing*, 2018.