MyMinder: A User-centric Decision Making Framework for Intercloud Migration

Esha Barlaskar, Peter Kilpatrick, Ivor Spence and Dimitrios S. Nikolopoulos
The School of Electronics, Electrical Engineering and Computer Science,
Queen's University Belfast, BT7 1NN, Belfast, U.K.

Keywords: Cloud Computing, Dynamic Decision Making, QoS Monitoring, Inter-cloud Migration.

Abstract: Each cloud infrastructure-as-a-service (IaaS) provider offers its own set of virtual machine (VM) images and hypervisors. This creates a vendor lock-in problem when cloud users try to change cloud provider (CP). Although, recently a few user-side inter-cloud migration techniques have been proposed (e.g. nested virtualisation), these techniques do not provide dynamic cloud management facilities which could help users to decide whether or not to proceed with migration, when and where to migrate, etc. Such decision-making support in the post-deployment phase is crucial when the current CP's Quality of Service (QoS) degrades while other CPs offer better QoS or the same service at a lower price. To ensure that users' required QoS constraints are achieved, dynamic monitoring and management of the acquired cloud services are very important and should be integrated with the inter-cloud migration techniques. In this paper, we present the problem formulation and the architecture of a Multi-objective dYnamic MIgratioN Decision makER (MyMinder) framework that enables users to monitor and appropriately manage their deployed applications by providing decisions on whether to continue with the currently selected CP or to migrate to a different CP. The paper also discusses experimental results obtained when running a Spark linear regression application in Amazon EC2 and Microsoft Azure as an initial investigation to understand the motivating factors for live-migration of cloud applications across cloud providers in the post-deployment phase.

1 INTRODUCTION

With the addition of more and more Cloud Infrastructure-as-a-Service (IaaS) providers in the cloud market, cloud IaaS users have to face a number of challenges while selecting a particular cloud provider (CP). One of the main challenges in CP selection stems from performance variability amongst the CPs. Different CPs provide similar services (i.e. same virtual machine (VM) specification) but with differing performance levels, which results in considerable variations in the QoS. Another challenge in CP selection arises due to the diversified prices and provisioning policies of the instances (from here on we will use the terms VM and instance interchangeably), e.g. different CPs offer their services at different prices for different provisioning policies of the instances, such as, on demand instances, spot instances, reserved instances, etc.

Cloud users face further challenges even after selecting an appropriate CP as they need to verify whether their applications are performing in a stable manner with minimum or acceptable variations after being deployed in the CP's instances. Performance variability in the post-deployment phase usually arises when the instances of multiple users are running on the same physical node, which leads to multi-tenancy problems for I/O-bound applications even if the users select specialised dedicated instances (Leitner and Cito, 2016) (Li et al., 2012) (Li et al., 2013). Multi-tenancy problems arise because most of the computing resources (network and disk I/O) except for CPU cores are shared amongst all the users' instances running in a node. Such variations in QoS for latency-sensitive applications like web services may become a matter of concern and sometimes performance of a user application running in an instance may significantly degrade and violate QoS due to the behaviour of the co-located instances from other users. In addition, the price for the current cloud service may rise or other providers may offer special discounts after the deployment of the instance.

In order to help cloud users verify the performance of their applications, cloud service monitoring tools are provided by CPs and third party companies (Scheuner et al., 2014) (Silva-Lepe et al., 2008) (Ciuffo-

Barlaskar, E., Kilpatrick, P., Spence, I. and Nikolopoulos, D.
MyMinder: A User-centric Decision Making Framework for Intercloud Migration.
DOI: 10.5220/0006355905880595
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 560-567
ISBN: 978-989-758-243-1

letti, 2016). However, the monitoring tools do not provide any decision support on what steps a cloud user should follow if he/she realises that even their minimum QoS requirements are not met by the selected instances in the current provider. In order to meet the desired QoS requirements cloud users may require to migrate their applications to new instance type from the same provider with higher configuration than the existing one or from a new provider with a similar configuration. Taking decisions on whether to migrate applications for better QoS poses further decision making and technical challenges in migrating applications across CPs.

Although some of the researchers have tried to address the issues in inter-cloud migration, they have not provided any decision-making support. Others have focussed mainly on pre-deployment decision-making and there has been some work on post-deployment phase support, but these latter do not consider realistic migration overheads in the evaluation of their decision making framework. Therefore, naive cloud users should have an efficient dynamic decision making framework, which can help to provide a composite decision on the following:

1. How to detect if the current provider is not performing as required by user's application?
2. How to decide that the user's application needs to be migrated from the current provider?
3. Which alternative CP should be chosen to migrate the VM?
4. What instance type(s) will provide the best trade-off between cost and performance?
5. Whether the migration overhead will be more significant compared to the performance degradation in the current CP?

Considering the above queries, we propose a Multi-objective dYnamic MIgratioN Decision makER (MyMinder) framework. MyMinder offers a catalogue of metrics based on performance, cost and type of resources, from which cloud users can choose their requirement metrics depending on their application. Also, while choosing these metrics users can set certain ranges for the desired minimum and maximum QoS performance and cost values, which define acceptable variability from the preferred QoS and user's budget. MyMinder takes these requirements as inputs to carry out the monitoring and computes user satisfaction values based on their QoS requirements. In the event of any QoS violation or performance degradation MyMinder supports the user in finding alternative cloud services which can provide near-optimal performance, and efficiently migrates the application to that service provided by either the same or different CP. This paper makes the following contributions:

- Proposal of the architecture of MyMinder, a post-deployment decision making framework, which can dynamically decide whether user's VM should be migrated from the current CP to another CP.
- Problem formulation for selecting the most suitable CP to migrate the VM to.
- Preliminary experimental results from evaluating the performance of a user application running on public clouds, which motivates the requirement for a live VM migration in the post-deployment phase.

The remainder of the paper is organised as follows. Section 2 presents background and related work in user centric live VM migration and decision making. Section 3 and Section 4 provide detail of the problem formulation and the MyMinder architecture, respectively. Preliminary experimental results are evaluated and discussed in Section 5. Section 6 concludes the paper and discusses future work.

2 BACKGROUND AND RELATED WORK

With the proliferation of CPs it has become very difficult for cloud users to select the one that best meets their needs. Once they select the perceived optimal cloud service from a CP, cloud users encounter further challenges as they need to verify whether their applications are performing in a stable manner with minimum or acceptable variations after being deployed in the CP's instances. If the user realises that even their minimal QoS requirements are not met by the selected instances then they may require to migrate their applications to a new instance type from the same provider or to an instance with a similar configuration from a new provider. Taking decisions on whether to migrate applications for better QoS poses further decision-making and technical challenges for the user. We discuss how current work in the literature addresses these challenges in the following sections.

2.1 Post-deployment Decision Making

Although researchers have proposed different decision making methods in the pre-deployment phase (Li et al., 2010), (Brock and Goscinski, 2010), (Han et al., 2009), (Silas et al., 2012), (Rehman et al., 2014), (u. Rehman et al., 2013) decision making in

the post-deployment phase has not received much attention, other than the works in (ur Rehman et al., 2015) and (Li et al., 2011).

The authors in (ur Rehman et al., 2015) address decision making in the post-deployment phase by proposing a multi-stage decision-making approach. In the first stage, the available CP instances are shortlisted on the basis of the user's minimum QoS and cost criteria, and in the second stage, migration cost and time are evaluated. After completing these stages, they use the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) (Behzadian et al., 2012) and ELimination Et Choix Traduisant la REalit (ELimination and Choice Expressing REality), commonly known as ELECTRE (Roy, 1991), to find the most appropriate migration suggestion. They demonstrate their approach using a case study example. However, in their evaluation, they consider the overhead of a manual migration process where they assume that the network throughput between the source and the destination hosts remains constant during the migration process, which is unlikely to be true in real scenarios.

In (Li et al., 2011) a linear integer programming model for dynamic cloud scheduling via migration of VMs across multiple clouds is proposed in the context of a cloud brokerage system. The migration is triggered if a CP either offers a special discount or introduces a new instance type, and also if the user needs to increase the infrastructure capacity. They do not consider QoS violation or degradation in their migration decision. Moreover, they performed their experiments in a simulation based environment and the metrics that they considered for measuring migration overhead may not be feasible to obtain in real world scenarios.

2.2 User-centric Inter-cloud Migration

Although cloud users should not be worried about the complexities involved in VM migration - which is the essence of the 'cloud philosophy', experienced cloud users may wish to have the flexibility that migration brings in the form of inter-cloud migration. However, there are complexities in migrating VMs from one CP to another CP due to vendor lock-in issues. For users to avail of the benefits of VM migration independent of the cloud provider's permission, recent studies proposed different inter-cloud migration techniques which use a second layer of hardware virtualisation called nested virtualisation (Williams et al., 2012), (Jia et al., 2015), (Razavi et al., 2015). Nested VMs are usually migrated by using an NFS-based solution or an iSCSI-based solution. In some cases such as that of (Ravello, 2016) the focus is not on providing storage and network support for wide-area network (WAN) application but rather on providing an enclosed environment for distributed application development and debugging. In an NFS-based and iSCSI-based solution the WAN VM migration experiences increased latencies, low bandwidth, and high internet cost in accessing a shared disk image if the shared storage is located in a different data centre or region. To address this issue (Shen et al., 2016) proposed Supercloud using nested virtualisation with a geo-replicated image file storage that maintains the trade-off between performance and cost. However, Supercloud does not provide any decision-making framework.

Other state-of-the-art techniques which allow multi-cloud deployment are Docker (Docker, 2013) and Multibox containers (Hadley et al., 2015). Although containers are lightweight, they are not ideal for stateful applications due to limited volume management in the case of container failover. Moreover, containers are not as pliable as VMs because all containers must share the same kernel and also both Docker and Multi-box container migration techniques are in their preliminary stages. Our proposed MyMinder system will adopt the most robust and resilient technique amongst the available inter-cloud migration techniques after evaluating their complexities and migration overhead. We envisage a system which can handle inter-cloud migration automatically along with a decision making framework, thus delivering the best of both worlds.

3 PROBLEM FORMULATION

In this paper we present the MyMinder framework (Figure 1), which can assist cloud users in achieving a stable QoS performance in the post-deployment phase by helping decide on actions to be taken as well as providing support to achieve such actions. MyMinder can monitor the performance of the deployed users' applications and provide the required measurements to determine the satisfaction level of the user's requirements described in their requests. In the event of QoS violation or degradation in the current CP's service, MyMinder can trigger a migration decision after identifying a suitable CP to which the overhead of migration and the chances of QoS violation are the least. For performing these actions MyMinder needs to evaluate the satisfaction values based on the QoS requirements specified in the user's requests. In the following subsections we illustrate user requirements, details of the CP instance type model, and the related

measures.

3.1 User Requirements

A user sends a request describing his/her resource requirements and QoS requirements. This request is represented by a requirement vector : $r_i = [r_{i1}, r_{i2}, ..., r_{ij}]$ where r_{ij} specifies the $jth(j = 1, 2, ...J)$ requirement of user i that has to be satisfied by the selected CP and these requirements may include the following information criteria: 1) Resource criteria: amount of resources required for running the user's application (e.g. memory, storage, CPU etc.). 2) Budget constraint: prices of the instances should be within the cost limit of the user. 3) Migration overhead constraint: cost of migration and performance overhead of migration should be acceptable. 4) QoS criteria: Quality of service requirements of the user's application that has to be fulfilled (e.g. desired and maximum execution time, response time, etc.).

Here, criteria 1 to 3 will be evaluated before deploying the application and only if these criteria are met then the application will be deployed and after deploying the application criteria 4 will be measured using a satisfaction value.

3.2 CP Instance Types Model

Instances of different CPs differ in performance depending on their characteristics such as VM instance size, hardware infrastructure, VM placement policies used for load balancing or power optimisation etc. Reasons affecting the QoS obtained from a particular instance type of a CP are typically not known by the user and so the QoS data of a given CP are not available in advance. It is possible to measure the QoS parameters only after the instance is deployed and these measurements may be evaluated against the requirements specified in the user request by determining the runtime performances such as execution time of applications, instructions committed per second (IPS), etc. These measurements constitute the evaluation of the extent to which the QoS requirements specified in the user's request r_{ij} are satisfied. The satisfaction level of user requirement r_{ij} is denoted by $s_{ij} \in [0, 1]$, where $s_{ij} = 1$ if the requirement r_{ij} is fully satisfied, otherwise $0 \leqslant s_{ij} < 1$.

If a user provides the requirement vector r_i along with the desired QoS requirement and acceptable maximum variability in the QoS, then standard deviation (*SD*) is used as a measure of QoS performance variability. The closer the *SD* is to 0, the greater is the uniformity of performance data to the desired value ($r_{Qd(r_{ij})}$) and greater is the satisfaction value. The closer the *SD* is to 1, the greater is the variability of performance data to the desired value and smaller is the satisfaction value. Hence, the satisfaction value is given as follows:

$$s_{ij} = 1 - SD \tag{1}$$

$$SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Qa(r_{ij}) - \overline{M(r_{ij})})^2} \tag{2}$$

$$M(r_{ij}) = \frac{1}{N-1}\sum_{i=1}^{N}(Qa(r_{ij})) \tag{3}$$

where,
$Qa(r_{ij})$=Actual QoS value obtained after deploying the user's application (e.g. actual execution time, response time, etc.). These values are in normalised form.
$M(r_{ij})$= The arithmetic mean of $Qa(r_{ij})$.
$r_{Qd(r_{ij})}$= Desired QoS requirements of the user's applications (e.g. desired execution time, response time, etc.) for the QoS requirement r_{ij}. This value is used as a standard value against which QoS variability is compared.
N= total number of measurements.

3.3 Utility Function

The utility function $f(r_i)$ for each user request r_{ij} is a linear combination of the satisfaction value s_{ij} and the associated weights w_{ij} multiplied by an indicator function $\phi(r_i)$. The weight for each of the user request indicates its importance to the user and the indicator function sets the satisfaction level to zero when the request is not satisfied. In the case of satisfied requests the value of the indicator function is selected such that: $\phi(r_i) = (\sum_j w_{ij})^{-1}$ normalises the weight vector and limits the maximum possible value of $f(r_i)$ to 1. Thus, the utility function is defined as:

$$f(r_i) = \phi(r_i)\sum_{n=1}^{J} w_{ij}s_{ij} \tag{4}$$

where

$$\phi(r_i) = \begin{cases} 0, & \text{if } QoS \text{ not met}. \\ (\sum_j w_{ij})^{-1}, & \text{otherwise}. \end{cases} \tag{5}$$

If all the requirements of a user are fully satisfied then $f(r_i) = 1$; otherwise if the requirements are partially satisfied then the value of $f(r_i)$ will vary with the extent of requirements being satisfied by a particular instance type of a CP. To demonstrate this we consider an example.

Let $r_i = [r_{i1}, r_{i2}, ..., r_{ij}]$ be the user's requirement vector while making his/her initial request. The request contains the user's requirements constraints and

the type of the requirement attributes are presented below:

1) r_R: Requested amount of resources required for running the user's application (e.g. memory, storage, CPU etc.) where $r_i \in micro, small, medium, large, xlarge$.

2) r_B: Prices of the instances specified in the user's budget where $r_B \in Max_{price}$

3) r_{Mo}: Maximum migration overhead a user can accept where $r_M \in Overhead\ of\ migration$.

4) r_{Qd}: Desired QoS requirements of the user's applications (e.g. desired execution time, response time, IPS, etc.) where $r_{Qd} \in D_{val}$.

The value of the satisfaction vector is calculated with the help of monitoring and detection modules (see Section 4) which is given by S_i^T (Equation 1). We assume that for a user's request with a requirement vector r_i=[micro, 200s, 400s, £5/hr, 30%], the satisfaction vector is calculated as:

$$S_i^T = [1,0,1,1] \tag{6}$$

For simplifying the example we did not consider partial satisfaction values, and here 0 denotes fully satisfied and 1 denotes not satisfied. Therefore the utility value is calculated as follows if the weight vector is $W_i^T = [0.1,0.1,0.1,0.3]$:

$$f(r_i) = \begin{cases} 0, & \text{if } QoS\ not\ met. \\ \phi(r_i) W_i^T S_i^T = 0.5, & \text{otherwise.} \end{cases} \tag{7}$$

The indicator function's value is considered to be 1 in this case and also the utility function's value did not exceed 0.5 even though more than half of the requirements were fully satisfied.

These utility values will be used to predict the QoS for each CP's instance types model.

4 MyMinder ARCHITECTURE

In this section we describe the architecture of MyMinder that will implement the system. Figure 1 depicts the MyMinder architecture, which includes modules for: monitoring, detection, prediction, and decision making. We describe each of these modules in the following subsections.

4.1 Monitoring Module

The monitoring module is designed for monitoring the QoS performance of the user's application deployed in the VM. The performance data are collected by local monitoring agents deployed in each user's VM. The local monitoring agents send the collected data periodically to the global monitoring component in the monitoring module and then finally the data are stored in the QoS performance repository. Also, the monitoring module maintains another repository, which stores information regarding the list of available VMs from different CPs and their prices. This information is collected by CP profiling components.

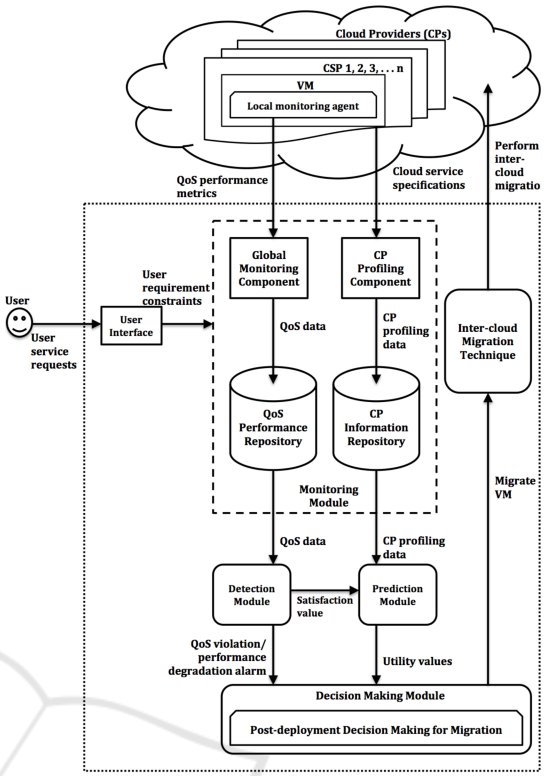

Figure 1: MyMinder Architecture.

4.2 Detection Module

The detection module is responsible for detecting any QoS violation or degradation in the performance. The performance data are retrieved from the QoS performance repository. It uses a window-based violation detection technique (Meng and Liu, 2013) to generate QoS violation or performance degradation alarms based on the user's QoS requirement constraints and the user can decide the size of the window. This module generates QoS violation alarms if the current performance value falls outside the acceptable range as defined by the QoS statement. It also can be tuned to generate a degradation alarm if the performance moves to and stays within a defined distance of the QoS limits throughout a defined period. Degradation alarms may be used to predict likely breach of QoS and so may contribute to preventative migration. The

module reports QoS and degradation alarms on a continuous basis by sending them to the decision making module.

4.3 Prediction Module

The objective of the prediction module is to help determine a suitable CP instance to which the user's application may be migrated. Based on the user's QoS satisfaction values (measured by the detection module) and the user's requirements, the prediction module calculates the utility function (see Equation 4) for each of the available CP instances. The satisfaction values for the current as well as previously deployed CP instances by the same user or different users are stored with their corresponding utility values. These perceived utility values are used to train the prediction models for each of the CP instances using machine learning techniques. Thus, the prediction models are capable of predicting the QoS satisfaction values in the destination CP for the new user's instance which needs migration.

4.4 Decision-making Module

The decision making module receives alarms from the detection module if any QoS violation or degradation is detected, and also it takes utility function values as input from the prediction module. It then checks with user requirement constraints to know whether the user wants to be informed before reaching the minimum requirement levels, i.e. performance degradation alert or to be informed if the minimum requirements are not met, i.e. QoS violation alert. After confirming user requirements, this module verifies whether the instances with different utility values provided by the prediction module are currently available for selection. If the instances are available then it evaluates the migration overhead of each of the instances and finally ranks the instances based on their utility value and migration overhead values. The instance with highest utility value and lowest migration overhead is chosen for migration. The migration overhead will depend on the type of inter-cloud migration technique being used. The migration overhead can be defined either in terms of monetary loss or performance loss and it usually denotes the service downtime penalty per time unit.

5 PRELIMINARY EXPERIMENTS IN PUBLIC CLOUDS

In this section we present some experimental results to showcase that user application performance varies across CPs and even when running in the same instance of the same CP, which may lead to QoS degradation and violation. The goal of this experimental evaluation is to understand why a cloud user may require a dynamic decision making framework and inter-cloud migration of VMs. For the experiments we chose two public cloud providers: Amazon Elastic Cloud Compute (EC2) and Microsoft Azure. We considered the t2.micro instance with 1GB memory and 1 CPU core in Amazon EC2, and Dsv2 (which is a small instance) with 3GB memory and 1 CPU core in Azure. We ran a Spark application which performs linear regression between two randomly generated variables with 1 million rows of data each containing two floating variables in both the cloud providers.

5.1 Experimental Result

We evaluate the experimental results from both cloud providers in order to understand how user application performance varies per day and per week. We collected performance metrics (application execution time) of two weeks (7 October to 20 October, 2016) from both the cloud providers. Figure 2 and Figure 3 present box-plots, which show performance variability in both Amazon and Azure.

In Amazon (Figure 2), there is significant variability in week 1 with a relative standard deviation of 81%. In week 2, although the performance does not improve, there is less variability with a relative standard deviation of 10%. Whenever we stop the application and restart the instance, the application seems to perform better for a number of executions but after it is executed continuously the performance degrades and shows significant variation compared to the initial performance. However, after degrading to a certain extent the performance seems to remain in that range without further degradation or improvement. Such variability is primarily because of the instance type t2.micro which is a general purpose instance and suffers from CPU stealing as they share their processor with other tenants based on a credit system.

In Azure (Figure 3), while there is some variation in both weeks, the difference in performance variation from week 1 to week 2 is less as compared to Amazon. In week 1, the relative standard deviation is 25%, whereas in week 2, although the performance does not improve, there is less variability with a relative standard deviation of 18%. Thus, we observe that QoS

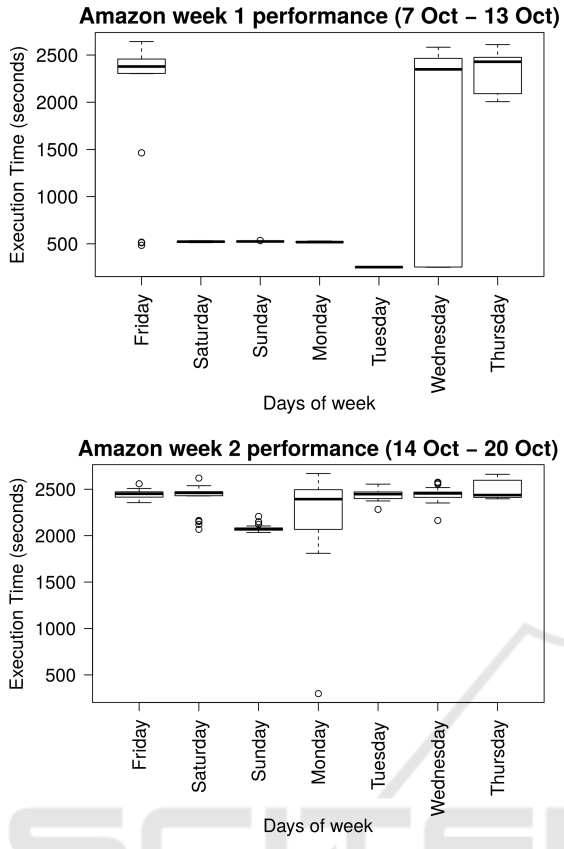

Figure 2: (Top) Amazon week 1 performance. (Bottom) Amazon Week 2 performance.

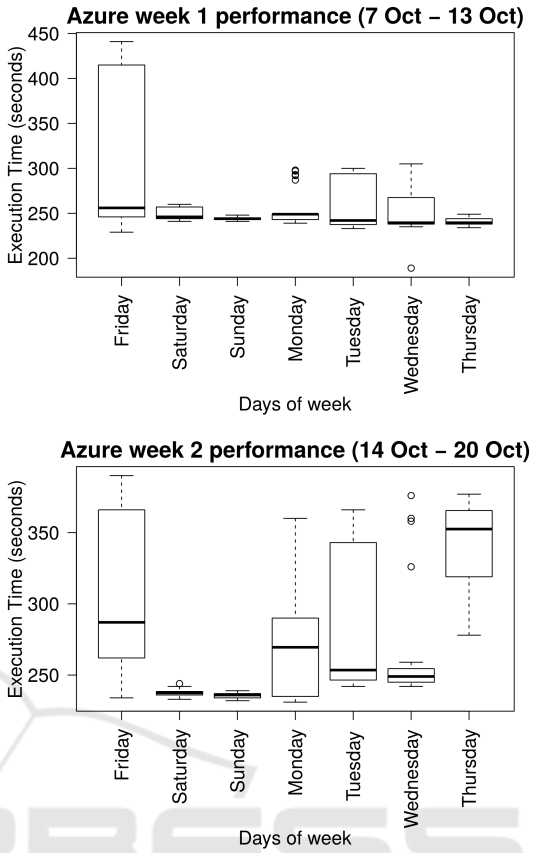

Figure 3: (Top) Azure week 1 performance. (Bottom) Azure Week 2 performance.

varies across CPs and even within the same instance type of the same CP while running a user application. This observation motivates us further in implementing MyMinder in order to dynamically manage cloud user applications in maintaining a stable QoS throughout the lifetime of the applications.

6 CONCLUSION AND FUTURE WORK

In this paper we have proposed the architecture of MyMinder, a post-deployment decision making framework, which can detect Cloud QoS violation and performance degradation and dynamically decide whether a user's VM requires migration from the current provider to another provider. In addition, we have presented the problem formulation for selecting the most suitable CP in the case that the VM requires migration from the current provider. We performed some initial experiments to understand how a real application performs after being deployed in public cloud and whether significant performance variation actually occurs. These experiments motivate the need for live VM migration from one CP to another.

As part of MyMinder implementation, we are currently experimenting with different inter-cloud migration techniques as discussed in Section 2.The key challenge in adopting these techniques is to maintain the baseline performance in spite of the extra layer of indirection due to the second-layer hypervisor. We executed sysbench CPU benchmark and a Spark application in the nested VMs of Xen-Blanket in OpenStack VMs and observed an overhead of 20-30% on the application performance. This overhead is mainly due to the indirection and two levels of scheduling in nested virtualisation. We assume that this overhead can be reduced by using Docker containers. However, Docker containers bring other challenges in the form of creating a distributed storage for migrating containers with its associated storage across different CPs. Currently, Docker containers can be migrated by using a storage plugin called Flocker within a data centre, i.e. within one single CP.

Our future work includes building the proof of concept for MyMinder and incorporating within it one of the state-of-the-art inter-cloud migration techniques. Also, we will consider a suitable machine

learning algorithm in our prediction module.

REFERENCES

Behzadian, M., Khanmohammadi Otaghsara, S., Yazdani, M., and Ignatius, J. (2012). Review: A state-of the-art survey of TOPSIS applications. *Expert Syst. Appl.*, 39(17):13051–13069.

Brock, M. and Goscinski, A. (2010). Toward ease of discovery, selection and use of clusters within a cloud. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 289–296.

Ciuffoletti, A. (2016). Application level interface for a cloud monitoring service. *Computer Standards and Interfaces*, 46:15 – 22.

Docker (2013). Docker Containers. `https://www.docker.com/`. [Online; accessed 25-October-2016].

Hadley, J., Elkhatib, Y., Blair, G., and Roedig, U. (2015). *MultiBox: Lightweight Containers for Vendor-Independent Multi-cloud Deployments*, pages 79–90. Springer International Publishing, Cham.

Han, S.-M., Hassan, M. M., Yoon, C.-W., and Huh, E.-N. (2009). Efficient service recommendation system for cloud computing market. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 839–845, New York, NY, USA. ACM.

Jia, Q., Shen, Z., Song, W., van Renesse, R., and Weatherspoon, H. (2015). Supercloud: Opportunities and challenges. *SIGOPS Oper. Syst. Rev.*, 49(1):137–141.

Leitner, P. and Cito, J. (2016). Patterns in the chaos – a study of performance variation and predictability in public iaas clouds. *ACM Trans. Internet Technol.*, 16(3):15:1–15:23.

Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA. ACM.

Li, W., Tordsson, J., and Elmroth, E. (2011). Modeling for dynamic cloud scheduling via migration of virtual machines. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 163–171, Washington, DC, USA. IEEE Computer Society.

Li, Z., OBrien, L., and Zhang, H. (2013). Ceem: A practical methodology for cloud services evaluation. In *2013 IEEE Ninth World Congress on Services*, pages 44–51.

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

Meng, S. and Liu, L. (2013). Enhanced monitoring-as-a-service for effective cloud management. *IEEE Transactions on Computers*, 62(9):1705–1720.

Ravello (2016). Ravello Systems: Virtual Labs Using Nested Virtualization. `https://www.ravellosystems.com`. [Online; accessed 15 November-2016].

Razavi, K., Ion, A., Tato, G., Jeong, K., Figueiredo, R., Pierre, G., and Kielmann, T. (2015). Kangaroo: A tenant-centric software-defined cloud infrastructure. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, pages 106–115.

Rehman, Z. U., Hussain, O. K., and Hussain, F. K. (2014). Parallel cloud service selection and ranking based on qos history. *Int. J. Parallel Program.*, 42(5):820–852.

Roy, B. (1991). The outranking approach and the foundations of electre methods. *Theory and Decision*, 31(1):49–73.

Scheuner, J., Leitner, P., Cito, J., and Gall, H. C. (2014). Cloud workbench - infrastructure-as-code based cloud benchmarking. *CoRR*, abs/1408.4565.

Shen, Z., Jia, Q., Sela, G.-E., Rainero, B., Song, W., van Renesse, R., and Weatherspoon, H. (2016). Follow the sun through the clouds: Application migration for geographically shifting workloads. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 141–154, New York, NY, USA. ACM.

Silas, S., Rajsingh, E. B., and Ezra, K. (2012). Efficient service selection middleware using electre methodology for cloud environments. *Information Technology Journal*, 11(7):868.

Silva-Lepe, I., Subramanian, R., Rouvellou, I., Mikalsen, T., Diament, J., and Iyengar, A. (2008). *SOAlive Service Catalog: A Simplified Approach to Describing, Discovering and Composing Situational Enterprise Services*, pages 422–437. Springer-Verlag Berlin Heidelberg.

u. Rehman, Z., Hussain, O. K., and Hussain, F. K. (2013). Multi-criteria iaas service selection based on qos history. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1129–1135.

ur Rehman, Z., Hussain, O. K., Chang, E., and Dillon, T. (2015). Decision-making framework for user-based inter-cloud service migration. *Electronic Commerce Research and Applications*, 14(6):523 – 531.

Williams, D., Jamjoom, H., and Weatherspoon, H. (2012). The xen-blanket: Virtualize once, run everywhere. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 113–126, New York, NY, USA. ACM.