Evaluator Services for Optimised Service Placement in Distributed Heterogeneous Cloud Infrastructures

Frederik Vandeputte, Luc Vermoesen
Alcatel-Lucent Bell NV, Belgium

David Griffin, T. Khoa Phan, Miguel Rio
University College London, UK

Pieter Simoens, Piet Smet
iMinds/University of Ghent, Belgium

Dariusz Bursztynowski
Orange, Poland

Folker Schamel, Michael Franke
Spinor, Germany

***Abstract*—Optimal placement of demanding real-time interactive applications in a distributed heterogeneous cloud very quickly results in a complex tradeoff between the application constraints and resource capabilities. This requires very detailed information of the various requirements and capabilities of the applications and available resources. In this paper, we present a mathematical model for the service optimization problem and study the concept of evaluator services as a flexible and efficient solution for this complex problem. An evaluator service is a service probe that is deployed in particular runtime environments to assess the feasibility and cost-effectiveness of deploying a specific application in such environment. We discuss how this concept can be incorporated in a general framework such as the FUSION architecture and discuss the key benefits and tradeoffs for doing evaluator-based optimal service placement in widely distributed heterogeneous cloud environments.**

Keywords—Service-oriented networking, orchestration, distributed heterogeneous cloud platform, placement algorithms

I. INTRODUCTION AND MOTIVATION

Over the past few years, cloud computing has quickly become a popular paradigm for automatically deploying and scaling various types of services such as Web services. However, these centralized homogeneous cloud computing infrastructures are not optimized for efficiently running geo-localized, personalized, bandwidth and/or processing-intensive real-time applications. For these types of applications (as well as others), the concept of edge computing and distributed heterogeneous clouds recently have gained a lot of interest [2][5][12]. These infrastructures allow a much better use of available network and computing resources, which can have a significant impact on overall cost-efficiency and QoE.

However, optimally deploying applications with such stringent requirements onto distributed resource-constrained heterogeneous cloud infrastructure is a complex problem. First, a service may have specific hardware and software resource or performance requirements to deliver consistent QoE towards all end users. Second, the resource capabilities as well as perceived performance/QoS may vary significantly in such heterogeneous environment. Third, different environments across different data centers may vary significantly in price, and service providers may want to decide on how much they are willing to pay for a particular QoS provided by some pool of (virtualized) resources.

Specifying all these requirements and trade-offs in static manifests, extracting detailed static and runtime knowledge of the available hardware resources, and finally combining all this information to be able to infer feasible and optimal deployment locations, would result in a **very complex system** (e.g., a rule engine) that needs to be able to *understand* all requirements, capabilities and their corresponding relationships. Moreover, it would still be **incomplete**, as new applications may have different requirements that cannot be captured, processed or understood by current available system. It would also require to explicitly identify and expose all application requirements or resource capabilities and constraints, which can prove to be difficult, complex, or may result in unacceptable overhead, or is practically impossible due to intellectual property concerns.

In this paper, we present a framework and mechanism for optimal service placement in widely distributed heterogeneous cloud infrastructures. Specifically, we further study the concept of an evaluator service introduced in [7] in more detail for efficiently and flexibly coping with application requirements and resource constraints in such complex environment. Basically, a evaluator service is an active service probe that is (deployed and) triggered prior to service deployment, and that evaluates a particular (virtual) runtime environment by generating a **score**, comprising all application-specific functional tests and trade-offs. These scores can subsequently be used as key input for building efficient optimal service placement algorithms. Secondly, we present a mathematical model for optimizing service placement in such dynamic distributed heterogeneous environment.

The main contributions of this paper are as follows. First, we present a framework and mechanism for optimal service placement in a distributed heterogeneous cloud. Second, we study evaluator services for efficiently and flexibly abstracting the feasibility and cost-efficiency of a particular (virtual) runtime environment for deploying a demanding real-time interactive service. Third, we discuss how evaluator services could be used for doing optimal service placement and discuss how to minimize the overhead of these active probes.

Section II first discusses related work, followed by an introduction of the FUSION architecture as a novel service oriented network architecture for managing services in a widely distributed cloud environment in Section III. Section IV then discusses the service placement optimization problem in more depth in such environment. We introduce the concept of evaluator services in Section V, and provide an initial analysis of the tradeoffs and benefits in Section VI. We conclude in Section VII.

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) in the FUSION (Future Service Oriented Networks) project under grant agreement n° 318205.

II. Related Work

There is a large body of papers on online quality prediction techniques in service oriented systems, e.g., [3][6][8][11] [13][14]. Based on the general taxonomy for related techniques [10], the FUSION evaluator service approach falls into the scope of run-time verification and online testing methods. In a recent paper [15], the authors propose a method for QoS prediction of candidate services for adaptation of dynamic composite services. Their approach builds on QoS prediction, using historical data obtained from different users to estimate the unknown QoS values, and thus does not requires additional service invocations. Matrix factorization is used to derive unknown values from relatively small set of observed QoS data. This approach does not cater for functional metrics of services and cannot be directly applied to evaluate data centers where candidate services have not been deployed so far.

In [9] the authors propose a desired functional scope of a monitoring system for assessing cloud infrastructure based on metrics required for service deployment. Different levels of evaluation criteria are considered, namely functional capabilities of the infrastructure (e.g., VM CPU/memory/disc, supported OSes, VPN models, VM measurements, available load balancers, etc.) as well as performance assessment capabilities of services (support for load tests, stress tests, capacity tests). For the latter, a notion of a gauge system is introduced to assess cloud service performance using dedicated agents and a measuring tool. However, no implementation of such a tool nor algorithms are proposed.

On the network side, different approaches to evaluate latency and packet loss ratio have been developed that use active monitoring agents deployed either in dedicated locations or directly on user devices. For example, Akamai uses site analyzer agents [1] that download Web objects and measure their failure rates and download times. Those agents are configured as a dedicated network that is independent of the content delivery network. They are deployed in major end-user networks worldwide to serve as landmarks and their measurements can be used to improve the quality of network performance map.

Conversely, the Radar approach of Cedexis [4] collects real-time user statistics of every major cloud & CDN provider using the end-device monitoring agent model. The measurements are crowd-sourced by client agents accessing websites that have a Radar tag embedded. Essentially, when a user visits a Radar-enabled website, a small JavaScript client agent is downloaded, it receives instructions from Cedexis specifying which platform to measure next, and initiates a set of specific measurements for this platform that are then uploaded to Cedexis. Such techniques can be adopted to create FUSION evaluator services that jointly assess data center and network level performance.

III. FUSION Architecture

Today, service providers only need to deploy their service in the cloud infrastructure of a single incumbent to be globally accessible. These cloud platforms provide supporting technologies for load distribution and automatic scaling in a single datacenter or redirection of users between datacenters in a handful of regions.

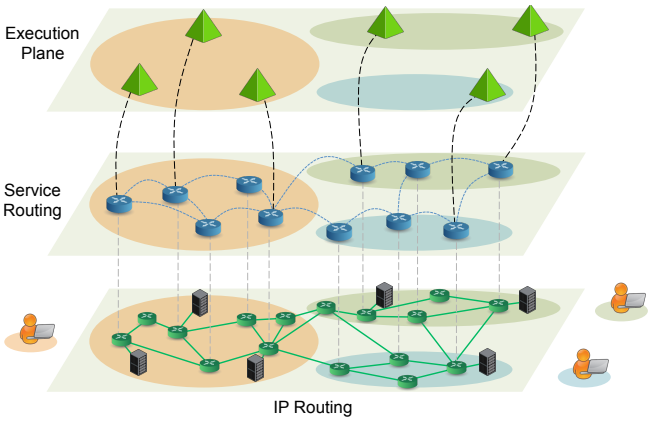

Figure 1. High-level FUSION Architecture

In contrast, demanding interactive applications must be deployed in a large set of execution nodes with a sufficient degree of distribution to ensure that users can access the service with the desired QoE. Preserving sufficient coverage for users in a wide area will likely involve execution nodes exploited by different entities (e.g. ISPs). In addition to the complexity of dealing with various interfaces offered by different parties, the design of logic for placement decision, service orchestration and instance selection forms another barrier for application providers to enter the market. To be efficient, detailed knowledge is needed about the capabilities of heterogeneous nodes, the IP network topology and expected user demand patterns. This knowledge is only available at ISPs and infrastructure providers; yet as of today a common framework is missing that integrates service deployment functionality with network-aware instance selection.

The FP7 FUSION project has incepted a 3-layer architecture as shown in Figure 1. The basic operation of our system is that orchestration domains, consisting of a potentially large number of geographically distributed execution zones (EZs), deploy services on behalf of service providers in one or more EZs according to the expected user demand. This is depicted in the upper layer of Figure 1. The middle layer provides service resolution capabilities for finding the *best* available instance. Once a specific service instance in a specific EZ (managed by a zone manager) has been selected for the user request, data plane communications take place in the data forwarding plane depicted by “IP Routing” in the lower layer of Figure 1. We refer the reader to [7] for a more detailed description of the architecture.

IV. Service Placement Optimization Problem

The decision where to place a service can be based on various criteria, such as taking into account hardware requirements, its proximity to external sources or the clients, the network latency between clients and service instances, but also costs for running the service on a specific EZ and zone policies. Service providers have to balance these objectives when deciding where to place their services. In this section, we first define a utility function, which uses network performance metrics between a user and an EZ to measure user satisfaction, and describe service placement as a multi-objective optimization problem in which we first guarantee max-min fairness between users and then maximize the total utility of all users. We also consider a trade-off between the service deployment cost and the performance (total utility) of users.

A. Problem description

As input to the placement problem, we consider the estimated user requests, network performance model (e.g. latency between users and EZs), deployment cost of service instances in EZs and resource constraints (e.g. number of session slots that each EZ can support). The output then is a service placement solution that maximizes performance (total utility) while achieving max-min fairness between users. The objective also considers the trade-off between the performance and the service deployment cost.

B. Mathematical model

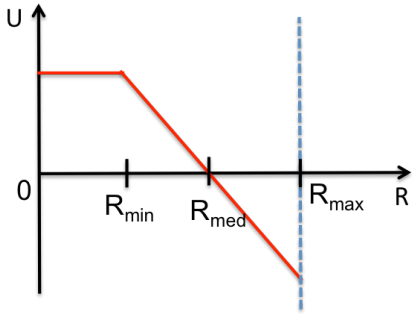

Figure 2. Utility function vs. latency

As shown in Figure 2, depending on the service type, different values of R_{min}, R_{med} and R_{max} are defined. The utility function should express the following meanings:

- For some services (e.g. voice), further reducing the latency will not improve QoE; therefore the utility remains constant if R <=R_{min}
- If R_{min} < R <= R_{med}: the utility value is positive, meaning that the QoE is good. However, the user satisfaction is reducing when the latency is increasing.
- If R_{med} < R <= R_{max}: the utility value is negative but the QoE is still in an acceptable range.
- If R_{max} < R: the service request is blocked.

We model the problem as a linear programming formulation. The key idea is that we include the utility function into the objective of the formulation. Moreover, we add constraints on the available session slots and the total budget for deploying service instances at EZs. In summary, the algorithm works in two steps:

- Step 1: we first maximize the minimum user utility. In this step, we guarantee that the solution achieves max-min fairness between all the users.
- Step 2: we then maximize the total utility. Moreover, we add new constraints to ensure that the minimum utility of all users equals the max-min fairness value. Therefore, the output of the algorithm achieves max-min fairness between users and also maximizes the total utility of all the users.

There is always a trade-off between the utility and the deployment cost. In general, given a solution, we can plot its cost and utility on a 2-D plane as Figure 3.

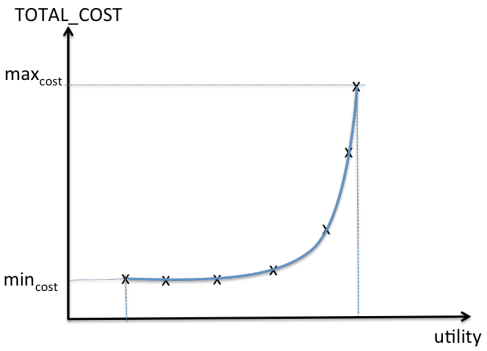

Figure 3. Trade-off between total utility and cost

Given min_{cost} and max_{cost} as a constraint, we can find the corresponding utility values. Depending on the granularity of the graph and how much time we can pay for computation, we can choose a number of points in between [min_{cost}, max_{cost}]. Finally, we get a trade-off relationship of the cost and the utility as in Figure 3. Based on this figure, the service provider can easily choose a solution with their desired trade-off.

In the above optimization formulation we have used network latency to determine user utility. We can extend this to include the capabilities and performance of heterogeneous EZs. These capabilities can be efficiently measured by evaluation functions, as discussed in the following section.

V. Evaluator Services

As a service provider will be charged for running the service, he should be able to decide what he is willing to pay for. Rather than trying to capture all detailed service requirements and placement policy trade-offs in a complex static manifest (thereby also requiring the orchestration layer to be able to interpret and cope with these complex manifests), the core idea is to offload this complexity into active probes that evaluate the feasibility and/or cost-effectiveness of running a particular service in a particular execution environment and return **a score** to grade the evaluation. These probes may be generic, resource specific or service specific.

For example, services may require specific hardware or a certain proximity to other service instances or have different runtime requirements towards the execution environment. For instance, a real-time rendering service may depend on a GPU, or certain GPU capabilities, and a corresponding streaming service nearby. These requirements may include specific OpenGL extensions, a specific minimum OpenCL version or supporting specific OpenCL extensions, or vendor specific hardware and APIs such as NVIDIA CUDA support. Effectively, most 3D games have particular minimal GPU requirements. Additionally, a service provider may offer specific quality modes for a particular game, which translate into more specific hardware requirements. For example, certain realistic real-time lighting and shadowing techniques depend on the availability of a geometry shader.

Describing these kinds of restrictions in static manifest files would result in requiring the orchestrator to understand all possible use cases of all applications. The descriptions would also need to be updated whenever new hardware or hardware revisions become available. Particular runtime environments would also need to be characterized to capture and expose their runtime behavior.

Therefore, a static approach using detailed manifests quickly becomes difficult to manage for large numbers of services and therefore unscalable (though it could be sufficient for an architecture designed for a specific and limited subset of services with a priori known requirements). Instead, we propose using evaluator services that are deployed on the actual environments as active probes. These evaluator services can be provided by the application service provider (or possibly an evaluator service provider), allowing application and service provider specific checks and cost-utility trade-offs to be made by the service provider. They are deployed within various execution environments of several EZs (based on the service provider policies) and are automatically triggered by the domain orchestrator placement function as part of service placement for determining the optimal location(s) for deploying new instances of a particular (set of) service(s).

A. Design considerations

The three main design considerations are simplicity, flexibility and efficiency. **Simplicity** means that it should be simple to leverage evaluation scores optimizing service placement. As such, we envision these scores to be as basic as a float or integer, abstracting the complex tradeoff between static requirements, runtime behavior, QoS and cost, rather than using complex scores containing multiple values representing different aspects (cost, efficiency, etc.).

The second design decision is **flexibility**. Service providers should be in the loop when deciding where their services should be deployed, a feature that is naturally supported by evaluator services. On the other hand, a domain orchestrator or zone manager also may want to enforce some policies or reserve some (compute or networking) resources for more profitable services. As such, we allow for a dynamic pricing model where the price of execution environments can change based on changing policies or changing runtime behavior, allowing domains and zones to steer the decisions of the evaluator services by dynamically changing the price. The cost of running a service in a particular environment is one of the key input parameters of an evaluator service, allowing the service provider to return a proper score with respect to the cost. Note that this score can also change over time due to changing policies of the service provider. As such, we envision a score to be typically only valid for a limited amount of time.

The third design decision is **efficiency**. Three key factors are the overall response time and the deployment and runtime overhead. As such, evaluator services should be optimized for quickly returning a score upon an evaluation request. For some evaluators, this could involve doing part of the evaluation as a background process. Also, as evaluator services may need to be deployed just-in-time in remote data centers, the provisioning and deployment time should be minimal. Hence, a good candidate for quickly provisioning and deploying new instances in particular locations are lightweight containers such as Docker. The issue of runtime overhead is discussed further in Section V.

B. Modeling & implementation

In general, an evaluator service is a function that, given a set of input parameters (including the environment, historical data, policies, etc.), returns a value that can be considered as a score or rank, indicating how suitable that environment is for deploying a number of session slots of a particular service:

$$score = evaluator(Service, InstParams, Env, Cost(Env,t))$$

Service represents the service as well as its requirements, *InstParams* the instantiation and configuration parameters for deploying that service (e.g., UHD quality, premium QoS, etc.), *Env* represents the execution environment, and *Cost(Env, t)* represents the cost of that environment in a given time frame.

A minimal property is that the resulting scores should be (partially) **ordered** for a particular service type: a (slightly) preferred environment should have a (slightly) higher score, allowing to simply order all tested environments based on their score.

In the context of global multi-service placement algorithms, we are currently also investigating the benefits of associating additional properties to these scores, such as proportionality (i.e., an environment that is twice as good results in a score that is twice as high), or allow for a more specific interpretation of the scores (e.g. as a bidding value in an auctioning placement algorithm or an average execution time or runtime latency).

It is important to stress that these evaluators could be implemented in any framework or environment and only need to implement an API; in FUSION, we developed a simple REST API for triggering these evaluators. We recommend that these evaluators are packaged for example in lightweight containers to minimize deployment and runtime overhead.

C. Service-centric evaluator-based service placement strategy

Evaluator services can be integrated into service placement algorithms in various ways. One possible high-level strategy goes as follows. First, a domain orchestrator preselects a number of EZs in which to run an evaluator for a particular service, for example based on a priori knowledge. Next, the orchestrator triggers all selected zones in parallel for doing an evaluation (possibly with a deadline). Each zone subsequently selects a set of (virtual) execution environments onto which to make the evaluation. This may involve first deploying the corresponding evaluator service in such environment (if time permits). Each zone manager then triggers all selected evaluators to generate a score for the particular deployment request, after which all received evaluations are collected and returned to the domain orchestrator. The latter can then use these scores are part of its service placement strategy.

In case of a simple service-centric placement strategy, this may entail selecting the zone(s) and/or virtual environment(s) with the highest relative scores. In such model, it is the responsibility of the domain orchestrator and zone manager for fairly pricing each environment (which can change with e.g. time, popularity and/or internal policies).

The service placement optimization algorithm as described in section III can be also extended to include evaluation scores as follows. First, evaluation scores for services that have the semantic of execution time/latency can be added to the network latency thereby optimizing placement on a combination of network performance, evaluation score and costs. Second, if evaluation scores have a more complex semantic (e.g. they imply a quality level of video resolution, accuracy of computation, etc.) then a multi-dimensional utility function can be used.

One way of achieving this is for a matrix of network latency values to be supplied to allow the evaluator service to

combine network performance with other evaluation metrics in a service-specific manner and return a mapping between latency and evaluation score. The utility function as shown in Figure 2 can then be replaced with one describing the relationship between evaluation score and utility.

It should be noted that evaluation and service placement optimization are off-line orchestration actions taken at the epochs of service deployment and periodic service redeployment, e.g. when user demand patterns change significantly and new EZs are required to house additional service instances closer to the sources of increased demand. As such, computation time is not as significant an issue compared to a scheme where evaluation and instantiation is undertaken for each service request.

VI. Analysis and Discussion

In this section, we discuss the benefits of evaluator services for service placement, analyze the trade-off between efficiency and overhead and high-light possible further optimizations.

A. Benefits for service placement

Evaluators will identify the feasible EZs in terms of their capabilities and efficiency, while service placement will select between them for optimizing the utility function as discussed in section III. Prior service evaluation will ensure that the selected EZs for placing service instances have the required software and hardware capabilities, which will significantly reduce the total number of EZs under consideration for the placement optimization algorithm, improving scalability and performance. Secondly, no detailed information of the application specifications or available EZs resources need to be disclosed with the central orchestrator to be able to do optimal service placement.

B. Trade-off between efficiency and overhead

A key challenge of evaluator services is to minimize the amount of overhead they could introduce, especially in case a large number of these probes need to be deployed in a distributed and heterogeneous environment. Consequently, there is an interesting trade-off in the amount of active evaluators compared to the number of active application instances and the relative QoS, QoE or cost benefits which deploying a particular amount of evaluators may induce.

As an example, the runtime overhead for a particular service deployment can be approximated as follows:

$$Overhead = D*(Z*V)*E/I$$

In this formula, I represents the number of active application service instances, Z represents the total number of EZs, V the average number of environment types per zone, D the fraction of all environments onto which an evaluator service is deployed, and E represents the fraction of time an evaluator service on average is running in a particular environment. Let us assume $Z=100$ EZs, and $V=10$ environment types per zone, and $D=1/10$, meaning only 10% of all environments are preselected for evaluation on average. In case $I=100$ (i.e., on average, 100 service instances are active), then E must remain below 1/10 to keep the overall runtime overhead below 1%. This means that each evaluator can only be active for about 2 hours per day, which should be largely sufficient in case there are only a few evaluations per day. In case $I=1000$, the coverage D could be increased to e.g. 100% when keeping the runtime overhead fixed at 1%.

Especially for services with very stringent requirements, it may be beneficial to deploy evaluators more aggressively to quickly find the most cost-effective environment(s) for hosting that service. For others, or for more homogeneous environments, deploying evaluators more sparsely may suffice.

C. Further optimizations

To further optimize the efficiency of evaluator services, we are currently exploring the benefits of optionally splitting the evaluation process into two sub functions, namely a *probing* function and a *cost-benefit analysis* function. As it can easily become very costly to deploy evaluators in a large number of resource-constrained or expensive environments, it may be beneficial if the probing function (i.e., where the actual environment is evaluated w.r.t. static features and runtime behavior) is only run very sporadically for a short period of time on those distributed heterogeneous execution environments. The second part of the evaluation process, namely the cost-benefit analysis, could then be done on a central cloud environment, where it can run in a cheaper execution environment and closer to the global placement function.

D. Planned evaluation

A complete and realistic evaluation of the placement strategies in a distributed heterogeneous cloud environment is hard to achieve through simulation. Instead, we are aiming for a prototype deployment of the FUSION prototype on the iLab.t testbed infrastructure [16], which allows for flexible testing in various configurations. This testbed comprises nodes of various hardware generations and various support of hardware accelerators such as GPUs. To further increase the heterogeneity, we will leverage on the existing federation of this testbed with the CloudLab in Utah [17], which allows us to include ARM-based servers in the set-up.

Based on these testbeds we plan to conduct experiments based on industry-typical scenarios in the area of digital media (e.g. VOD) and gaming. For example, we will validate the concept of evaluator services in a thin-client multi-player gaming scenario consisting of a game server running in Linux being connected to multiple rendering clients requiring a Windows operating system and GPU hardware supporting a specific Direct 3D shader model.

VII. Conclusion

In this paper, we proposed a framework and mechanism for doing optimal placement of demanding services in widely distributed heterogeneous cloud infrastructures. We introduced the concept of evaluator services for assessing the feasibility and cost-efficiency of deploying services in particular distributed execution environments, without having to rely on complex static manifests or resource descriptions. We discussed how these active probes could be efficiently leveraged for doing optimal service placement, trading off efficiency and overhead for different types of services.

In future work, we will evaluate a number of evaluator-based global service placement strategies and investigate mechanisms for further reducing the overhead of these probes.

REFERENCES

[1] Akamai. Akamai Site Analyzer - Service Description. July 2009.: http://www.atoll.gr/media/brosures/Akamai_Site_Analyzer_Service_Description.pdf

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proc. of the 1st workshop on Mobile cloud computing. ACM, 2012.

[3] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A framework for QoS driven runtime adaptation of service-oriented systems," IEEE Trans. Software Eng., vol. 38, no. 5, pp. 1138-1159, 2012.

[4] Cedexis. http://www.cedexis.com

[5] A. Chandra, J. Weissman, and B. Heintz, "Decentralized edge clouds," Internet Computing, IEEE, vol. 17, no. 5, 2013.

[6] J. Ejarque, A. Micsik, R. Sirvent, P. Pallinger, L. Kovacs, and R. Badia, "Semantic resource allocation with historical data based predictions," in CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization, 2010, pp. 104–109.

[7] D. Griffin et al., "Service oriented networking", Proc. of EUCNC, Bologna, 2014

[8] D. Ivanovic, M. Carro, and M. V. Hermenegildo, "Constraint based runtime prediction of SLA violations in service orchestrations," in Proc. 9th Intl Conf. on Service-Oriented Computing (ICSOC 2011) vol. 7084. Springer, 2011, pp. 62–76.

[9] Hangoo Jeon, Young-Gi Min, and Kwang-Kyu Seo, "A Framework of Performance Measurement Monitoring of Cloud Service Infrastructure System for Service Activation, " International Journal of Software Engineering and Its Applications, Vol.8, No.5 (2014), pp.127-138.

[10] A. Metzger, C.-H. Chi, Y. Engel, and A. Marconi, "Research challenges on online service quality prediction for proactive adaptation," in Proc. of the 2012 Workshop on European Software Services and Systems Research-Results and Challenges (S-Cube), 2012, pp. 51-57.

[11] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, prediction and prevention of SLA violations in composite services," in IEEE International Conference on Web Services (ICWS) Industry and Applications Track, 2010.

[12] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," Pervasive Computing, IEEE, vol. 8, no. 4, pp. 14–23, 2009.

[13] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based online testing for proactive adaptation of service-based applications (short)," in COMPSAC 2011 – The Computed World: Software Beyond the Digital Society. IEEE Computer Society, 2011.

[14] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," IEEE Trans. Software Eng., vol. 30, no. 5, pp. 311-327, 2004.

[15] Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu., "Towards Online, Accurate, and Scalable QoS Prediction for Runtime Service Adaptation, " IEEE 34th International Conference on Distributed Computing Systems (ICDCS), 2014

[16] iLab.t, ilabt.iminds.be

[17] https://www.cloudlab.us/