Challenges for orchestration and instance selection of composite services in distributed edge clouds

Pieter Simoens, Lander Van Herzeele
Ghent University - iMinds
G. Crommenlaan 8 bus 201, 9050 Gent, Belgium
psimoens@intec.ugent.be

Frederik Vandeputte, Luc Vermoesen
Alcatel-Lucent Bell Labs
Copernicuslaan 50, 2018 Antwerpen, Belgium
frederik.vandeputte@alcatel-lucent.com

***Abstract*—Today's centralized cloud-computing infrastructures have not been designed with geo-localized, personalized, bandwidth/processing-intensive, real-time applications in mind. High network delay and low throughput can have a significant impact on the user experience. Instead, such services could be deployed in distributed service nodes at the edge of the network, closer to the user. In this paper we focus on composite services of which the components are running in different service nodes. We present a two-layer framework that provides service orchestration and instance selection. We present the orchestration mechanisms to enable the flexible re-use of components across different composite services. For the resolution layer of our framework, we present two modes of operation that combine network and service availability information for efficient per-request instance selection among a multitude of service replicas.**

I. Introduction

Despite the advantages of elasticity and cost efficiency brought by the cloud paradigm, the requirements of many innovative services do not match well with a deployment in today's cloud infrastructure. The high, variable network delays [1] and low throughput [2], [3] to a relatively small number of data centers deep in the network can have a serious impact on the QoE experienced by many users of resource-demanding multimedia interactive services like personalised real-time video and games. Real-time analysis of the big data produced by the billions of sensors and devices in the Internet-of-Things is an enabler for a new generation of services that provide geolocalized and/or personalized results. The continuous transfer of this sensor data stream to centralized clouds would simply congest the network. A much better approach is to bring the computational application logic closer to the user and to the input data.

Essentially, these services with low-latency, high-throughput requirements must be deployed on a collection of distributed, smaller clouds at the network edge. This concept has been proposed in the literature under a variety of names: edge computing [4], fog computing [5] or cloudlets [6]. Already today, we see a number of commercial efforts and trends that introduce service execution capabilities in ISP networks. Relatively small data centers in the ISP network previously only used for ISP-specific services like IPTV and network management are already being exposed as virtualized resources, e.g. to CDNs [7]. In the context of Network Function Virtualization, network management functions are being aggregated on general purpose hardware deployed at central offices and customer premises [8]. Even traditional network elements are being equipped with computing resources and storage capacity. Services can now run at LTE eNB base stations [9] or on IP routers [10]. Compared to over-the-top players, operators can leverage the unique position of this distributed cloud infrastructure in the network for providing better QoS to demanding applications.

We thus arrive in a situation where replicas of the same service are deployed in many nodes - in this paper referred to as execution zones - spread over the network. In this paper, we focus on the challenges for orchestration and instance selection for composite services in such an environment. Each component of this service is subject to particular deployment, orchestration and management rules and is potentially provided by different entities . An example composite service is an interactive multimedia dashboard, combining different media like photos, private videos, video-on-demand mash-ups and video chats into a personalized video stream. This application involves multiple decoders, encoders, streamers and application logic. Other examples can be found in the domain of the Internet-of-Things where data from geographically distributed sensors must be combined with application logic.

Our goal is to realize a flexible ecosystem in which individual service components can be part of multiple composite services. This introduces challenge for the orchestration (deployment, scaling) of the individual components. Moreover, when a client wants to connect to a composite service, we need to find a suitable instance of each components. It is unlikely that all components of a composite service are available in a single execution zone, e.g. because the limited edge node capacity is fully occupied by already connected users or because necessary hardware (GPU, input sensor) for some components is not present on all nodes.

In the context of the FP7 FUSION project [11], we investigate how service-request resolution capabilities can be natively embedded in the network to address these challenges. In this paper, we present the necessary key concepts to realize flexible re-use and selection of service components for composite services. We introduce the concept of session slots as service availability metric and present early results on efficient selection between multiple instances that are deployed in geographically distributed execution zone. The remainder of this paper is structured as follows. In section II, we discuss the overall architecture of our two-layered service-centric overlay framework. In section III, we elaborate on the orchestration layer of the framework. In section IV, we discuss the instance selection provided by the resolution layer. Early

978-3-901882-76-0 @2015 IFIP

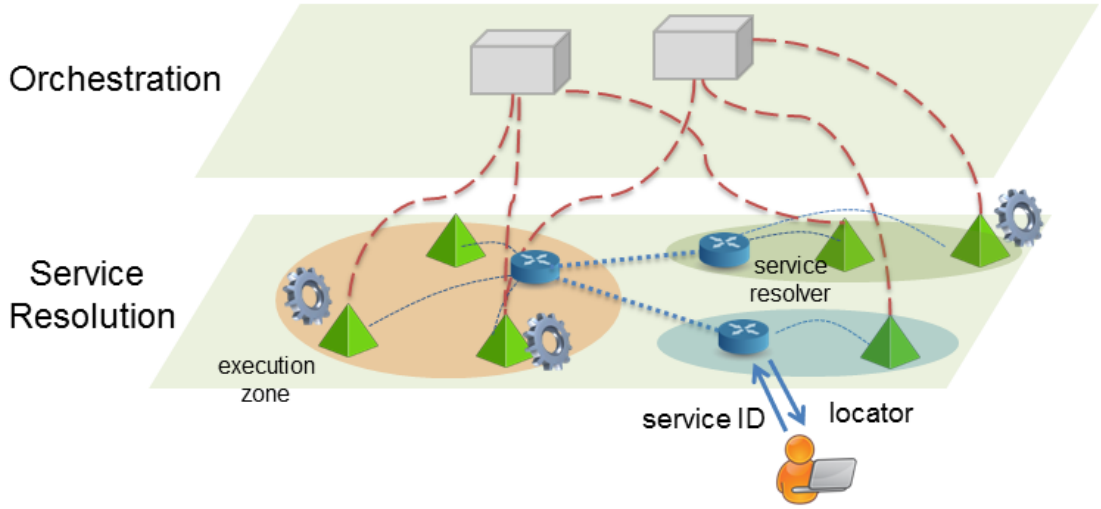


Fig. 1. Schematic overview of the FUSION architecture. The distributed execution zones of the edge cloud are grouped logically in service resolution domains. Execution zones report to the service resolver of their domain on service availability. Users send a location-agnostic service identifier to their local resolver, who returns the locator (IP/port) of the best currently available service replica.

experimental results indicate the benefits of taking into account the overall composite service structure in the design of service selection algorithms. Related work in this domain is presented in section V.

II. Architecture

The main functional blocks of the FUSION architecture are illustrated in Figure 1. Application developers register their service with an *orchestrator*, which takes care of the deployment, provisioning, placement and scaling of service instances in a number of *execution zones* that are geographically distributed in the edge cloud. An execution zone is the logical representation of physical computational resources in a specific location where services are deployed in virtual machines or containers. The main primitive of the *service resolution layer* is to resolve service requests to the locator of the "best" instance among many replicas of the same service component running in execution zones throughout the Internet. This fine-grained resolution capability is performed on the grounds of network metrics, service performance metrics and service instance load.

We refer the reader to [11] for a more in-depth discussion on the motivation and structure of the FUSION architecture. In the remainder of this section, we briefly describe two architectural aspects relevant to the scope of the current paper.

A. Name-based service request resolution

The service resolution layer is conceived as an overlay of the IP layer. In an anycast-style process, clients send a service ID to their closest service resolver. Following the principles of name-based networking, these service IDs are agnostic to the network location of the corresponding services. The resolution layer selects one or more replicas of the specified service, taking into account resolution metrics, service availability and network metric constraints (e.g. maximum latency). The client is returned with a list of one or more locators (IP address and port) of the selected instances. Clients subsequently connect to a service component instance, beyond the control of the service resolution layer.

Execution zones are grouped in multiple administrative domains. Each resolution domain is governed by one (logically) centralized *service resolver* that applies an internal resolution policy. Service state information is exchanged between the different domains, possibly subject to particular advertisement policies. One particular business model is where resolution domains coincide with today's Autonomous Systems. In such a scenario, each Internet Service Provider is operating one logical service resolver that peers with service resolvers of other ISPs. Service resolvers will first try to resolve requests to an instance running in one of their local execution zones. If no suitable instance is available, and if allowed by the service QoS requirements and deployment policies, the request can be resolved to an instance running in another domain.

Service resolvers need to combine network metrics with service availability information to select the best replica on a per-request basis. Each resolution domain builds its own database with information on end-to-end network metrics, e.g. using the Archipelago measurement infrastructure [12]. Service availability information is expressed by means of the concept of resource session slots, as discussed below.

B. Advertising service availability through session slots

Service instances typically can handle a number of requests in parallel with sufficient performance, depending on the allocated resources and the service characteristics. Since we are targeting services like monitoring and multimedia where active connections have durations in the order of minutes or even hours, using a queuing model would lead to unacceptably long waiting times. Moreover, the smaller infrastructure of execution zones in the edge cloud, compared to clouds, drastically reduce scaling opportunities to provide additional capacity. For these reasons, service resolvers should be provided with up-to-date information on service availability to ensure that service

requests are optimally resolved. Service availability is however a complex metric that is influenced by many factors.

First, service *availability* can only be assessed by instances at runtime because the capacity of a single instance in terms of concurrent requests that can be served is not a direct function of the resources that were allocated to it. Underlying hardware differences, multi-tenancy resource contention and varying data center network topology are only a few causes of performance variation of the same instance [13]. Therefore, each instance deployed in the FUSION architecture must report at regular intervals on its service availability to the gateway component of each execution zone.

Second, it should be noted that service availability cannot directly be derived from service *utilization*. Continuous utilization metrics like average CPU percentage provide information on the service load generated by connected clients, but provide no indication on how many additional connections can be served. We therefore introduce the concept of *session slots*, as an abstract, non-service specific, discrete metric indicating the number of new sessions that an instance can handle without loss of service performance. Each running instance reports at regular intervals its number of available slots to the zone gateway. The gateway in turn reports to its resolver the sum of the slots reported by running instances and the additional maximal capacity that can be deployed if needed by booting additional instances (e.g. up to a specified maximum cost). An additional advantage is that zone-internal scaling policies remain hidden to other entities.

By combining network metrics with service availability information (session slots); service resolvers are able to select the public service endpoint of the best matching execution zone for a given client request.

III. Composite Service Orchestration

A composite service is an abstract representation for a collection of components that each provide part of the functionality of the overall service. The communication links between these components for implementing the composite service can be modeled in a connected graph. Typically, the connection with the client is handled by one of the nodes in the graph, which we call the *entry service*. Examples of composite services S and T are shown in the left hand side of Figure 2. In this section, we will discuss the orchestration of such services.

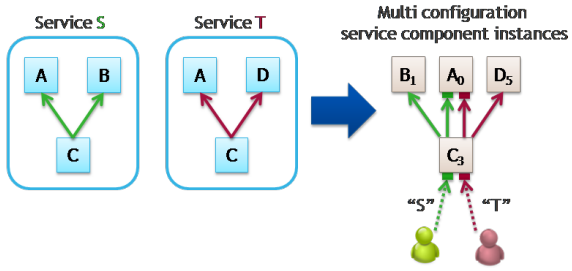


Fig. 2. Component C is part of the service graph of composite services S and T. At runtime, instance C3 serves users of both composite services.

A. Service registration using TOSCA

Service providers can register their application with the FUSION framework by means of a manifest that specifies the structure of the composite service, design time parameters and deployment and scaling policies. In FUSION, we build on the TOSCA specification language that enables inter-operable description of application and infrastructure cloud services, the relationships between service components and operational behavior like deploy operations [14].

The structure of the composite service is described using an XML-file in the *service-template* directory of the Cloud Service Archive file. An example is provided in Listing 1.

The composite service is registered with serviceID `CompService` at a FUSION orchestrator. The manifest states that this service comprises two components: `ServiceA` and `ServiceB`. The relationship between these components is defined in RelationshipTemplates. Various types of Relationships are defined, however in the scope of this paper we focus on the `connectsTo` type, which denotes that the source element will initiate the connection to the target element. One possible means for the source element to get the locator of an instance is to send a request to the FUSION service resolution plane for service ID `ServiceB`.

Listing 1. FUSION manifest for a composite service with two components

```
<ServiceTemplate id="CompService",
                 name="Composite Service" >
...
 <TopologyTemplate>
   <NodeTemplate id="ServiceA" name="Service A" type="FUSION
        :component">
   </NodeTemplate>
   <NodeTemplate id="ServiceB" name="Service B"
                              type="FUSION:Component">
   </NodeTemplate>

   <RelationshipTemplate id="AtoBConnection"
                         type="my:connectsTo">
        <SourceElement ref="Service A" />
    <TargetElement ref="Service B" />
   </RelationshipTemplate>
 </TopologyTemplate>

</ServiceTemplate>
```

The orchestrator will parse the manifest and validate that enough resource session slots are available by the already deployed instances, e.g. because they were registered previously in another manifest - either as standalone service or as part of another composite service.

B. Multi-configuration of service components

A key feature of our flexible service ecosystem is that service components can belong to multiple composite services, each with their own set of instantiation and configuration parameters. In the example of Figure 2, the composite services S and T have service components A and C in common. Instead of deploying instances of A and C that are uniquely assigned to a single composite service, the FUSION orchestration supports the sharing of instances between different composite services.

Such shared instances will be provided with multiple configurations by the orchestration layer, one per composite service. Each service configuration is mapped on a different port. The instance knows to which composite service an incoming connection belongs by the port on which it is received. The multi-configuration of instances remains hidden to the resolution layer. In the example of Figure 2, the same IP but

different ports of an instance of C are advertised as endpoint for composite services S and T.

Although each composite service is subject to proprietary scaling rules, the budget of session slots of an instance must be shared between all composite service configurations. To comply with the different scaling policies, one suboptimal approach is to fix the allocation of session slots between the different configurations. Imagine that instance C0 in Figure 2 can provide 4 resource session slots. A straightforward approach is to fix the allocation of resource session slots between configurations, e.g. 2 each for service S and service T. When a third user requests an instance of composite service S, the scaling rules force the orchestration to deploy an additional instance of component C, although instance C0 can still handle the third connection.

To allow for more advanced orchestration and more efficient resource usage, we introduce the concept of virtual elastic scaling. We mandate that each instance not only reports at regular intervals its number of actively running connections per configuration, but also the maximum number of connections it can handle, depending on the amount of infrastructure resources allocated to it. The latter number must be reported only once, when the instance is first booted. The orchestrator can now virtually overscribe this capacity by advertising to the resolution layer more session slots for each composite service than the running instance(s) can handle. Note that this oversubscription can be done for any instance (not only entry services). Instead of deploying or terminating instances, the zone orchestration logic can adjust the virtual scaling over the different configurations. This should lead to a more stable environment, since there will be fewer instance deployment and termination operations.

IV. Service Request Resolution

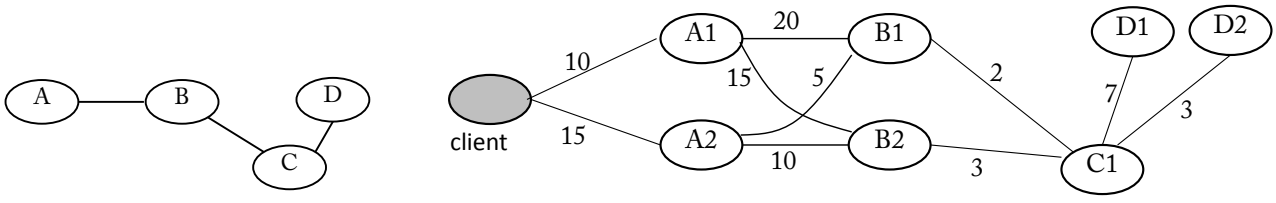
Clients send a request with the service ID of the composite service to their local resolver. This resolver is continuously collecting information on the availability of service instances, that is either received by reports from the zone gateways in its resolution domain, or passed by resolvers of other domains. Depending on the amount of information on the composite service structure that is available in the resolution layer, more intelligent instance selection algorithms can be applied. This comes at the cost of increased complexity in the resolution layer. We will detail this trade-off by means of Figure 3.

The left figure shows the *deployment* graph that is constructed from the TOSCA service manifest. The right figure shows the *instance* graph. The nodes in this graph are the instance of each component that has available session slots at the time of the request. The weight edges reflect the cost metric (e.g. latency, bandwidth) of the data path.

A. Resolution models

In the most basic resolution model, the structure of the composite service is transparent to the service resolution layer. In this scenario, the client is returned with the IP and port of an instance of the *entry* service component. This role is assumed by component A in Figure 3(a). When the client connects to this instance, this will result in a cascade of requests and subsequent data connections. In the example of Figure 3(a), when the instance of A receives a connection from a user, it will in turn request the service resolution layer for an instance of component B. When A is connected to an instance of B, the latter instance will similarly establish a connection to the next component. In this scenario, which we refer to as *nexthop* in the remainder of this section, the structure of the composite service remains hidden to the service resolver, and it will always return the instance with the lowest data path cost from the requesting entity. Referring to Figure 3(b), the service router would first return the IP and port of instance A1 to the client. In turn, this instance would be provided with the locator of instance B1, etc. This would result in a total instantiation cost of 31 (10 + 15 + 3 + 3). However, the lowest subgraph consists of (A2, B1, C1 and D2) and has an instantiation cost of 25 (15 + 5 + 2 + 3). This example shows that this heuristic may lead to suboptimal instance selection, especially for composite services with complex deployment graphs.

In this paper, we propose an alternative approach where the structure of the composite service is registered at the service resolver by the orchestration layer. In this model, which we refer to as *overall* in the remainder of this section, the service resolver is able to determine the subgraph of the instance graph, that contains exactly one instance of each atomic service and that has the lowest total edge cost. Note that in general, shortest path algorithms cannot be applied since there is not necessarily a single source and/or single sink node in the composite graph structure. Moreover, the order and size of the instance graphs grow rapidly with the number of instances of each atomic service. If *N* instances of atomic service A are deployed, and *M* instances of atomic service B, then the graph will contain $N \times M$ edges since in principle each instance of B can be reached from any instance of A. This model might provide better selection results, but comes at the cost of additional complexity in the resolution layer.



(a) Deployment graph

(b) Instance graph at request time

Fig. 3. The illustrated composite service graph consists of 4 components. Per request, the optimal combination of available instances must be selected.

B. Experiment set-up

To evaluate the benefits of embedding the structure of composite service graphs in the resolution layer, we have built a simulator using the discrete event simulation library MASON [15]. Instead of using artificially generated network topologies, we decided to perform simulations on the CAIDA IP router-level topology that is continuously collected and updated by a set of globally distributed probes [16]. Since we assume that service routing domains are approximately aligned to Internet AS, we reduced the dataset to the routers that were geo-located in France, the largest country in the European Union. IP routers with the same geographical coordinates in the dataset were aggregated, and only links between the reduced set of nodes were maintained. The resulting topology comprised 77 nodes and 98 edges. The link cost was set to the geographical distance between the resolvers. The edge cost in the instance graph is the total path cost of the shortest path between these instances. It has been shown that the geographical distance is an acceptable approximation of network latency [17]. Note however that any cost metric can be used.

Service requests are generated from 16 randomly selected client nodes and served by an instance running on any of the other 61 server nodes. From each client node and for each service, requests are generated following a Poisson distribution (mean λ) with exponentially distributed request inter-arrival times. Each client node thus contributes equally to the demand, which is in line with the fact that we are using a network topology of a medium-sized country where service popularity or diurnal patterns are likely to be homogeneous across the country. We generated 10 composite service structures by generating graphs with 2 components, randomly selected from a set of 10. Each instance is provided with 10 session slots and placed on a randomly selected server node. In the simulation, each connection lasts for a fixed period $T = 40$. Some early results are presented in Figure 4. The cost metric in this figure refers to the sum of the data path link costs between the instances that were selected as reply to a particular service requests.

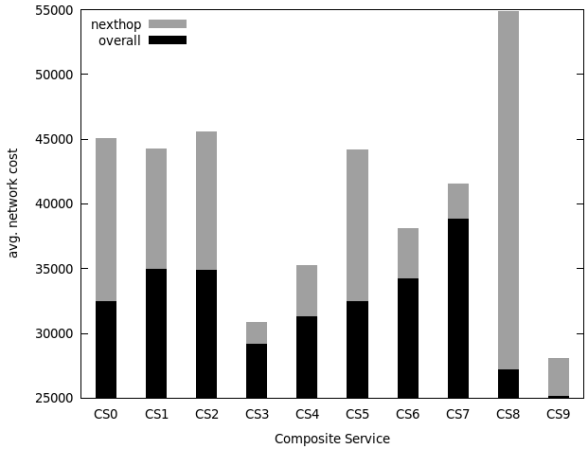


Fig. 4. Average cost of all instantiations for requests from one client node, grouped per composite service, and generated following a Poisson distribution with mean $\lambda = 0.1$.

Figure 4 compares the average cost of composite service instantiations for both resolution models, for requests generated from one client node in our topology and for each of the 10 composite services. The average cost of the *nexthop* resolution is between 6 % and 201 % higher than *overall* resolution. We expect even higher gains for larger composite services.

V. Related work

Several distributed service management architectures such as IRMOS [18] or NGSON [19] have been proposed in recent years. In contrast to FUSION, IRMOS is based on pre-allocation in managed networks for providing strict QoS guarantees. We augment the NGSON paradigm by providing a powerful service orchestration layer that is capable of allocating and load balancing service instances through dynamic cooperation with a distributed execution environment.

In the domain of instance selection algorithms, Malik et al. [20] present an algorithm that divides execution nodes in mutually exclusive groups, based on incomplete inter-node latency information. In [21], the authors use a locality-sensitive hashing scheme that relies on a node coordinate system for network-aware instance selection. The feasibility of these algorithms for the FUSION framework and how they can be extended to include session slot information is subject to future work.

VI. Conclusion

This paper introduces the architectural aspects of the FUSION framework related to the orchestration and service selection for composite services. These new concepts can enable the flexible deployment of demanding applications across distributed execution environments in distributed edge clouds. Current and future work include the further design of instance selection algorithms, both inside a single execution zone, as across execution zones. This involves extensive simulation with more heterogeneous demand patterns, as well as building an integrated prototype.

Acknowledgment

Part of this research was funded by the 7th Framework Programme of the European Commission through the FUSION project under grant agreement no. 318205.

References

[1] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proceedings of the 11th annual workshop on network and systems support for games*, 2012.

[2] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *INFOCOM*. IEEE, 2013.

[3] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proc. of the 11th annual intl. conf. on Mobile systems, applications, and services*. ACM, 2013.

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

[5] 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.

[6] 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.

[7] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing cdn-isp collaboration to the limit," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 34–44, 2013.

[8] "White paper: Why distribution matters in nfv," NFV Insights Series, Alcatel-Lucent and Telefonica, Tech. Rep.

[9] "Nokia siemens networks intelligent base stations," Nokia Siemens Networks, Tech. Rep., November 2013.

[10] Cisco fog computing with iox. [Online]. Available: http://www.cisco.com/web/solutions/trends/iot/cisco-fog-computing-with-iox.pdf

[11] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, and F. Schamel, "Service oriented networking," in *Networks and Communications (EuCNC), 2014 European Conference on*. IEEE, 2014, pp. 1–5.

[12] Center for applied internet data analysis. archipelago measurement infrastructure. [Online]. Available: http://caida.org/projects/ark

[13] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift, "More for your money: exploiting performance heterogeneity in public clouds," in *Proc. of the Third ACM Symposium on Cloud Computing*. ACM, 2012.

[14] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: Portable automated deployment and management of cloud applications," in *Advanced Web Services*. Springer, 2014.

[15] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.

[16] The caida ucsd ipv4 routed /24 topology dataset -2012-07. [Online]. Available: http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml.

[17] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive p2p systems," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, 2009.

[18] A. Menychtas, D. Kyriazis, S. V. Gogouvitis *et al.*, "A cloud platform for real-time interactive applications." in *CLOSER*, 2011, pp. 397–403.

[19] S.-I. Lee and S.-G. Kang, "Ngson: features, state of the art, and realization," *Communications Magazine, IEEE*, vol. 50, no. 1, 2012.

[20] S. Malik, F. Huet, and D. Caromel, "Latency based group discovery algorithm for network aware cloud scheduling," *Future Generation Computer Systems*, vol. 31, 2014.

[21] A. Klein, F. Ishikawa, and S. Honiden, "Towards network-aware service composition in the cloud," in *Proc. of the 21st international conference on World Wide Web*. ACM, 2012.