Hierarchical Service Placement for Demanding Applications

Elisa Maini, Truong Khoa Phan, David Griffin, Miguel Rio
University College London, UK
Email: {e.maini, t.phan, d.griffin, miguel.rio}@ucl.ac.uk

***Abstract*—The increasing scale of cloud environments requires more scalable orchestration systems for determining which physical resources are responsible for processing service requests. A centralised service placement lacks scalability while a fully decentralised approach only has a limited view of the system. For these reasons, this paper investigates a hierarchical approach for service placement in a distributed environment, which increases scalability while maintains high service placement quality. First, we design a polynomial optimisation algorithm to place services in cloud data centers based on our novel utility function. Then, we describe a hierarchical model with the need to only know a small subset of the data required by the global optimisation formulation. Simulations show that our approach is scalable and performs well, close to the centralised model.**

I. Introduction and Motivation

The rapid growth of broadband communications has led to many new applications such as on-line interactive maps, social networks, video streaming, cloud computing and CDN (Content Distribution Network) services. Those services are provided by application providers which instantiate their replicas in clouds consisting of hundreds of servers distributed over the Internet. Application providers can instantiate directly their service replicas or delegate service management functions to a service orchestrator (SO), whose key role is to undertake service placement taking into account both computational and networking factors. To optimise service deployment, the SO balances application requirements (for instance the service latency) and costs as well as ensures service availability (in terms of quality-of-service) for all users.

However, the increasing scale of cloud infrastructures complicates the service orchestration task, leading to scalability issues of the orchestration system itself. A single centralised orchestration approach has the advantage of ultimate level of visibility and control, as the placement algorithm has full visibility of individual users and servers. However, this solution is prone to scalability issues with millions of users using services across hundreds or thousands of servers. A centralised service orchestration is impractical in real deployments, as a single global orchestrator would be required to collect information from all servers and networks and also to model predicted demand from all users.

On the other hand, with a completely decentralised approach, scalability can be handled more easily. However, this model implies that service providers need to register, deploy and manage their services with multiple providers in multiple locations, making the configuration and management of widely deployed services more complex.

Hierarchical approaches try to combine the advantage of both centralised and decentralised solutions. Specifically, hierarchical approaches scale better compared with the centralised solutions, maintaining a good overview of the global system. In addition, at lower levels, the algorithms use only local information, increasing the scalability. For these reasons, in this paper we present a hierarchical solution for service placement.

In brief, the contributions of our work are as follows:

- Using the *utility function* introduced in our previous work [15], we design a polynomial centralised optimisation algorithm that allows service providers to deploy their services to the best locations. We consider both max-min fairness policy and maximizing QoS in our model.
- To address scalability issues, we propose a hierarchical model that allows service providers to run a local version of the placement algorithm without the need for global knowledge of all service replicas and network conditions.

This paper is organised as follows: in Section II we present related work. Section III introduces the terms and concepts that we will use in the following sections. Section IV presents the optimisation formulation and is followed by hierarchical model in Section V. We present simulation results in Section VI and draw conclusions of the work in Section VII.

II. Related work

Content replication has been a topic of extensive research. There exists a large number of works related to service placement problem and most of the solutions can be divided into centralised ([17], [10], [9], [3], [4]) and fully distributed approaches ([1], [18], [11], [12]). In [17] a solution based on a centralised controller has been proposed to dynamically allocate instances according to changes in application demands. Then, the algorithm has been expanded in [3]. However, these solutions only work well for small environments, but do not scale well for large data centers. [10] considers an optimization problem which models the dynamic placement of applications. It allows different types of resources to be managed and aims at maximizing satisfied application demand while minimizing placement changes compared to the previous placement. Such algorithm has been modified in [9] in order to produce application placement that allows application load to be better balanced across server machines. In this work, we use the novelty of utility function [15] for the placement model. In addition, we consider both max-min fairness and maximizing utility as the multi-objective in the optimization formulation.

In [18] a gossip-protocol is used to provide a fully centralised approach. The basic idea is that a node is an individual entity which manages itself. In addition, it exchanges information with others and shifts load between them. [1] introduces an approach that involves selective update propagation to achieve a processing load on a node that is independent of the system size. The basic idea is that a node propagates further only those updates that cause a modification to its forwarding table. Other decentralised approaches are proposed in [11], [12]; however, all those solutions there is no higher-level overview of the network and are not able to achieve good placement quality, while still maintaining good scalability. In addition, hierarchical techniques are used in [8], [13]. In this work, we use a hierarchical approach with our novel utility framework to improve scalability for the placement system.

III. Definition of terms

In this section we give brief definitions for the terms that we will introduce in the next sections.

- *A*n application (service) provider (ASP) represents an organisation that wish to deploy an application (service) in data-centers/clouds over the Internet. When deploying services, they need to consider the trade-off between deployment cost and QoS for their users.
- Services will be deployed in datacenters/clouds. We use the term *execution zone (EZ)* as a logical representation of physical computational resources in which the services can be deployed. An ASP can rent a portion of resources which are then under its control for deploying services.
- A *session slot* is a unit of measurement representing how many user sessions can be accommodated simultaneously by a given service instance. If the number of users is more than an EZ's capacity, we will deny (block) some users to guarantee good QoS for the others.

In this work, instead of using raw latency, we use our utility framework to evaluate QoS [15]. In general, for some services, if latency is below a specific value, the improvement is not perceived by the users of that service. For example, for interactive voice services, humans can not perceive any improvement in quality if the latency is reduced below 20 ms [16]. We visualize this in Figure 1 in which the utility function is defined by three parameters of latency: T_{min}, T_{fair} and T_{max} to rate QoS in term of *excellent*, *good*, *fair*, *poor* and *no service (or blocked)*. Note that the utility is not restricted to only latency. In future work, we will extend the utility to be a combination of any QoS metrics such as latency, bandwidth, loss, etc. More details on the utility characterization are discussed in [15].

IV. Service placement optimisation

In this section, we present a mathematical formulation for the service placement. Given the key notations in Table I, we use linear programming to formulate the service placement problem. In particular, we guarantee max-min fairness between users and also maximise the total utility of all users. In addition, we consider a trade-off between the service deployment cost and the performance (utility) of users. The problem is described as follows:

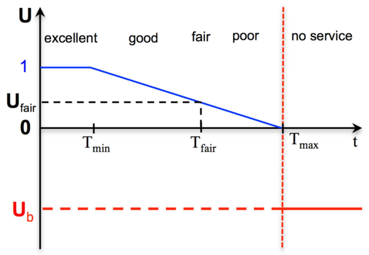


Figure 1. Utility function vs. latency

Table I
Key Notation (in Alphabetic Order)

c_z	unit deployment cost at EZ z
d_i	requested session slot of user i $\forall i \in \mathcal{I}$
$\mathcal{I}$	set of user groups $\mathcal{I} = \{i\}$
l_{iz}	latency between user i and EZ z
S_z	capacity at EZ z
t_i	average perceived latency of user i
$\mathcal{Z}$	set of execution zones (EZ) $\mathcal{Z} = \{z\}$
U_b	utility value of a blocked user
u_i	utility of users i
x_{iz}	fraction of user i connects to EZ z
y_i	variable used to compute the utility

- Input: estimated user requests ($\mathcal{D}$); two threshold values (T^i_{min} and T^i_{max}) to define the utility function for each user group i; latency between user group i and EZ z is l_{iz}; unit service deployment cost at EZ z (c_z); the maximum budget ($COST$) and capacity at each EZ (S_z).
- Objective: maximize the performance (total utility) of users while achieving max-min fairness between users. The objective also considers the trade-off between the performance and the service deployment cost.
- Output: $x_{iz} \in [0,1]$: fraction of user group i connects to EZ z to get the service. It is noted that, instead of individual user, we consider i as a group of users. For example, a user i represents for all individual users of a city. For that reason, we can use x_{iz} as a real variable to indicate the percentage of users in the city i to connect to which EZ.

In this work, we consider a multi-objective optimization model which achieves max-min fairness for all users as well as maximizing the total utility. To do this, the algorithm works in two steps as follows:

- Step A: maximizing the minimum user utility to achieve max-min fairness between all users.
- Step B: given the max-min fairness in step A as a constraint, the objective in this step is to maximize the total utility over all users.

A. Linear Program - Max-min Fairness

$$U_{fairness-min} = max\ (U) \tag{1}$$

s.t.

$$\sum_{z \in \mathcal{Z}} x_{iz} = 1 \qquad \forall i \in \mathcal{I} \tag{2}$$

$$\sum_{i \in \mathcal{I}} d_i x_{iz} \leq S_z \qquad z \in \mathcal{Z} \tag{3}$$

$$t_i = \sum_{z \in \mathcal{Z}} l_{iz} x_{iz} \leq T^i_{max} \qquad \forall i \in \mathcal{I} \tag{4}$$

$$y_i \geq 0 \qquad \forall i \in \mathcal{I} \tag{5}$$

$$y_i \geq t_i - T^i_{min} \qquad \forall i \in \mathcal{I} \tag{6}$$

$$u_i = \frac{T^i_{max} - T^i_{min} - y_i}{T^i_{max} - T^i_{min}} \qquad \forall i \in \mathcal{I} \tag{7}$$

$$U \leq u_i \qquad \forall i \in \mathcal{I} \tag{8}$$

$$\sum_{z \in \mathcal{Z}} \sum_{i \in \mathcal{I}} c_z d_i x_{iz} \leq COST \tag{9}$$

$$x_{iz} \in [0, 1], u_i \leq 1 \qquad i \in \mathcal{I}, z \in \mathcal{Z} \tag{10}$$

Explanation:

- The objective function (1) is to maximize the minimum utility U to achieve max-min fairness policy.
- Constraint (2): all the requests of user group i have to be served.
- Constraint (3) limits physical capacity at an EZ when deploying service instances.
- Equation (4) is used to compute the average latency t_i for the user group i to get the service. This latency should be less than the maximum value T^i_{max}.
- Constraint (5) - (6) ensure that $y_i \geq 0$ if $t_i \leq T^i_{min}$, otherwise $y_i \geq t_i - T^i_{min}$.
- Equation (7) is used to model the utility function. If $t_i < T^i_{min}$, y_i is forced to be 0 as in the objective function we try to maximize the utility U. If $t_i \geq T^i_{min}$, the formulation will choose $u_i = \frac{T^i_{max} - t_i}{T^i_{max} - T^i_{min}} < 1$. More detail of the utility formulation can be found in [15].
- Constraint (9) set a limit of the total budget.

B. Linear Program - Maximizing Utility

$$\max[\sum_{(i) \in \mathcal{D}} u_i] \tag{11}$$

s.t.

(2) - (10)

$$U \geq U_{max-min} \tag{12}$$

Explanation: In this step, we add the constraint (12), where U is the minimum utility of users (constraint (8)) and $U_{max-min}$ is the objective value from step A, to ensure that the solution still achieve the max-min fairness policy. We keep the same constraints (2) - (10) and change the objective function as (11) which will find a solution that maximizes the total utility.

The optimization formulations in both step A and B are pure linear programming models (there is no integer or binary variables), therefore it can be solved in polynomial time. The number of variables x_{iz} in the LP problem is $|\mathcal{I}| \times |\mathcal{Z}|$ where $|\mathcal{I}|$ is the number of user groups and $|\mathcal{Z}|$ is the number of EZs. Since $|\mathcal{Z}|$ is much smaller than $|\mathcal{I}|$, the worst case complexity of the LP problem will be $O(|\mathcal{I}|^{3.5})$ [19].

V. Hierarchical service placement

In the centralised model, the service orchestrator has a detailed view of all EZs and all forecasted user demands for a particular service and it optimises the placement of service instances in the EZs to maximise total utility within cost constraints set by the application provider. It may be impractical, for scalability reasons, for a globally centralised placement algorithm to maintain detailed knowledge of all users and all EZs and so here we investigate a hierarchical solution where the overall orchestration domain is split into geographical sub-domains.

In this model the high-level orchestrator has limited visibility of EZs and user demands within a sub-domain - it sees only the aggregate of user demands and the aggregate of EZ capacities within a particular sub-domain. The high-level orchestrator places service instances at the coarse granularity of sub-domain only and subsequently each sub-domain orchestrator undertakes a further placement algorithm with the scope of that sub-domain only to determine in which specific EZs what quantity of service instances should be placed to supply the required number of session slots to meet the specific detailed demand pattern of user requests within that sub-domain.

There are many ways of sub-dividing an overall orchestration domain into sub-domains. One option is to map sub-domains onto the same geographical area covered by resolution domains: the entity responsible for resolving user requests to EZs with available session slots. Equating sub-domains for orchestration and service placement purposes with resolution domains is not essential as other coarser or finer grained sub-domains could be considered. However, in the rest of this section we assume that the lower-level orchestration domains have been mapped onto resolution domains and the term resolution domain is used to mean the lower-level sub-domain for orchestration and service placement. Note that is also possible to consider multiple hierarchical levels of service orchestration and placement; however, we only model two levels in the analysis described in this section.

In summary, as each lower-level domain is treated as a black box with respect to the high-level orchestrator the overall service placement problem can be divided into smaller units – one at the high level working at coarse granularity and several (one per sub-domain) operating at a lower level with more detailed information but with limited geographical scope. In this way the optimisation algorithms can be executed with reduced quantities of information, increasing scalability.

Table II
KEY NOTATION (IN ALPHABETIC ORDER)

$\mathcal{D}_k$	set of user requests of M_k
d_i^k	requested session slot of user i for M_k
K	number of most populated cities in each region
M_k	k^{th} resolution domain
m_k	the centroid of the resolution domain M_k
N	number of resolution domains
S_{z_k}	available session slots at EZ z_k
S_z	available session slots for the resolution domain M_k
$\mathcal{Z}_k$	set of EZs $\mathcal{Z}_k = \{z_k\}$ of the resolution domain M_k

A. Dataset description

Here we present the dataset used in our model in order to analyse and evaluate the performance of our framework. We use a dataset with 2048 data centers distributed in 525 cities all over the world [6]. Since data centers in a city are geographically close to each others, we group them as one execution zone. The capacity of one EZ is proportionally to the number of data centers in that city. Then, we split the whole world into eleven geographical regions and each EZ belongs to one specific region according to its geographical location. The regions are based on the continents, but with larger continents split into two or three regions to make the population of each region roughly the same. In the end, the resulting regions are: Western Europe (EUW), Eastern Europe (EUE), Central Asia (ASC), Southern Asia (ASS), Pacific Asia (ASP), Africa (AFR), Northern North America (NAN), Southern North America (NAS), Eastern South America (SAE), Western South America (SAW) and Oceania (OCE). In fact, using smaller regions can speed-up the optimization algorithm but we will lose details on the dataset and would end-up with more local optimal solutions.

B. Problem description and methodology

In our modelling approach, we further sub-divide each region into N low-level orchestration sub-domains, which are termed resolution domains (RDs) in the remainder of this section, but it should be born in mind that although the low-level orchestration sub-domains have been mapped to resolution domains here this is not the only granularity of orchestration sub-domain that can be considered. For our simulation models we select the K most populated cities in each region which become the centroids m_k of the resolution domains M_k in that region. Users and EZs are mapped to their geographically closest centroid, and are said to belong to that resolution domain. Each resolution domain (low-level orchestration sub-domain) consists of a number of EZs and users at specific locations. But, from the perspective of the high-level orchestrator the capacities of the EZs are aggregated into a single logical EZ (termed high-level EZ) located at the centroid of the resolution domain (Eq. 14) and the individual users are modelled as a single group of users (termed high-level user), also located at the centroid (Eq. 13).

$$d_i^k = \sum_{i \in M_k} d_i \qquad \forall i \in \mathcal{D}_k \subseteq \mathcal{D} \quad (13)$$

$$S_z = \sum_{z_k \in \mathcal{Z}_k} S_{z_k} \qquad \forall z_k \in \mathcal{Z}_k \subseteq \mathcal{Z} \quad (14)$$

As an example, we split the Western Europe (EUW) in three resolution domains (RD1, RD2, and RD3). The individual EZs and users are grouped into their high-level counterparts, which are located at the resolution domains centroid.

The high-level orchestration algorithm only sees the set of high-level EZs and high-level users, located at the centroids of the lower level sub-domains. It runs the centralised optimization which is solved to find high-level EZs should deploy what quantity of service instances to meet the predicted demand of the high-level users. Fig. 2 shows an example of the placement decision made by the high-level orchestrator. In each resolution domain some service demand is allocated to the local high-level EZ (local-to-local requests), i.e. the high-level orchestrator has matched local demand to local EZs, some are allocated to a remote EZ (local-to-remote requests) or come from a remote user (remote-to-local requests). When there are not enough resources in terms of EZ capacity to match user requests to EZs within the maximum utility for the service or if the cost of the solution would exceed the cost constraints, user requests are blocked.

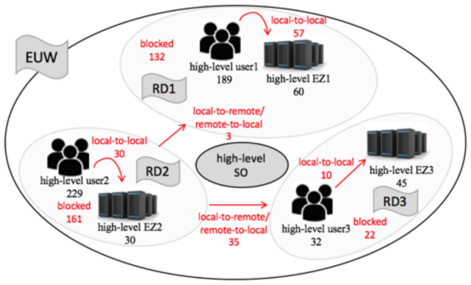


Figure 2. Solution of the centralised optimisation algorithm

C. Post-processing phase

Using the hierarchical approach, the optimisation problem has been split into a number of sub-problems, each of which can be undertaken with a smaller quantity of information, reducing the time to find a feasible solution. Since the high-level orchestrator does not know detailed information for each sub-domain in terms of the individual EZs and individual users belonging to each domain it is unable to determine the precise quantity of session slots to be allocated to EZs and which of these will be allocated to which individual users. Hence the first task of the low-level optimisation algorithms is to map the output of the high-level placement into a more detailed input for the low-level placement optimisation function.

A post-processing phase is locally used in each low-level orchestrator to attribute the total local-to-local, local-to-remote and blocked demand to individual low-level users within that sub-domain and how the total remote-to-local requests are allocated to individual low-level EZs in that sub-domain. In particular, local-to-local requests and blocked requests are simply allocated proportionally to the initial quantity of individual user demand. For instance, consider RD2 where the total blocked requests are 161; after allocating these proportionally 144 requests from $user_4$ are blocked and 17 from $user_5$. Similarly, the total local-to-remote requests are 38 (the sum of those to $RD1$ and $RD3$), these are split between the low-level users such that 35 are from $user_4$ and 3 from $user_5$. After having attributed the blocked requests and the local-to-remote requests to each low-level user, the remaining service requests are 34 for $user_4$ and 4 for $user_5$, which are the total local-to-local requests for RD2. Local-to-remote request for the $RD2$ are remote-to-local requests for other resolution domains; in particular, 3 remote-to-local requests for RD1 and 35 for $RD3$. Such requests must be allocated to the EZs belonging to $RD1$ and $RD3$, respectively. We choose to allocate proportionally, for instance for $RD3$ we consider a remote user with 35 requests from $RD2$ and for $RD1$ a remote user with 3 requests from $RD2$. Local-to-local service requests are then allocated using the placement optimization algorithm running in each low-level sub-domain (Fig. 3).

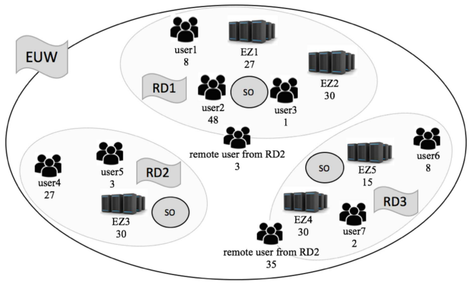


Figure 3. Requests and available session slots after post-processing phase

Now that the aggregate local-to-remote and remote-to-local demands which had previously been determined by the high-level placement algorithm have been allocated to individual EZs and users by each low-level optimisation algorithm, this is used as input to the full placement optimisation algorithm running in each sub-domain.

VI. Simulation results

We solve the linear program model using the IBM CPLEX solver [5]. All computations were carried out on a computer equipped with a 3 GHz CPU and 8 GB RAM. For the data set presented in section V-A, it takes less than 1 minute to find an optimal solution of the LP formulations.

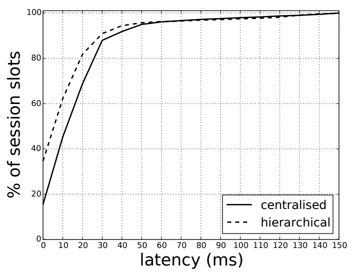
The data deployment cost is based on the Amazon EC2 charging model. The user demand is modeled as a Poisson process and is proportional to the population of each city [7]. The latency between users and execution zones are collected based on Haversine distance, the shortest distance between two points around the planet's surface [2]. For the utility function, we use the three latency thresholds which work for voice services [15]: $T_{min} = 20$ ms, $T_{fair} = 100$ ms, $T_{max} = 150$ ms. We consider to minimize the number of blocked user requests by setting small negative value of $U_b = -100$. We show in Fig. 4 a comparison between the CDF latency of the hierarchical and the centralised algorithms with different mismatch levels between supply and demand ("X% rand."). In particular, we first run the centralised model but without the capacity to find a *perfect placement* solution assume that we do not have any resources' constraints. Next, we create different levels of mismatch between supply and demand by varying a parameter "X% rand.". This is for each EZ, we remove the $X\%$ of its session slots from the *perfect placement* configuration. Then, we mix the removed session slots of all EZs and scatter them uniformly in all EZs. This guarantee that the total session slots of EZs in all cases (*perfect placement* and "X% rand.") are the same. "0% rand." is equivalent to the *perfect placement* while in "100% rand." there is a uniform distribution of session slots between all EZs. The reason we vary "X% rand." is to test how the algorithms adapt with different configurations. $X = 0\%$ (Fig. 4 (a)) corresponds to the "easy" case for the placement algorithms as it aligns with the *perfect placement* solution. On the other hand, $X = 100\%$ (Fig. 4 (c)) is the "hard" configuration for finding a good placement solution. As shown in Fig. 4, we see that the gap between the hierarchical and the centralised algorithms increases when we increase $X\%$. However, in general, the hierarchical algorithm performs well, close to the centralised one.

In Fig. 5 we show evaluating results for the hierarchical algorithm in terms of latency and utility with different value of "X% rand.". As mentioned before, increasing "X% rand." would result in worse QoS in term of latency and utility which can be seen clear in Fig. 5. However, as our model try to maximize the total utility, the gap between the two cases $X = 0\%$ and $X = 40\%$ is small meaning that we still can find good solution even with 40% of mismatch configuration. In case, $X = 100\%$, we do not have enough resource at EZs but the algorithm is successful to minimize the number of blocked user requests.

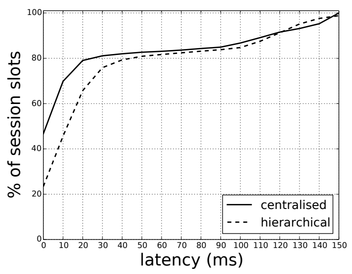
We show in Fig. 6 the benefit of max-min fairness policy over all users. We test with different cost budget in the model. We first consider a minimum cost value that can find a feasible placement solution (constraint (9)), then increase this budget by the "budget multiplication faction" shown in the x-axis of Fig. 6. For each test, we check the latency of the worst user - the one that has the maximum latency over all users. As shown in Fig. 6, with the max-min fairness constraint, the worst user still can get a good QoS (latency is less than 25 ms) while without max-min fairness, some users suffer from high latency (but it is always less than T_{max}). This confirms that with the max-min fairness, all users will have a better fair share of the resources at EZs.

VII. Conclusion

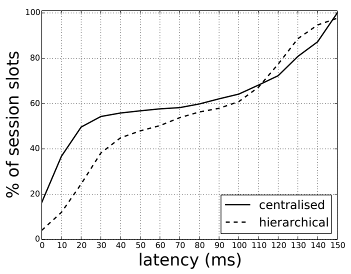
This paper present a hierarchical service placement approach for cloud environments. Our method to implement



(a) X = 0%

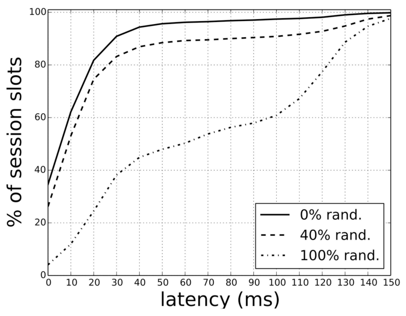


(b) X = 40%

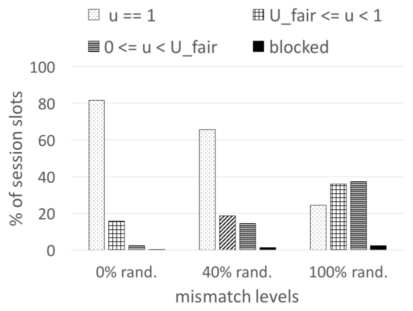


(c) X = 100%

Figure 4. CDF latency for the centralised and the hierarchical placements



(a) Latency



(b) Utility

Figure 5. Latency and utility results for hierarchical placement under different mismatch levels

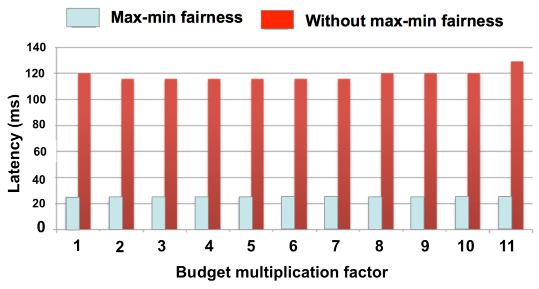


Figure 6. Latency of the worst user with and without max-min fairness

service placement allows for trading-off user QoS with deployment costs. Compared with the existing approach, our formulation is simpler while maintains good utility for users. Compared with the centralised approach, the hierarchical method performs well, close to the centralised one.

Acknowledgment

This research has received founding from the 7th Framework Programme (FP7/2007-2013) of the European Union, through the FUSION (grant agreement 318205) projects.

References

[1] C. Adam and R. Stadler, *Service Middleware for Self-Managing Large-Scale Systems*, in IEEE Transactions on Network and Service Management, 2007.
[2] G.V. Brummelen, *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton Uni. Press, 2013.
[3] D. Carrera, M. Steinder, I. Whalley, J. Torres and E. Ayguade, *Utility-based Placement of Dynamic Web Applications with Fairness Goals*, in Network Operations and Management Symposium (NOMS), 2008.
[4] S. Clayman, E. Maini, A. Galis, A. Manzalini, N. Mazzocca, *The dynamic placement of virtual network functions*, in NOMS, 2014.
[5] www-01.ibm.com/software/commerce/optimization/cplex-optimizer
[6] http://www.datacentermap.com/
[7] http://github.com/richardclegg/multiuservideostream
[8] B. Hudzia, M.T. Kechadi and A. Ottewill, *TreeP: A Tree Based P2P Network Architecture*, in CLUSTER, 2005.
[9] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko and A. Tantawi, *Dynamic Placement for Clustered Web Applications*, in World Wide Web (WWW), 2006.
[10] T. Kimbrel, M. Steinder, M. Sviridenko and A. Tantawi, *Dynamic Application Placement under Service and Memory Constraints*, in Workshop on Experimental and Efficient Algorithms (WEA), 2005.
[11] Y. Li, F.H. Chen, X. Sun, M.H. Zhou, W.P. Jiao, D.G. Cao and H. Mei, *Self-Adaptive Resource Management for Large-Scale Shared Clusters*, in Journal of Computer Science and Technology, 2010.
[12] C. Low, *Decentralised Application Placement*, in Future Generation Computer Systems, 2005.
[13] H. Moens, J. Famaey, S. Latre , B. Dhoedt and F. De Turck, *Hierarchical Network-Aware Placement of Service Oriented Applications in Clouds*, in NOMS, 2014.
[14] J. Nielsen, *Usability Engineering: Response Times: The Three Important Limits,* 1993.
[15] T.K. Phan, D. Griffin, E. Maini, M. Rio, *Utility-maximizing Server Selection*, in IFIP Networking, 2016.
[16] M.A. Stone, B.C. Moore, *Tolerable Hearing Aid Delays. Est. of Limits Imposed by the Auditory Path Alone using Simulated Hearing Losses*, in Ear and Hearing, 1999.
[17] C. Tang, M. Steinder, M. Spreitzer and G. Pacifici, *A Scalable Application Placement Controller for Enterprise Data Centers*, in World Wide Web (WWW), 2007.
[18] F. Wuhib, R. Stadler and M. Spreitzer, *Gossip-based Resource Management for Cloud Environments*, in CNSM, 2010.
[19] Z. Zhang, Y. Hu, M.Zhang, R.Mahajan, A. Greeberg, B. Christian, *Optimizing Cost and Performance Online Service Provider Networks,* in NSDI, 2010.