Utilitarian Placement of Composite Services

Truong Khoa Phan, Miguel Rocha, David Griffin, and Miguel Rio

***Abstract*—The emergence of distributed clouds opens up new research challenges for service deployment. Composite services consist of multiple components, potentially located in different geographical locations, which need to be interconnected and invoked in the correct order according to the overall service work-flow. The placement of composite services over distributed cloud node locations raises new challenges for efficient deployment and management. In this paper, we design exact models of the composite service placement problems using mixed integer linear program, and compare these to solutions based on genetic algorithms. We use a utility function, based initially on latency metrics, to evaluate the quality of service (QoS) of the deployed composite service. By maximizing the utility with respect to deployment cost, our approach can provide good QoS for users while satisfying budget constraints for service providers. Based on simulations using real data-center locations and traffic demand patterns, we show that our algorithms are scalable under a range of scenarios.**

***Index Terms*—Utility function, network function virtualization, composite service, service placement, optimization.**

I. Introduction

A WIDE variety of emerging Internet services such as virtual & augmented reality, network gaming and instrumentation for the Internet of Things require tight constraints on quality of service (QoS) that can only be achieved by moving computation closer to the users. In this context, application providers will make use of any of the thousands of third-party, publicly available, cloud data centers to instantiate parts of their services. Therefore, placement policies will play a key role in delivering good QoS for users. There are many drivers for service placement, including server resilience, network diversity, and proximity of servers to users. In this paper, we focus on deploying services close to users to improve QoS metrics, such as latency and/or throughput. We thus arrive in a situation where replicas of the same service are deployed in many data-centers, spread over the Internet. Service quality has two major sets of component metrics, relating to processing at servers and networking latency. The service placement system needs to take into account both computation and networking factors to optimize its performance.

Choosing a concrete service placement solution that is optimal with regard to QoS and cost constraints is an NP-hard problem [19]. In this work, we focus on the composite service deployment problem, which is even more complex. A composite service is formed from several components, potentially located in different data-centers. When in use, we must determine an efficient way to connect those components as an adequate work-flow. QoS of a composition is the aggregated QoS of the individual services according to the work-flow patterns. With the emergence of distributed cloud paradigms and virtualization technologies, the decomposition of complex services into components provides options for increasing flexibility and reducing costs compared to single atomic service deployment models. For instance, some components can be deployed once and used to serve multiple service sessions in order to reduce overall deployment costs.

One way to achieve good QoS is to reduce network latency by deploying services close to users [7], [19], [20], [26], [27]. In this paper, based on our previous work [23], [26], [27], we translate latency (both network latency and processing time at servers) to a utility score, which has been proven to be a better way to evaluate QoS [26], [27]. For composite services, the end-to-end latency is the aggregated latency of the path connecting each component together. In addition, we also consider one-hop utility constraints, as some services require a short delay for a special hop (e.g., from the users to the rendering component). Those latencies are then converted into a utility score. Our algorithms try to maximize the utility, given the deployment cost constraints.

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

- We consider both network latency and processing time at data-centers in evaluating QoS. We then convert those latencies into utility scores and propose algorithms to maximize the utility.
- We consider three basic composite service structures and propose Mixed Integer Linear Program (MILP) formulations to find the exact solutions for each structure. In addition, we propose genetic algorithms as meta-heuristics to address these optimization problems.
- We evaluate our algorithms on a real dataset of data-centers and users distributed around the globe. Simulation results show that our approaches are scalable and can provide close-to-optimal solutions.

The rest of this paper is structured as follows. In Section II, we introduce the three main composite service structures with example use cases. To evaluate QoS, we review the idea of the utility function in Section III. We then propose MILP models in Section IV to find the exact optimal solution for each

Manuscript received April 24, 2017; revised September 28, 2017; accepted January 19, 2018. Date of publication January 25, 2018; date of current version June 8, 2018. This work has been supported in part by the FP7 FUSION (grant agreement 318205), in part by the U.S. Army Research Laboratory and the U.K. Ministry of Defence (agreement number W911NF-16-3-0001), in part by the H2020 5G-MEDIA (grant agreement 761699) and in part by the CHIST-ERA CONCERT (grant agreement I1402) projects. The associate editor coordinating the review of this paper and approving it for publication was E. Elmroth. *(Corresponding author: Truong Khoa Phan.)*

T. K. Phan, D. Griffin, and M. Rio are with the Department of Electronic and Electrical Engineering, University College London, London WC1E 6BT, U.K. (e-mail: t.phan@ucl.ac.uk; d.griffin@ucl.ac.uk; miguel.rio@ucl.ac.uk).

M. Rocha is with the Centre of Biological Engineering, University of Minho, 4710-057 Braga, Portugal (e-mail: mrocha@di.uminho.pt).

Digital Object Identifier 10.1109/TNSM.2018.2798413

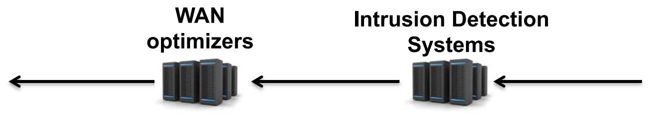


Fig. 1. Network service chaining in NFV.

composite service structure. In Sections V and VI, we design genetic algorithms to quickly find close-to-optimal solutions. We show evaluation results of those algorithms in Section VII. We present a literature review of related work in Section VIII and conclude the work in Section IX.

II. Composite Service Structures

The emergence of new networking technologies like Network Function Virtualization (NFV) and Software Defined Networking (SDN), opens up a new venue for flexibly deploying new network services. This approach consists in delivering network functions as software that can be deployed at required locations in the network, without the need to install specific equipments for new services. However, we do not limit ourselves to network functions. Applications, such as video processing and gaming, can also be deployed as a set of interconnected service components. Application providers will buy computation resources in a open market from a myriad of cloud providers allowing for thousands (even millions in the future) of potential placement servers, without any fixed infrastructure cost. A composite service requires connecting several components running in different locations (data-centers). We present in this section the three basic structures of a composite service.

A. Chain Structure

One of the most common structures of NFV composite services is the chain structure. In Figure 1, an *intrusion detection system* is used to monitor network for malicious activities or policy violations and a *WAN optimizer* is used to maximize the efficiency of data flow across a wide area network. The two components can be deployed in different data-centers and are connected together as a chain to provide the necessary functionalities for the network. For this structure, the end-to-end latency will be accumulated from each hop in the chain (including network latency and processing delay at data-centers).

B. Parallel Structure

In Figure 2, we present an example of a parallel composite service structure. An example use case could be a user watching a video stream, with automatic subtitling or language translation with re-encoding of a specific video codec suitable for rendering on the user's end device. The streaming service, therefore, needs to send: (1) the audio to a real time translator, and, (2) the video to a encoder, which can be deployed in different locations. The translated audio and encoded video are then forwarded to a rendering box that will stream the merged audio and video channels to the user. The steps of translating and encoding can be achieved in parallel and the end-to-end latency will be the longest of the two branches.

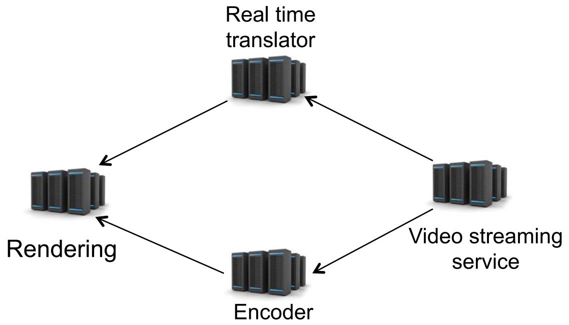


Fig. 2. Video streaming with translator and decoder.

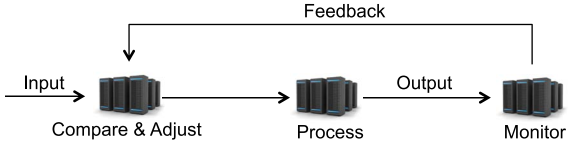


Fig. 3. Closed loop system.

C. Cycle Structure

The third structure we are presenting is a cycle. An example could be the closed loop system shown in Figure 3. It is similar to the chain structure except there is a loop to provide feedback to make decisions in subsequent rounds. The end-to-end latency is accumulated for each hop. Examples of this include services where components send any sort of application feedback to the source. For example a camera that needs to be steered remotely based on the image it sends, or a drone that needs to be tele-guided based on sensory information it is reporting to the source.

These three basic structures can be combined to form any complex composite service. In this paper, we present algorithms to maximize the utility for each of the basic composite structures. The algorithms can be combined to find solutions for more complex structures. By maximizing the utility, we consider several constraints for the composite service deployment problem:

- Fixed cost: the cost of deploying the service for the first time at a data-center. This can be thought as the cost to transfer, install and store the software in that location and could include one-off software license costs. The fixed cost is incurred only once and does not vary with the number of service instances at a certain location.
- Linear cost: this cost is proportional to the resources used by the service. The more service instances are required the more resources are consumed and hence the cost incurred.
- Latency: this includes both the network latency and processing time at data-centers. There is a trade-off, for example, between deploying services in a distant data center with a higher network latency but faster processing time, or choosing a closer low-latency location with slower processing time. Our algorithms consider this

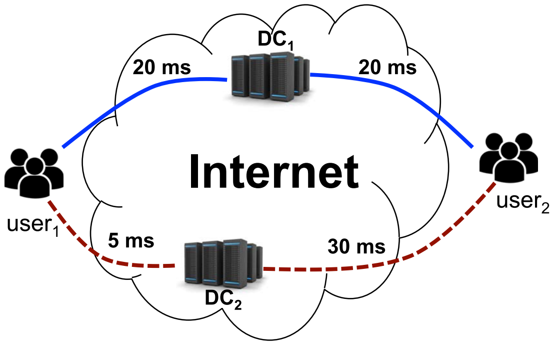


Fig. 4. Utility based vs. closest based placement.

trade-off in the optimization model. In addition, some services require a higher performance connection between users and the first hop component. For example, users should connect to a low-latency rendering component in an on-line game service to reduce lag as the player moves viewpoint, while the game simulation engine itself could be located more remotely if the position of other players does not change rapidly and so a longer latency would not impact game play. Therefore, along with the end-to-end latency, we also consider the first hop latency as a constraint when deploying a composite service.

In this paper, we convert latency (both network latency and processing time at a data-center) into a utility score as a metric to evaluate QoS. We first summarize the idea of the utility function presented in our previous work [26].

III. Utility Function

A. Illustrative Example

For simplicity, we show the idea of the utility function based on network latency of an atomic service. As an example, Figure 4 shows two groups of users who wish to access a service, e.g., a real-time audio translation service for two users located in either group 1 or 2 whose conversation is translated in real time by a cloud-based service deployed in one of the shown DCs. Due to deployment cost constraints, we assume that the service instance can only be deployed in a single DC (DC_1 or DC_2). Latencies between users and DCs are shown in Figure 4. Users do not perceive a degradation in the quality of voice services, when the latency is equal to or less than 20 ms [30]. Therefore, 5 ms or 20 ms latency gives equivalent (and the best) QoS for interactive voice services.

As shown in Figure 4, the classical closest based algorithm would minimize average latency and result in deploying the service at DC_2 (dashed lines) with an average latency from DC to the users is $(5 + 30)/2 = 17.5$ ms. However, using this deployment solution, user 1 will experience excellent QoS while user 2 experiences degradation (compared to the threshold of 20 ms for voice services). For that reason, a better solution would be to deploy the service at DC_1 where both users receive the highest QoS, with a latency of 20 ms. This shows that we can design a utility function which can be adapted for specific services and provides a better approach in evaluating QoS than a classical one that simply minimizes latency.

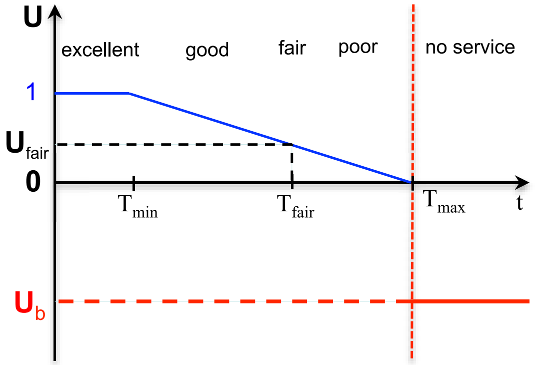


Fig. 5. Utility function (U) vs. latency (t).

B. Utility Function

Our general utility function is grounded on practical research on quality of service utility [12], [14] and years of investigation on Mean Opinion Scores [3]. Our interval data points map to user ratings of *excellent*, *good*, *fair*, *poor* and *no service or blocked* (Figure 5).

In the utility framework, application providers determine the utility function by setting two latency thresholds: T_{min} and T_{max} (the utility is not restricted to only latency, it can be extended to other QoS metrics such as bandwidth, jitter, etc.). As shown in Figure 5, we use a non-increasing piece-wise linear utility function that is characterized by:

- If $t \le T_{min}$: depending on the service type, an appropriate value of T_{min} is selected; even if the latency is reduced below this value, the improvement is not perceived by the users of that service, thus the utility is unchanged ($U_{max} = 1$). For instance, *voice over IP* requires $T_{min} =$ 20 ms [30]; for *simple Web services*, $T_{min} = 100$ ms gives users the feeling of instantaneous response [24].
- If $T_{min} < t \le T_{max}$: QoS is within an acceptable range ($0 \le U < 1$). User satisfaction reduces as the latency increases. We also define $T_{fair} \in [T_{min}, T_{max}]$ as the point from which users start to feel disappointed about the services as QoS is getting poorer. Note that the value of T_{fair} is set depending on services and does not change the slope of the utility graph.
- If $T_{max} < t$: the request is *blocked* (*no service*) because the latency is beyond the acceptable range.

Given these definitions, the utility function can be computed as follows (see [26] for more details):

$$U = \begin{cases} 1 & \text{if } t \le T_{min} \\ 0 \le \frac{T_{max}-t}{T_{max}-T_{min}} < 1 & \text{if } T_{min} < t \le T_{max} \\ U_b < 0 & \text{otherwise} \end{cases}$$

Based on this utility function, the utility-maximizing solution for the problem in Figure 4 should be to deploy the service at DC_1, as both users will get the maximum utility with $t = T_{min} = 20$ ms. Given the idea of the utility function, we design optimization formulations for each composite

TABLE I
SUMMARY OF KEY NOTATIONS (ALPHABETICAL ORDER)

C_v	unit cost to deploy the service at DC v
$\mathcal{D}$	set of user requests $\mathcal{D} = \{D_i, \forall i \in \mathcal{I}\}$
D_i	required resource from user i
E	set of links in auxiliary graph
F_v	fixed cost to deploy the service at DC v
i	user $i \in \mathcal{I}$ where $\mathcal{I}$ is a set of users
L_{uv}	network latency for link (u, v)
$N(v)$	set of neighbors of v in the auxiliary graph
p_v^i	if DC v is selected by user i or not
P_v	processing time at DC $v \in V$
s	virtual source
t_i	end-to-end latency perceived by user i
U_i	end-to-end utility for user i
V	set of nodes in the auxiliary graph
x_{uv}^i	link (u, v) is chosen by user i or not
y_v	DC v is selected or not (for fixed cost)
z_i	a variable used for utility calculation of user i

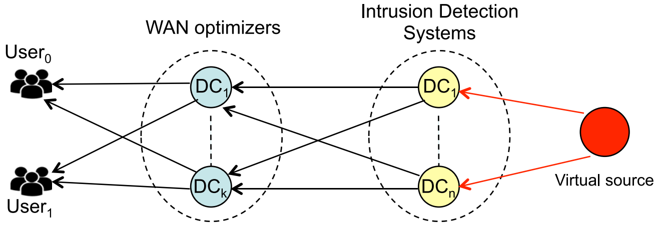


Fig. 6. Auxiliary graph for the chain structure.

service structure, where the objective is to maximize the total utility over all users.

IV. COMPOSITE SERVICE FORMULATION

A. *Chain Structure*

To formulate the optimization problem, we first create an auxiliary service graph as in Figure 6 for the given chain structure in Figure 1. Each instance in a dashed circle represents a data-center location, where we can deploy that type of component of the composite service. If a DC is able to deploy multiple types of components, this DC appears in several circles. For example, as shown in Figure 6, DC_1 is able to deploy both WAN optimizer and intrusion detection system components, therefore it appears in both the circles. When in use, we need to create a chain work-flow connecting each instance toward the user as shown in Figure 1. The auxiliary graph is created as follows:

- We create a virtual source connecting to all instances in the last group (e.g., intrusion detection system in Figure 6). The links between the virtual source and those instances have zero latency.
- A full mesh connection is defined between each component as in Figure 6. Each link has an associated network latency and we can remove those which have latencies exceeding the maximum end-to-end or one-hop latency constraints.

Based on this auxiliary graph, we develop a mixed integer linear program formulation (see the notations in Table I) to find optimal service placement solutions for the chain structure:

$$\max \sum_{i \in \mathcal{I}} U_i \tag{1}$$

$$\text{s.t.} \quad \sum_{v \in N(u)} \left(x_{uv}^i - x_{vu}^i\right) = \begin{cases} 1 & \text{if } u = s \\ -1 & \text{if } u = i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$p_u^i \geq x_{uv}^i \;\; \forall (u, v) \in E, \;\; i \in \mathcal{I} \tag{3}$$

$$p_v^i \geq x_{uv}^i \;\; \forall (u, v) \in E, \;\; i \in \mathcal{I} \tag{4}$$

$$t_i = \sum_{(u,v) \in E} L_{uv} x_{uv}^i + \sum_{v \in V} P_v p_v^i \quad \forall i \in \mathcal{I} \tag{5}$$

$$y_v \geq p_v^i \quad \forall v \in V, i \in \mathcal{I} \tag{6}$$

$$\sum_{i \in \mathcal{I}} \sum_{(u,v) \in E} D_i C_v x_{uv}^i + \sum_{v \in V} F_v y_v \leq COST \tag{7}$$

$$z_i \geq 0 \tag{8}$$

$$z_i \geq t_i - T_{min}^i \tag{9}$$

$$U_i = \frac{T_{max}^i - T_{min}^i - z_i}{T_{max}^i - T_{min}^i} \tag{10}$$

$$x_{uv}^i, y_v, p_u^i \in \{0, 1\} \quad \forall i \in \mathcal{I}, (u, v) \in E, v \in V \tag{11}$$

where:

- Objective function (1) is to maximize total utility over all users.
- (2) represents flow conservation constraints, making sure that a flow from the virtual source to each of the users can be found. There will be no flow outgoing from the user i and no flow incoming to the virtual source s. For intermediate nodes, the outgoing and incoming flows should be equal.
- We use binary variable p_u^i and p_v^i in constraints (3)-(4) to determine if user i uses node u or node v, then equation (5) is used to compute the end-to-end latency which includes network latency ($\sum_{(u,v) \in E} L_{uv} x_{uv}^i$) and processing time ($\sum_{v \in V} P_v p_v^i$).
- Constraints in (6) are used to determine if a DC is selected (by any users) or not. These will be used to compute the fixed cost.
- Constraint (7) limits the deployment cost which includes both the fixed cost ($\sum_{v \in V} F_v y_v$) and the linear cost ($\sum_{i \in \mathcal{I}} \sum_{(u,v) \in E} D_i C_v x_{uv}^i$). The cost of establishing a relationship with a DC and other one-off costs such as installing the application software is represented by the fixed cost. The cost of the computational resources consumed by the running instances of an application component is represented by the linear cost.
- Constraints (8), (9) and (10) are used to compute the utility function mentioned in Section III-B.
  - If $t_i \leq T_{min}^i$, based on constraints (8) -(9), z_i can take any value that is greater or equal to 0; however, due to the objective function maximizing the utility (1), the minimum value of z_i is chosen, or in other words, z_i is set to 0 and thus, $U_i = 1$ (the maximum utility, when $t_i \leq T_{min}^i$).
  - Similarly, if $t_i > T_{min}^i$, the formulation will choose $z_i = t_i - T_{min}^i$ and thus $U_i = \frac{-t_i + T_{max}^i}{(T_{max}^i - T_{min}^i)}$.

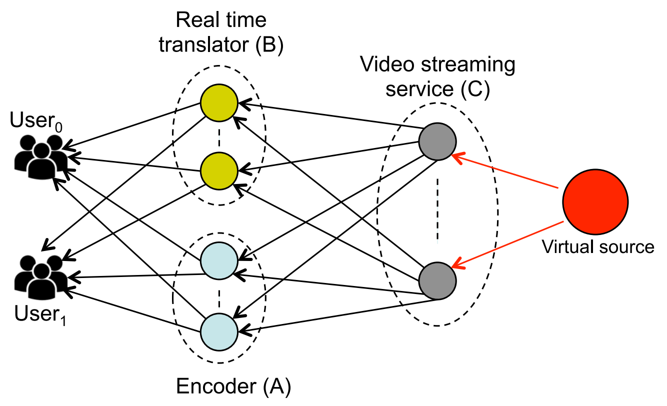


Fig. 7. Auxiliary graph for the parallel structure.

B. Parallel Structure

For a composite service with three components such as in Figure 2, we create an auxiliary service graph as in Figure 7. The technique used here is similar to the one applied for the chain structure. We then introduce an integer linear program formulation based on the auxiliary graph.

Let A, B and C be the groups of DCs that are capable to deploy "encoder", "real time translator" and "video streaming" services, respectively and assume that each user connects to its local rendering box.

$$\textit{objective} \quad (1)$$

s.t.

$$\sum_{v \in N(u)} \left(x^i_{uv} - x^i_{vu}\right) = \begin{cases} 1 & \text{if } u = s \text{ or } u \in C \\ -2 & \text{if } u = i \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\sum_{v \in B, u \in C} x^i_{uv} = 1 \ \forall i \in \mathcal{I} \tag{13}$$

$$\sum_{v \in A, u \in C} x^i_{uv} = 1 \ \forall i \in \mathcal{I} \tag{14}$$

$$\sum_{u \in B} x^i_{ui} = 1 \ \forall i \in \mathcal{I} \tag{15}$$

$$\sum_{u \in A} x^i_{ui} = 1 \ \forall i \in \mathcal{I} \tag{16}$$

$$t_i \geq L_{uv}\left(\sum_{u \in C, v \in B} x^i_{uv} + \sum_{u \in B} x^i_{ui}\right) + \sum_{v \in V} P_v p^i_v \quad \forall i \in \mathcal{I} \tag{17}$$

$$t_i \geq L_{uv}\left(\sum_{u \in C, v \in A} x^i_{uv} + \sum_{u \in A} x^i_{ui}\right) + \sum_{v \in V} P_v p^i_v \ \forall i \in \mathcal{I} \tag{18}$$

$$\textit{Constraints} \quad (3, 4, 6 - 11)$$

where:

- Constraints (12) -(16) are used to make sure that we will find a parallel structure from s to each of the users.
- Constraints (17) -(18) are used to find the maximum latency between the two branches. This maximum value will be the end-to-end latency for the composite services.

C. Cycle Structure

For a composite service with three components such as shown in Figure 3, we create an auxiliary service graph as in Figure 8. The technique used here is similar to the one applied for the chain or the parallel structures.

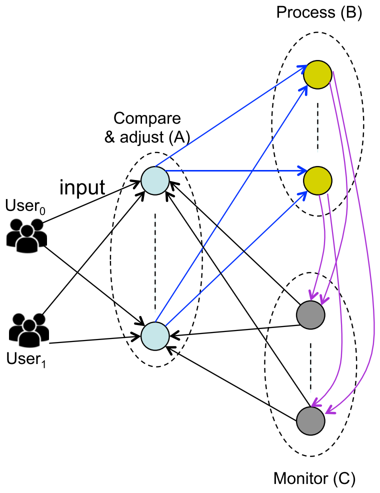


Fig. 8. Auxiliary graph for the cycle structure.

Let A, B and C be the groups of DCs that are capable to deploy "compare and adjust", "process" and "monitor" services, respectively.

$$\textit{objective} \quad (1)$$

s.t.

$$\sum_{v \in N(u)} \left(x^i_{uv} - x^i_{vu}\right) = \begin{cases} 1 & \text{if } u = i \\ -1 & \text{if } u \in A \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\sum_{u \in A, v \in B} x^i_{uv} = 1 \ \forall i \in \mathcal{I} \tag{20}$$

$$\sum_{u \in B, v \in C} x^i_{uv} = 1 \ \forall i \in \mathcal{I} \tag{21}$$

$$\sum_{u \in C, v \in A} x^i_{uv} = 1 \ \forall i \in \mathcal{I} \tag{22}$$

$$\sum_{u \in A} x^i_{iu} = 1 \ \forall i \in \mathcal{I} \tag{23}$$

$$t_i = \sum_{v \in N(u)} L_{uv} x^i_{uv} + \sum_{v \in V} P_v p^i_v \ \forall i \in \mathcal{I} \tag{24}$$

$$\textit{Constraints} \quad (3, 4, 6 - 11)$$

where:

- Constraints (19) -(23) are used to make sure that we will find a cycle flow for each user as in Figure 3.
- Constraints (24) are used to compute the end-to-end latency of the cycle structure.

D. Single Hop and End-to-End Utility

There could be different utility requirements for each hop in a composite service graph. For instance, some composite services require the first hop to have much lower latency (e.g., between the user and the rendering component of an on-line game). Therefore, in addition to end-to-end utility, we should also consider the first hop utility (U^f_i). To achieve this, we

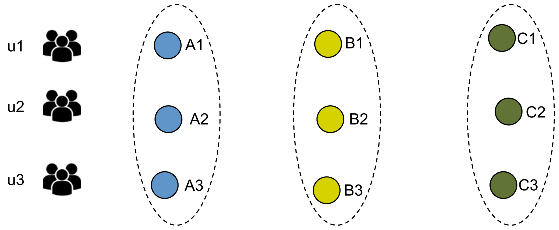


Fig. 9. Example of chain structure.

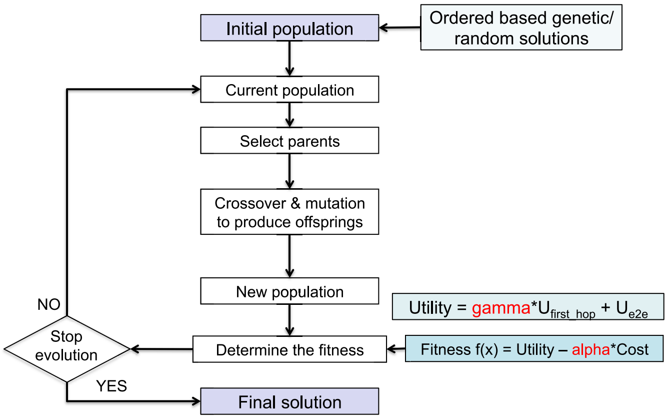

Initial population
Ordered based genetic/ random solutions
Current population
Select parents
Crossover & mutation to produce offsprings
New population
Utility = gamma*U_first_hop + U_e2e
NO
Stop evolution
Determine the fitness
Fitness f(x) = Utility – alpha*Cost
YES
Final solution

Fig. 10. Genetic algorithm flow chart.

replace the objective function (1) by:

$$\max \quad \gamma \sum_{(s,i)\in\mathcal{D}} U_i^f + \sum_{(s,i)\in\mathcal{D}} U_i \tag{25}$$

where U_i^f is calculated in the same way as U_i but only for the first hop. Depending on how important the first hop utility is, we set an appropriate value for $\gamma \geq 0$. To extend this concept further we could define a general utility function which is a combination of the utility of any single hop (not just the first hop) and the end-to-end utility.

We have introduced MILP models for the three basic structures of composite services. Service providers can create a MILP model for their specific composite service by combining the appropriate structures. However, as the composite placement is in the family of facility location problems and these are known to be NP-hard, we can only find the optimal solution for small input datasets (number of users, number of data centers). In the next subsections, we introduce genetic algorithms which can scale better with the size of input data.

V. Genetic Algorithm (GA)

To illustrate the optimization of composite services, we present a genetic algorithm for the chain structure with three components, shown in Figure 9. We can apply this strategy to solve for other composite structures. Each group in Figure 9 represents a set of DCs capable of deploying one kind of component (e.g., some components can only be deployed in DCs equipped with special hardware like GPU, etc.). If a DC can deploy all three components, it will appear in all groups.

The genetic algorithm is based on the flow chart depicted in Figure 10. We define a fitness function $f(x)$ to evaluate each chromosome x in the population. Based on the fitness score, good chromosomes are selected for the next generations.

In this genetic algorithm, we use the utility function in (25). Then the algorithm tries to maximize the following fitness function:

$$\max \quad f(x) = (utility - \alpha \times cost) \tag{26}$$

where *cost* is the sum of the linear and the fixed costs at DCs. We use α as a parameter to give a trade-off between the utility and the cost. Good chromosomes are the ones that have higher values of $f(x)$. We explain in detail the steps of the GA:

1) Start: The initial population is a set of potential solutions to the problem. Each solution contains a list of vectors. Each vector is a list of integers representing a chain structure connecting a user to each of the three components. As an example, [(1, 1, 2, 1), (2, 2, 1, 1), (3, 1, 2, 3)] represents a possible solution for the three users: (u_1, A_1, B_2, C_1), (u_2, A_2, B_1, C_1), and (u_3, A_1, B_2, C_3). We randomly choose some solutions to be in the initial population. Some of them may not be feasible, but they will be eliminated in next generations of the genetic algorithm. Moreover, we can use output of an order-based genetic algorithm (Section VI) to be the initial population.
2) New population: After each round, a new population is created by the following steps until reaching the stopping criterion (maximum number of generations):
   - Selection: select parents' chromosomes from a population according to their fitness (the higher the fitness, the higher chance to be selected).
   - Crossover: with a crossover probability, swap part of the information between pairs of parents to form new children.
   - Mutation: with a mutation probability, randomly alter some genes inside a chromosome to get a new chromosome.
3) Stop: If the end condition is reached (e.g., maximum number of generations), the algorithm stops, and returns the best solution in the current population.

VI. Order-Based Genetic Algorithm

By introducing the fixed cost, we try to minimize the number of DCs in use. We observe that if we pick up users one by one, and try to maximize the fitness of those users, then the order of users to be picked up is important and affects the final solution. This occurs since the flows of later users that reuse components deployed in DCs used by earlier user flows do not incur the fixed cost. In this section, we introduce an order-based algorithm, which is actually a genetic algorithm with a different representation (based on permutations) aiming to find the best sequence of user flows to consider.

1) Initial population: Random orders of users, e.g., (u_2, u_3, u_1), (u_1, u_3, u_2) are used to create the initial population.
2) To decode a solution, for each user in the sequence, greedily select an assignment that minimizes $f = (latency + \alpha \times cost)$. The *cost* here includes the fixed and

linear cost. For more detail on the greedy approach, for instance, we choose the order (u_2, u_3, u_1). Then starting from u_2, we need to select A_1, A_2 or A_3 so that the value of f is minimal. Let's say we choose A_1, then from A_1 we need to decide to go to B_1, B_2 or B_3 such that the value of f is minimal and so on. After finishing for u_2, we continue with u_3 in a similar way. Because of the fixed cost, the order of users is important.

3) Stop: When reaching a maximum number of generations, the algorithm returns the best order of users that maximizes $fitness = (utility - \alpha \times cost)$, where $utility = \gamma \times U_{first\_hop} + U_{e2e}$.

We implemented the genetic algorithms using "inspyred" - the open source framework for creating biologically-inspired computational intelligence algorithms in Python.[1] The inspyred library provides basic components such as a generator to define how solutions are created and an evaluator that defines how fitness values are calculated for solutions. In addition, several evolutionary operators are also available to use such as "selector", "variator", "replacer" (to determine parents and new population) and "terminator" (to say whether the evolution should end).

VII. Simulation Results

First, we solve the mixed integer linear program model using IBM's CPLEX solver [2]. All computations were carried out on a computer equipped with a 3 GHz CPU and 8 GB RAM. We use a dataset with 2508 data centers distributed in 656 cities all over the world (the dataset is described in more detail in [1]). The fixed deployment cost is based on the Amazon EC2 charging model. The user demand is proportional to the population of each city [4]. Latency between users and execution zones are computed based on Haversine distance between two points around the planet's surface [11].

In the following, we use an example chain of three components, where the first component has much tighter constraints on network latency to the users than the others. For example, this could be a cloud-based rendering service for a 3D virtual reality environment or online game, where the scene, from the point-of-view of the user needs to track the user's head movement in real time and, hence, the latency between the user and this component should be very low to avoid noticeable lag. The other components may be managing the environment state and providing background objects and textures, and although end-to-end latency to the distant component needs to be within the overall utility bounds, the constraints are not as tight as to the renderer. This pattern of component chains, with extra constraints on the positioning of the nearest component, could be applicable to many services, including virtual and augmented reality, games and video conferencing.

For the purpose of simulation, based on measurements of QoS of on-line games [13], we configure the utility function and other related parameters as follows:

- The first hop represents the graphic rendering component. As shown in [13], users in a first-person shooter game are aware of latencies above 20 ms, while in car racing simulations, only latencies above 50 ms affect game results [25]. Thus, we set $T_{min} = 20$ ms and $T_{max} = 50$ ms for first hop utility.
- Most players in impaired games can tolerate latencies of up to 150 ms [13], and so we set $T_{min} = 50$ ms and $T_{max} = 150$ ms for the end-to-end (E2E) utility, which covers the full chain of components.
- We consider latency to be the sum of network latency and processing time at DCs. This processing time is considered inversely proportional to DC's fixed cost (the intuition is the more expensive a DC is, the faster processing time it has).

We summarize the notations used in the simulations as follows:

- Improvement score of algorithm (A) vs. algorithm (B) $= \frac{100(f(A)-f(B))}{|f(A)|}$
- Fitness f = Utility $-\alpha\times$ Cost
- Utility $= \gamma \times U_{first\_hop} + U_{e2e}$
- *GA (order)*: we first run the order-based genetic algorithm to find a solution, then put this solution along with random ones into the initial population.
- *GA (rand.)*: we run the genetic algorithm in which the initial population is totally random.
- *Order-based*: we run the order-based generic algorithm as described in Section VI.
- *Best random*: we randomly select 2×10^5 solutions for both the small and the large dataset, then get the one with the best fitness value.

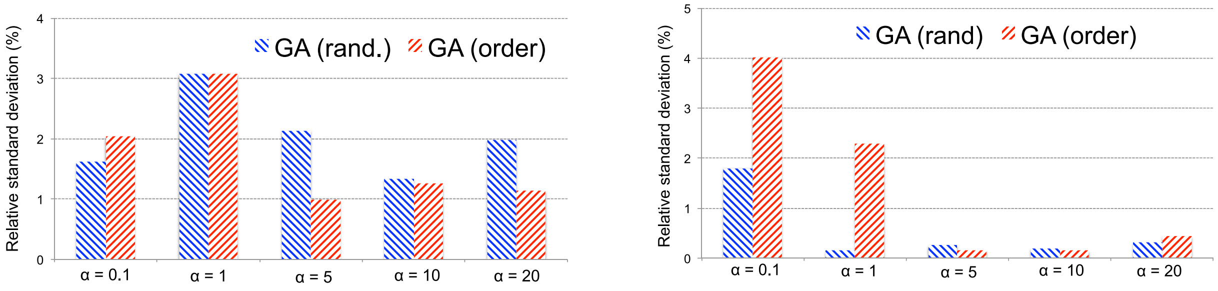
First, we show the comparison of the random, the order-based genetic, the genetic and the optimal solution for a small dataset. Next, we present evaluation results of genetic algorithms for a larger dataset. Both datasets are subsets of our full dataset (detail is presented in the next subsections). Since GAs are stochastic, we run each simulation ten times and take the mean result. Figure 11 shows the relative standard deviation (RSD) of the fitness function of the scenario where $\gamma = 2$, over ten simulation runs; a low RSD indicates that the data points are close to the mean. We omit scenarios for other values of γ, as the results are similar. As shown in Figure 11, in both the small and the large data sets, variation over the ten runs is small, less than 4%.

A. Small Dataset

We focus on the simulation results for the chain structure as the observations we found are also similar for the parallel and the cycle structures. The dataset includes 25 users, 3 groups of DCs (group A, B and C as in Figure 9), each respectively has 58, 78 and 85 DCs in which we randomly select from the full dataset (around 10% of the total 656 DCs). We set a stopping criterion for the genetic algorithm (GA) so that it will explore around 2% of the searching space (2×10^5 solutions).

1) Pareto Graph Cost vs. Utility: Given a placement solution, we can plot its cost and utility on a 2-D plane as in Figure 12. Note that Figure 12 is for $\gamma = 2$ (first hop utility is twice as important as end-to-end utility; we omit other γ cases as the results are similar). For the MILP, given cost

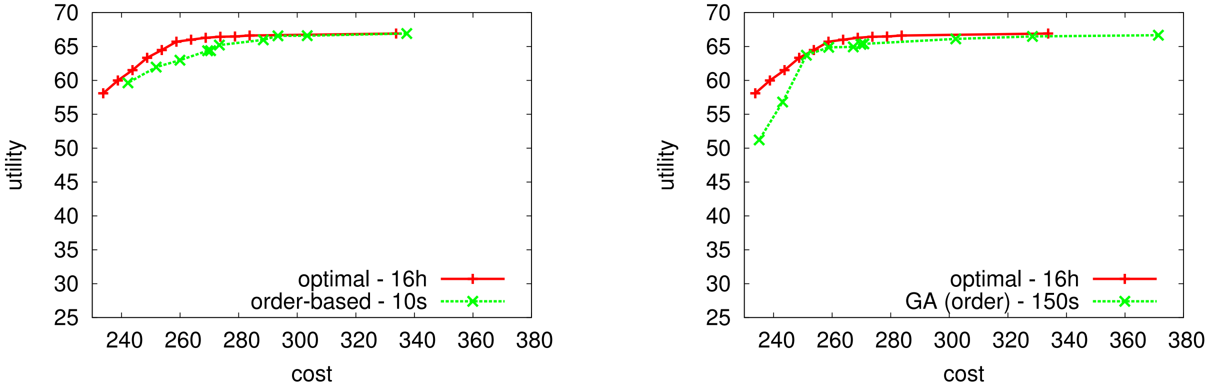
[1] https://pypi.python.org/pypi/inspyred



(a) Small data set

(b) Large data set

Fig. 11. Relative standard deviation (RSD) of fitness value.



(a) Optimal vs. order-based

(b) Optimal vs. GA

Fig. 12. Pareto graph of utility vs cost.

as a constraint, we try to maximize the utility for that cost and find each point in the Pareto curve (in red). We identify 10 points in which the cost is varied from 0 to a maximum value ($COST_{max}$) that allows to find the best utility (note that this utility cannot be improved, even when the cost is larger than $COST_{max}$). Then, we set each value of cost in the constraint (e.g., inequality (7) in Section IV-A) and find the optimal solution which maximizes the utility. Effectively, the curve connecting these optimal points forms a Pareto front on the plane. Using more than 10 intervals can help to create a smoother graph, but requires a longer computation time.

For the order based and GA algorithms, as we do not explicitly have a cost constraint, we adjust the value of α in the fitness function to see different solutions, i.e., different trade-offs of the two components of the objective function. Each solution provides a pair of (utility, cost) values. Then, we pick up 10 pairs which have the best utility for a given cost and draw the Pareto front as in Figure 12.

In Figure 12, we also show the computation time of the algorithms. It took the MILP 16 hours to find the optimal curve (1.6 hours in average to find one point in the curve), while only 150 seconds and 10 seconds for the GA and order-based algorithms, respectively, to find their Pareto curves. While reducing execution time significantly, the solutions of the order based and the GA algorithms are close to optimal.

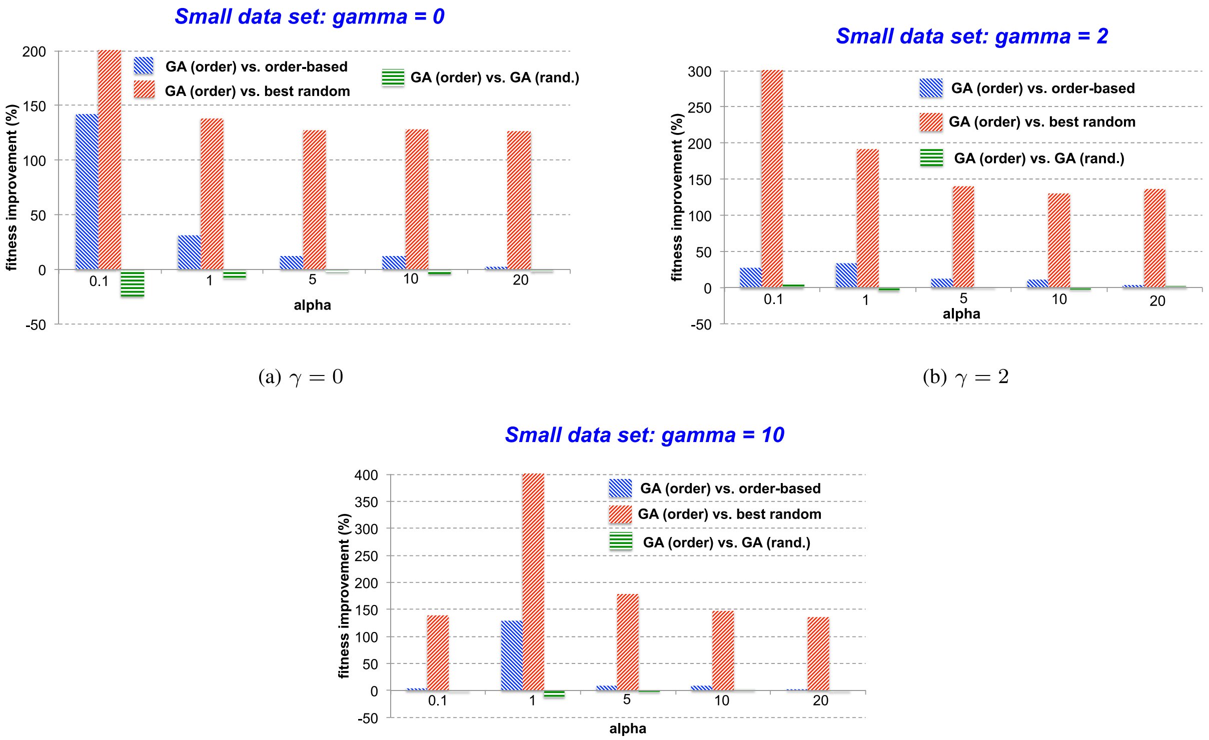
2) GA vs. Order Based vs. Random Algorithms: In figure 13, we show a comparison between several algorithms: GA (order), GA (rand.), order-based and the best random algorithms. By showing the improvement score of algorithm A vs. algorithm B, we can see which one is better for different values of the γ parameter. Positive values of the improvement score means that the algorithm A is better than algorithm B.

We can make the following observations based on Figure 13:

- The *order-based*, *GA (order)* and *GA (rand.)* algorithms are much better than the *random* solutions.
- In general, *GA (order)* and *GA (rand.)* are the best algorithms. They show a significant improvement over the *order-based* and *random* ones.
- In some cases, *GA (rand.)* is better than *GA (order)* and vice versa. For the *GA (order)* algorithm, as we start with a reasonably good solution this could be a locally optimal point and the algorithm can be trapped there. On the other hand, the *GA (rand.)* algorithm starts with a totally random solution, reducing the chance of starting in a local optimum, and because the search space for this small dataset is not huge ($\sim 96 \times 10^5$) it can find a good solution despite starting from a random position. This explains why the *GA (rand.)* algorithm can end up with better solutions than the *GA (order)* algorithm (negative improvement score in Figure 13).

It is worth noting that the difficulty level of the problem we solve depends on the parameters we set:

- The overall objective is to maximize the fitness function (Fitness = Utility $-\alpha\times$ Cost where Utility = $\gamma \times U_{first\_hop} + U_{e2e}$). When α is small (i.e., positive and close to zero), utility is dominant over cost in the fitness function. Also, when γ is large, the first hop utility (U_{first_hop})



(a) $\gamma = 0$

(b) $\gamma = 2$

(c) $\gamma = 10$

Fig. 13. Improvement score with different γ for small data set.

is dominant over the end-to-end utility (U_{e2e}). In this case, maximizing the fitness function is equivalent to maximizing the first hop utility. Optimizing the first hop utility is relatively easy as it simply chooses the best next hop in terms of latency. Therefore, when first hop utility is dominant (small α and large γ), a simple heuristic algorithm can also find a good solution.

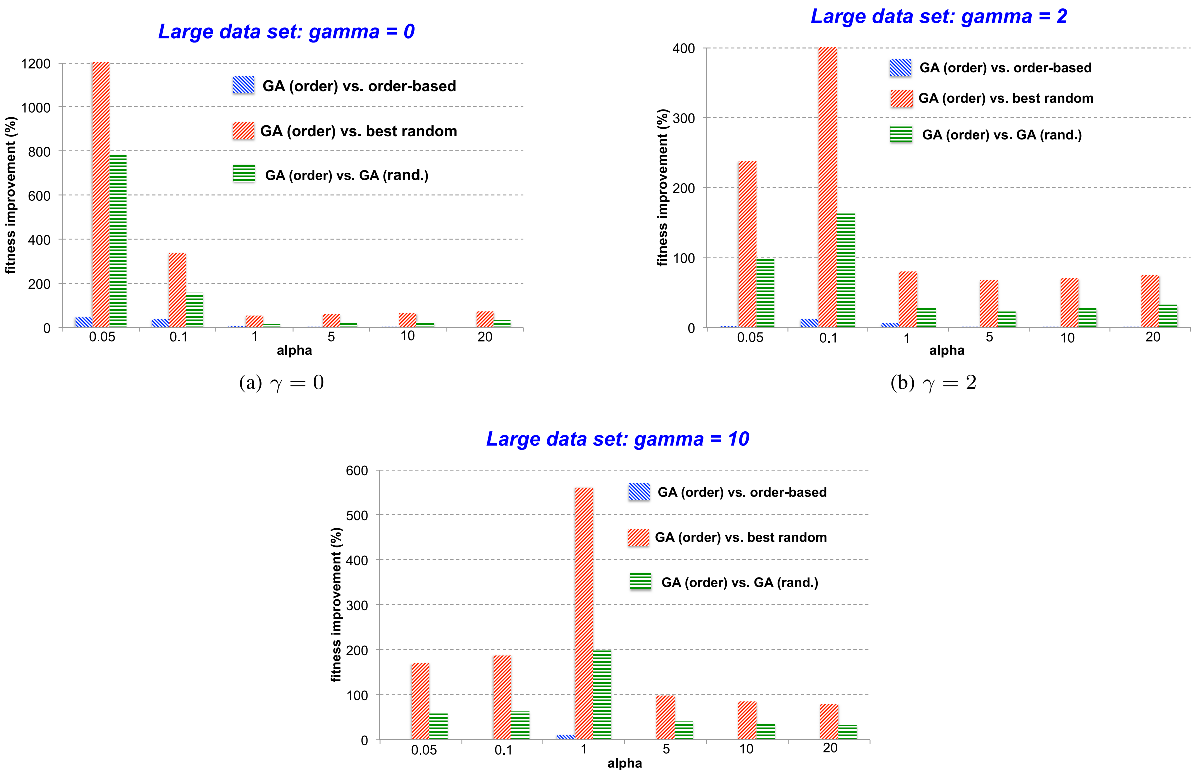
- On the other hand, when α is large enough, maximizing the fitness function is equivalent to minimizing the cost. The cost we are using here includes fixed (deployment) and linear cost. However, the fixed and linear costs are proportional (high fixed cost will lead to high linear cost). Therefore, minimizing the total cost is equivalent to avoid using high cost DCs. This is a simple task as the algorithm tries to use the low cost DCs first.
- The more general case is when both utility and cost are important (α is not too small and γ is not too large). This makes the task more difficult as it is a multi-objective optimization problem: trying to maximize utility and minimize cost at the same time. In this case, a much more intelligent algorithm is needed to find a good solution.

The results shown in Figure 13 can be explained as follows.

- Figure 13a: because $\gamma = 0$, the first hop utility is not considered in the optimization. This is the case when both utility and cost are important and it is difficult to find a good solution. When increasing α, the cost becomes more dominant and it is easier to find a good solution (a simple heuristic algorithm is enough). This explains why the gaps between GA (order) and order based algorithms reduce when α increases. Simple order based algorithm can find as good solution as the intelligent GA (order).
- Figure 13b and Figure 13c: when α is small (i.e., $\alpha < 1$), this is the case where the *first hop utility is dominant*. On the other hand, when $\alpha > 1$ the *cost is dominant*. That is why the gaps between GA (order) and order based algorithms increase and then reduce as α increases. The more difficult the task is, the better the GA (meta-heuristic) performs compared to the simple heuristic (order based) case. Note that we can see this behavior clearer in Figure 13c than in Figure 13b, since in the latter case, when $\gamma = 2$, it is not large enough for the first hop utility to be fully dominant over the end-to-end utility. Therefore, the optimization task is still hard even when α is small. That is why we see a similar gap between GA (order) and order based algorithms when $\alpha = 0.1$ and $\alpha = 1$.

B. Large Dataset

The larger dataset consists of 1834 users, 3 groups of DCs (groups A, B and C as in Figure 6), each has 656, 328 and 656 DCs, respectively. We use only 328 DCs in group B to avoid trivial solutions where all three components of one chain are in the same DC (to minimize latency). By eliminating those trivial cases, we are able to highlight the benefits of intelligent algorithms over the greedy ones. In trivial cases, there is not a major difference between the results of the algorithms as all of them are able to find the optimal solutions.



(a) $\gamma = 0$

(b) $\gamma = 2$

(c) $\gamma = 10$

Fig. 14. Improvement score with different γ for large data set.

We set the stopping criterion for the GA so that it will explore approximately 0.00008% of the searching space (2×10^5 solutions - taking around 30 minutes to execute).

1) GA vs. Order Based vs. Random Algorithms: We observe the following in Figures 14a to 14c:

- GA (order) is always the best algorithm. Even when exploring just 0.00008% of the search space, GA (order) can show 25% improvement over the order based algorithm (Figure 14a).
- In many cases, the gaps between GA and other algorithms reduce as α increases (as optimizing for cost is more straightforward than for utility).
- In many cases, the gaps between GA and order-based algorithm reduce when increasing γ (as optimizing first hop utility is more straightforward than end-to-end utility).

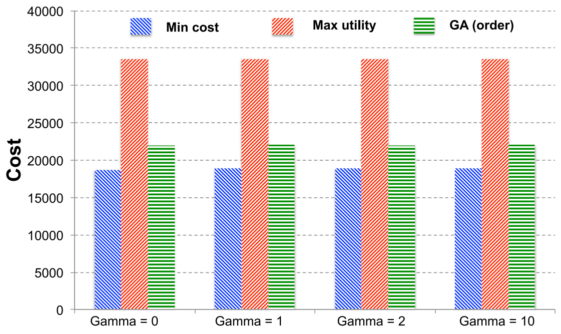


Fig. 15. Cost of min cost vs. max utility vs. GA algorithm.

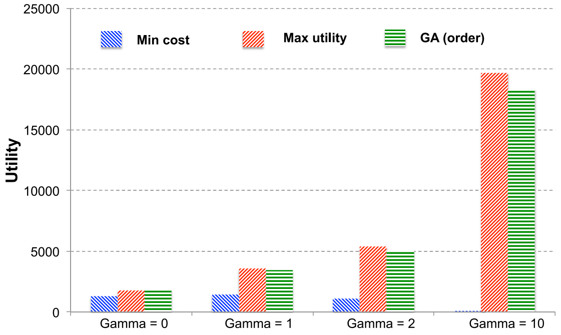


Fig. 16. Utility of min cost vs. max utility vs. GA algorithm.

2) Comparison (Minimum Cost vs. Maximum Utility vs. GA Algorithms): Elsewhere in the literature, for example [8]–[10], [28], minimizing cost is the main objective. As in our work they also consider both fixed and linear costs, but include a wider variety of costs such as energy and traffic, which could also be included in our future work. In this section, we made a comparison between our utility-based optimization versus the cost-minimization approach widely adopted in other work in the literature, as indicated by our results on *minimum cost GA*. In the minimum cost GA, we set α to be large enough in the fitness function ($fitness = utility - \alpha \times cost$) to force the algorithm to minimize the cost. On the other hand, in the maximum utility GA we set $\alpha = 0$, meaning that the algorithm tries to maximize the utility. The GA (order) used in Figures 15 and 16 is the GA in which we test with different values of α and choose the one that has a good trade-off between utility and cost. As shown in Figures 15 and 16, the GA results in marginally higher costs than the minimum cost algorithm (Figure 15), while its utility is close to the maximum utility one (Figure 16). It is noteworthy to mention that the maximum utility and the GA are also close to the optimal maximum utility solution. For example, when $\gamma = 10$ (Figure 16), the maximum total utility is $1834 \times 11 = 20174$ (1834 users, each has a maximum utility score $= 10 \times U_{first\_hop} + U_{e2e} = 11$). This confirms that the GA can find good solutions even for a larger dataset.

VIII. Related Work

In this section, we survey related work in the following three areas.

A. Composite Services

There are several studies in literature on Web service compositions [6], [15], [16], [32], [33]. One example of a composite Web service would be a "Travel Planner" which requires multiple components (e.g., flight booking, travel insurance, hotel booking, car renting, etc.) executed sequentially or concurrently. The paper [33] proposed linear programming methods to optimally select components during the execution of a composite service. As this problem is NP-hard, many studies propose the use of genetic and heuristic approaches to solve the QoS-aware composition problem [6], [15], [16], [32]. In general, these studies focus on Web services with specific requirements and constraints on execution duration, reliability, and availability. In addition to execution time at server, the papers [15], [16] also take into account network latency in the selection solutions. Similar to [15] and [16], in this paper we focus on general services which have constraints on network latency, processing time and service deployment cost at data-centers. In addition, we use a utility function instead of latency to evaluate QoS. Recently, there are some studies close to ours focusing on NFV deployment [9], [10]. However, they only work for the chain structure. In this paper, we propose algorithms for three general structures of composite services.

B. Atomic Service Placement and Selection

Our work is closely related to recent work on optimizing performance-cost for server selection [7], [31], [34]. For example, Wendell *et al.* [31] introduce DONAR - a decentralized replica-selection system that pushes clients to servers close-by. Zhang *et al.* [34] focus on optimizing cost and performance in online service provider networks. Auspice [29] uses a heuristic placement algorithm to determine the locations of active replicas to minimize client-perceived latency. In general, these works use network latency as a main metric to validate QoS and they only consider atomic services. In this work, we move a further step by consider to maximize the utility function for composite services.

C. Network Latency and Traffic Demand Estimation

Our work uses network latency and translates it into a utility value for evaluating QoS. Therefore, techniques to estimate network latency are important as this gives inputs for our algorithms. Recent works have shown that the IP geolocation of the user provides accurate and predictable network latency [5]. This has been confirmed by third-party datasets such as Peerwise [21] and iPlane [22]. In addition, those observations have been proved by our own extensive active measurements [17], [18]. On the other hand, how to predict traffic demand is also important for our optimization algorithms. Wendell *et al.* has shown in [31] that client request rate can be sufficiently predictable under short interval (e.g., 10 minutes). This work confirms that it is possible to provide an accurate predicted traffic demand for our optimization model.

IX. Conclusion

In this paper, we propose a utility framework used to optimize QoS for composite service placement. Since this is a NP-hard problem, we introduce a MILP formulation to find the optimal solutions for small datasets, while also designing genetic algorithms for larger inputs. Based on simulation results, we show that our approaches work well and close to the optimal solutions. Although our work has concentrated on optimizing network and processing latency within the constraints of deployment costs, the optimization formulation is easily extendable to include other network parameters such as bandwidth and the associated network transit costs. In future work we will further develop on-line algorithms to propose on-demand dynamic service placement algorithms. In addition, we will extend the genetic algorithms with multi-objective functions.

References

[1] *Data Center Map*. Accessed: Jan. 20, 2018. [Online]. Available: http://www.datacentermap.com/

[2] *ILOG CPLEX Optimization*. Accessed: Jan. 20, 2018. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio

[3] *Methods for Subjective Determination of Transmission Quality*. Accessed: Jan. 20, 2018. [Online]. Available: http://www.itu.int/rec/T-REC-P.800-199608-I/en

[4] *Video Distribution Modelling*. Accessed: Jan. 20, 2018. [Online]. Available: https://github.com/richardclegg/multiuservideostream

[5] S. Agarwal and J. Lorch, "Matchmaking for online games and other latency-sensitive P2P systems," in *Proc. ACM SIGCOMM*, 2009, pp. 315–326.

[6] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. Int. Conf. World Wide Web*, Madrid, Spain, 2009, pp. 881–890.

[7] A. Ascigil *et al.*, "On uncoordinated service placement in edge-clouds," in *Proc. IEEE CloudCom*, 2017, pp. 41–48.

[8] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[9] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of virtualized deep packet inspection functions in SDN," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, San Diego, CA, USA, 2013, pp. 992–997.

[10] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," in *Proc. IEEE NetSoft*, London, U.K., 2015, pp. 1–9.

[11] G. V. Brummelen, *Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*. Princeton, NJ, USA: Princeton Univ. Press, 2013.

[12] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility-based placement of dynamic Web applications with fairness goals," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, Salvador, Brazil, 2008, pp. 9–16.

[13] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proc. ACM SIGCOMM Workshop Netw. Syst. Support Games*, 2005, pp. 1–7.

[14] M. A. Khan and U. Toseef, "User utility function as quality of experience (QoE)," in *Proc. Int. Conf. Netw.*, 2011, pp. 99–104.

[15] A. Klein, F. Ishikawa, and S. Honiden, "Towards network-aware service composition in the cloud," in *Proc. Int. Conf. World Wide Web*, Lyon, France, 2012, pp. 959–968.

[16] A. Klein, F. Ishikawa, and S. Honiden, "SanGA: A self-adaptive network-aware approach to service composition," *IEEE Trans. Services Comput.*, vol. 7, no. 3, pp. 452–464, Jul./Sep. 2014.

[17] R. Landa *et al.*, "Measuring the relationships between Internet geography and RTT," in *Proc. IEEE ICCCN*, 2013, pp. 1–7.

[18] R. Landa *et al.*, "The large-scale geography of Internet round trip times," in *Proc. IFIP Netw.*, Brooklyn, NY, USA, 2013, pp. 1–9.

[19] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros, "Distributed placement of service facilities in large-scale networks," in *Proc. IEEE INFOCOM*, Barcelona, Spain, 2007, pp. 2144–2152.

[20] J. Li *et al.*, "DR-Cache: Distributed resilient caching with latency guarantees," in *Proc. IEEE INFOCOM*, Honolulu, HI, USA, 2018, pp. 1–9.

[21] M.-T. Lu *et al.*, "Design and evaluation of a P2P IPTV system for heterogeneous networks," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1568–1579, Dec. 2007.

[22] H. Madhyastha *et al.*, "iPlane: An information plane for distributed services," in *Proc. USENIX NSDI*, Seattle, WA, USA, 2006, pp. 367–380.

[23] E. Maini, T. K. Phan, D. Griffin, and M. Rio, "Hierarchical service placement for demanding applications," in *Proc. IEEE GlobeCom Workshop (CCSNA)*, Washington, DC, USA, 2016, pp. 1–6.

[24] J. Nielsen, *Usability Engineering*. San Fransisco, CA, USA: Morgan Kauffman, 1993.

[25] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proc. NOSSDAV*, 2002, pp. 23–29.

[26] T. K. Phan, D. Griffin, E. Maini, and M. Rio, "Utility-maximizing server selection," in *Proc. IFIP Netw.*, Vienna, Austria, 2016, pp. 413–421.

[27] T. K. Phan, D. Griffin, E. Maini, and M. Rio, "Utility-centric networking: Balancing transit costs with quality of experience," *IEEE/ACM Trans. Netw.*, to be published.

[28] W. Racheg, N. Ghrada, and M. F. Zhani, "Profit-driven resource provisioning in NFV-based environments," in *Proc. IEEE ICC*, Paris, France, 2017, pp. 1–7.

[29] A. Sharma *et al.*, "A global name service for a highly mobile internetwork," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, 2014, pp. 247–258.

[30] M. A. Stone and B. C. Moore, "Tolerable hearing aid delays. I. Estimation of limits imposed by the auditory path alone using simulated hearing losses," *Ear Hear.*, vol. 20, no. 3, pp. 182–192, 1999.

[31] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized server selection for cloud services," in *Proc. ACM SIGCOMM*, New Delhi, India, 2010, pp. 231–242.

[32] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007, Art. no. 6.

[33] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality driven Web services composition," in *Proc. Int. Conf. World Wide Web*, Budapest, Hungary, 2003, pp. 411–421.

[34] Z. Zhang *et al.*, "Optimizing cost and performance online service provider networks," in *Proc. USENIX NSDI*, 2010, pp. 1–15.

Truong Khoa Phan received the M.Sc. and Ph.D. degrees from INRIA/I3S, Sophia, France, in 2011 and 2014, respectively. He is currently a Senior Research Associate with the Department of Electronic and Electrical Engineering, University College London, U.K. His research interests include network optimization, cloud computing, software-defined networks, network function virtualization, P4, multicast, and P2P.

Miguel Rocha is an Associate Professor with the Department of Informatics, University of Minho, where he is also a Senior Researcher with the Centre of Biological Engineering, and the Director of the Master in Bioinformatics. His main research interests lie in the fields of bioinformatics, systems biology, and data mining, areas where he applies methods such as evolutionary algorithms, neural networks and other machine learning models to problems in biological data analysis, metabolic engineering, and the optimization of computer networks.

David Griffin received the B.Sc. degree in electronic and electrical engineering from Loughborough University and the Ph.D. degree in electronic and electrical engineering from University College London. He is a Principal Research Associate with the Department of Electronic and Electrical Engineering, University College London. His research interests include planning, management and dynamic control for providing QoS in multiservice networks and novel routing paradigms for the future Internet.

Miguel Rio is a Professor with the Department of Electronic and Electrical Engineering, University College London, where he researches and lectures on Internet technologies. His research interests include on real-time overlay streaming, network support for interactive applications, quality of service routing, and network monitoring and measurement.