CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Ghent University Academic Bibliography

Network-aware Service Placement and Selection Algorithms on Large-scale Overlay Networks

J. Famaey[a,*], T. Wauters[a], F. De Turck[a], B. Dhoedt[a], P. Demeester[a]

[a] *Department of Information Technology (INTEC), Ghent University – IBBT, Gaston Crommenlaan 8, bus 201, B-9050 Gent, Belgium*

Abstract

Currently many service providers offer their services on a private and proprietary hard- and software infrastructure. These infrastructures often share many similarities. Hence we believe a generic service management architecture, that allows service providers to offer a large array of different services on a single infrastructure or multiple providers to offer their services cooperatively, would provide many advantages over current silo-based approaches. Additionally, by allowing the distributed service management components to cooperate in a peer-to-peer overlay network, scalability and resilience of the system could be greatly improved.

In this paper we propose an optimal algorithm, based on an Integer Linear Programming (ILP) formulation, and several heuristics to support such a generic overlay-based service management architecture. More specifically, we propose algorithms for dynamically allocating server and network resources to a set of services and selecting a suitable service instance for each client. Service instances are placed on a set of servers, taking into account server resource constraints (e.g. CPU and memory). Unlike existing algorithms for this problem, those proposed in this paper also support Service Level Agreements (SLAs), which take the form of Quality of Service demands such as transmission latency constraints and bandwidth requirements. The optimization goal is to maximise the percentage of satisfied demand (answered requests) and minimise the total number of required

*Corresponding author; Tel: +32 9 331 49 38; Fax: +32 9 331 48 99
Email addresses: jeroen.famaey@intec.ugent.be (J. Famaey), tim.wauters@intec.ugent.be (T. Wauters), filip.deturck@intec.ugent.be (F. De Turck), bart.dhoedt@intec.ugent.be (B. Dhoedt), piet.demeester@intec.ugent.be (P. Demeester)

overlay servers, while satisfying the SLAs and resource constraints. Additionally, we propose an extension that allows the algorithms to find overlay routing paths to improve the transmission latency for latency-sensitive services.

Extensive simulations were performed to evaluate the performance and scalability of the heuristics. They showed that in many cases the heuristics perform close to optimal and they scale well in terms of network size.
Key words: service management, service placement, server selection, overlay routing

---

1. Introduction

The exponential growth of the Internet allows an ever-growing number of service providers to reach more and more users. Currently, many service providers offer their services on a private and proprietary infrastructure, designed specifically for a single service. Nevertheless, many components of such a service management infrastructure could be reused for offering a wide range of different services. By designing generic service management components, service providers could focus on designing the actual service logic, instead of the management infrastructure.

A generic service management infrastructure could be used by a single provider to offer a range of different services. Alternatively, many providers could collaborate to offer their services on a shared, global infrastructure. This provides additional advantages, such as: a larger shared resource pool, longer reach across the Internet, and composition of services of different providers (i.e. workflows [1]).

Recently, peer-to-peer overlay networks have been widely proposed in the context of service management infrastructures [2, 3]. Using such peer-to-peer principles provides several advantages, such as improved scalability, resilience towards failures and support for overlay routing.

In this paper, an overlay-based service management architecture is proposed. The assumed deployment scenario is shown in Fig. 1. It is assumed that the infrastructure consists of several datacenters spread across the Internet. In

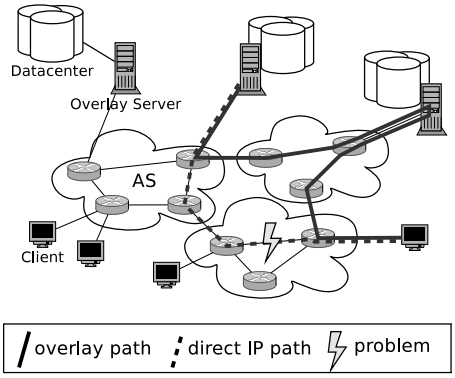


Figure 1: The assumed deployment scenario for a generic service management infrastructure

front of each datacenter are one or more overlay servers. These servers are responsible for managing the resources of the datacenter, and performing service management tasks. Additionally, they are connected together in an overlay network, which can be used for disseminating information or routing service requests. The figure also shows how overlay routing via an intermediate overlay server can be used to circumvent problems in the underlying physical network (e.g. bandwidth bottlenecks, failures, or high-latency links).

In addition to the generic service management architecture itself, some of its required algorithmic components are also studied in this paper. More specifically, an Integer Linear Programming (ILP) based optimal algorithm and several heuristics are proposed for performing service placement and server selection. The proposed algorithms take into account server resources, such as CPU and memory. In contrast to many existing service placement algorithms, they also take into account Service Level Agreements (SLAs) in the form of bandwidth requirements and transmission latency constraints. Additionally, an extension to the algorithms is proposed that allows them to find overlay routes between the overlay servers and clients, to improve the transmission latency of requests.

The algorithms presented in this paper improve upon our previous work [4]. First, the algorithms have been adapted in order to optimise performance on the level of client-service connections, instead of single request messages. This greatly improves scalability and real-world applicability. Second, an extension to the heuristics is presented. It allows them to find the lowest latency path,

while satisfying bandwidth requirements. Third, the devised Integer Linear Programming (ILP) formulation of the service placement problem is presented in full. Finally, more thorough simulation results are presented. This allows us to better evaluate and compare the performance and behaviour of the presented algorithms.

The rest of this paper is structured as follows. Related work is discussed in Section 2. The components of which the service management architecture is composed are further discussed in Section 3. Section 4 gives a formal description of the service management network model and the service placement and server selection problems. Section 5 gives an in-depth overview of the designed algorithms. In Section 6, overlay-aware versions of the algorithms are discussed. The algorithms are evaluated based on simulation results for different scenarios in Section 7. And finally, conclusions are drawn in Section 8.

2. Related Work

In the past, variants of the service placement problem have been solved by translating them into *facility location* [5] and *knapsack* [6] problems. The facility location problem is defined as selecting a subset of available locations for the placement of facilities, minimising the total transportation and placement costs. In the context of service placement, a facility represents a service instance, while a facility location represents a server. The knapsack problem can be described as follows. Given a set of items with weight and profit, determine the number of each item-type to place in a container, maximising total profit. In this case, the items represent services, while knapsacks represent servers. The general form of both problems is NP-hard.

Facility location has been a point of interest in research for many years. As a consequence, many sub-optimal heuristics, that decrease complexity, have been proposed. They solve the problem using a wide plethora of different techniques, such as: linear program relaxation [7, 8], local search [9, 10, 11, 12], dynamic programming [13], greedy principles [14], and many others [15, 12, 16, 17]. Most related to the problem presented in this paper is the capacitated facility location problem (CFLP) [10, 11]. The CFLP variant introduces an additional

constraint that stipulates the maximum number of clients that can be handled by each facility. This can be mapped to the server-resource constraints in the service placement problem. However, these heuristics often assume the distance measure satisfies the triangle inequality, while it has been shown that Internet transmission latency does not [18]. Additionally, the facility location problems are formulated for placing several copies of a single facility. On the other hand, the algorithms proposed in this paper place one or more instances of several different service types. This cannot be easily generalised, as the CFLP assumes a capacity per facility instance, while we assume the capacity is shared among all facility (i.e. service) instances placed in the same location (i.e. server). As a consequence, it is not possible to merely execute the CFLP algorithm for all facility types sequentially.

The knapsack problem has also been a topic of thorough study [6, 19]. The variant most related to the problem presented in this paper is the class-constrained multiple-knapsack problem (CMKP) [20, 21]. In CMKP, items of multiple types need to be placed in multiple knapsacks, each with limited capacity. This can be easily mapped to the different service types that need to be placed on multiple servers, with limited resources. The classic CMKP has been successfully adapted to the service placement problem [22, 23]. Karve et al. designed a centralized application placement middleware [22]. Their heuristic maximizes satisfied demand and minimizes total number of placement changes compared to a previous placement scheme. However, only server resource constraints are taken into account. Adam et al. argued that any centralized service placement algorithm has a limited scalability and created a decentralized variant of [22]. In [24, 25], they propose an application placement middleware that constructs overlays, places services, selects service instances for clients and routes the client requests to the correct instance. Their service placement heuristic maximizes satisfied demand, also taking into account only CPU and memory constraints. In [3], the scalability of their previous design is improved. Here every node requires knowledge on only a limit number of other nodes, instead of global knowledge on all nodes in the network. Although these algorithms take into account several server resources, none of them consider network related resources and limitations. We argue that in this age, where many applications require large

amounts of bandwidth (e.g. video streaming, IPTV) and guarantees concerning transmission latency (e.g. online gaming, video/audio conferencing) these are just as important as server resources. As a consequence, our algorithms also consider bandwidth and transmission latency.

No one-to-one mapping is possible between the facility location or knapsack problems and the problem solved in this paper. The service placement algorithms presented in this work take into account both restrictions in the network and on the servers. Additionally, they attempt to place one or more instances of multiple services on multiple servers. As a consequence, the problem can be translated into a combination of the facility location and knapsack problems. Existing algorithms can thus not be directly adapted.

In [26], an algorithm is proposed for placing service components that cooperate in a service composition. The algorithm minimises traffic between service components and variance in processing power used on each server. However, it is assumed only a single instance of each service component should be placed in the network.

Service Level Agreements (SLAs) provide a formal mechanism for defining contracts between service providers and consumers [27]. Research on SLAs has focused on negotiation protocols [28], translation into device configurations [29], and monitoring of compliance [30]. SLAs have also been integrated into service management architectures, for instance [31, 2]. However, this architecture only considers the migration of existing service instances between servers, and not the problem of deciding how many instances of each service to place. Additionally, the server selection and request routing problems are not tackled.

3. Service Management Architecture

The components of the considered generic overlay-based self-managing service management architecture are shown in Fig. 2. It consists of 4 major components, which operate on top of the transport layer. The *resource monitoring* component is responsible for monitoring server and network resource usage. The *peer-to-peer overlay* maintains a virtual peer-to-peer network topology, which

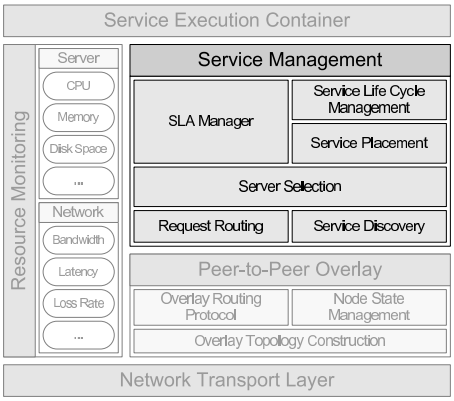


Figure 2: The functional components of our proposed self-managing generic service management infrastructure

can then be used by the service management components. These *service management* components are responsible for allocating resources to services, monitoring for SLA violations, discovering service instance locations, server selection and routing requests. Finally, the *service execution container* executes service instances on every server, as decided by the service placement algorithms. All these components can either be centralised, or, to improve scalability and resilience, distributed across a set of overlay servers. In this last case, decentralised algorithms are needed for all management components. This paper focusses on the algorithms for the service management components. Therefore, these components are discussed in more detail in the rest of this section. For clarity and brevity, a detailed description of the other components is omitted.

The *service placement component* is responsible for allocating server and network resources to every service. Because of resource constraints it is usually impossible to run every service on every server. Therefore it must be decided where to place instances of which services, and how many instances to place. Closely related, is the *server selection component*. As multiple instances of a service might be run across the network, it must be decided which instance to use for which clients. The decision of both the service placement and server selection algorithms is based on the available resources and SLAs, which they monitor via the resource monitor and SLA manager. They should make sure server and network resources do not become overloaded, and no SLA violations occur. Algorithms that solve the service placement and service selection

problems are further discussed in Section 5.

In a large-scale system, a scalable mechanism is needed for discovering the location of service instances, before server selection can take place. Several solutions, which operate on top of the overlay topology, exist for this *service discovery* problem. For example, a Distributed Hash Table (DHT) [32] or a flooding-based approach [3] could be used.

Finally, client *request routing* can either be done directly via the underlying IP network, or via the overlay network, using an overlay routing protocol. The algorithms for finding suitable overlay routing paths are discussed in Section 6.

4. Problem Formulation

Informally, the *service placement* and *server selection* problems can be described as follows. The goal of service placement is to decide which services to execute on which servers, without violating any server resource constraints. The objective is to use as few overlay servers as possible. The goal of server selection is to decide which instance of a service to use for the requests of a certain client, without violating any server or network resource constraints. The objective of this problem, is to satisfy as much client demand as possible. In the rest of this section, a more formal description of the problems is given.

4.1. Network Model

The overlay network consists of a set of overlay servers $\mathcal{O}$ and a set of clients $\mathcal{C}$. Together these form the set of overlay nodes $\mathcal{N} = \mathcal{O} \cup \mathcal{C}$. Every node $n \in \mathcal{N}$ has a maximum incoming and outgoing bandwidth limit, respectively B_n^{in} and B_n^{out}, which represent the available access-link bandwidth. The link between every pair of nodes $m, n \in \mathcal{N}$ has a transmission latency $\Delta_{m,n}$. Every overlay server o has a memory capacity Γ_o and CPU capacity Ω_o.

Additionally, a set of services $\mathcal{S}$ is defined. Every service $s \in \mathcal{S}$ has several server resource requirements and an associated SLA. The required server resources consist of memory per instance γ_s^{in}, memory per client γ_s^{cl}, and CPU per client ω_s. The SLA consists of the required request bandwidth β_s^{req} (on the path from the client to the server), reply bandwidth β_s^{rep} (from server to client),

and maximum round trip time (RTT) δ_s. This is the maximum transmission latency from the client to the server and back. It is used for modelling time critical services, such as online gaming or VoIP. Finally, every service has a priority π_s, which indicates its importance.

Every client c has a set of services $\mathcal{S}_c$, which it wants to use. The set $\mathcal{C}_s$ contains all clients that want to use service s, thus

$$c \in \mathcal{C}_s \Leftrightarrow s \in \mathcal{S}_c \tag{1}$$

Finally, as some services have certain soft- or hardware requirements, we use the boolean variable $R_{o,s}$ to indicate if overlay server o is capable of running service s.

4.2. Formal Problem Formulation

In this section, a formal problem formulation of the service placement and server selection problems is given. It is structured in the form of an Integer Linear Programming (ILP) formulation. An ILP formulation consists of decision variables, constraints and an objective function. The decision variables represent the solution space of the problem. The constraints make sure the values of the decision variables are valid for the given problem and the objective function is used to pick the optimal values of the decision variables. In the rest of this section, the decision variables, constraints and objective function for the service placement and server selection problem are explained.

Decision Variables. The formulation consists of three types of decision variables

- $S_{o,s} \in [0, 1]$ equals 1 if overlay server o executes service s
- $U_o \in [0, 1]$ equals 1 if overlay server o is used to execute services
- $O_{c,o,s} \in [0, 1]$ equals 1 if overlay server o answers the requests of client c for service s

Constraints. An overlay server can only run services if it satisfies their requirements

$$\forall o \in \mathcal{O}, \forall s \in \mathcal{S}: \quad S_{o,s} \quad \leq R_{o,s} \tag{2}$$

If an overlay server hosts any services, it is used

$$\forall o \in \mathcal{O}: \quad \sum_{s \in \mathcal{S}} S_{o,s} \quad \leq |\mathcal{S}| \times U_o \tag{3}$$

An overlay server can only answer requests of services it executes and if the client wants to use that service

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}: \quad O_{c,o,s} \quad \leq S_{o,s} \tag{4}$$

$$\forall c \in \mathcal{C}, s \in \mathcal{S}: \quad \sum_{o \in \mathcal{O}} O_{c,o,s} \quad \leq \sigma_{c,s} \tag{5}$$

With $\sigma_{c,s} = 1$ if $s \in \mathcal{S}_c$, and $\sigma_{c,s} = 0$ otherwise.

The amount of used resources cannot exceed available resources

$$\forall o \in \mathcal{O}: \quad \sum_{s \in \mathcal{S}} \gamma_s^{in} \times S_{o,s} + \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \gamma_s^{cl} \times O_{c,o,s} \quad \leq \Gamma_o \tag{6}$$

$$\forall o \in \mathcal{O}: \quad \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \omega_s \times O_{c,o,s} \quad \leq \Omega_o \tag{7}$$

$$\forall c \in \mathcal{C}: \quad \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} \beta_s^{req} \times O_{c,o,s} \quad \leq B_c^{out} \tag{8}$$

$$\forall c \in \mathcal{C}: \quad \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} \beta_s^{rep} \times O_{c,o,s} \quad \leq B_c^{in} \tag{9}$$

$$\forall o \in \mathcal{O}: \quad \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \beta_s^{req} \times O_{c,o,s} \quad \leq B_o^{in} \tag{10}$$

$$\forall o \in \mathcal{O}: \quad \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \beta_s^{rep} \times O_{c,o,s} \quad \leq B_o^{out} \tag{11}$$

The transmission latency from the client to the server and back, cannot exceed the maximum RTT of the service

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}: \quad (\Delta_{c,o} + \Delta_{o,c}) \times O_{c,o,s} \quad \leq \delta_s \tag{12}$$

Objective Function. The objective function maximises the satisfied demand and minimises the number of used overlay servers

$$\max \left(\alpha \times \frac{\sum_{c \in \mathcal{C}} \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} \pi_s \times O_{c,o,s}}{\sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}_c} \pi_s} - (1 - \alpha) \times \frac{\sum_{o \in \mathcal{O}} U_o}{|\mathcal{O}|} \right) \tag{13}$$

The parameter α represents a weight factor, which can be used to change the relative importance of both goals. Both satisfied demand and number of used overlay servers are normalised. The number of answered requests is multiplied by the priority of the service, which effectively makes the objective function prioritise more important services.

5. Service Management Algorithms

In this section, several algorithms that solve the *service placement* and *server selection* problems are discussed. Service placement and server selection are closely intertwined. A different placement for example, will influence to optimal solution of the selection problem. Therefore, to optimise both objectives as much as possible, our proposed algorithms solve both problems simultaneously.

As previously stated, this problem can be translated into a combination of the capacitated facility location and class-constrained multiple-knapsack problems. Both these problems are NP-hard [5, 6]. As a consequence, the problem solved here is NP-hard as well. Therefore, we propose not only an optimal algorithm, but also several sub-optimal heuristics, with a polynomial time complexity.

5.1. ILP Algorithm (ILP)

The ILP algorithm finds the optimal solution to both the placement and selection problem. It is based on the formal problem formulation given in Section 4 and solved using the ILOG CPLEX [33] software package, with the simplex and interior point methods [34].

5.2. Greedy Service Placement Heuristic (GSP)

Although the ILP algorithm returns the optimal solution to the problem, it has a very high time complexity and is therefore unusable for larger scenarios. As an alternative, a suboptimal heuristic was devised with a lower time complexity.

As its name implies, this heuristic is based on the greedy principles. It is greedy in the way that it will separately place instances of each service, without taking into account services that have not been placed yet (it does however take into account resources used by already placed services). Pseudo-code for the algorithm is shown in Algorithm 1.

The algorithm starts by initialising the required variables (lines 1–11). The variables S_o and $C_{o,s}$ are used by the algorithm to store the obtained solution. The set S_o will contain all services executed by overlay server o, while $C_{o,s}$ will contain all clients that use the instance of s on o to satisfy their demand. The

Algorithm 1 Pseudo-code of the greedy service placement heuristic

function gsp()

1: **for all** $o \in \mathcal{O}$ **do**
2: $S_o \leftarrow \{\}$
3: $\gamma_o^{avl} \leftarrow \gamma_o$
4: $\omega_o^{avl} \leftarrow \omega_o$
5: $B_o^{in,avl} \leftarrow B_o^{in}$
6: $B_o^{out,avl} \leftarrow B_o^{in}$
7: **for all** $s \in \mathcal{S}$ **do**
8: $C_{o,s} \leftarrow \{\}$
9: **for all** $c \in \mathcal{C}$ **do**
10: $B_c^{in,avl} \leftarrow B_c^{in}$
11: $B_c^{out,avl} \leftarrow B_c^{in}$
12: sort($\mathcal{S}$) //by decreasing prioritised requests
13: **for all** $s \in \mathcal{S}$ **do**
14: **for all** $c \in \mathcal{C}_s$ **do**
15: $U_c \leftarrow \{\}$
16: **for all** $o \in \mathcal{O}$ **do**
17: $u_o \leftarrow \sum_{s \in \mathcal{S}} |C_{o,s}|$
18: **if** $B_c^{in,avl} \geq \beta_s^{rep}$ **and** $B_c^{out,avl} \geq \beta_s^{req}$ **then**
19: **for all** $o \in \mathcal{O}$ **do**
20: **if** $(s \in S_o$ **or** $\gamma_o^{avl} \geq \gamma_s^{in} + \gamma_s^{cl})$ **and** $\gamma_o^{avl} \geq \gamma_s^{cl}$ **and** $\omega_o^{avl} \geq \omega_s$ **and** $B_o^{in,avl} \geq \beta_s^{req}$ **and** $B_o^{out,avl} \geq \beta_s^{rep}$ **and** $R_{o,s} =$ **true and** $\Delta_{c,o} + \Delta_{o,c} \leq \delta_s$ **then**
21: $U_c \leftarrow U_c \cup \{o\}$
22: $u_o \leftarrow u_o + 1$
23: sort($\mathcal{O}$) //by decreasing u_o
24: sort($\mathcal{C}$) //by increasing $|U_c|$
25: **for all** $o \in \mathcal{O}$ **do**
26: **for all** $c \in \mathcal{C}$ **do**
27: **if** $B_c^{in,avl} \geq \beta_s^{rep}$ **and** $B_c^{out,avl} \geq \beta_s^{req}$ **then**
28: **if** $(s \in S_o$ **or** $\gamma_o^{avl} \geq \gamma_s^{in} + \gamma_s^{cl})$ **and** $\gamma_o^{avl} \geq \gamma_s^{cl}$ **and** $\omega_o^{avl} \geq \omega_s$ **and** $B_o^{in,avl} \geq \beta_s^{req}$ **and** $B_o^{out,avl} \geq \beta_s^{rep}$ **then**
29: **if** $s \notin S_o$ **then**
30: $\gamma_o^{avl} \leftarrow \gamma_o^{avl} - \gamma in_s$
31: $S_o \leftarrow S_o \cup \{s\}$
32: $C_{o,s} \leftarrow C_{o,s} \cup \{c\}$
33: $B_c^{in,avl} \leftarrow B_c^{in,avl} - \beta_s^{rep}$
34: $B_c^{out,avl} \leftarrow B_c^{out,avl} - \beta_s^{req}$
35: $\gamma_o^{avl} \leftarrow \gamma_o^{avl} - \gamma_s^{cl}$
36: $\omega_o^{avl} \leftarrow \omega_o^{avl} - \omega_s$
37: $B_o^{in,avl} \leftarrow B_o^{in,avl} - \beta_s^{req}$
38: $B_o^{out,avl} \leftarrow B_o^{out,avl} - \beta_s^{rep}$
39: **return** $\{S_o\}_{\forall o \in \mathcal{O}}, \{C_{o,s}\}_{\forall o \in \mathcal{O}, s \in \mathcal{S}}$

other variables represent the remaining amount of resources of each type for the clients and overlay servers (i.e. CPU, memory and bandwidth). Subsequently,

the services are sorted from most to least prioritised request count (line 12). The prioritised request count equals the number of clients that want to use the service multiplied by its priority.

The algorithm iterates over all services to decide where to place an instance, and which instance to use for every client (lines 13–38). First, the set of overlay servers that can be used by the clients of the service is determined (lines 14–22). These overlay servers are called the candidate servers, and the set is denoted by U_c. The variable u_o equals the number of clients that have server o as a candidate plus the number of clients that have already been assigned to it for other services. These two variables are initialised on lines 15–17. Then, the algorithm checks if the client has enough bandwidth available to use the service (line 18). Finally, all overlay servers that have enough resources, and for which the transmission latency to the client is within the delay bound of the service, are added to U_c (lines 19–22).

Then, the candidate sets are used to determine the actual service placement and server selection schemes (lines 23–38). First, the set of overlay servers is sorted from most to least clients that use it, or can use it (line 23). This allows the algorithm to minimise the number of used servers. Second, the clients are sorted from least to most candidates (line 24). This makes sure that clients with only few candidates also have a chance to select a server. In the third step, the actual placement and selection take place (lines 25–38).

It can be easily seen that the worst-case time complexity of the greedy service placement heuristic is $O\left(|\mathcal{S}| \times |\mathcal{C}| \times |\mathcal{O}|\right)$.

5.3. Overlay-Subset Service Placement Heuristic (OSP)

The greedy service placement heuristic requires interaction between the different clients and overlay servers to perform service placement and server selection. Before the placement of a service is actually performed, it must first iterate over all clients of that service and check which overlay servers are possible candidates for that client. Although this is not a problem in a centralised architecture, where information on all servers and clients is available, it is difficult to translate this to a decentralised environment. To leverage this, a second heuristic

Algorithm 2 Pseudo-code of the overlay-subset service placement heuristic

function osp($N \in [1..|\mathcal{O}|]$)

1: **for all** $o \in \mathcal{O}$ **do**
2: $S_o \leftarrow \{\}$
3: $u_o \leftarrow 0$
4: $\gamma_o^{avl} \leftarrow \gamma_o$
5: $\omega_o^{avl} \leftarrow \omega_o$
6: $B_o^{in,avl} \leftarrow B_o^{in}$
7: $B_o^{out,avl} \leftarrow B_o^{in}$
8: **for all** $s \in \mathcal{S}$ **do**
9: $C_{o,s} \leftarrow \{\}$
10: **for all** $c \in \mathcal{C}$ **do**
11: $B_c^{in,avl} \leftarrow B_c^{in}$
12: $B_c^{out,avl} \leftarrow B_c^{in}$
13: sort($\mathcal{S}$) //by decreasing prioritised requests
14: **for all** $s \in \mathcal{S}$ **do**
15: **for all** $c \in \mathcal{C}_s$ **do**
16: **if** $B_c^{in,avl} \geq \beta_s^{rep}$ **and** $B_c^{out,avl} \geq \beta_s^{req}$ **then**
17: $\mathcal{O}_c \leftarrow$ getNearestServers(c, N)
18: sort($\mathcal{O}_c$) //by decreasing u_o and first if $s \in S_o$
19: **for all** $o \in \mathcal{O}_c$ **do**
20: **if** ($s \in S_o$ **or** $\gamma_o^{avl} \geq \gamma_s^{in} + \gamma_s^{cl}$) **and** $\gamma_o^{avl} \geq \gamma_s^{cl}$ **and** $\omega_o^{avl} \geq \omega_s$ **and** $B_o^{in,avl} \geq \beta_s^{req}$ **and** $B_o^{out,avl} \geq \beta_s^{rep}$ **and** $R_{o,s} =$ **true and** $\Delta_{c,o} + \Delta_{o,c} \leq \delta_s$ **then**
21: **if** $s \notin S_o$ **then**
22: $\gamma_o^{avl} \leftarrow \gamma_o^{avl} - \gamma in_s$
23: $S_o \leftarrow S_o \cup \{s\}$
24: $C_{o,s} \leftarrow C_{o,s} \cup \{c\}$
25: $u_o \leftarrow u_o + 1$
26: $B_c^{in,avl} \leftarrow B_c^{in,avl} - \beta_s^{rep}$
27: $B_c^{out,avl} \leftarrow B_c^{out,avl} - \beta_s^{req}$
28: $\gamma_o^{avl} \leftarrow \gamma_o^{avl} - \gamma_s^{cl}$
29: $\omega_o^{avl} \leftarrow \omega_o^{avl} - \omega_s$
30: $B_o^{in,avl} \leftarrow B_o^{in,avl} - \beta_s^{req}$
31: $B_o^{out,avl} \leftarrow B_o^{out,avl} - \beta_s^{rep}$
32: **return** $\{S_o\}_{\forall o \in \mathcal{O}}, \{C_{o,s}\}_{\forall o \in \mathcal{O}, s \in \mathcal{S}}$

was devised. The overlay-subset service placement heuristic does not require this interaction and uses only the information available from past placement decisions. Additionally, a client will only consider the N nearest overlay servers as possible candidates, which greatly improves the scalability and reduces the information that is needed on every client. Pseudo-code for this algorithm is shown in Algorithm 2.

First, as for GSP, the solution variables are initialised (lines 1–12) and the list

of services is sorted by decreasing prioritised request count (line 13). Then, the algorithm iterates over all services (lines 14–31).

The algorithm will attempt to find a server for every client that uses the service (lines 15–31). If the client has enough available incoming and outgoing bandwidth for the service (line 16), the algorithm will retrieve the N overlay servers with the lowest transmission delay from the client c (line 17) and sort them from most to least clients already using the server (line 18). Additionally, to preserve as much memory as possible, servers that already run an instance of s are prioritised. This prevents the algorithm from placing too many instances of a single service. Finally, the first of the N servers that has enough resources and a low enough transmission latency (line 20), will be used by the client (lines 21–31).

The worst-case time complexity of this algorithm is $O\left(|\mathcal{S}| \times |\mathcal{C}_s^m| \times N \times \log N\right)$, with $\mathcal{C}_s^m = \max_{s \in \mathcal{S}} \mathcal{C}_s$. Therefore, if N is small compared to $\mathcal{O}$ (e.g. $N < 30$ if $\mathcal{O} = 100$ or $N < 190$ for $\mathcal{O} = 1000$) and the number of clients per service $\mathcal{C}_s$ is smaller than $\mathcal{C}$, then the time complexity of OSP is less than that of GSP.

6. Overlay-Aware Algorithms

The algorithms described in Section 5 route service requests and replies directly from the client to their selected overlay server and back. Nevertheless, routing via the overlay network could improve the transmission latency and thus the perceived Quality of Experience (QoE) significantly. Fig. 3 shows an example of how overlay routing could improve the latency between client c and overlay server o. This is the case because the triangle inequality does not hold for transmission latencies over the Internet [18]. However, routing via the overlay network comes at the cost of additional bandwidth consumption, as all the requests and replies will now need to pass through additional overlay servers.

In the rest of this section, the changes needed for the algorithms to support overlay routing are discussed. The adapted algorithms will not only select an overlay server for the service requests of the clients, but will also calculate a suitable path through the overlay network.

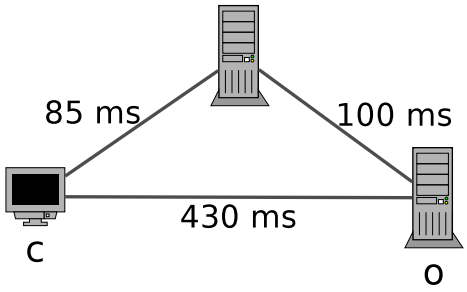


Figure 3: An example where routing via an additional overlay server reduces the transmission latency (from 430 to 185 ms)

6.1. ILP Formulation

Several changes are needed in the ILP formulation to support overlay routing. These are discussed in the rest of this section.

Decision Variables. An additional decision variable is needed to support the notion of overlay paths. It is assumed the path from the client to the overlay server is equal to the path back. Therefore, only 1 extra decision variable is needed

- $P_{c,o,s,m,n} \in [0,1]$ equals 1 if the overlay link between nodes m and n is used in the path between client c and overlay server o for sending requests and replies of service s

Constraints. In the original algorithm, an overlay server was activated if it ran any services. When using overlay routing, servers that are used for routing should also be considered active. Therefore, an additional constraint is needed

$$\forall o \in \mathcal{O}: \quad \sum_{c\in\mathcal{C}} \sum_{p\in\mathcal{O}} \sum_{s\in\mathcal{S}} \sum_{n\in\mathcal{N}} P_{c,p,s,o,n} + P_{c,p,s,n,o} \quad \leq \nu \times U_o \tag{14}$$

With $\nu = 2 \times |\mathcal{C}| \times |\mathcal{O}| \times |\mathcal{S}| \times |\mathcal{N}|$.

Constraints 8, 9, 10 and 11 are no longer valid, as now bandwidth needs to be reserved on the entire path between the client and the server. The following 2 constraints replace the previous 4 bandwidth reservation constraints

$$\forall m \in \mathcal{N}: \quad \sum_{c\in\mathcal{C}} \sum_{o\in\mathcal{O}} \sum_{s\in\mathcal{S}} \sum_{n\in\mathcal{N}} \beta_s^{req} \times P_{c,o,s,m,n} + \beta_s^{rep} \times P_{c,o,s,n,m} \quad \leq B_m^{out} \tag{15}$$

$$\forall m \in \mathcal{N}: \quad \sum_{c\in\mathcal{C}}\sum_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}}\sum_{n\in\mathcal{N}} \beta_s^{req} \times P_{c,o,s,n,m} + \beta_s^{rep} \times P_{c,o,s,m,n} \quad \leq B_m^{in} \quad (16)$$

Constraint 12, limiting the transmission latency between the client and server, must now be replaced by a constraint that takes into account the latency on the entire overlay path

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}: \quad \sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N}} (\Delta_{m,n} + \Delta_{n,m}) \times P_{c,o,s,m,n} \quad \leq \delta_s \quad (17)$$

Additionally, a number of new flow conservation constraints are needed. They make sure the P variables form a valid path. These are discussed in the rest of this section.

The number of incoming links must equal the number of outgoing links and there may be at most one of each (except for the client and the overlay server answering the requests)

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}, m \in \mathcal{O} \setminus \{o\}: \quad \sum_{n\in\mathcal{O}\cup\{c\}} P_{c,o,s,n,m} - P_{c,o,s,m,n} \quad = 0 \quad (18)$$

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}, m \in \mathcal{O} \setminus \{o\}: \quad \sum_{n\in\mathcal{O}\cup\{c\}} P_{c,o,s,n,m} + P_{c,o,s,m,n} \quad \leq 2 \quad (19)$$

The client must have one outgoing link part of the path, and the overlay server answering the requests must have one incoming link part of the path

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}: \quad O_{c,o,s} \leq \sum_{n\in\mathcal{N}} P_{c,o,s,c,n} \quad \leq 1 \quad (20)$$

$$\forall c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}: \quad O_{c,o,s} \leq \sum_{n\in\mathcal{N}} P_{c,o,s,n,o} \quad \leq 1 \quad (21)$$

The client cannot have any incoming links on the request path and the overlay server answering the request no outgoing links. Additionally, other clients cannot be part of its path

$$\sum_{c\in\mathcal{C}}\sum_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}}\sum_{n\in\mathcal{O}} P_{c,o,s,n,c} + P_{c,o,s,o,n} \quad = 0 \quad (22)$$

$$\sum_{c\in\mathcal{C}}\sum_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}}\sum_{m\in\mathcal{N}}\sum_{d\in\mathcal{C}\setminus\{c\}} P_{c,o,s,m,d} + P_{c,o,s,d,m} \quad = 0 \quad (23)$$

6.2. Heuristics

There are only minor changes needed for the heuristics to be able to support overlay routing. The idea is to only use overlay routing, if the latency on the underlay link directly between the client and the server is too high. For this, an additional function that finds an overlay path between the client and server with enough bandwidth and low enough latency is needed. The greedy service placement heuristic should then check if such a path exists at line 14 (only if the direct transmission latency is too high). The overlay-subset service placement heuristic should do this check at line 14 as well. Additionally, a third solution variable is needed that keeps track of these overlay paths.

Algorithm 3 shows pseudo-code for the algorithm that finds the shortest hop-count path between a client and its server. The algorithm makes sure that the path contains enough bandwidth for the given service and the transmission latency is low enough. It is based on Yen's "K shortest loopless paths" algorithm [35]. The extra K parameter gives an upper bound to the number of paths that can be tested before the algorithm returns. This parameter helps in limiting the complexity of the algorithm in very large networks. The algorithm selects the shortest hop-count path that satisfies the constraints, because every additional hop in the path incurs a bandwidth penalty on that node. To waste as little bandwidth for routing as possible, there should be as few hops in the path as possible .

First, the algorithm will check if the client and overlay server have enough bandwidth available for the service (lines 1–2). If this is not the case, it returns and no path is found. Then, the solution variables are initialised (lines 3–5). The variable p represents the current path, P represents the list of shortest paths, and C the list of candidate shortest paths. Finally, the algorithm will find the next shortest path, until one is found that satisfies the maximum delay bound of the service (lines 6–23).

The next shortest path is found as follows. In the first step, the algorithm iterates over i from 1 to the number of nodes in the current path p (lines 9–17). For every path $q \in P$, it will check if the path consisting of the first i nodes of p is equal to that of the first i nodes of q (lines 13–14). If this is the case, the

Algorithm 3 An adjusted version of Yen's algorithm to find an overlay path between the client and server with lowest hop-count, enough bandwidth to support the given service and a low enough transmission latency for the service

function findPath($c \in \mathcal{C}, o \in \mathcal{O}, s \in \mathcal{S}, K \in \mathbb{N}^*$)

1: **if** $B_c^{out,avl} < \beta_s^{req}$ **or** $B_c^{in,avl} < \beta_s^{rep}$ **or** $B_o^{in,avl} < \beta_s^{req}$ **or** $B_o^{out,avl} < \beta_s^{rep}$ **then**
2: **return null**
3: $p \leftarrow \{c, o\}$
4: $P \leftarrow \{\}$
5: $C \leftarrow \{\}$
6: **while** $K > 0$ **and** $\Delta_p > \delta_s$ **do**
7: $K \leftarrow K - 1$
8: $P \leftarrow P \cup \{p\}$
9: **for** $i = 1$ to $|p|$ **do**
10: $E \leftarrow \{\}$
11: **for** $j = 1$ to $|P|$ **do**
12: $q \leftarrow P_j$
13: **if** $p_{(1..i)} == q_{(1..i)}$ **then**
14: $E \leftarrow E \cup \{[p_i, p_{i+1}]\}$
15: $t \leftarrow p_{(1..i-1)} \cup$ getShortestPath$(p_i, o, E, p_{(1..i-1)})$
16: **if** $t \notin P$ **and** $t \notin C$ **then**
17: $C \leftarrow C \cup \{t\}$
18: **if** $|C| > 0$ **then**
19: $p \leftarrow$ getShortestCandidatePath(C)
20: $C \leftarrow C \setminus \{p\}$
21: $P \leftarrow P \cup \{p\}$
22: **else**
23: **return null**
24: **if** $\Delta_p \leq \delta_s$ **then**
25: **return** p
26: **else**
27: **return null**

edge consisting of the i^{th} and $(i+1)^{th}$ nodes of p is added to the list of excluded edges E (line 14). Then, a new path is constructed (line 15). It consists of the concatenation of the first $i-1$ nodes of p and the shortest hop-count path from the i^{th} node of p to o. This path cannot contain any of the first i nodes of p or any of the edges in E. Additionally, nodes that do not have enough incoming and outgoing bandwidth available to be used for routing requests and replies of service s are not used. The path can be calculated using an altered version of Dijkstra's shortest path algorithm [36], that does not use the forbidden edges and nodes. The path t is then added to C (lines 16–17). In the second step, p is set to the shortest hop-count path in C and the process is repeated.

The worst-case time complexity of Yen's original algorithm equals $O\left(K/2 \times |\mathcal{O}|^3\right)$, which is the same as that of our adjusted algorithm. Note, that in the case where $\Delta_{c,o} + \Delta_{o,c} \leq \delta_s$, the function returns the path $\{c, o\}$ (or **null**) and the complexity is reduced to $O(1)$.

7. Simulation Results

In this section, the performance and scalability of the heuristics is evaluated, based on simulation results. Performance is measured by comparing the solution to the optimal solution, calculated with the ILP algorithm. Scalability is evaluated by comparing results for different sized networks.

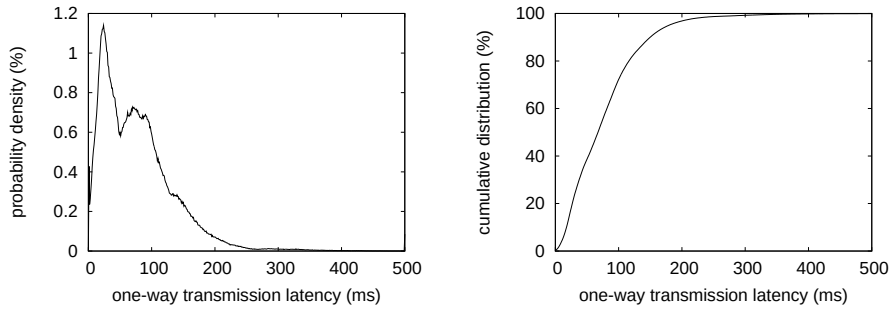
As a metric for solution quality, the *satisfied request count* and *used servers* are used. An server is used, if it runs any services or is part of any overlay routing path. The load on the network is expressed as the total number of requests. The error bars in the graphs represent the standard error of the mean over 10 (for ILP) or 30 (for the GSP and OSP) iterations.

7.1. Simulation Setup

The incoming and outgoing bandwidth of all clients and servers was set to 10 Mbps. Every server had a total of 1000 CPU cycles per second, and 4 GiB of memory. Heterogeneity was introduced by randomizing service requirements. Service maximum round trip times (RTT) were selected according to a random uniform distribution from the set $\{200, 500, 1000\}$ ms. Required CPU per client was chosen according to a random uniform distribution from the set $\{100, 200, 300, 400\}$ cycles per second. Required memory was chosen from the sets $\{256, 512, 1024\}$ MiB per instance and $\{32, 64, 128, 256\}$ MiB per client. Finally, request and reply bandwidth requirements were selected from the set $\{250, 500, 1000, 2000\}$ Mbps.

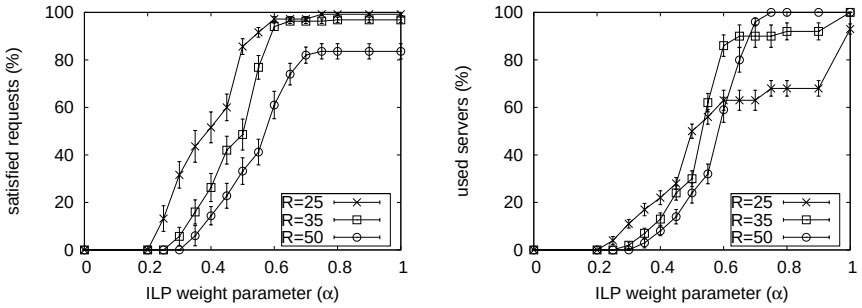
The transmission latency between every pair of nodes (clients and servers) was determined using a distribution function based on the King [1] dataset [38]. The probability density and cumulative distribution functions are shown in Fig. 4.

[1] `http://pdos.csail.mit.edu/p2psim/kingdata/`



(a) probability density function (b) cumulative distribution function

Figure 4: The one-way transmission latency distribution, based on the King dataset



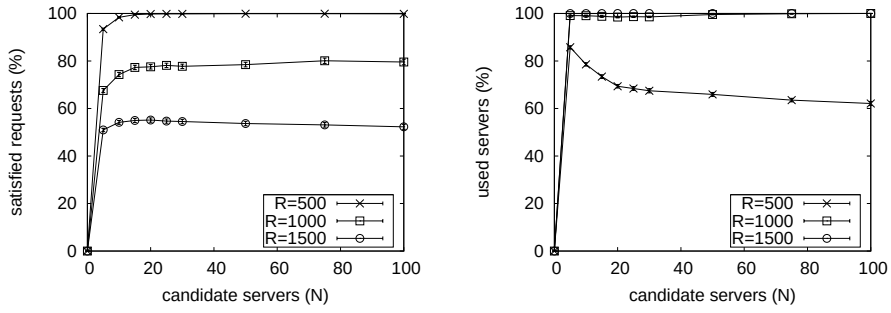
(a) satisfied requests (b) used servers

Figure 5: Satisfied request count and used servers as a function of the ILP weight parameter α and request count R; $C = 25$ represents a low-load scenario, $C = 35$ a slightly overloaded one, and $C = 50$ corresponds to a heavily overloaded scenario

7.2. Influence of ILP α Parameter

The goal of this scenario, is to study the effect of the weight parameter α on the solution of the ILP algorithm (ILP). The weight parameter provides a trade-off between maximising satisfied demand and minimising the number of used overlay servers. By varying α, its exact effects can be measured, and a good value for the parameter can be derived. As the ILP algorithm scales poorly, these simulations were performed on a small network topology, with 10 servers, 25, 35, and 50 clients, and 10 randomly generated service types. The simulation results are shown in Fig. 5.

As expected, α is directly proportional to the number of used servers and sat-



(a) satisfied requests (b) used servers

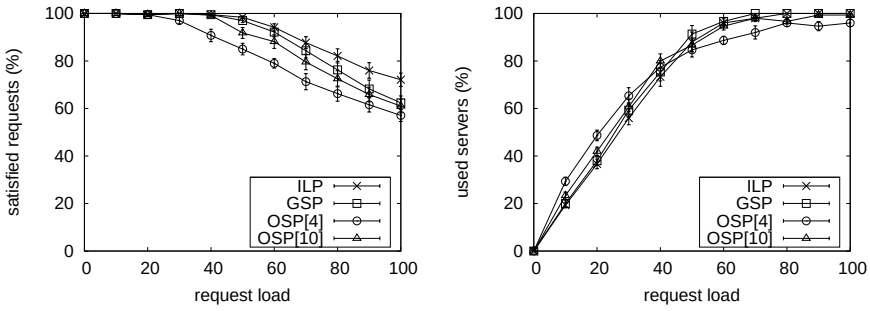
Figure 6: Satisfied requests and used servers as a function of the OSP N parameter and request count R

isfied requests. The results show that the network load (i.e. request count) influences the effect of varying α. For lower loads, increasing α starts affecting the evaluation metrics faster. In this paper, maximizing satisfied demand is deemed a more important goal than minimizing the number of used servers. Therefore, a higher value for α will be used. The graphs show that for $\alpha = 0.8$, satisfied demand is maximized for all network loads, while not more than necessary servers are used. This value was thus used in the other simulations.

7.3. Influence of OSP N Parameter

The overlay-subset service placement heuristic (OSP) N parameter denotes the maximum number of nearby servers that are considered as candidates by every client. It provides a trade-off between result quality and execution time. In this section, the effect of this parameter on the result quality is studied. The goal is to find out if a small value for N is still capable of providing high quality results. The network consists of 100 servers, 50 service types, and 500, 1000, and 1500 clients. The simulation results are shown in Fig. 6.

As expected, initially both satisfied requests and number of used servers increase as N increases. What is important, is that both stay about the same for all loads and $N \geq 20$. The drop in number of used overlay servers for $N \geq 6$ and $R = 500$ is a consequence of the fact that the algorithm can pack more clients on the same server if N increases. On the other hand, at higher loads ($cl \geq 60$) the number of used overlay servers increases for higher N. This is because if



(a) satisfied requests (b) used servers

Figure 7: Satisfied requests and used servers as a function of request load for different algorithms

the load becomes higher, clients can no longer be packed on a limited number of servers.

In conclusion, in the simulated scenarios OSP give good results for N as low as 10. This is only 10% of the total number of available servers.

7.4. Optimality of the Heuristics

In this scenario, the solution quality of the heuristics is compared to that of the optimal ILP algorithm. To allow the ILP algorithm to solve the scenario's within a reasonable time-frame, again a small network with 15 servers and 10 service types was used. The number of client requests was varied from 0 to 100. The value of the N parameter of the OSP algorithm, is supplied between square brackets. Thus, OSP[a] means OSP with $N = a$. The simulation results are shown in Fig. 7.

In terms of satisfied requests, the heuristics perform close to optimal if the load is low. Though, at higher loads ($R \geq 40$) their performance degenerates faster than that of the optimal ILP algorithm. For $N = 4$ OSP's performance in terms satisfied requests degenerates faster than for $N = 10$. Additionally, Fig. 7 shows that GSP performs slightly better than OSP[10] in terms of satisfied demand. In terms of used servers, there is no significant performance difference between the optimal ILP algorithm and the heuristics.

In summary, for high enough N, there is little difference in performance between GSP and OSP. Compared to the optimal solution, both heuristics perform well

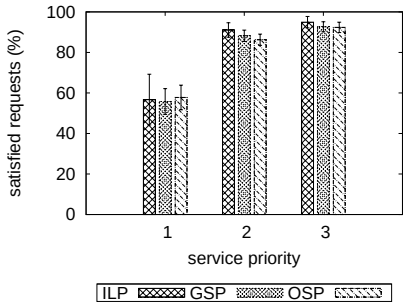


Figure 8: Satisfied requests per algorithm for the three priority classes

for both performance metrics. However, the ILP algorithm can handle slightly more requests. Additionally, at higher loads the performance of the heuristics degenerates faster in terms of satisfied demand. For 100 requests, ILP can handle, on average, 10% more requests than GSP.

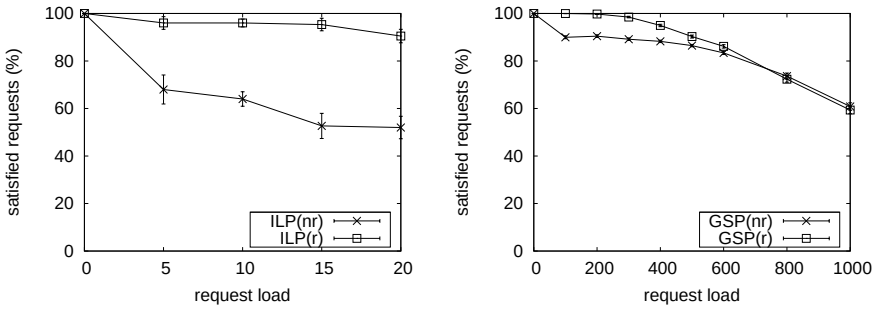
7.5. Prioritized Services

In this scenario, the effect of the priority variable π on the services is inspected. The simulation was performed on a small network with 10 overlay servers, 50 client requests, and 10 service types. This corresponds to an overloaded scenario. The priority of each service type was selected uniformly at random from the set $\{1, 2, 3\}$. Results are shown in Fig. 8

The figure shows the satisfied request percentage per priority type. It is clearly visible that each algorithm satisfies a larger demand for the higher priority classes. For example, ILP handles 95% of the requests with priority 3, while only satisfying 56% for class 1. This corresponds to the desired behaviour of the algorithms.

7.6. Overlay Routing

Routing service requests and replies via the overlay topology allows QoS constraints to be better satisfied. For example, it allows problems in the underlying network, such as bandwidth bottlenecks or high latency links, to be circumvented. In this section, the ability of the overlay routing components (as described in Section 6) to overcome high latency links is studied in more detail. For this purpose, a number of additional *routing servers* were introduced into



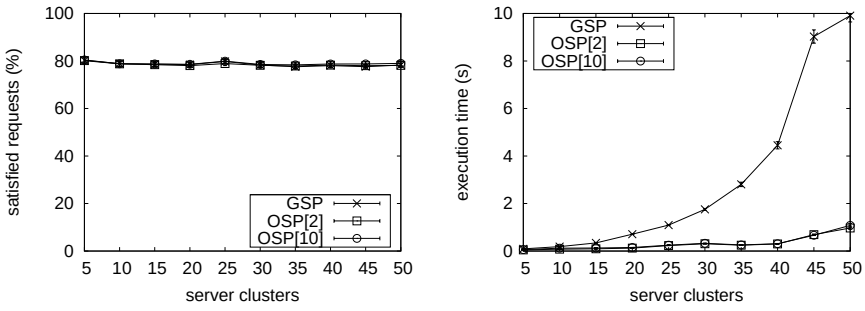
(a) results for ILP (b) results for GSP

Figure 9: Satisfied requests with and without overlay routing as a function of request load for different algorithms

the network. They act as normal overlay servers, but are not capable of processing service requests. Therefore, they only forward traffic to other servers or clients. Additionally, for this simulation, service RTTs were selected uniformly at random from the set $\{100, 150, 200\}$. The services in this scenario thus represent interactive multimedia applications with stringent latency demands (e.g. online gaming, or VoIP). The effect of overlay routing on the solution is studied by performing the simulation with and without overlay routing capable algorithms. It is expected that the overlay routing capable algorithm will be able to satisfy more demand, as they will be able to route around the high latency links to make sure the RTT constraints remain satisfied.

As the overlay routing variant of the ILP algorithm scales poorly, the simulations were performed on a small network for the ILP algorithm. It contained 5 servers, 2 routing servers, 5 service types, and up to 20 service requests. The heuristics were evaluated using a larger network with 15 clusters counting each 10 servers, 5 routing servers, 20 service types, and up to 1000 simultaneous service requests. The results are shown in Fig. 9. As the results for OSP show a similar trend to those of GSP, only GSP and ILP are shown.

The figure clearly shows that the satisfied demand greatly improves when overlay routing is used (denoted as GSP[r] and ILP[r]). This is because the algorithms are then able to route around the high latency links between the clients and a large portion of the overlay servers. Because of this, the transmission latency constraint of the lower latency services can also be satisfied and the number of



(a) satisfied requests (b) algorithm execution time

Figure 10: Satisfied requests and algorithm execution time as a function of server cluster count for different algorithms

satisfied requests increases. As shown in Fig. 9a using the routing component gives an increase in satisfied demand of up to 43% in the simulated scenario for the ILP algorithm. However, Fig. 9b shows that for the larger scenario the GSP overlay routing heuristic achieves only a smaller gain of up to 10%. Eventually, when the request load becomes very high, no more performance is gained from using overlay routing.

7.7. Scalability

The scalability of the heuristics is evaluated by varying the size of the network, while leaving the overall load constant. A heuristic is considered to be scalable if the quality of its solution does not degenerate as the network size grows. A network with 20 service types was used. The number of server clusters was varied from 0 up to 50, with each cluster consisting of 10 servers. The number of simultaneous service requests was set to $X \times 50$, with X the number of server clusters. This gave a slightly overloaded scenario. Therefore, the number of used servers was always equal to the number of available servers and is thus not depicted. The simulation results are shown in Fig. 10.

The results in Fig. 10a show that the solution of the heuristics does not degenerate as the network size grows. These results also confirm the performance results, showing that there is very little difference between GSP and OSP in terms of satisfied requests. Notable in these results is the fact that even for very low values of the OSP N parameter, the solution scales well even for very large

networks. The average execution time is shown in Fig. 10b. The figure shows that even for a large scenario with 500 servers and 2500 service requests, both heuristics solve the problem in less than 10 seconds on average. Additionally, the results show that OSP scales much better in terms of execution time than GSP, while achieving similar performance in terms of satisfied demand.

8. Conclusions

In this paper we proposed an optimal algorithm and two heuristics that solve the NP-Hard service placement and server selection problems in the context of service management platforms. The optimal algorithm is based on a ILP formulation, while the heuristics use greedy principles to tackle the problem. Additionally, we discussed the required changes to allow the algorithms to find overlay routing paths, in order to satisfy the Quality of Service (QoS) constraints of latency-sensitive services.

Unlike existing algorithms, ours did not only take into account server resources, such as CPU and memory, but also Service Level Agreements in the form of QoS requirements, such as transmission latency and bandwidth demands.

Based on extensive simulation results, we showed that, our heuristics perform close to optimal in most scenarios, while having only a polynomial time complexity. Additionally, the use of overlay routing allows both the optimal algorithm and the heuristics to significantly reduce the number of SLA violations (in terms of transmission latency bounds) in face of high-latency links. Finally, it was also shown that the heuristics scale well to large-sized networks, both in terms of solution quality and execution time.

Acknowledgments

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

Tim Wauters is funded by the Fund for Scientific Research Flanders (FWO).

References

[1] T. Yu, K. Lin, Service selection algorithms for composing complex services with multiple QoS constraints, in: Third International Conference on Service Oriented Computing (ICSOC 2005), Vol. 3826, 2005, pp. 130–143.

[2] C. Reich, K. Bubendorfer, R. Buyya, An autonomic Peer-to-Peer architecture for hosting stateful web services, in: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08), 2008, pp. 250–257.

[3] C. Adam, R. Stadler, Service middleware for self-managing large-scale systems, IEEE Transactions on Network and Service Management (TNSM) 4 (3) (2007) 50–64.

[4] J. Famaey, T. Wauters, F. De Turck, B. Dhoedt, P. Demeester, Towards efficient service placement and server selection for large-scale deployments, in: Proceedings of the 4th Advanced International Conference on Telecommunications (AICT'08), 2008, pp. 13–18.

[5] P. B. Mirchandani, R. L. Francis (Eds.), Discrete Location Theory, Wiley-Interscience, 1990.

[6] S. Martello, P. Toth, Knapsack problems: algorithms and computer implementations, John Wiley & Sons, 1990.

[7] J.-H. Lin, J. S. Vitter, Approximation algorithms for geometric median problems, Information Processing Letters 44 (5) (1992) 245–249.

[8] M. Sviridenko, An improved approximation algorithm for the metric uncapacitated facility location problem, in: 9th International IPCO Conference on Integer Programming and Combinatorial Optimization, 2002, pp. 240–257.

[9] M. R. Korupolu, C. G. Plaxton, R. Rajaraman, Analysis of a local search heuristic for facility location problems, in: Symposium on Discrete Algorithms, 1998.

[10] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit, Local search heuristics for k-median and facility location problems, SIAM Journal on Computing 33 (3).

[11] F. A. Chudak, D. P. Williamson, Improved approximation algorithms for capacitated facility location problems, Mathematical Programming: Series A and B 102 (2) (2005) 207–222.

[12] D. Krivitski, A. Schuster, R. Wolff, A local facility location algorithm for sensor networks, Lecture notes in computer science (2005) 368–375.

[13] A. Tamir, An o(pn2) algorithm for the p-median and related problems on tree graphs, Operations Research Letters 19 (2) (1996) 59–64.

[14] K. Jain, M. Mahdian, E. Markakis, A. Saberi, V. V. Vazirani, Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp, Journal of the ACM (JACM) 50 (6).

[15] K. Jain, V. V. Vazirani, Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation, Journal of the ACM (JACM) 48 (2).

[16] T. Moscibroda, R. Wattenhofer, Facility location:Â distributed approximation, in: Annual ACM Symposium on Principles of Distributed Computing, 2005, pp. 108–117.

[17] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, A. Bestavros, Distributed placement of service facilities in large-scale networks, in: 26th IEEE International Conference on Computer Communications (INFOCOM), IEEE, 2007, pp. 2144–2152.

[18] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, IDMaps: A global internet host distance estimation service, IEEE/ACM Transactions on Networking 9 (5) (2001) 525–540.

[19] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, 2004.

[20] H. Shachnai, T. Tamir, Noah's bagels – some combinatorial aspects, in: Proceedings of the 1st International Conference on FUN with Algorithms, 1998, pp. 65–78.

[21] T. Tamir, H. Shachnai, On two class-constrained versions of the multiple knapsack problem, Algorithmica 29 (3) (2001) 442–467.

[22] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi, Dynamic placement for clustered web applications, in: 15th International Conference on World Wide Web (WWW 06), 2006, pp. 595–604.

[23] B. Urgaonkar, A. L. Rosenberg, P. Shenoy, Application placement on a cluster of servers, International Journal of Foundations of Computer Science 18 (05) (2007) 1023.

[24] C. Adam, R. Stadler, A middleware design for large-scale clusters offering multiple services, IEEE Electronic Transactions on Network and Service Management 3 (1).

[25] C. Adam, R. Stadler, C. Tang, M. Steinder, M. Spreitzer, A service middleware that scales in system size and applications, in: 10th IFIP/IEEE International Symposium on Integrated Management (IM'07), 2007, pp. 70–79.

[26] S. Graupner, A. Andrzejak, V. Kotov, H. Trinks, Adaptive service placement algorithms for autonomous service networks, in: Proceedings of the 2nd International Workshop on Engineering Self-Organising Applications (ESOA), Springer, 2005, pp. 280–297.

[27] A. Paschke, M. Bichler, SLA representation, management and enforcement, in: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE), 2005, pp. 158–163.

[28] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems, in: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2002, pp. 153–183.

[29] M. D'Arienzo, A. Pescapè, G. Ventre, Dynamic service management in heterogeneous networks, Journal of Network and Systems Management 12 (3) (2004) 349–370.

[30] J. Sommers, P. Barford, N. Duffield, a. Ron, A framework for multi-objective SLA compliance monitoring, in: 26th IEEE International Conference on Computer Communications (INFOCOM), 2007, pp. 2446–2450.

[31] C. Reich, K. Bubendorfer, M. Banholzer, R. Buyya, A SLA-Oriented management of containers for hosting stateful web services, in: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007, pp. 85–92.

[32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), 2001, pp. 149–160.

[33] ILOG, CPLEX 10.0 User's Manual, ILOG Inc., Mountain View, CA, 2006.

[34] G. L. Nemhauser, L. A. Wolsey, Integer and combinatorial optimization, Wiley-Interscience, 1988.

[35] J. Y. Yen, Finding the K shortest loopless paths in a network, Management Science 17 (11) (1971) 712–716.

[36] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.

[37] T. Hill, P. Lewicki, Statistics: Methods and Applications, StatSoft, Inc., 2006.

[38] K. P. Gummadi, S. Saroiu, S. D. Gribble, King: Estimating latency between arbitrary internet end hosts (2002), in: 2nd ACM SIGCOMM workshop on Internet Measurement (IMW'02), 2002, pp. 5–18.

[39] W.-C. Feng, F. Chang, W. chi Feng, J. Walpole, A traffic characterization of popular on-line games, IEEE/ACM Transactions on Networking 13 (3) (2005) 488–500.