On the Benefit of Virtualization: Strategies for Flexible Server Allocation

Dushyant Arora, Anja Feldmann, Gregor Schaffrath, Stefan Schmid
Deutsche Telekom Laboratories / TU Berlin, Germany

1 Introduction

Network virtualization [2] is an intriguing paradigm which loosens the ties between services and physical infrastructure. The gained flexibility promises faster innovations, enabling a more diverse Internet and ensuring coexistence of heterogeneous *virtual network (VNet)* architectures on top of a shared substrate. Moreover, the dynamic and demand driven allocation of resources may yield a "greener Internet" without sacrificing (or, in the presence of the corresponding migration technology: with improved!) quality-of-service (QoS) / quality-of-experience (QoE).

This paper attends to a fundamental challenge in the field of network virtualization: the flexible allocation and migration of servers. As a generic use case, we consider a network operator offering a flexible service to a set of dynamic or mobile users, and we present a model that captures the main cost factors in such a system. This allows us to shed light on the benefit of the flexible allocation and the use of migration.

Although our cost model is described from a network virtualization perspective, it is not limited to such architectures: similar tradeoffs exist, e.g., in classic cloud networks, in content distribution networks, in the deployment of multicast reflectors or mirrored web content, or in cache placement. Our algorithms and insights are quite general and applicable to various scenarios, ranging from business applications such as SAP services in the cloud, to entertainment applications such as mobile gaming.

Concretely, the algorithms presented in this paper guarantee a low access latency by adapting the resources over time while taking into account the corresponding costs: communication cost, allocation cost, migration cost (e.g., service interruption), and cost of running the servers. The algorithms come in two flavors, exploring the extremal perspectives: *online algorithms* where allocation decisions are done without any information on future requests, and *offline algorithms* where the (e.g., periodic) demand is known ahead of time. Both algorithms are applicable to various delay models (access latency, delay due to different load functions, etc.). Moreover, we also describe an optimal offline but static algorithm that allows us to *quantify the cost-benefit tradeoffs of dynamic resource allocation*, and thus to shed light on fundamental questions such as the use of migration compared to solutions using static servers. For example, our simulations show that the overall cost can be higher (by up to hundred percent), if resources are static, in particular if the demand dynamics is moderate.

The content of this workshop paper is based on the longer *arXiv report 1011.6594*; please refer to the technical report for more details.

2 A Flexible Service Provider

Our work is motivated by the network virtualization paradigm that decouples services from the underlying physical infrastructure (the *substrate*) and for which we are in the process of developing a prototype architecture [8]. However, the model and tradeoffs studied in the following are rather general: they arise in many contexts beyond network virtualization.

We consider a service provider offering a service to users and which can benefit from the flexibility of network and service virtualization. The goal of the service provider is to minimize the round-trip-time (RTT) of its service users to the servers, by triggering migrations depending, e.g., on (latency) measurements.

Graph Model and Access Cost. Formally, we consider a substrate network $G = (V, E)$ managed by a physical network provider, where $v \in V$ are the substrate nodes and $e = (u, v) \in E$, with $u, v \in V$, are the substrate links; we will refer to the total number of substrate nodes by $n = |V|$. Each substrate node v has a certain strength $\omega(v)$ associated with it (number of CPU cores, memory size, bus speed, etc.). A link is charac-

terized, e.g., by a bandwidth capacity $\omega(e)$ and a latency $\lambda(e)$.[1] In addition to the substrate network, there is a set T of external terminals (the *thin clients*, or the *users*) that access G by issuing requests for a given virtualized service hosted on a set of virtual servers S on G. We will assume that a service is offered redundantly by up to $k = |S|$ servers, and that there is at most one server per substrate node ($k \leq n$). In order for the clients in T to access the servers S, a fixed subset of nodes $A \subseteq V$ serve as *Access Points* where clients in T can connect to G. Due to the request dynamics, the popularity of access points can change frequently, which may trigger the migration algorithm.

We define σ_t to be the multi-set of requests at time t where each element defines a request access point $a \in A$. For ease of notation, when clear from the context, we will simply write $v \in \sigma_t$ to denote the multi-set of access points used by the different requests. Our main objective is to shed light onto the tradeoff between the access costs Cost_{acc} of the users to the service (delay of requests), the server migration cost Cost_{mig}, and the cost Cost_{run} for running the servers: While moving the servers closer to the requester may reduce the access costs and hence improve the quality of service, it also entails the overhead of migration; moreover, the more active servers, the more resources are needed (processing requests, CPU, storage, etc.). In this paper, an approximate model is considered where the cost of accessing the server Cost_{acc} depends on the sum of the requests' latencies along the links to the corresponding servers and the latency due to the server's load, which, for server v and at time t, is given by $\text{load}(v,t) = f(\omega(v), \eta(v,t))$, a function of the node strength $\omega(v)$ and the number of requests arriving at the servers hosted by v at time t, $\eta(v,t)$. For example, a simple model where the load increases linearly would be $\text{load}(v,t) = \eta(v,t)/\omega(v)$:

$$\text{Cost}_{\text{acc}}(t) = \sum_{r_t \in \sigma_t} \text{delay}(r_t) + \sum_{v \in V} \text{load}(v,t).$$

We assume that requests are routed to the closest server (w.r.t. access cost).

Server Model and Migration. Each of the (at most k) servers can assume three different states: *not in use*, *inactive*, and *active*. If a server is not in use, there are no costs. An inactive server comes at a certain cost R_i per time: this cost includes storing the application software (e.g., the game) plus certain maintenance costs. The running cost of an active server R_a is larger, as it also includes CPU costs, maintaining state in the RAM, or bandwidth costs. In order to startup a server which is not in use, a fixed creation cost c is assumed. For instance, this cost captures the installation of the Linux box and the template (copy if already on disk or download from an NFS share), configuration of the template (e.g., setting up IP addresses manually or via DHCP), starting the server etc. Finally, we assume that the cost of changing from inactive to active state is negligible. Also the cost of migration depends on many different factors. While the operating system is typically replicated (copy Linux box from disk), the virtual server's configuration and data/state component must be transmitted over the network (the entire disk, or simply the `/etc` and `/var` partition). Besides the cost for the bulk data transfer of the server state, there are opportunistic costs that depend on whether the system supports live migration or not, possibly some requests need to be routed to other servers during migration, there can be periods of service interruption, etc. We will consider a scenario where migration cost can be approximated by a constant β and where inactive servers are not migrated. How does β relate to c? Again, it depends on the scenario. For example, $c \gg \beta$ for systems which support live migration (almost no opportunistic or outage costs during migration), where there is an NFS share and only the server state is migrated, and where new servers need to be configured manually. On the other hand, for example $c \ll \beta$ in systems where configuration is simple and where migration happens over multiple provider domains. For the formal description of the algorithms we will focus on the more interesting case that $\beta < c$: if $\beta \geq c$, migration is never beneficial, and the problem boils down on when and where to create and delete servers; our algorithms can easily be adapted for these situations. Also note that in our model, migrating a server from node v to an empty node v' costs β and that subsequently, node v is empty. It is not possible to maintain a (for example inactive) copy of the server at v "for free"; rather, this would require to set up a new server which costs c, as the template needs reconfiguration (e.g., new unique network addresses are needed). However, we emphasize that our approach can be adopted in many alternative scenarios where the cost models are slightly different, e.g., where server copies can be kept during migration and c denotes the cost of investing into an additional server.

Request Model. How to model the terminal dynamics (e.g., due to user mobility)? One could assume arbitrary request sets σ_t, where σ_t is completely independent of σ_{t-1}. However, for certain applications it may be more realistic to assume that the requests move "slowly" between the access points. We can distinguish between two different sources of request dynamics: *time zone effects* (users from different countries access a service at different times of the day) and *user mobility*. Note that while users typically travel between different cities or countries at a limited speed, these geographical movements may not translate to the topology of the substrate network.

[1] We assume that capacities are given and cannot be changed/increased, e.g., by investing into the infrastructure.

Thus, rather than modeling the users to travel along the links of G, we may consider on/off models where a user appears at some access point $a_1 \in A$ at time t, remains there for a certain period Δt, before moving to another arbitrary node $a_2 \in A$ at time $t + \Delta t$. Often, it is reasonable to assume some form of correlation between the individual users' movements. For example, in an urban area, workers commute downtown in the morning and return to suburbs in the evening.

Online and Offline Algorithms. In the worst case, resources in a virtual network need to be allocated *online*, without any knowledge of the request sequence or demand in advance. In order to focus on the main properties and tradeoffs involved in the the dynamic allocation and migration problem, we assume a simplified online framework. We assume a synchronized setting where time proceeds in time slots (or *rounds*). In each round t, a set of σ_t terminal requests arrive in a worst-case and online fashion at an arbitrary set of access nodes A. Thus the allocation problem is equivalent to the following synchronous game, where an online algorithm ALG has to decide on the server allocation and migration strategy in each round t, without any information on the future access requests. Concretely, in each round $t \geq 0$: (1) The requests σ_t arrive at some access nodes A. (2) The online algorithm ALG pays the requests' access costs $\text{Cost}_{\text{acc}}(t)$ to the corresponding servers. (3) The online algorithm ALG decides where in G to allocate new or remove existing servers, which servers should be active and which inactive, and where to migrate the servers in S. Accordingly, ALG incurs running costs $\text{Cost}_{\text{run}}(t)$ as well as migration costs $\text{Cost}_{\text{mig}}(t)$. To evaluate the efficiency of an online algorithm, its performance is often compared to the performance of a (sometimes hypothetical) optimal offline algorithm for the given request sequence. The ratio of the two costs is called the *competitive ratio* and captures the "price of not knowing the future". Moreover, an optimal offline algorithms is useful in scenarios with predictable or periodic demand.

3 Allocation Strategies

3.1 Online Algorithms

A natural and rather general approach to design allocation algorithms is based on *configurations*.

Definition 3.1 (Configuration) *A* configuration γ *describes, for each server, whether it is* not in use*,* inactive*, or* active*. In case of inactive and active servers, γ specifies where—i.e., on which node—the server is located.*

The following algorithm ONCONF is a natural extension of [1] to multi-server scenarios; it is based on configurations. We assume that migration is cheap, i.e., that $\beta < c$; a scenario where $\beta > c$ is analogous and not investigated further here.

Online Algorithm ONCONF: ONCONF uses a counter $\text{C}(\gamma)$ for each configuration γ. Time is divided into *epochs*. In each epoch ONCONF monitors, for each configuration γ, the cost of serving all requests from this epoch by servers kept in configuration γ, including the access costs (latency plus induced load) of the requests, the server running costs, and possible creation costs. ONCONF stores this cost in $\text{C}(\gamma)$. The servers are kept in a given configuration $\widehat{\gamma}$ until $\text{C}(\widehat{\gamma})$ reaches $k \cdot c$, where k is the maximal number of servers. In this case, ONCONF changes to a configuration $\widehat{\gamma}'$ chosen uniformly at random among configurations with the property $\text{C}(\gamma) < k \cdot c$. If there is no such configuration left, the epoch ends in that round; the next epoch starts in the next round and the counters $\text{C}(\gamma)$ are reset to zero.

We can implement ONCONF in such a way that inactive servers are managed by a queue of constant size. Inactive servers in the queue are managed in a FIFO manner (older servers are replaced first); in addition an inactive server expires after x epochs for some constant parameter x. (This also means that in configurations γ in ONCONF, inactive servers are not included.)

During an epoch, as there are $\sum_{i=1}^{k} \binom{n}{i}$ many configurations in total, ONCONF visits at most $O(k \log n)$ many configurations (a subset [1] of logarithmic size). The cost per epoch is at most $k \cdot c$ (this is overly pessimistic). An optimal offline algorithm has cost at least $\min\{\beta, c/2\}$ on average per epoch: in order to achieve a cost smaller than $\text{C}(\gamma)$ per epoch (for any γ!), the optimal algorithm needs to change configuration (by migrating or creating/deleting a server). Thus, in competitive analysis parlor [1], we have a worst-case performance guarantee: the competitive ratio is at most $O(c/\beta \cdot k^2 \log n)$.

Clearly, ONCONF needs to be optimized in many respects. For instance, it can make sense to switch between "close" (with respect to costs) configurations only, or to deterministically switch to the configuration with the lowest counter. However, the main problem of ONCONF is different: due to the configuration complexity, the runtime is only acceptable for a small number of servers k.

There are many ways to speed up the computations, and we present one efficient variant for ONCONF: ONTH is a sequential variant of ONCONF, where only one server of the configuration changes state per epoch. (TH stands for the threshold parameter c on which ONTH relies.) Although no competitive ratio is derived, our simulations show that ONTH performs well (see the technical report).

Online Heuristic ONTH: ONTH divides time is divided into two types of epochs: a "small" epoch ends when we have accumulated a cost of $y \cdot \beta$ in a given configuration for some constant parameter y, and a "large" epoch ends when the accumulated access cost is larger than the accumulated running cost (of the active servers); concretely, we will use the following condition: $\text{Cost}_{\text{acc}}/(k_{\text{cur}}+1) - \text{Cost}_{\text{run}} > c$, where k_{cur} denotes the current number of active servers. When a small epoch ends ONTH changes to the cheapest (w.r.t. the passed epoch and including access, migration, and running cost) configuration among the following: (1) γ (no change), (2) γ but where one server s is migrated to a different location, (3) γ but where one server s becomes inactive (at most k options). Inactive servers are organized in a first-in-first-out (FIFO) queue of constant size, i.e., inactive servers which fall out of the queue are no longer in use. Inactive servers in the queue expire after $x \cdot \beta$ rounds for some parameter x ($x = 20$ in our simulation). When a large epoch ends, a new server is activated at an optimal position with respect to the access cost of the latest large epoch.

3.2 Offline Algorithms

The online perspective assumed in the last section may be overly pessimistic in reality. For example, if the requests follow a regular pattern (e.g., a periodic pattern per day or week), offline algorithms can be used for optimal server allocation over time. Offline algorithm are also useful to study the theoretical benefits of dynamic allocation compared to optimal static solutions, e.g., in simulations; moreover, when comparing the optimal offline algorithms to the online algorithms, the competitive ratios—"the price of not knowing the future"—can be computed.

This section presents an optimal offline algorithm OPT for our resource allocation optimization problem. It turns out that offline strategies can be computed for many different scenarios, and we describe a general algorithm here. Algorithm OPT is based on dynamic programming techniques and also uses the concept of configurations (cf Definition 3.1). Recall that given a configuration γ, access costs Cost_{acc}, migration costs Cost_{mig}, and the running costs Cost_{run} over time can be computed.

OPT exploits the fact that the migration problem exhibits an optimal substructure property: Given that at time t, the k servers are in a configuration γ, then the most cost-efficient *path* (migrations, activation and deactivation of servers, creation, etc.) that leads to this configuration consists solely of optimal sub-paths. That is, if a cost minimizing path to configuration γ at time t leads over a configuration γ' at time $t' < t$, then there cannot be a cheaper migration sub-path that leads to γ' at time t' than the corresponding sub-path.

OPT essentially fills out a matrix $opt[\text{time}][\text{configuration}]$ where $opt[t][\gamma]$ contains the cost of the minimal path that leads to a configuration where the servers satisfy the requests of time t in a configuration γ. Recall from Definition 3.1 that a configuration γ describes for each virtual server s at which physical node v it is hosted and whether s is *not in use*, *inactive*, or *active*.

Assume that in the beginning, the system is located in configuration γ_0. Thus, initially, $opt[0][\gamma] = \text{Cost}(\gamma_0 \rightarrow \gamma) + \text{Cost}_{\text{run}}(\gamma) + \left[\sum_{v \in \sigma_0} \text{Cost}_{\text{acc}}(v, \gamma)\right]$, where $\text{Cost}(\gamma_1 \rightarrow \gamma_2)$ denotes the cost of changing from configuration γ_1 to γ_2 (cost of migrations, creation costs, etc.), $\text{Cost}_{\text{run}}(\gamma)$ denotes the cost of running the inactive and active servers for one time unit in configuration γ, and $\sum_{v \in \sigma_0} \text{Cost}_{\text{acc}}(v, \gamma)$ denotes the access costs (request latency and server load) resulting from the requests of σ_0 accessing the active servers in configuration γ. (W.l.o.g., we assume that the cost Cost_{acc} contains the first wireless hop from terminal to substrate network.)

For $t > 0$, we find the optimal values $opt[t][\gamma]$ by considering the optimal paths to any configuration γ' at time $t-1$, and adding the migration cost from γ' to γ. That is, in order to find the optimal cost to arrive at a configuration with servers at γ at time t:

$$\min_{\gamma'} \left[opt[t-1][\gamma'] + \text{Cost}(\gamma' \rightarrow \gamma) + \text{Cost}_{\text{run}}(\gamma) + \sum_{v \in \sigma_t} \text{Cost}_{\text{acc}}(v, \gamma) \right]$$

where we assume that Cost_{acc} includes the first (wireless) hop of the request from the terminal to the substrate network, and where $\text{Cost}(\gamma \rightarrow \gamma) = 0$.

Observe that the computational complexity of OPT is high for scenarios with many servers, and clustering or sampling heuristics may be used to speed up the computations.

4 Benefit of Dynamic Allocation

We conducted several experiments to study the performance of our algorithms and shed light onto the benefits of dynamic server allocation. We studied both artificial *Erdös-Rényi graphs* random graphs (with connection probability 1%) as well as more realistic graphs taken from the *Rocketfuel project* (including the corresponding latencies for the access cost). To simulate OPT, we constrain ourselves to line graphs. If not stated otherwise, we assume that $\beta = 40$, that $c = 400$, and that $R_a = 0.25$ and $R_i = 0.05$; for experiments with $\beta > c$, we set $\beta = 400$ and $c = 40$.

We consider two different scenarios: *time zone effects* (users from different countries access the service at different times of the day) and *user mobility*.

Time Zones Scenario: This scenario models an access pattern that can result from global daytime effects. We divide a day into T time periods. For each time t, $p\%$ of all requests originate from a node chosen uniformly at random from the substrate network (we use the same locations for each day). The sojourn time of the requests at a certain location is given by a parameter λ. In addition, there is background traffic: the remaining requests originate from nodes chosen uniformly at random from all access points.

Commuter Scenario: This scenario models an access pattern that can result from commuters traveling downtown for work in the morning and returning back to the suburbs in the evening. The scenario comes in two flavors: *static* and *dynamic*.

Static Load: We use a parameter T to model the *frequency* of the changes. At time $t \bmod T < T/2$, there are $2^{t \bmod T}$ requests originating from access points chosen uniformly at random around the center of the network. In the second half of the day, i.e., for $t \in [T/2, ..., T]$, the pattern is reversed. The total number of requests per round is fixed to $2^{T/2}$. Concretely, at time $t_i < T/2$, the requests originate from $p = 2^{t_i \bmod T}$ of all access points including the network center ($2^{T/2}/p$ requests per access point), until single requests originate from $2^{T/2}$ access points. Then, the same process is reversed until all $2^{T/2}$ requests originate from a single access point: the network center. We assume that the time period between t_i and t_{i+1} is given by a fixed parameter λ.

Dynamic Load: The total number of requests per round is not fixed to $2^{T/2}$. At time $t_i < T/2$, the requests originate from $p = 2^{t_i \bmod T}$ of all access points including the network center (one request per access point), until single requests originate from $2^{T/2}$ access points. Then, the same process is reversed until we have a single request originating from a single access point: the network center. We assume that the time period between t_i and t_{i+1} is fixed and we denote it by parameter λ.

The main objective of our algorithms is to adapt to dynamically changing demands in an efficient manner. Where to allocate or migrate how many servers depends on the origins and the size of the requests, and also on how access cost increases as a function of load. Due to space constraints, in the following, we will focus on the important question of the use of flexible allocation in virtual networks. For additional simulation results (e.g., for the competitive ratio of ONTH), see *arXiv report 1011.6594*.

When is dynamic allocation and migration technology most useful? As a reference point, we use the following static (but offline) algorithm STAT.

Static Heuristic STAT: For a given request sequence σ, STAT determines the number of servers k_{opt} as follows. For each $i \in \{1, \ldots, k\}$, we compute the cost of the following greedy static configuration for σ: one active server $j \in \{1, \ldots, i\}$ after the other is placed greedily at the location which yields the lowest cost for σ, given the already placed servers $\{1, \ldots, j-1\}$. k_{opt} is defined as the i with minimal cost.

Figure 1 (left) plots the ratio of the total cost incurred by STAT and the total cost incurred by OPT in the dynamic load commuter scenario as a function of λ. As can be seen, for very high dynamics as well as for very low dynamics, the allocation flexibility is of limited benefit. However, for moderate dynamics, it is worthwhile for OPT to exploit the request patterns, and a better performance can be achieved (up to a factor of two). This result also meets our expectation. Note that the two cases $\beta < c$ and $\beta > c$ are not directly comparable. In both cases, the creation costs dominate due to a low T and small number of inactive servers, but differ significantly in the two scenarios, resulting in different absolute costs and hence not directly comparable ratios.

Also in a commuter scenario with static load (Figure 1, middle), $\beta < c$ yields the lower ratio, fluctuating more or less constantly around 1.2 until it goes down to one for static access patterns. For $\beta > c$, flexible allocation pays off and the ratio goes up to almost two for intermediate λ values, but the variance is higher as well.

The dependency on λ is more accentuated in the time zone scenario: the benefit of flexible allocation highly depends on the dynamics. Figure 1 (right) shows that while a very high dynamic yields a moderate ratio, the ratio goes up quickly already for small λ, and then the use of dynamic allocation declines more or less linearly with lower dynamics. In contrast to the commuter scenario, in the time zone scenario the requests move highly correlated and migration is attractive.

5 Related Work

Our work is motivated by the network virtualization paradigm which promises to overcome the "ossification" of the Internet (see e.g., [2]). It is expected that in the future, virtual networks will be allocated, maintained and managed like clouds, offering flexibility and elasticity of resources allocated for a limited time, driven by the demand. Contrariwise, virtual networks can be an enabler for novel cloud architectures.

Our work builds upon the single server architecture studied in [1], where a competitive online algorithm has been described to migrate a single server depending on the dynamics of mobile users. In contrast to [1] which attends to the question of *where* to migrate a server, we,

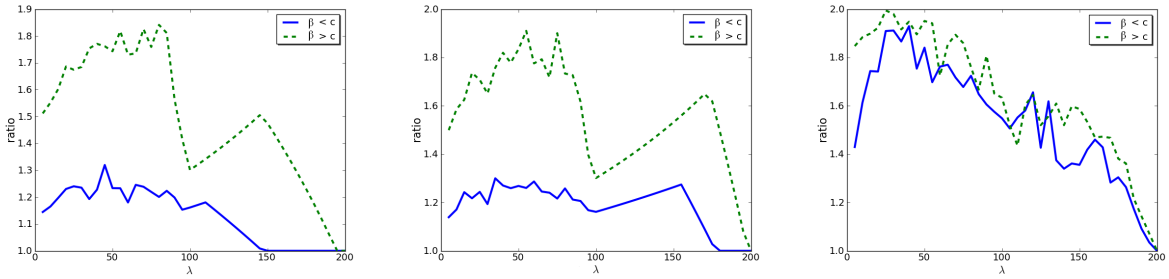

Figure 1: *Left:* The use of flexible allocation: ratio of STAT and OPT costs in dynamic load commuter scenario as a function of λ, where runtime was 200 rounds, $T = 4$, network size 5, and averaged over 10 runs. *Middle:* The use of flexible allocation: ratio of STAT and OPT costs in static load commuter scenario as a function of λ, where runtime was 200 rounds, $T = 4$, network size five, and averaged over 10 runs. *Right:* Ratio of STAT and OPT cost as a function of λ in time zone scenario ($p = 50\%$). Runtime 200 rounds, three requests per round, network size five, and averaged over ten runs.

in this paper, initiate the study of when to allocate *additional servers*. This also differs from our work on online VNet embeddings [3] where virtual networks are statically packed in the most resource efficient locations.

Hao et al. [4] provide a motivation for our work by showing that under certain circumstances, migration of a Samba front-end server closer to the users can be beneficial even for bulk-data applications. Of course, the question of how to dynamically embed and migrate virtual servers is fundamental and has been studied in several contexts, e.g., in *cloud computing* or *content distribution networks* [7, 9]. For instance, [6] presents the *Mobitopolo* infrastructure to facilitate flexible deployment and migration of distributed applications.

Our problem is a special instance of a *metrical task system* and is related to *uncapacitated facility location* (UFL) where the number of facilities (the virtual servers) is jointly derived along with the locations as part of a solution that minimizes the combined service hosting and access cost. Laoutaris et al. [5] propose a heuristic algorithm to solve the facility location problem in a distributed manner, and allow for facility migration (unlike in our case, the corresponding cost is proportional to path length); by aggregate and central re-computations of neighborhoods, an increase/decrese of resources occurs over time. Although eventual convergence is shown, the quality of the resulting equilibrium is not analyzed formally. Zhang et al. [9] extend the problem formulation of [5] to server loads and prove a constant (static) approximation guarantee. However, the performance over time in a dynamic setting is unclear, and the benefits of dynamic resource allocation cannot be assessed. Moreover, in our more granular model, there are not only facility creation but also running costs and the notion of mobility of requests (i.e., a dynamic demand over time) is made explicit. Finally, our algorithms use efficient inactive states to save costs while avoiding to shut down a server entirely.

References

[1] BIENKOWSKI, M., FELDMANN, A., JURCA, D., KELLERER, W., SCHAFFRATH, G., SCHMID, S., AND WIDMER, J. Competitive analysis for service migration in vnets. In *Proc. 2nd ACM SIGCOMM VISA* (2010).

[2] CHOWDHURY, M. K., AND BOUTABA, R. A survey of network virtualization. *Elsevier Computer Networks 54*, 5 (2010).

[3] EVEN, G., MEDINA, M., SCHAFFRATH, G., AND SCHMID, S. Competitive and deterministic embeddings of virtual networks. In *ArXiv Technical Report 1101.5221* (2011).

[4] HAO, F., LAKSHMAN, T. V., MUKHERJEE, S., AND SONG, H. Enhancing dynamic cloud-based services using network virtualization. *SIGCOMM Comput. Commun. Rev. 40*, 1 (2010), 67–74.

[5] LAOUTARIS, N., SMARAGDAKIS, G., OIKONOMOU, K., STAVRAKAKIS, I., AND BESTAVROS, A. Distributed placement of service facilities in large-scale networks. In *IEEE INFOCOM* (2007).

[6] POTTER, R., AND NAKAO, A. Mobitopolo: A portable infrastructure to facilitate flexible deployment and migration of distributed applications with virtual topologies. In *Proc. 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures (VISA)* (2009), pp. 19–28.

[7] PRESTI, F. L., PETRIOLI, C., AND VICARI, C. Distributed dynamic replica placement and request redirection in content delivery networks. In *Proc. 15th MASCOTS* (2007).

[8] SCHAFFRATH, G., WERLE, C., PAPADIMITRIOU, P., FELDMANN, A., BLESS, R., GREENHALGH, A., WUNDSAM, A., KIND, M., MAENNEL, O., AND MATHY, L. Network virtualization architecture: Proposal and initial prototype. In *Proc. VISA* (2009).

[9] ZHANG, Q., XIAO, J., GÜRSES, E., KARSTEN, M., AND BOUTABA, R. Dynamic service placement in shared service hosting infrastructures. In *NETWORKING* (2010), pp. 251–264.