Online Inter-Datacenter Service Migrations

Nikos Tziritas, Samee U. Khan, *Senior Member, IEEE*, Thanasis Loukopoulos, Spyros Lalis, Cheng-Zhong Xu, *Senior Member, IEEE*, Keqin Li, *Fellow, IEEE*, and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Service migration between datacenters can reduce the network overhead within a cloud infrastructure; thereby, also improving the quality of service for the clients. Most of the algorithms in the literature assume that the client access pattern remains stable for a sufficiently long period so as to amortize such migrations. However, if such an assumption does not hold, these algorithms can take arbitrarily poor migration decisions that can substantially degrade system performance. In this paper, we approach the issue of performing service migrations for an unknown and dynamically changing client access pattern. We propose an online algorithm that minimizes the inter-datacenter network, taking into account the network load of migrating a service between two datacenters, as well as the fact that the client request pattern may change "quickly", before such a migration is amortized. We provide a rigorous mathematical proof showing that the algorithm is 3.8-competitive for a cloud network structured as a tree of multiple datacenters. We briefly discuss how the algorithm can be modified to work on general graph networks with an $O(\log|V|)$ probabilistic approximation of the optimal algorithm. Finally, we present an experimental evaluation of the algorithm based on extensive simulations.

Index Terms—Cloud computing, online service migrations, online virtual machine migrations, online algorithms

1 Introduction

Economies of scale, the pay-as-you-go model, and automated tools that facilitate the porting of legacy services to cloud environments [1] have made clouds an attractive solution for enterprises and organizations. The key technology for enabling service portability as well as elastic operation is virtualization. On the one hand, legacy programs can be executed on top of a virtual machine (VM) that can be configured according to client requirements. On the other hand, VMs (and the enterprise services that run on top of them) can migrate between datacenters, at runtime, to provide better response times and achieve a more efficient operation of the cloud environment with minimal disruptions to the service clients.

In this paper, we focus on performing VM migrations to minimize the network overhead within a given cloud infrastructure. For the sake of generality we use the term *service migrations*, rather than VM migrations, as one can imagine each enterprise service or service bundle running in a separate VM or a span of VMs. Even though we don't consider the case where the VMs of a service may split between datacenters, such a case can be tackled by considering a composite aggregated VM as a candidate for migration. The algorithm presented in the paper is also applicable for containers or dockers, somce it can be simply adjusted to run in the context of an operating system or in the context of a docker to migrate a container. The service placement is tackled in a coarse-grained fashion, with all of the VMs comprising a service being migrated from one datacenter to another. On the other extreme, the intra-datacenter VM placement can be tackled as a fine-grained approach, which is complementary to our work and does not come in conflict with the inter-datacenter service migration problem.

Most importantly, we tackle the problem in an *online fashion*, assuming an unpredictable, dynamically changing client access pattern, and considering the expected benefit of a service migration while also taking into account the network overhead for performing such an operation. The difficulty of the problem resides in the fact that in the generalized case, future client access patterns are unknown and cannot be predicted based on the history [6]. Consequently, poor decisions may be taken. Not only can one miss out on the beneficial migrations that would lead to better performance, but one may also decide on non-beneficial migrations, which must be reverted in the subsequent decisions; thereby, significantly degrading the system performance. The proposed algorithm does not make any assumption about the access patterns. The reason is that the parameters are adjusted according to the theoretical analysis performed in Section 6, since such goals are aligned with intra-datacenter service migrations but not with inter-datacenter service migrations.

Although the tackled problem is complex it is also generic enough to tackle specific cases of interest. For instance a transcoding service that gets as input video streams, encodes them in different bit rates and sends them to one or more delivery servers, could benefit from the algorithm described in the paper, since both video sources and destinations possibly vary in time (e.g., video conferencing).

- *N. Tziritas and C.-Z. Xu are with the Chinese Academy of Sciences, Beijing 100864, China. E-mail: {Nikolaos, cz.xu}@siat.ac.cn.*
- *S.U. Khan is with the Department of Electrical & Computer Engineering, Mississippi State University, Starkville, MS 39762 USA. E-mail: skhan@ece.msstate.edu.*
- *T. Loukopoulos and S. Lalis are with the University of Thessaly, Filellinon, Volos 382 21, Greece. E-mail: luke@dib.uth.gr, lalis@inf.uth.gr.*
- *K. Li is with the State University of New York, Albany, NY 12246 USA. E-mail: lik@newpaltz.edu.*
- *A.Y. Zomaya is with the University of Sydney, Camperdown, NSW 2006, Australia. E-mail: albert.zomaya@sydney.edu.au.*

Manuscript received 31 Jan. 2016; revised 8 Oct. 2016; accepted 1 Dec. 2016. Date of publication 9 Mar. 2017; date of current version 3 Dec. 2020. Recommended for acceptance by M. Parashar, O. Rana, and R.C.H. Hsu. Digital Object Identifier no. 10.1109/TCC.2017.2680439

The specific contributions of this paper are: (a) we formulate the service migration problem, and propose an online algorithm that minimizes the inter-datacenter network overhead; (b) we provide a competitive analysis which reveals a 3.8-competitive ratio, when the underlying network topology is a tree; (c) we provide insights into how our algorithm can be extended to work on generalized graph networks; and (d) we present an experimental evaluation showing that our algorithm achieves a network load reduction of up to 80 percent, when compared with a static offline algorithm and up to 30 percent, when compared with the best known algorithm [4] pertaining to issue under discussion.

The remainder of this paper is organized as follows. Section 2 describes the related work. In Section 3, we introduce the system model and problem formulation. Sections 4 and 5 describe the algorithm. Section 6 provides a detailed proof for the competitive ratio of the algorithm. In Section 7, we describe the experimental evaluation. Finally, Section 8 concludes the paper.

2 Related Work

A lot of works have studied the service or VM placement/migration problem in an offline fashion, under various objectives. Specifically, with "offline" we mean that the application characteristics are static. Ref. [16] minimizes the total network overhead by co-locating VMs that communicate heavily with each other. Ref. [5] proposes VM consolidation algorithms to reduce energy consumption. An interference aware migration strategy for VMs is proposed in [24]. This strategy tackles the problem of server consolidation by considering co-located VMs that do not interfere with each other regarding cpu utilization. An algorithm that minimizes the total network overhead by co-locating services with increased communication requirements is provided in [19]. A similar approach is described in [22] for wireless sensor networks. The minimization of the communication delay between VMs was tackled in [2], with the focus on reducing the interactions among various geographically distributed data centers.

Our work is closer to that of [2], [16], [19], and [22]. The main difference being that: (a) we tackle the problem in an online fashion and (b) we minimize the communications due to the forwarding of client requests to the respective services rather than due to any inter-dependencies between different services.

Many efforts have also been done to tackle the service or VM placements in an online fashion [23]. A distributed rate allocation algorithm is discussed in [9]. The bandwidth sharing problem is tackled in [10] and [11] as a Nash bargaining game. The authors propose allocation principles by defining a tunable base bandwidth for each VM. Ref. [3] presents an algorithm for solving the problem of energy- and performance-efficient VM consolidation. The VM consolidation problem is also tackled in [18] through online bin packing. In [13] and [25], the authors solve the same problem by using Kalman filter and Nash equilibrium techniques, respectively. An approach migrating agents in an online fashion under WSNs is discussed in [21]. Complementary to our work are [7], [14], [20] that propose load and power aware controllers. The target of [20] is to balance the load and maximize quality of service, while [7] and [14] to optimize power consumption. Ref. [17] studies the trade-off between communication overhead and delay overhead in the context of message aggregation in WSNs. The authors in [12] proposed an online algorithm for the joint problem of energy minimization and network congestion. Lastly, in [4], the authors solve the problem of service placement in virtual networks, with the objective of minimizing the total service access delay experienced by the clients. The authors prove that their proposed algorithm is $O(\mu \log n)$-competitive, with μ being the ratio between maximal and minimal link capacity in the underlying network, and n being the number of datacenters in the system.

Our work fundamentally differs from [3], [12], [13], [18], and [25] in that such works are concerned with a different objective function. Our work differs from [21] as: the proposed algorithm in [21] is not scalable when the system exhibits frequent changes in load because it keeps historical information potentially for every access request. In contrast, our approach keeps the changes in an aggregated way. Ref. [17] is similar to our work in terms of its objective functions; however, it assumes a different system model (message aggregation). The work that is closest to ours is reported in [4] as it adopts a similar model and objective function. The pitfall of the algorithm presented in [4] is that it divides time into epochs, and resets the information kept about the traffic/load between services and datacenters at the end of each epoch. To understand why the above is a pitfall, we refer the reader to the proof of Theorem 1 in Section 6 which shows that the competitive ratio crucially depends on the time instance where such a reset takes place, which advocates against an epoch-based approach.

3 System Model and Problem Definition

Let a cloud infrastructure be captured as a graph $G = (V, E)$, where each vertex $u \in V$ represents a datacenter n_u and each edge $e = (u, v) \in E$ represents a communication link between n_u and n_v in the intra-cloud network. From now on we will use interchangeably the intra-cloud and inter-datacenter network.

Each communication link e is characterized by a possibly different data transfer overhead weight w_e. The data transfer overhead is computed by the average delay experienced when transferring data over that link. The aggregated network overhead for a path p between two datacenters n_u and n_v is denoted by $w^p_{uv} = \sum_{\forall e \in p} w_e$, and the minimum network overhead over all such paths is denoted by $W_{uv} = \min_{\forall p} w^p_{uv}$. We assume that the network is direction-neutral ($W_{uv} = W_{vu}$) and that the network overhead of local data transfers is zero ($W_{uu} = 0$).

A datacenter can host several services; however, each service is hosted on a single datacenter. Each client connects to the cloud through an entry-point datacenter, e.g., the one closest to its physical location. Client requests are forwarded over the inter-datacenter network to the datacenter hosting the respective services, and responses are sent back to the clients in the same way. Without the loss of generality, we assume that a reply follows the same path that was used for the requests (in reverse direction). Client mobility is captured implicitly, through the dynamically changing client access pattern.

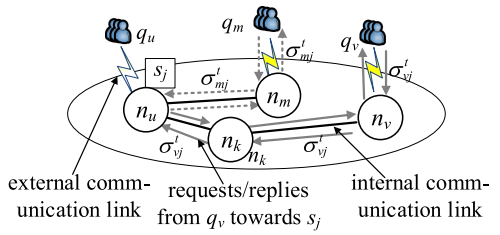

Fig. 1. An example with datacenters and clients.

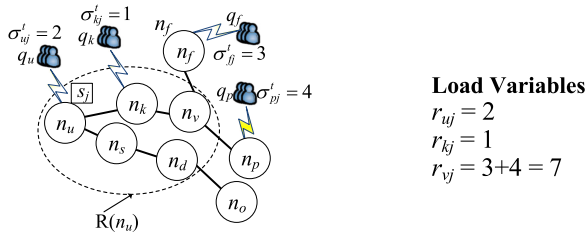

Fig. 2. An example of network awareness and load variables. The dashed line marks the awareness circle of datacenter n_u.

Let s_j be the jth service of the system, hosted on datacenter n_v. Moreover, let q_u be the set of clients that connect to the cloud through n_u, and let σ^t_{uj} denote the data volume of the requests/replies sent/received by clients in q_u through n_u for service s_j at time t. Such traffic must be forwarded between n_u and n_v (the host of s_j). It is noteworth to mention that no such forwarding is required if s_j is hosted on n_u. Fig. 1 depicts an example of the three different sets of clients q_u, q_m, and q_v that connect to the cloud via datacenters n_u, n_m, and n_v, respectively. The clients interact with the service s_j hosted on n_u with the respective client volume being σ^t_{uj}, σ^t_{mj}\$ and σ^t_{vj}.

To deal with a dynamically changing client request/reply volume, a service s_j can be migrated from its current host n_u to another datacenter n_v. Let $M_{uvj}(t)$ represent such a migration being performed at time t. Moreover, let MC_j be the data transferred associated with s_j. Finally, let h be a service hosting scheme/function, where $h(t, j) = u$ means that s_j is hosted on n_u at time t. Due to service migrations, it could be that $h(t,\ j) \neq h(t', j)$. That is, a service may be hosted on various datacenters at different points in time. Let ε be strictly less than the time needed to migrate any service between any pair of nodes.

Based on the above, Eq. (1) captures the network overhead due to the client-datacenter interactions at time t, as a function of the service hosting function h. The extra network overhead due to the service migrations that take place at time t is given by Eq. (2). It must be noted that when $h(t, j) = h(t + \varepsilon, j)$, then s_j has not been migrated during $[t, t + \varepsilon]$. Given that a service (s_j) migration started any time before t' and ended exactly at $t' + \varepsilon$, then we demand $\varepsilon > 0$ such that MC_j data units are uniformly distributed within $[t', t' + \varepsilon]$. Note that Eq. (2) seems counter-intuintive because the service migration overhead is divided by time. However, the time is vanished when Eq. (2) is integrated as happens in Eq. (3). The total intra-cloud network overhead for a given time interval [0, T] is equal to the sum of these two components over that interval, see Eq. (3).

$$c^t_1(h) = \sum_{\forall j} \sum_{\forall u} \sigma^t_{uj} \times W_{u,h(t,j)} \tag{1}$$

$$c^t_2(h) = \sum_{\forall j} \frac{MC_j}{\varepsilon} \times W_{h(t,j),h(t+\varepsilon,j)} \tag{2}$$

$$C(h) = \int_0^T c^t_1(h)dt + \int_0^T c^t_2(h)dt \tag{3}$$

The optimization problem that we are addressing in this artile can be stated as follows: *Given a network $G = (V, E)$, the network communication overhead between each datacenter pair W_{uv}, a set of services s_j and a client traffic pattern σ^t_{uj} over the time interval [0, T], find a service hosting function/scheme h for this interval such that* Eq. (3) *is minimized*.

Note that if all links of the intra-cloud network have the same data transfer overhead, then the network overhead of path p between two datacenters can be replaced by its length $w^p_{uv} = len(p)$. In this case, Eq. (3) is equal to the total network traffic for the interval [0, T], and by solving the above problem one minimizes the intra-cloud network traffic for that interval.

4 Network Overhead Minimization Algorithm

In this section, we present an algorithm that decides, in an online fashion, whether (or not) to migrate a given service from its current host to another datacenter to minimize the total network overhead as per Eq. (3). We refer to the algorithm as the Network Overhead Minimization (NOM) algorithm. It is noteworthy to mention that the NOM is designed for tree-based networks. Because each datacenter is responsible for the service migration it hosts, there is no way of a conflict between service migrations.

A key parameter of NOM is the extent to which a datacenter knows the topology of the neighborhood, referred to as *network awareness radius R*. Let $R(n_u)$ be the set of datacenters that are at most R hops away from n_u, including the datacenter itself. For each locally hosted service s_j and each datacenter $n_m \in R(n_u)$, n_u keeps a load variable r_{mj}, which is used to record the client traffic related with s_j and n_m, as follows: (a) if n_m is less than R hops away from n_u, then r_{mj} records the client traffic generated by all of the clients residing at n_m; and (b) if n_m is exactly R hops away from n_u, then r_{mj} captures the aforementioned traffic as well as the traffic related with the clients residing at datacenters using n_m as a router to reach s_j. The datacenter also keeps a load variable r_{uj} for recording the client traffic for the local service s_j residing at n_u.

Fig. 2 reports an example where datacenter n_u hosts service s_j and where R is 2. For the purpose of illustration, let us focus on the time instance t. The entry points for client traffic that concerns service s_j are datacenters n_u, n_k, n_f and n_p, with the respective traffic volume being $\sigma^t_{uj} = 2$, $\sigma^t_{kj} = 1$, $\sigma^t_{fj} = 3$ and $\sigma^t_{pj} = 4$. Given that $R(n_u) = \{n_k, n_v,\ n_s,\ n_d\}$, n_u has five load variables for s_j: r_{kj}, r_{vj}, r_{sj}, r_{dj} and r_{uj}. Note that only $r_{uj} = 2$, $r_{kj} = 1$, $r_{vj} = 7$ are greater than zero. Moreover, r_{vj} aggregates the client traffic coming from n_f and n_p, which is routed via n_v, even though n_v is not the entry point for any clients of s_j.

Based on the aforementioned load variables, n_u computes the benefit of s_j being prospectively hosted on n_m versus the

```
1: for each s_j hosted by n_u
2:    newHost = n_u; //the current host of s_j
3:    do
4:       b = 0; //the current benefit of migrating s_j
5:       candHost = newHost; // init best candidate for migrating s_j
6:       for each ( n_m ∈ R(n_u): hops(newHost, n_m) == 1 ) {
7:          if ( B_umj > 0 ) //pick this host
8:             b = B_umj;
9:             candHost = n_m;
10:            break;
11:         end if
12:      end for
13:      if ( (candHost == newHost) || (Eq. (6) is false ) ) break;
14:      else newHost = candHost;
15:   while ( hops(newHost, n_u) < R );
16:   if ( b != 0 ) migrate s_j on newHost; // the migration decision
17:   else do nothing;
18:  if ( Eq. (7) is true ) then reset the load variables for s_j
19: end for
```

Fig. 3. Pseudocode of NOM for datacenter n_u.

current hosting of s_j on n_u, as per Eqs. (4) and (5). First, the network overhead for the current placement of s_j on n_u, as well as for the prospective case of s_j being hosted on each of the datacenters $n_m \in R(n_u)$, is calculated using Eq. (4). Thereafter, for each such datacenter n_m, the benefit of migrating s_j from n_u to n_m is calculated using Eq. (5), as the difference between the network overheads of the respective placements. Finally, n_u decides to migrate s_j to the datacenter that gives the largest benefit, provided this benefit is greater than a threshold (which we discuss in the subsequent text).

$$cost_{jx} = \sum_{\forall v \in R(n_x)} r_{vj} W_{xv} \tag{4}$$

$$B_{umj} = cost_{ju} - cost_{jm} \tag{5}$$

Checking Eq. (5) for each datacenter $n_m \in R(n_u)$ can be time-consuming. To reduce the time-complexity of the algorithm, we employ the following technique. Initially, the algorithm calculates the benefit of migrating s_j only to 1-hop neighbors of n_u, as if R were equal to 1. As will be shown in Section 6 (Theorem 2), at most one 1-hop neighbor, say n_m, can have a positive benefit B_{umj}. If B_{umj} is greater than the migration threshold (see next), then the calculation is repeated by considering only the datacenters that are one hop further away from n_m. The iteration stops when: (a) no beneficial migration can be found for s_j; or (b) n_m is R hops away from n_u. The algorithm decides to migrate s_j to the last datacenter (if any) for which the benefit was greater than the threshold. The pseudocode of the NOM algorithm, is given in Fig. 3. Note that the NOM is distributed, and runs periodically at every datacenter.

In addition to the aforementioned steps, NOM makes two important checks or actions. The first check (line 13 of the pseudocode) is to decide for a service migration only if Eq. (6) holds (if the benefit is at least twice the network overhead for actually performing the migration). The second check (line 18 of the pseudocode) is to reset the load variables when Eq. (7) holds (when the aggregate reaches the so-called reset threshold RT_j).

$$B_{umj} \geq W_{um} \times 2MC_j \tag{6}$$

$$\sum_{n_m \in R(n_u)} r_{mj} \geq RT_j \tag{7}$$

The introduction of a migration threshold and the resetting of the load variables play a crucial role for the competitive ratio of the NOM, as we will discuss in Section 6. In particular, if the migration threshold is chosen too small the competitive ratio increases dramatically, which we capture in Theorem 3.

5 Adapting the NOM for Generalized Network Graphs

As mentioned previously, NOM is designed to work on tree networks. In this section, we detail an adaptation of the NOM that can also work in on generalized network graph structures.

First, we define the metric for the problem of minimizing the network overhead in general graphs as $(V, \boldsymbol{\rho})$. Specifically, V is the set of datacenters, while $\rho(n_u, n_v)$ is the minimum network overhead for transferring one data unit between n_u and n_v. We define the diameter of $\boldsymbol{\rho}$ as: $\Delta = \max_{\forall n_u, n_v \in G} \rho(n_u, n_v)$.

Next, we find a distribution DS over a family of tree metrics, TM, so as to a-probabilistically approximate our metric. When we say that metric (V, ρ) is a-probabilistically approximated by (DS, TM), then for every datacenter pair (n_u, n_v), $E_{\rho' \in (DS, TM)}[\rho'(n_u, n_v) \leq a \times \rho(n_u, n_v)$, must hold true. Note that $E[\rho'(n_u, n_v)]$ is the expected value of $\rho'(n_u, n_v)$ .According to [8], for any given metric (V, ρ), one can find a (DS', TM') that is the $O(\log|V|)$ approximation of $\boldsymbol{\rho}$. Therefore, we can generate a tree T according to (DS', TM'), and solve the problem on T, using NOM. For further information of how T can be generated, we refer the reader to [8]. As the last step, we map the produced solution onto the original graph.

In simple words, according to [8] a tree T is constructed with each node belonging to G being a leaf in T. The upper levels of T consist of clusters. For example, assume that n_1 and n_2 have parent the cluster c_1, while n_3 and n_4 have parent the cluster c_2. Then, c_1 and c_2 are cluster nodes consisted of n_1, n_2 and n_3, n_4, respectively. It must be noted that c_1 and c_2 belong to the second level of T. Assume now that c_3 is a parent of c_1 and c_2 and belongs to the third level of T. Therefore, c_3 is a cluster consisting of c_1 and c_2, and thus consisting of n_1, n_2, n_3 and n_4. The weight of an edge between a node located at ith and a node located at $(i+1)$th level equals 2^i. Therefore, the distance between n_1 and c_2 is $\rho'(n_1, c_2) = 2 + 2^i$, while the distance between n_1 and n_4 is $\rho'(n_1, n_4) = 2(2 + 2^i)$.

After constructing T we run NOM on that tree and get a solution consisting of all of the service migrations. Then, the next step is to map the solution based on T onto a solution based on G. There are two cases: (a) a service migrates (based on T) from a leaf node n_u towards a leaf node n_v; (b) a service migrates (based on T) from a leaf node n_u towards a cluster node c_k. We must note that there is no case of migrating a service between two clusters. In (a) the migration is mapped onto G by migrating the respective service from n_u towards n_v following the shortest path (based on G)

between n_u and n_v. In (b) the migration is mapped onto G by migrating the respective service(s) from n_u towards a node n_m belonging in c_k. Specifically, n_m must satisfy the following requirement: *the amount of communication load between the respective service(s) and the node n_m is greater than or equal to any datacenter belonging in c_k.*

As discussed previously, we can generate a tree topology that $O(\log|V|)$ probabilistically approximates a general-structured network. Therefore, we can apply NOM on that tree and result in a competitive ratio of 3.8 (see Theorem 1 below). The solution of NOM in *T* is modified such that any service is assigned on a leaf node of *T*, at the expense of an approximation ratio of two. The above ratio is justified by the following: (i) *T* is structured as a tree, and (ii) when a service is (virtually) hosted by a cluster c_k, it is assigned on the leaf with the highest communication load among leaves belonging in c_k. According to the above, the communication load is strictly less than that of assigning the respective service on c_k. Therefore, the competitive ratio remains constant. The above means that the resulted algorithm can $O(\log|V|)$ probabilistically approximate the optimal algorithm for general structured networks.

6 Competitive Analysis

Competitive analysis is a method used to compare the output of an online algorithm (ALG) that is unaware of the future with the output of the offline optimal algorithm (OPT) that has the complete knowledge of the future. The input is chosen by a cognizant adversary, such that the competitive ratio between ALG and OPT is maximized. Given a set of sequences, $S = (\sigma_1, \sigma_2, \ldots)$, of requests, the competitive ratio between ALG and OPT may be given as:

$$\max_{\sigma \in S} \frac{ALG(\sigma)}{OPT(\sigma)} \tag{8}$$

An algorithm is competitive if and only if its competitive ratio is bounded. Specifically, an algorithm is called $\boldsymbol{\zeta}$-competitive if the following holds:

$$ALG(\sigma) \leq \zeta \times OPT(\sigma),\ \forall \sigma \in S \tag{9}$$

Therefore, we must devise an online algorithm such that $\boldsymbol{\zeta}$ is as small as possible.

6.1 The Competitive Ratio of the NOM

In this section, we prove that NOM is 3.8-competitive when the underlying network is structured as a tree. Because the network overhead incurred between datacenters and clients is specified by a constant, we can omit it from our proof. Specifically, the aforementioned overhead is independent of the decisions of OPT and NOM. For our analysis, we assume that there is only one service (called s_j) within the cloud. Such an assumption is necessary to find the worst-case bounds. Moreover, we also assume that $r_{uj}(z, y)$ is the same as r_{uj}, with the difference being that the former captures the load for a specific time interval $[z, y]$. Furthermore, we use the notation $B_{uvj}(z, y)$ to capture the migration benefit according to the time interval $[z, y]$. To keep our analysis tractable, we also assume that the awareness of the NOM is equal to the diameter of the network.

Below we provide a sketch of our proofs. Particularly, in Lemma 1 – Lemma 8, we assume that: (a) NOM and OPT are allowed to perform at most one migration and (b) when NOM performs a migration, the destination is the same as that of the OPT algorithm. Lemma 1 provides the intuition of why Eq. (7) is important to improve the competitive ratio of the NOM. In Lemma 2 – Lemma 6, we identify the competitive ratio, under the assumptions that load variable resets are disallowed. Lemma 7 and Lemma 8 identify the competitive ratio when the load variable resets are allowed. In Lemma 8 and Lemma 9, we prove that the assumptions (a) and (b) as stated above do not compromise the competitive ratio of NOM.

Lemma 1. *The competitive ratio between NOM and OPT is unbounded, provided that NOM does not apply Eq. (7).*

Proof. It suffices to show that there is an $\boldsymbol{\sigma}$ such that the ratio between NOM and OPT tends to become infinity. To proceed, we make the following assumptions:

- The life of our system ends at *T* time instance.
- The cloud consists of two datacenters (n_u and n_v, with $w_{uv} = 1$), and initially, s_j is located on n_u.
- The OPT and NOM perform $M_{uvj}(z)$ and $M_{uvj}(y)$ migrations, respectively.

Consequently, we have that $OPT(\sigma) = r_{vj}(0, z) + r_{uj}(z, T) + MC_j$, and $NOM(\sigma) = r_{vj}(0, y) + r_{uj}(y, T) + MC_j$. Therefore, the following must also hold: (a) $y > z$, due to optimality of OPT and (b) $B_{uvj}(0, y) = r_{vj}(0, y) - r_{uj}(0, y) > 2MC_j$, due to the first action/check of the NOM. The adversary can choose an $\boldsymbol{\sigma}$ such that $r_{vj}(0, y) = r_{uj}(0, y) + 2MC_j + \varepsilon$. We also demand that the chosen $\boldsymbol{\sigma}$ meets the following requirements: (a) $r_{vj}(0, z)$ and $r_{uj}(z, T)$ equal to zero, (b) $r_{uj}(0, y)$ equals a sufficiently large number (or with a slight abuse we can say that it tends to infinity) such that the ratio between MC_j and $r_{uj}(0, y)$ tends to zero, and (c) $r_{uj}(y, T)$ equals to zero. Note that $r_{uj}(0, y)$ is not bounded by any number (there is no mechanism preventing $r_{uj}(0, y)$ from being as large as possible), while OPT($\boldsymbol{\sigma}$) is a constant. Because $r_{uj}(0, y)$ is a component of $NOM(\sigma)$, it entails that the latter is not bounded. Therefore, the competitive ratio is unbounded. □

It is noteworthy to mention that by applying the second check/action of the NOM, the load variables are reset to zero when RT_j is reached. Therefore, $r_{uj}(0, y)$ is bounded RT_j, which in turn means that NOM($\boldsymbol{\sigma}$) is also bounded because all of its components are bounded. Subsequently, we show the consequences of not applying the second check/action of the NOM procedure. It is prudent to choose the value of RT_j to be greater than the double the network overhead of migrating s_j towards a 1-hop neighboring datacenter ($RT_j > 2MC_j$); otherwise, we compromise the performance of the NOM, as the load variables will be reset before being able to decide for a migration. We also must note that when resetting the load variables there is a special case of resetting a variable r_{vj}, while some $B_{uvj}(0, y)$ is greater than zero. Below we show whether the competitive ratio is dependent on the above or otherwise.

Lemma 2. *The competitive ratio between NOM and OPT is at least three, provided that NOM performs all of its checks/actions.*

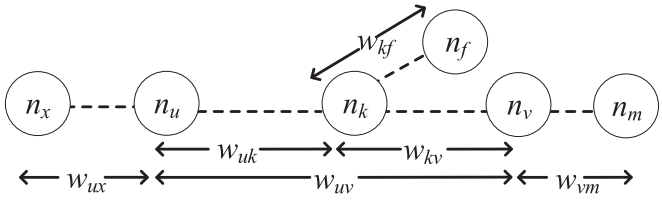

Fig. 4. Location of additional datacenters.

Proof. Assume that we have a network with at least two datacenters (n_u and n_v), with s_j being initially located on n_u. If OPT decides to perform a migration, $M_{uvj}(z)$, then the best case for OPT is burdened with the migration overhead equaling $w_{uv} \times MC_j$. On the other hand, the adversary will choose an $\boldsymbol{\sigma}$ such that the NOM is burdened by at least $w_{uv} \times 2MC_j + w_{uv} \times MC_j$. The first factor concerns the load that must be generated to satisfy the first check/action of NOM. The second factor represents the network overhead of migrating s_j from n_u to n_v. Therefore, the ratio between NOM and OPT is three. □

Lemma 3. *The competitive ratio between NOM and OPT becomes one, under the requirement that both do not perform any migration.*

Proof. Because of the requirement that neither the NOM nor OPT perform migrations, it means that they are burdened with the same network overhead. Therefore, the competitive ratio is one. □

Lemma 4. *The competitive ratio between NOM and OPT is 2-ε, provided that (a) OPT performs one migration but NOM does not perform, (b) $RT_j > 2MC_j$, and (c) the load variables are not reset.*

Proof. We assume that the life of our system is *T*, and the OPT performs $M_{uvj}(z)$. The ideal case for the OPT is that only n_u generates load before time instance *z*, while only n_v generates load after *z*. In that way the OPT will pay only the network ovherhead of performing $M_{uvj}(z)$. The adversary will choose an $\boldsymbol{\sigma}$ such that n_v generates $2MC_j - 1$ during [*z*, *T*]. Note that for the case that only n_v generates load, n_v cannot generate more load than $2MC_j - 1$. The above is because in that case NOM would migrate s_j at n_v, which comes in contradiction due to the precondition that NOM is not allowed to perform migrations. Therefore, NOM is burdened with $w_{uv} \times (2MC_j - 1)$.

Previously, we showed that OPT is not burdened with the load generated by n_v during $[z, T]$, while NOM is burdened with w_{uv} for each unit of load. Consequently, the adversary will find an $\boldsymbol{\sigma}$ such that n_v generates as much load as possible, provided that the ratio between NOM and OPT increases, while the preconditions are not violated. Therefore, we have to investigate the behavior of the ratio when additional datacenters (other than n_v) generate load, giving in that way the ability of n_v to generate load more than $2MC_j - 1$. We proceed by assuming that n_v generates load equaling $2MC_j$, and discern the cases where an additional datacenter n_d generates load of one unit: (a) $n_d \equiv n_u$, (b) $n_d \equiv n_x$, (c) $n_d \equiv n_k$, (d) $n_d \equiv n_m$, and (e) $n_d \equiv n_f$. The locations of the aforementioned datacenters are given in Fig. 4.

We now discuss each of the above scenarios in detail. For the case (a), when OPT is burdened with extra w_{uv} overhead, NOM is not burdened with extra overhead. The above means that the ratio between NOM and OPT will decrease. Therefore, the adversary will not choose the above scenario. For the case (b), when OPT incurs extra $w_{uv} + w_{ux}$ overhead, NOM is burdened with extra w_{ux} overhead. Because the ratio lessens, the adversary will not choose such a scenario. It can be seen, that if case (c) happens, then n_k will satisfy Eq. (6). Therefore, NOM will migrate s_j at n_k. The above comes in contradiction with the fact that NOM is not allowed to perform migrations. Case (d) matures if n_m generates a load of one unit, then n_v will satisfy Eq. (6), which contradicts the precondition (a) of Lemma 4. If case (e) does exist, then we have a contradiction because n_k will satisfy Eq. (6).

Therefore, the adversary will not choose any $\boldsymbol{\sigma}$ that satisfies the above scenarios. We must also note that the adversary will also not choose any combination of them. For example assume that case (b) is combined with case (c), then the above scenario is feasible (no contradiction). However, the adversary will not choose such a combination, as the ratio decreases. As a result, if the adversary tries to push n_v to generate a load more than $2MC_j - 1$, then either we have a violation of precondition (a) of Lemma 4 or the ratio decreases. Therefore, the ratio between NOM and OPT becomes $2 - \varepsilon\,(\varepsilon = 1/2MC_j)$. We must note that we do not consider any case that the load variables are reset due to the precondition (c) of Lemma 4. □

Lemma 5. *The competitive ratio between NOM and OPT is given by (5.7), provided that: (a) NOM performs all of its checks/actions, (b) OPT and NOM perform $M_{uvj}(z)$ and $M_{uvj}(y)$, respectively, (c) only n_u and n_v generate load within the cloud, and (d) the load variables are not reset.*

Proof. Because only n_u and n_v generate load within the cloud, we result in the Eq. (5.1) and Eq. (5.2). Note that because OPT takes the optimal decision, it holds that $y > z$. In the subsequent text, we discuss the values that aforementioned load variables must take on, such that the ratio between the NOM and OPT be maximized.

$$OPT(\sigma) = w_{uv} r_{vj}(0, z) + w_{vu} r_{uj}(z, T) + w_{uv} MC_j \qquad (5.1)$$

$$NOM(\sigma) = w_{uv} r_{vj}(0, y) + w_{vu} r_{uj}(y, T) + w_{uv} MC_j \qquad (5.2)$$

First, the adversary will try to identify whether the ratio (between NOM and OPT) increases when the load variables of OPT are set to zero. We can see that the ratio increases when either $r_{vj}(0, z)$ or $r_{uj}(z, T)$ decreases. Therefore, we have that $r_{vj}(0, z) = r_{uj}(z, T) = 0$.

On the other hand, we also need to check the load variables of the NOM. Because $y > z$, it holds that $r_{uj}(y, T) < r_{uj}(z, T)$, which in turn means $r_{uj}(y, T) = 0$. Now it remains to be decided on the value of the last load variable $r_{vj}(0, y)$. On a first look we expect that the adversary will choose $r_{vj}(0, y)$ to be as large as possible. However, the variable $r_{vj}(0, y)$ needs more investigation. First of all, because NOM will perform $M_{uvj}(y)$, it must hold that $B_{uvj}(0, y) \geq 2w_{uv} \times MC_j$. Note that $B_{uvj}(0, y)$ cannot be much greater than $2w_{uv} \times MC_j$, because when NOM

identifies that the above inequality is satisfied it will migrate s_j onto n_v. According to the above, it holds that there is an $\boldsymbol{\varepsilon}$ such that $B_{uvj}(0, y) + \varepsilon = 2w_{uv} \times MC_j$, which in turn translates through Eqs. (4) and (5) into Eq. (5.3). Note that because $y > z$, it holds that $r_{uj}(0, y) = r_{uj}(0, z) + r_{uj}(z, y)$. The adversary will choose $r_{uj}(z, y) = 0$, because otherwise the OPT must be burdened with the load measured in $r_{uj}(z, y)$. Therefore, Eq. (5.3) can be rewritten as Eq. (5.4).

We observe that due to Eq. (5.4), $r_{uj}(0, z)$ comes into play. The adversary will demand that $r_{uj}(0, z)$ be as large as possible, because only NOM will be burdened with the load measured in $r_{uj}(0, z)$. However, $r_{uj}(0, z)$ cannot be arbitrarily large, because the load variables are reset when they reach RT_j. Therefore, it holds that $r_{vj}(0, y) + r_{uj}(0, z) \leq RT_j$. The adversary will choose the largest value for the above load variables which means that the above inequality becomes Eq. (5.5). We must note that the adversary will choose Eq. (5.5) to be satisfied, provided that NOM first check for a possible migration and then for resetting the load variables.

$$w_{uv}r_{vj}(0, y) = 2w_{uv}MC_j + w_{vu}r_{uj}(0, y) + \varepsilon \tag{5.3}$$

$$w_{uv}r_{vj}(0, y) = 2w_{uv}MC_j + w_{vu}r_{uj}(0, z) + \varepsilon \tag{5.4}$$

$$r_{vj}(0, y) + r_{uj}(0, z) = RT_j \tag{5.5}$$

$$w_{uv}r_{vj}(0, y) = \big(2w_{uv}MC_j + w_{uv}RT_j + \varepsilon\big)/2 \tag{5.6}$$

By solving Eqs. (5.4) and (5.5), we obtain Eq. (5.6). By substituting all of the above into Eqs. (5.1) and (5.2), we obtain the following competitive ratio.

$$\big(4MC_j + RT_j + \varepsilon/w_{uv}\big)/2MC_j \tag{5.7}$$

We define $\boldsymbol{\varepsilon}$ as the network overhead incurred until the NOM identifies that there is a beneficial migration and successfully executes the migration. We expect that $\boldsymbol{\varepsilon}$ is much smaller that MC_j. □

Lemma 6. *The competitive ratio between NOM and OPT is given by Eq. (5.7), provided that the preconditions are the same with those of Lemma 5, with the difference that all datacenters are able to generate load within the cloud.*

Proof. It suffices to show that if datacenters other than n_u and n_v generate load, then the ratio between NOM and OPT lessens compared to that of Eq. (5.7). Notice that the maximum load (associated with s_j) that can be generated within the cloud equals RT_j, irrespective of how many datacenters generate load within the cloud. In Eq. (5.7) we assumed that the total load within the cloud is RT_j. The OPT is burdened with zero network overhead for the above load. On the other hand, NOM is burdened with zero network overhead for each unit of $r_{uj}(0, z)$, while with w_{uv} network overhead for each unit of $r_{vj}(0, y)$.

In the subsequent text, we examine the behavior of the ratio between NOM and OPT when another datacenter (other than n_v and n_u) generates load. We discern the cases where the aforementioned datacenter (hereafter called n_d) is in the same location as: (a) n_x in Fig. 4, (b) n_k in Fig. 4, (c) n_f in Fig. 4, and (d) n_m in Fig. 4.

In terms of (a), for each unit of load of n_x, NOM is burdened with w_{ux} extra network overhead, while OPT with $w_{ux} + w_{vu}$ extra network overhead. We observe that the ratio decreases against that of Eq. (5.7).

Regarding (b), we need to adjust Eqs. (5.1) and (5.2) according to load generated by a datacenter n_d that has the same location as n_k in Fig. 4. Therefore, $w_{ud} \times r_{dj}(0, z) + w_{vd} \times r_{dj}(z, T)$ will be added into Eq. (5.1), while $w_{ud} \times r_{dj}(0, y) + w_{vd} \times r_{dj}(y, T)$ into Eq. (5.2). If n_d is closer to n_v compared to n_u, then the adversary will choose n_d to generate load after time instance z (i.e., $r_{dj}(0, z) = 0$), so that the ratio between NOM and OPT is maximized. On the other extreme, if n_d is closer to n_u, then the adversary will choose $r_{dj}(z, T) = 0$. If n_d is in the half of the distance between n_u and n_d, then by choosing any of the above is equivalent for the competitive ratio. We must note that the adversary will choose $r_{dj}(y, T) = 0$, irrespective of the location of n_d. The above is attributed to the fact that $r_{dj}(y, T) < r_{dj}(z, T)$. Note also that due to the aforementioned facts, it holds that $r_{dj}(z, T) = r_{dj}(z, y)$.

First, we assume that n_d is closer to n_v compared to n_u. According to the above, n_d can generate load during [z, y]. Consequently, the adversary will choose $r_{dj}(0, z) = 0$, and $r_{dj}(y, T) = 0$. Therefore, Eqs. (5.1), (5.2), (5.4), and (5.5) become,

$$OPT(\sigma) = w_{vd}r_{dj}(z, y) + w_{uv}MC_j \tag{5.1}$$

$$NOM(\sigma) = w_{uv}r_{vj}(0, y) + w_{ud}r_{dj}(z, y) + w_{uv}MC_j \tag{5.2}$$

$$\begin{aligned} w_{uv}r_{vj}(0, y) = {} & 2w_{uv}MC_j + w_{vu}r_{uj}(0, z) \\ & + w_{vd}r_{dj}(z, y) - w_{ud}r_{dj}(z, y) + \varepsilon \end{aligned} \tag{5.4'}$$

$$r_{vj}(0, y) + r_{uj}(0, z) + r_{dj}(y, z) = RT_j \tag{5.5'}$$

By combining Eqs. (5.4′) and (5.5′), and solving as per $r_{vj}(0, y)$, we get the equation Eq. (5.6′). To result in the previous equation we use also the fact that $w_{uv} = w_{ud} + w_{vd}$. By substituting Eqs. (5.6′) into (5.2′) and taking the ratio between NOM and OPT, we result in Eq. (5.7′). However, we can see that Eq. (5.7′) is smaller than Eq. (5.7), provided that $r_{dj}(z, y) > 0$. Note that when $r_{dj}(z, y) = 0$, then Eqs. (5.7) and (5.7′) become equal.

$$r_{vj}(0, y) = \frac{2w_{uv}MC_j + w_{uv}RT_j - 2w_{ud}r_{dj}(y, z) + \varepsilon}{2w_{uv}} \tag{5.6'}$$

$$\frac{4MC_j + RT_j + \varepsilon/w_{uv}}{2(w_{vd}/w_{uv})r_{dj}(z, y) + 2MC_j} \tag{5.7'}$$

By choosing n_d being closer to n_u against n_v, the only change is that instead of having $2(w_{vd}/w_{uv})r_{dj}(z, y)$ in the denominator of Eq. (5.7′), we have $2(w_{ud}/w_{uv})$ $r_{dj}(0, z)$. Consequently, the resulting ratio is also strictly smaller than that of Eq. (5.7), provided that $r_{dj}(z, y) > 0$ (otherwise they are equal). On the other hand, by choosing n_d being located in the half of the distance between n_u and n_v, then the ratio is equivalent with Eq. (5.7′). In conclusion, the ratio between NOM and OPT lessens if we demand from a datacenter like n_d to generate load.

Regarding case (c), we follow the same reasoning with case (b) and obtain Eqs. (5.1′), (5.2′), (5.4′) and (5.5′). First, we assume that n_d is closer to n_v against n_u. By combining Eqs. (5.4′) and (5.5′), and solving as per $r_{vj}(0, y)$, we do not result in Eqs. (5.6′) but in (5.6′′). The above is attributed to the fact that the equation $w_{uv} = w_{ud} + w_{vd}$ does not hold in case (c) because of the location of $n_d \equiv n_f$ (see Fig. 4). Specifically it holds that $w_{ud} - w_{uv} < w_{vd}$. Therefore, by incorporating Eqs. (5.6′′) into (5.2′), the ratio between NOM and OPT is given by Eq. (5.7′′). Consequently, the ratio of Eq. (5.7′′) is strictly smaller than that of Eq. (5.7), provided that $r_{dj}(z, y) > 0$ (otherwise they become equal).

$$r_{vj}(0,y) = \left[\begin{array}{c} 2w_{uv}MC_j + w_{uv}RT_j - \\ (w_{uv} + w_{ud} - w_{vd})r_{dj}(y,z) + \varepsilon \end{array}\right] / 2w_{uv} \quad (5.6'')$$

$$\frac{4MC_j + r_{dj}(z,y)(w_{ud} - w_{uv} + w_{vd})/w_{uv} + RT_j + \frac{\varepsilon}{w_{uv}}}{2(w_{vd}/w_{uv})r_{dj}(z,y) + 2MC_j} \quad (5.7'')$$

By choosing n_d being closer to n_u against n_v, the only change is that instead of having $2(w_{vd}/w_{uv})r_{dj}(z, y)$ in the denominator of Eq. (5.7′′), we have $2(w_{ud}/w_{uv})r_{dj}(0, z)$. However, due to the location of $n_d \equiv n_f$ (see Fig. 4), it holds always that $w_{vd} - w_{uv} < w_{ud}$. As a consequence, the resulting ratio is also strictly smaller than that of Eq. (5.7), provided that provided that $r_{dj}(z, y) > 0$. On the other hand, by choosing n_d being located in the half of the distance between n_u and n_v, then the ratio is equivalent with Eq. (5.7′′). Therefore, the ratio between NOM and OPT lessens if the adversary demands from a datacenter like $n_d \equiv n_f$ to generate load.

Regarding case (d), we observe that n_d is closer to n_v than n_u. Due to the above, the adversary will demand from n_d to generate load after time instance z. By applying the same reasoning as previously, we result in Eqs. (5.1′), (5.2′), (5.4′), and (5.5′). By combining Eqs. (5.4′) and (5.5′), and solving as per $r_{vj}(0, y)$, we get the equation Eq. (5.6′′′). To result in the previous equation we use the fact that $w_{ud} = w_{uv} + w_{vd}$ (see Fig. 4 for $n_d \equiv n_m$). By plugging Eqs. (5.6′′′) into (5.2′), we result in the ratio between NOM and OPT, described by Eq. (5.7′′′). We observe that Eq. (5.7′′′) is smaller than Eq. (5.6), under the premise that $r_{dj}(z, y) > 0$.

$$r_{vj}(0,y) = \frac{2w_{uv}MC_j + w_{uv}RT_j - 2w_{uv}r_{dj}(y,z) + \varepsilon}{2w_{uv}} \quad (5.7''')$$

$$\frac{4MC_j + 2r_{dj}(z,y)w_{vd}/w_{uv} + RT_j + \varepsilon/w_{uv}}{2(w_{vd}/w_{uv})r_{dj}(z,y) + 2MC_j} \quad (5.7''')$$

We showed in the preceding text, that when datacenters (other than n_u and n_v) generate load towards s_j, then the ratio between NOM and OPT becomes smaller compared to Eq. (5.7). Therefore, the adversary will demand from datacenters (other than n_u and n_v) not to generate load. According to the above, the competitive ratio between NOM and OPT is captured by Eq. (5.7). □

Lemma 7. *The competitive ratio between NOM and OPT is given by either Eqs. (5.7) or (7.8), provided that: (a) the NOM performs all of the checks/actions; (b) the OPT and NOM perform each at most one migration, respectively; (c) If NOM performs a migration, then it chooses the same destination with OPT; (d) The load variables are infinitely reset; and (e) RT_j is greater than $2MC_j$.*

Proof. Consider that s_j is initially located on n_u. We discern three cases: (a) NOM and OPT do not perform migrations, (b) OPT performs $M_{uvj}(z)$, while NOM does not perform any migration, and (c) OPT performs $M_{uvj}(z)$, while NOM performs $M_{uvj}(y)$. In terms of (a), the ratio between the NOM and OPT is always one (see Lemma 3). Therefore, case (a) does not depend on whether the load variables are reset or not. Regarding the case (b), we have two subcases: (b.i) the load variables are infinitely reset before OPT performs $M_{uvj}(z)$. Consequently, (b.i) reduces to (a). The second subcase (b.ii) is that the load variables are reset after OPT performs $M_{uvj}(z)$. Concerning case (c), we have three subcases to discuss: (c.i) The load variables are infinitely reset before the OPT performs $M_{uvj}(z)$. We can see that (c.i) reduces to (a); (c.ii) The load variables are infinitely reset after the OPT performs $M_{uvj}(z)$ and before NOM performs $M_{uvj}(y)$. We also can observe that (c.ii) reduces to (b.ii); and (c.iii) The load variables are reset after both OPT and NOM perform their migrations. From the above we conclude that the ratio between NOM and OPT is not identified for the cases (b.ii) ≡ (c.ii) and (c.iii).

Investigating (c.iii) further, we observe that after both OPT and NOM perform their respective migrations, they will be burdened with the same network overhead. Therefore, the resets of the load variables can take place by having n_v to generate infinite load, without changing the ratio between them. The above situation is fully investigated by Lemma 5 and Lemma 6. Therefore, the largest ratio of the above situation is captured by Eq. (5.7). All that remains is to find the largest ratio for the case (b.ii), with the procedure of finding such a ratio being explained analytically in the below text. Moreover, in the subsequent text, we also will identify the datacenters that will be selected by the adversary to generate load such that the ratio between NOM and OPT be maximized. Because OPT performs $M_{uvj}(z)$, we conclude that n_v must generate load. We also need to examine whether the adversary will choose another datacenter (other than n_v) to generate load. Consequently, we assume that an extra datacenter (other than n_v) called n_d generates load. All possible cases of n_d are described below and depicted in Fig. 4: (b.ii.1) $n_d \equiv n_v$, (b.ii.2) $n_d \equiv n_x$, (b.ii.3) $n_d \equiv n_m$, (b.ii.4) $n_d \equiv n_k$, (b.ii.5) $n_d \equiv n_f$, and (b.ii.6) $n_d \equiv n_u$.

If (b.ii.1) happens, then only n_d can generate load to reset the load variables. However, because $RT_j > 2MC_j$, it means that NOM will migrate s_j onto n_v. The above contradicts the assumption that we took as (b) that NOM must not perform migrations. When (b.ii.2) takes place, then for each unit of load generated by n_x,OPT is burdened with w_{vx}, while NOM is burdened with w_{ux}. However, w_{vx} is greater than w_{ux} (see Fig. 4). Therefore, the ratio between NOM and OPT decreases when n_x generates load.

In terms of (b.ii.3), (b.ii.4), and (b.ii.5), we observe that the extra datacenter is located in the direction that goes from n_u towards n_v. Therefore, it must hold that $r_{uj}(z, T) > r_{vj}(z, T) + r_{dj}(z, T) - 2MC_j$. Otherwise, NOM must perform a migration towards that direction, which results in a contradiction. Note that in the above cases, n_u does not generate load; therefore, it holds that $r_{uj}(z, T) = 0$. Because $RT_j > 2MC_j$ (precondition (e)), the loads generated by n_d and n_v cannot be reset before the aggregate exceeds $2MC_j$. According to Eq. (6), when both n_v and n_d generate load equaling $2MC_j$, NOM will migrate s_j towards a datacenter across the path between n_u and n_v, which results in a contradiction. Therefore, if we want to reach the threshold we must demand from n_u to generate always load.

If (b.ii.6) happens, then $\mathrm{OPT}(\sigma) = w_{uv}r_{vj}(0, z) + w_{vu}r_{uj}(z, T) + w_{uv}MC_j$, while $\mathrm{NOM}(\sigma) = w_{uv}r_{vj}(0, T)$. During [0, z] both OPT and NOM produce the same network overhead. Therefore, the adversary will choose $r_{vj}(0, z) = 0$. During [z, T], the adversary will choose only n_v to generate load (because OPT is not burdened with this load). However, in the above case, NOM will perform $M_{uvj}(y)$, which results in a contradiction. Therefore, during [z, T] both n_u and n_v generate load such that the load variables be infinitely reset. According to the aforementioned cases, the network overheads produced by OPT($\boldsymbol{\sigma}$) and NOM($\boldsymbol{\sigma}$) are given below (with F denoting the number of resets performed):

$$OPT(\sigma) = F \times w_{vu}r_{uj}(z, T) + w_{uv}MC_j \tag{7.1}$$

$$NOM(\sigma) = F \times w_{uv}r_{vj}(z, T) \tag{7.2}$$

Because NOM must not perform any migration, we mandate that $B_{uvj}(0, y) < 2w_{uv}MC_j$. Note that the ratio between NOM and OPT increases as the load generated by n_v increases. As a result the ratio between NOM and OPT is maximized when $B_{uvj}(0, y) - \varepsilon = 2w_{uv}MC_j$, with $\boldsymbol{\varepsilon}$ tending to zero. Consequently, $B_{uvj}(0, y) \approx 2w_{uv}MC_j$. Combining the above with Eqs. (4) and (5), we obtain Eq. (7.3). For resetting the load variables, we mandate that Eq. (7.4) be satisfied. By combining Eqs. (7.3) and (7.4), and solving as per $w_{vu}r_{uj}(z, T)$ and $w_{uv}r_{vj}(z, T)$, we obtain Eqs. (7.5) and (7.6), respectively.

$$w_{uv}r_{vj}(z, T) \approx 2w_{uv}MC_j + w_{vu}r_{uj}(z, T) \tag{7.3}$$

$$r_{vj}(z, T) + r_{uj}(z, T) = RT_j \tag{7.4}$$

$$w_{vu}r_{uj}(z, T) \approx w_{uv}RT_j/2 - w_{uv}MC_j \tag{7.5}$$

$$w_{uv}r_{vj}(z, T) \approx w_{uv}RT_j/2 + w_{uv}MC_j \tag{7.6}$$

By substituting Eqs. (7.6) in (7.1) and (7.6) in (7.2), and eliminating the common factor w_{uv}, we obtain the ratio (between NOM and OPT) described, as given by Eq. (7.7). The aforementioned ratio is maximized when F tends to infinity, as given in Eq. (7.8).

$$F\big(RT_j + 2MC_j\big)/\big[F\big(RT_j - 2MC_j\big) + 2MC_j\big] \tag{7.7}$$

$$\big(RT_j + 2MC_j\big)/\big(RT_j - 2MC_j\big) \tag{7.8}$$

Previously we examined the case where exactly two datacenters generate load. We concluded that the aformenetioned datacenters are n_u and n_v. Next, we investigate the case where extra datacenters (other than n_u and n_v) generate load. First of all, if the extra datacenters are located in the opposite direction (from n_u towards n_v), then OPT is burdened with more network overhead compared to that of NOM for each unit of load that comes from those datacenters. The above is because in case of NOM s_j is placed closer to the extra datacenters as compared to the case of OPT. On the other hand, if the aforementioned datacenters are located in the direction from n_u towards n_v, then it must hold that $r_{uj}(z, T) > r_{vj}(z, T) + r_{dj}(z, T) - 2MC_j$. In the above inequality, $r_{dj}(z, T)$ represents the load generated by the extra datacenters. Given that $r_{uj}(z, T) + r_{vj}(z, T) + r_{dj}(z, T) = RT_j$, the above inequality means that the load generated by n_v when only n_u and n_v generate load, it must be shared among the extra datacenters. However, we notice that OPT is burdened with zero load regarding the load generated by n_v. Therefore, in case of extra datacenters, OPT will be burdened with the load generated by n_d. The above means that the ratio between the NOM and OPT decreases. Consequently, under the preconditions of this lemma, the competitive ratio between the NOM and OPT is given by either Eq. (7.8) as captured in case (b.ii) or by Eq. (5.7) as described by case (c.iii). □

Corollary 1. *We assume the same preconditions of Lemma 7, with the difference being that in precondition (d) the number of resets are not infinite but finite. As a result of which, the competitive ratio is given by either Eqs. (5.7) or (7.7), with the latter being strictly smaller than Eq. (7.8).*

Lemma 8. *NOM and OPT choose the same destination when migrating a service.*

Proof. Assume that OPT makes a decision to migrate s_j onto n_v, and NOM takes the decision to migrate s_j onto n_d. Therefore, according to Eq. (6), n_d must generate more load than n_v. The above means that OPT did not take the optimal solution, which results in a contradiction. □

Lemma 9. *The competitive ratio is given by either Eqs. (5.7) or (7.8), and it does not depend on the number of migrations performed.*

Proof. According to the previous lemmas and Corollary 1, we have that the competitive ratio of NOM is given by either Eqs. (5.7) or (7.8). By incorporating a number of N migrations into the above equations, we can see that both the enumerators and denominators are multiplied by N. □

Theorem 1. *NOM is 3.8-competitive, provided that the underlying network is structured as a tree and the reset threshold is equal to that of $3.5MC_j$.*

Proof. According to Lemma 9, the competitive ratio between NOM and OPT is given by Eqs. (5.7) or (7.8). Therefore, we must identify the domination between them. We can observe that when RT_j attains large values of the order of $10MC_j$, it holds that Eq. (5.7) is greater than Eq. (L7.8). On the other hand, when RT_j tends to be equal to that of $2MC_j$, it holds that Eq. (7.8) is greater than Eq. (5.7). Therefore, we equate the above equations

to find the value of RT_j that simultaneously minimizes them. Before equating them, we must make an assumption about ε. As explained in Lemma 5, we expect that ε will be much smaller than MC_j (i.e., $\varepsilon << MC_j$). We solve the aforementioned equation by generously assuming that $\varepsilon/w_{uv} = MC_j/10$. By equating Eqs. (7.8) and (5.7), we find two roots of RT_j, one positive and one negative. Because RT_j cannot attain negative values, we exclude the negative root from our investigation. Consequently, Eqs. (7.8) and (5.7) become almost equal when $RT_j \approx 3.5MC_j$, with the competitive ratio becoming 3.8.

$$r_{mj} > r_{uj} + \sum_{\forall n_k \in A(n_u)=1, n_k \neq n_m} r_{kj}, \forall n_m \in A(n_u) = 1 \quad (10)$$

□

Theorem 2. *Given that an 1-hop migration of s_j is performed from n_u onto n_m, with n_m satisfying Eq. (10), then the network overhead decreases.*

Proof. In case s_j migrates onto n_m, then the network overhead increases by an amount equaling the second part of the above inequality. The above is justified by the fact that after s_j migrates onto n_m, it distances itself from the load associated with n_u or datacenters using n_u to communicate with n_m. The aforementioned load equals the second part of the above inequality. On the other hand, the network overhead increases by r_{mj}. The above is attributed to the fact that after s_j migrates onto n_m, s_j comes closer to the load associated with n_m and datacenters using n_m to communicate with n_u. The aforementioned load equals r_{mj}. Therefore, the total network overhead decreases when s_j migrates on n_m. In case s_j does not migrate on n_m, then the network overhead increases. □

Theorem 3. *The competitive ratio dramatically increases when the migration threshold is quite small.*

Proof. Consider the case of two datacenters n_u and n_v, with s_j being initially hosted on n_u. Assume that the migration threshold is one. Therefore, NOM migrates s_j onto n_v, when it holds that $B_{uvj} \geq 1$. Consider also that we have the time instances $z_1 < z_2 \ldots < z_n$. In case n_v generates load of one unit at z_1, then NOM will perform $M_{uvj}(z_1)$. In the sequel, if n_u generates load of one unit at z_2, then NOM will perform $M_{vuj}(z_2)$. If the above pattern repeats itself n times, then $\mathrm{OPT}(\sigma) = n$, while $\mathrm{NOM}(\sigma) = n \times 2MC_j + 2n$. Therefore, the ratio between NOM and OPT becomes $2MC_j + 2$. It can be seen that by increasing the migration threshold, the ratio between NOM and OPT decreases. The aforementioned ratio is minimized when the migration threshold is twice the network overhead of performing a migration from n_u onto n_v. □

7 Experimental Evaluation

This section presents an evaluation of the NOM algorithm based on simulations that performed using NS2 [26].

7.1 Setup

For the cloud infrastructure, 25 different (tree-structured) datacenter networks were generated. The number of hosted services was between 30 and 150 depending on the network size. The initial service placement on datacenters was on random. The number of client entry points to the cloud infrastructure varied by 10, 20, 40, and 60 percent the number of datacenters, chosen randomly.

Clients were clustered in groups, depending on their entry points. Unless otherwise stated, we assume that a client has two service usage modes M_H and M_L, with the first one generating ten times more request/reply traffic than the second one. To reflect the dynamic changes in client traffic, we let the clients alternate between the two modes periodically. We consider four different types of traffic variability, ultra-high (UH), high (H), low (L) and ultra-low (UL), where the period during which the clients stay within the same mode was randomly determined from a uniform distribution [1, 10], [1, 100], [50, 500], and [100, 1000], respectively. Finally, we differentiate between three client families F_1, F_2 and F_3, where at most 5, 10 and 20 percent of clients that use a specific service can be in M_H mode at the same time.

As a yardstick for the quality of the solutions derived by NOM, we used a *static* offline optimal algorithm (SOPT) that had *a priori* knowledge of the total request/reply traffic (see [15] assuming no dependencies between VMs). Note that we could exhaustively solve the problem in an optimal offline dynamic way; however, such an approach would lead in an acceptable execution time. To illustrate the working of SOPT, we use the following example. Let client groups q_1 and q_2 with entry points n_1 and n_2, respectively, generate traffic towards service s_j initially hosted on n_1. Datacenter n_1 is one hop away from n_2, and the the network overhead of migrating s_j between these two datacenters is one. Moreover, assume that during the time interval [1,5], the traffic for q_1 and q_2 is ten and 100 bytes, but then changes to 40 and ten bytes, respectively, for the interval [6,10]. An online optimal algorithm would decide to migrate s_j from n_1 to n_2 at time unit one, and then back to n_1 at time unit six, resuting in a total network overhead of 22 bytes (including the service migration network overhead). In the case of SOPT, the total traffic of q_1 and q_2 for s_j is 110 and 50 bytes, respectively, and the optimal static placement for s_j is to be hosted on n_2. Therefore, SOPT migrates s_j from n_1 to n_2 at time unit one that results in a total network ovherhead of 51 bytes (including the migration network overhead). SOPT performs all service migrations in the first time slot, and the placement remains unchanged for the entire duration of the experiment.

In addition, we compared NOM with the MIG algorithm presented in [4]. MIG divides time into epochs and makes the following considerations for each service s_j. In each epoch, MIG monitors for each datacenter n_u, the network overhead of serving all requests from this epoch by placing service s_j on n_u. This network overhead is kept in a variable named L_{uj}. MIG keeps the service s_j at n_u until L_{uj} reaches $\boldsymbol{\beta}$. Thereafter, MIG migrates s_j to n_w chosen uniformly at random among datacenters with the property $L_{wj} < \boldsymbol{\beta}$. If such a datacenter does not exist, MIG does not migrate s_j. For the next epoch, variables L_{uj} are reset to zero. In our experiments, β was expressed as a migration threshold factor multiplied by the migration network overhead of the service. The crucial difference between our NOM and MIG is that the latter performs a hard reset of load variables after

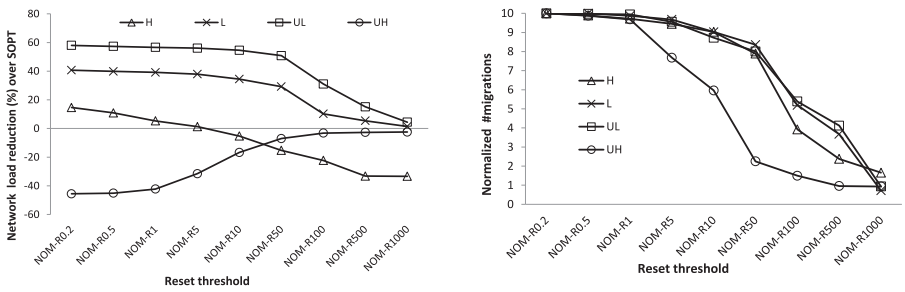

Fig. 5. Performance of NOM when varying its reset threshold and keeping fixed the migration threshold at 0.1.

starting a new epoch, while NOM does this based on our theoretical analysis.

7.2 Experiments

The experimental evaluation was split into three sections. Specifically, the first section concerns how the parameters of NOM and MIG affect their performance for various client traffic patterns and a fixed client family (F2). The second section investigates the behavior of NOM and MIG for different client families and a fixed client traffic pattern (L). Last, the third section compared NOM and MIG for both different traffic patterns and client families. In all experiments, the thresholds of NOM and MIG were set in a relative way, expressed as a factor of the service migration overhead. As defined previously, a migration/reset threshold equals the migration/reset threshold factor multiplied by the network overhead of migrating the respective service. From here onwards, when referring to a migration/reset threshold, it will mean the migration/reset threshold factor.

7.2.1 Varying the Client Traffic Patterns

In a first experiment we vary the reset threshold of NOM with the migration threshold fixed at 0.1. The above choice is justified by the fact that the smaller the value of the migration threshold the greatest the range of the eligible values for the reset threshold. Note that the reset threshold was always greater than the migration threshold because otherwise the migrations were suppressed.

Each variant was denoted as NOM-Rx, with x being the reset threshold. The experiment was conducted for all client traffic patterns UH, H, L, UL. Fig. 5a shows the network load reduction that was achieved vs. SOPT.

As it can be seen, the performance of NOM improved for H, L, and UL as the reset threshold increased. The opposite holds for UH. Recall that a smaller reset threshold made NOM more reactive to traffic changes which in turn led to more migrations. When traffic changes were very frequent, it led into migrations that could not be amortized. On the contrary, when traffic changes were not so frequent, migrations were more likely to be amortized, while delaying the migration decision (by having a higher threshold) did not bring any advantages. Overall, the greatest improvement versus SOPT was achieved for H and L. As expected, the improvement was comparatively small for UL, while no improvement was achieved for UH. The best variant was R10 closely followed by R50.

Fig. 5b shows the number of migrations. The reported values were normalized to the largest number of migrations performed in each case. We applied a separate normalization for each traffic pattern, as it was not meaningful to compare the number of migrations for different traffic patterns. As discussed above, the number of migrations decreased when increasing the reset threshold. The rate of this decrease was higher for UH, which further confirmed that in this case small reset thresholds made NOM overly reactive.

In a second experiment, we showed the behavior of NOM when varying the migration threshold while keeping the reset threshold constant at 10 (R10 was the best overall variant in the previous experiment). Note that the migration threshold could not be bigger than the reset threshold since this would suppress all migrations. As can be seen in Fig. 6a, the trends were similar to those of the previous experiment. However, for UH, the performance degraded for migration thresholds beyond 5.

As previously, the (normalized) number of migrations performed decreased as the migration threshold increased (see Fig. 6b), but the rate of decrease was stronger in all cases. The above was reasonable since the migration threshold suppressed migrations in a direct way. In fact, the number of migrations reached almost zero when the reset threshold was equal to the migration threshold.

As we can see in Fig. 7, considering the traffic patterns H, L, and UL the performance of MIG deteriorated as we

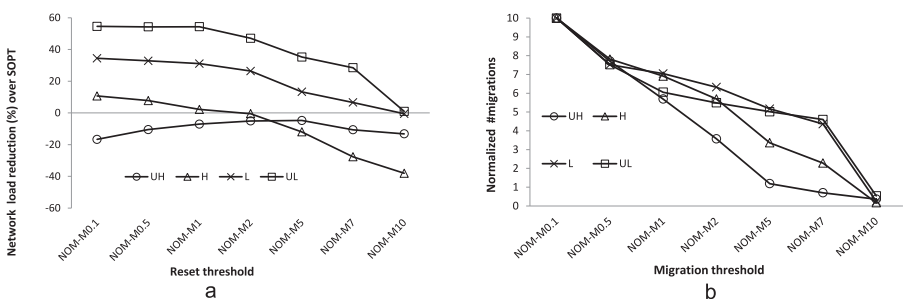

Fig. 6. Performance of NOM when varying its migration threshold and keeping fixed the reset threshold at 10.

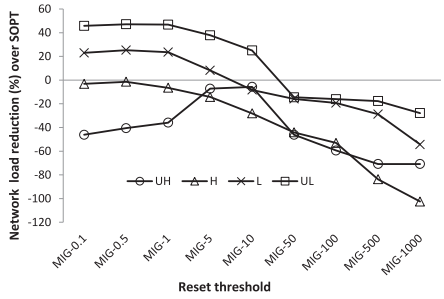

Fig. 7. Performance of MIG when varying its migration threshold.

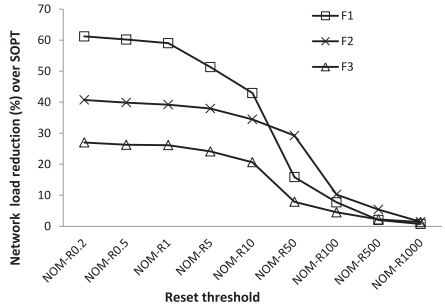

Fig. 9. Performance of NOM when varying the reset threshold and keeping fixed the migration threshold at 0.1.

increased its migration threshold. Specifically, in case of H, SOPT performed always better than MIG. On the other hand, for the case of L and UL MIG degraded to a point (MIG-10 for L and MIG-50 for UL) where from that point onwards MIG became worse than SOPT. Concerning UH, MIG was consistently outpaced by SOPT. Particularly, MIG resulted in better outcomes when increasing the migration threshold up to the point of MIG-M5. After that point, MIG worsened. Considering all of the client families, it is seen that MIG achieved best results when the migration threshold equaled 5.

Last, in Fig. 8, it is shown that the number of migrations lessened as the migrations threshold increased. It is remarkable that the number of migrations approached to zero when the migration threshold exceeds 50. The aforementioned is justified by the fact that when the migration threshold equaled 50, it means that no migration could be performed unless the total load collected was more than 50 multiplied by the size of the service.

7.2.2 Varying the Client Families

In this section, we conducted a series of experiments to examine the behavior of NOM and MIG compared to SOPT when varying the client families. Note that the traffic pattern was fixed (L).

Particularly, in the first experiment we considered the network load reduction of NOM when varying the client families and the reset threshold. As we can see (Fig. 9), NOM performance dropped when increasing the reset threshold. As explained earlier, the above is reasonable. It is also remarkable that when the reset threshold ranged from 0.2 to 10, the network load reduction achieved by NOM against SOPT in F1 was greater than that of NOM in F2, and by far better than that of NOM in F3. The reason of the above is that it was more probable for NOM to result in beneficial migrations when the number of clients changing their traffic patterns decreased. The above did not hold when the reset threshold ranged from 50 to 1000. The justification is that when the reset threshold increased excessively, then the load variables recorded huge amounts of network load. However, when the above happens, it is intrinsically more probable that some beneficial migrations be suppressed. Therefore, services may be locked-in at their current hosts. Unambiguously, NOM achieved best overall results in case of NOM-R0.2, with NOM-R0.5 and NOM-R1 being slightly worse.

As demonstrated in Fig. 10, the variants of NOM followed the same trend as that of the previous experiment (see Fig. 9). The main difference between Figs. 9 and 10 was that when varying the migration threshold, NOM became worse against varying its reset threshold. Another difference was that when increasing excessively the migration threshold at 10, NOM was inferior against SOPT; on the other hand, in case of setting the reset threshold to 1000, NOM remained slightly superior to SOPT. Last, it is seen that NOM in F1 became worse that that in F2 and equal to that in F3. The above was due to the services locked-in at their current hosts when a great amount of network load accumulated on the load variables (see the explanation in the previous paragraph).

Last in Fig. 11, we showed the behavior of MIG when varying the migration threshold size and keeping the same settings as previously. The trends remained the same as previously. It seems that services were locked-in at their current hosting datacenters when the migration threshold ranged from 10 to 1000.

7.2.3 Comparing the Best Variants of NOM and MIG

In this section, we conducted a series of experiments to evaluate the performance of best variants of NOM and MIG.

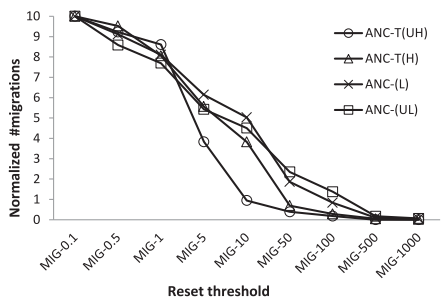

Fig. 8. Normalized number of migrations performed when varying the migration threshold of MIG.

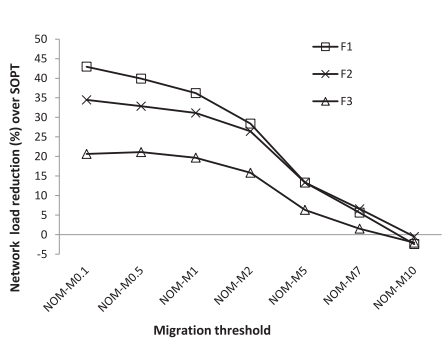

Fig. 10. Performance of NOM when varying the migration threshold and keeping fixed the reset threshold at 10.0.

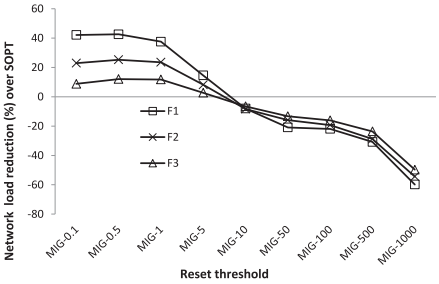

Fig. 11. Performance of MIG when varying the migration threshold.

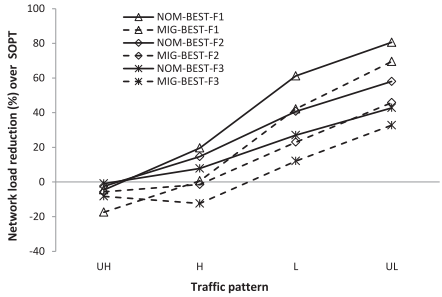

Fig. 13. Performance of best variants of NOM and MIG for each traffic pattern.

Specifically, in Fig. 12, we demonstrated the performance of best variants of NOM and MIG against SOPT under varying client families and traffic patterns. It must be noted that we have chosen NOM-R10 and MIG-5 as best variants of NOM and MIG, respectively. As shown, NOM achieved by far superior results against MIG irrespective of the chosen traffic pattern and client family. As observed NOM achieved network load reduction up to 70 percent against SOPT, while up to 30 percent against MIG.

In Fig. 13, we chose the best variant of NOM and MIG for each traffic pattern. As it can be seen from Fig. 5a and Fig. 6a, the best variant of NOM for the traffic pattern UH was NOM-R1000, while for the traffic patterns H, L, and UL was NOM-R0.2. On the other hand, it is observed in Fig. 7 that MIG-0.5 achieved the best results in terms of H, L, UL. Note that MIG-10 was superior against its counterparts regarding UH. In Fig. 13, the aforementioned variants were called NOM-BEST and MIG-BEST. As depicted in that figure, the NOM approached the performance of SOPT when the chosen traffic pattern was UH. Specifically, NOM achieved network load reduction of -4.5, -2.5, and -0.9 percent against SOPT when the client family was F1, F2, and F3, respectively. It is remarkable that when the traffic pattern was H, L, and UL, NOM achieved network load reduction from 7.8 up to 80 percent against SOPT. On the other hand, MIG achieved network load reduction from -17.5 up to -5.7 percent against SOPT under UH; while for the rest traffic patterns its network load reduction compared to SOPT ranged from -1.4 up to 69 percent. It must also be noted that by delving into the data of Fig. 13 we found that NOM achieves a performance difference against MIG of at least 10 percent (up to 36 percent) for F1, at least 3 percent (up to 22 percent) for F2, and at least 6 percent (up to 15 percent) for F3.

Last in Fig. 14, we showed the (normalized) number of migrations performed between NOM-BEST and MIG-BEST variants. As depicted in the figure, NOM performed always more migrations against MIG. Specifically, when delving into the details of both Figs. 13 and 14, we observed the following fact. When the difference between the performance of NOM and MIG increased, then we had also an increase in the difference between the number of migrations performed by NOM and MIG. The above meant that better performance reflected to more migrations.

The reason that NOM outperforms MIG is that the latter resets the load variables in a harsh way when an epoch ends without considering the collected load. On the other extreme, NOM resets the load variables in a sophisticated way, which is based on the collected load. By reducing the duration of an epoch it is likely to result in the following situation. Consider two services that the optimal algorithm would decide to migrate them. Assume that the load related with the first service increases quickly, while the load of the second service increases slowly. If the epoch is reduced, then MIG may migrate the first service in the same way as that of our algorithm. On the other hand, MIG may never migrate the second service, because the load variables may never reach the migration threshold.

8 Conclusions and Future Directions

In this work, we introduced the problem of deciding at what point in time a service must be migrated to reduce the network overhead. We proposed the network overhead migration algorithm (called NOM) as solution to the above problem. We gave an analytical proof about NOM is 3.8 competitive when the underlying network is structured as a tree. We conducted experimental evaluation to compare NOM and the best known online algorithm within the current literature (called MIG) to a static offline optimal algorithm (SOPT). We found that NOM outpaces MIG in all scenarios, while it is defeated by SOPT only in case of traffic patterns that change very frequently.

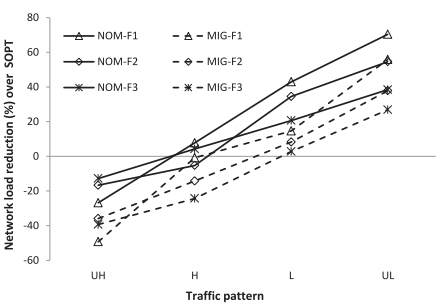

Fig. 12. Performance of NOM-R10 and MIG-5 when varying both client families and traffic patterns.

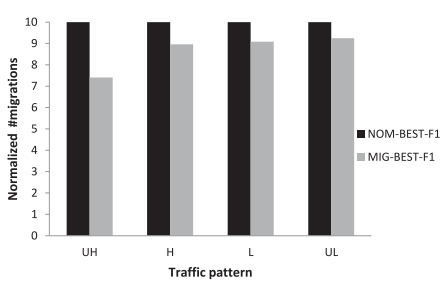

Fig. 14. Performance of best variants of NOM and MIG for each traffic pattern.

In the future, we plan to **(a)** incorporate machine learning techniques to predict the underlying system dynamics for improved performance of our online algorithms. **(b)** Adapt NOM to work on general graph networks according to the intuition given in Section 4.

Acknowledgments

Samee U. Khan's work was supported by (while serving at) the National Science Foundation (NSF). Nikos Tziritas' work was supported by NSFC and PIFI International Scholarship under the grants 61550110250 and 2017VCT0001, respectively. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation for cloud data centers," *Elsevier J. Netw. Comput. Appl.*, vol. 52, pp. 11–25, 2015.

[2] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. INFOCOM*, 2012, pp. 963–971.

[3] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy performance efficient dynamic consolidation of virtual machines in cloud data centers," *J. Concurrency Comput.: Practice Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[4] M. Bienkowski, et al., "Competitive analysis for service migration in vnets," in *Proc. SIGCOMM (workshop)*, 2010, pp. 17–24.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[6] N. Buchbinder and J. Naor, "The design of competitive online algorithms via a primal-dual approach," *Found. Trends Theoritical Comput. Sci.*, vol. 3, no. 2/3, pp. 93–263, 2009.

[7] B. Dietrich, D. Goswami, S. Chakraborty, A. Guha, and M. Gries, "Time series characterization of gaming workload for runtime power management," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 260–273, Jan. 2015.

[8] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," in *Proc. 35th Annu. ACM Symp. Theory comput.*, 2003, pp. 448–455.

[9] J. Guo, et al., "On efficient bandwidth allocation for traffic variability in datacenters," in *Proc. IEEE INFOCOM*, 2014, pp. 1572–1580.

[10] J. Guo, F. Liu, J. Lui, and H.-J. Jin, "Fair network bandwidth allocation in iaas datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.

[11] J. Guo, F. Liu, D. Zeng, J. C.S. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *Proc. IEEE INFOCOM*, 2013, pp. 2139–2147.

[12] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. INFOCOM*, 2012, pp. 2876–2880.

[13] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Comput.*, vol. 12, no. 1, pp. 1–15, 2009.

[14] P. Lama and X. Zhou, "Coordinated power and performance guarantee with fuzzy MIMO control in virtualized server clusters," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 97–111, Jan. 2015.

[15] C. H. Lee and K. G. Shin, "Optimal task assignment in homogeneous networks," *IEEE Trans. Comput.*, vol. 8, no. 2, pp. 119–129, Feb. 1997.

[16] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. INFOCOM*, 2010, pp. 1–9.

[17] Y. O. Oswald, S. Schmid, and R. Wattenhofer, "Tight bounds for delay-sensitive aggregation," in *Proc. 27th ACM Symp. Principles Distrib. comput.*, 2008, pp. 195–202.

[18] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2647–2660, Nov. 2014.

[19] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra, "Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration," in *Proc. 39th Int. Conf. Parallel Process.*, 2010, pp. 228–237.

[20] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 331–340.

[21] N. Tziritas, S. U. Khan, T. Loukopoulos, S. Lalis, C.-Z. Xu, and P. Lampas, "Distributed online algorithms for the agent migration problem in WSNs," *ACM/Springer Mobile Netw. Appl.*, vol. 18, no. 15, pp. 622–638, 2013.

[22] N. Tziritas, S. U. Khan, T. Loukopoulos, S. Lalis, C.-Z. Xu, and P. Lampsas, "Single and group agent migration: algorithms, bounds, and optimality issues," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3143–3161, Dec. 2014.

[23] F. Xu, F. Liu, H. Jin, and A. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proc. IEEE*, vol. 102, no. 1, pp. 11–31, Jan. 2014.

[24] F. Xu, F. Liu, L. Liu, H. Jin, B. Li, and B. Li, "iAware: Making live migration of virtual machines interference-aware in the cloud," *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3012–3025, Dec. 2014.

[25] Q. Zhang, Q. Zhu, M. F. Zhani, and R. Boutaba, "Dynamic service placement in geographically distributed clouds," *IEEE J. Selected Areas Commun.*, vol. 31, no. 12, pp. 762–772, Dec. 2012.

[26] Network Simulator2 (ns2). (1997). [Online]. Available: http://www.isi.edu/nsnam/ns/

Nikos Tziritas received the BSc degree from the Technological Educational Institute of Serres, Greece, in 2004, and the MSc and PhD degrees from the University of Thessaly, Greece, in 2006 and 2011, respectively. He currently works in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His work has appeared in more than 35 publications. He received the Award for Excellence for Early Career Researchers in Scalable Computing from IEEE Technical Committee in Scalable Computing.

Samee U. Khan received the PhD degree from the University of Texas, in 2007. Currently, he is the Department head and the James W. Bagley chair of Electrical & Computer Engineering at the Mississippi State University (MSU). Before arriving at MSU, he was Cluster lead (2016-2020) for Computer Systems Research at National Science Foundation and the Walter B. Booth professor at North Dakota State University. His research interests include optimization, robustness, and security of computer systems. His work has appeared in more than 400 publications. He is associate editor-in-chief of the IEEE IT Pro, and an associate editor of *Journal of Parallel and Distributed Computing* and *ACM Computing Surveys*.

Thanasis Loukopoulos received the Diploma in computer engineering and informatics from the University of Patras, Greece, in 1997. He received the PhD degree in computer science from the Hong Kong University of Science and Technology (HKUST), in 2002. Currently, he is an assistant professor in University of Thessaly, Greece. He has published 50 papers, and had the best paper award in ICPP 2001.

Spyros Lalis received the Diploma and PhD degree in computer science from ETH Zurich. He is and associate professor in the Electrical and Computer Engineering Department, University of Thessaly, Greece, and research associate in the Center for Research & Technology Thessaly (CERETETH). His main research interests are programming environments, operating systems, parallel & distributed systems, and ubiquitous computing systems.

Cheng-Zhong Xu received the PhD degree in computer science from the University of Hong Kong, in 1993. He is currently a professor in the Department of Electrical and Computer Engineering of Wayne State University, and the director of Cloud and Internet Computing Laboratory (CIC) and Sun's Center of Excellence in Open Source Computing and Applications (OSCA). His research interest is mainly in scalable distributed and parallel systems and wireless embedded computing devices. He has published two books and more than 160 articles in peer-reviewed journals and conferences in these areas. He is a fellow of the IEEE.

Keqin Li is a SUNY distinguished professor of computer science. He has published more than 350 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *Journal of Parallel and Distributed Computing*. He is a fellow of the IEEE.

Albert Y. Zomaya is the chair professor of High Performance Computing & Networking in School of Information Technologies, University of Sydney. He is the author/co-author of seven books, more than 400 papers, and the editor of 12 books and 15 conference proceedings. He was the editor in chief of the IEEE Transactions on Computers and servers as an associate editor for 19 leading journals. He is a fellow of the IET and the AAAS.