Distributed Approach to the Holistic Resource Management of a Mobile Cloud Network

William Tärneberg[1], Alessandro Vittorio Papadopoulos [2], Amardeep Mehta [3], Johan Tordsson [3], Maria Kihl [1]
[1]Department of Electrical and Information Technology, Lund University, Sweden
[2]Mälardalen University, Sweden
[3]Department of Computing Science, Umeå University, Sweden

***Abstract*—The Mobile Cloud Network is an emerging cost and capacity heterogeneous distributed cloud topological paradigm that aims to remedy the application performance constraints imposed by centralised cloud infrastructures. A centralised cloud infrastructure and the adjoining Telecom network will struggle to accommodate the exploding amount of traffic generated by forthcoming highly interactive applications. Cost effectively managing a Mobile Cloud Network computing infrastructure while meeting individual application's performance goals is non-trivial and is at the core of our contribution. Due to the scale of a Mobile Cloud Network, a centralised approach is infeasible. Therefore, in this paper a distributed algorithm that addresses these challenges is presented. The presented approach works towards meeting individual application's performance objectives, constricting system-wide operational cost, and mitigating resource usage skewness. The presented distributed algorithm does so by iteratively and independently acting on the objectives of each component with a common heuristic objective function. Systematic evaluations reveal that the presented algorithm quickly converges and performs near optimal in terms of system-wide operational cost and application performance, and significantly outperforms similar naïve and random methods.**

I. Introduction

With the advent of resource virtualisation and disaggregation in 5th-generation mobile networks as well as Edge- and Fog-computing, forthcoming cloud infrastructures are poised to be geographically distributed and capability- and cost-heterogeneous. In the literature, this paradigm goes by many names, such as; *Fog Computing* [8], *Telco-Cloud* [10], *Edge-cloud* [16], and *Mobile Cloud* [13], [14], [18]. Because of the network focus in this work, it is referred to as the Mobile Cloud Network (MCN) [17].

The MCN topological paradigm will proposedly enable and drive new types of services and applications that exploit the increased proximity to the end users and key infrastructure components.Contemporary cloud resources are housed in centralised Data Centres (DCs) that are separated from the end-users by the intermediate Wide Area Networks (WANs), core, and access networks. The added latency and weak-backhaul introduced by those networks has proven to inhibit the performance of cloud-based applications [2]. Furthermore, there is a large and growing set of mission critical real-time applications such as tele-robotic surgery [1], Radio Base Station (RBS) baseband signalling [11], gaming [15], and Augmented Reality (AR) [12] that are unable to operate in such a latency-, jitter-, and throughout-uncertain environment, provided by a centralised cloud paradigm. The decreased distance between the cloud infrastructure and the end-users, provided by an MCN, reduces the Round-Trip Time (RTT) and jitter, increases availability, and fault-tolerance [29] for the infrastructure's resident cloud applications.

To operate a viable MCN infrastructure, its operator needs to administer the admitted applications and the system's resources such that; resources are not over-provisioned, total operational cost is minimised, and that all applications' performance requirements are met. When managing an MCN, its operator's primary degree of freedom is the placement of the system's resident applications. Continuously and scalably evaluating the placement of a vast set of heterogeneous applications over a set of heterogeneous nodes is non-trivial and is the fundamental problem addressed in this paper.

Optimally placing the resident applications in an MCN, given the constraints above, is NP-hard [27]. Furthermore, the optimal placement of the MCN's resident applications was explored in our previous work [30], where it was concluded that a centralised solution is not scalable because it fundamentally fails to keep up with the system's rate of change.

The problem of placing applications in a cloud environment has been addressed in the literature for content routing [22], intra- and inter-DC application placement [5], [19], and in optimal content distribution in Content Delivery Networks (CDNs) [9]. To the best of our knowledge, none of the presented approaches simultaneously and holistically consider a set of criteria that are synonymous with a vast resource cost- and capacity-heterogeneous infrastructures with a high geographic granularity.

The broad challenges of Virtual Machine (VM) placement across DCs are taxamonised in [20] and formalised in [25]. The literature contains work on the placement of applications and their constituent VMs in DCs, to minimise cost [5], energy consumption [4], data-locality [24], network usage [6]. These methods primarily address the internal objectives of a DC and therefore inherently disregard the geographical discrepancy to the end users and the heterogeneity in both applications and infrastructure. Thus, they cannot be applied to this problem. Additionally, the internal administration of an MCN's DCs is beyond the scope of this work.

Furthermore, CDNs share much of the same distributed topological properties of an MCN but operate with the objective of maximising the hit-rate of a set of content over a finite

978-1-5090-3047-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICFEC.2017.10

set of resources as a function of the of content's popularity. In a CDN, content is static and resource usage is often not proportional to the demand and is confined to storage. Additionally, no performance guarantees can be given for all applications and instant scalability needs are not a concern. In contrast, in an MCN resources are heterogeneous and applications are highly dynamic with heterogeneous performance requirements that all must be accommodated.

The contributions of this paper are three-fold. Firstly, the paper outlines the application placement challenges and dynamics in an MCN on which a model is constructed, and presented in Section II. Secondly, this paper contributes with an iterative distributed algorithm that solves the application placement challenge. The algorithm takes a holistic approach by accommodating the system's primary objectives over a neighbourhood of DCs. By doing so the algorithm can accommodate the heterogeneous resources and applications in the system, without incurring additional cost and application placement oscillations. The algorithm is defined in section III. Thirdly, the algorithm is evaluated over a set of infrastructure topologies and contrasted with an optimal and a naïve method, detailed in Section IV. The results of the evaluation presented in Section V show that the algorithm can quickly and consistently converge while meeting all constituent entities' objectives. It is also shown that the algorithm approaches the system's optimal cost point within 8% and in reasonable time. Furthermore, the algorithm also outperforms the naïve method, both in term of convergence and cost. The evaluations also reveal some of the distinct challenges with the different topologies. Section VI summarises the results and provides a discussion on possible continuations of this work.

II. MCN System model

In this section a system model of the MCN is detailed. The model is used for defining the presented algorithm as well as for constructing a simulated environment for evaluating the algorithm. The model captures the fundamental challenges and properties of an MCN infrastructure. The model is an extension of our previous work [30]. An overview of the model's components can be seen in Figure 1.

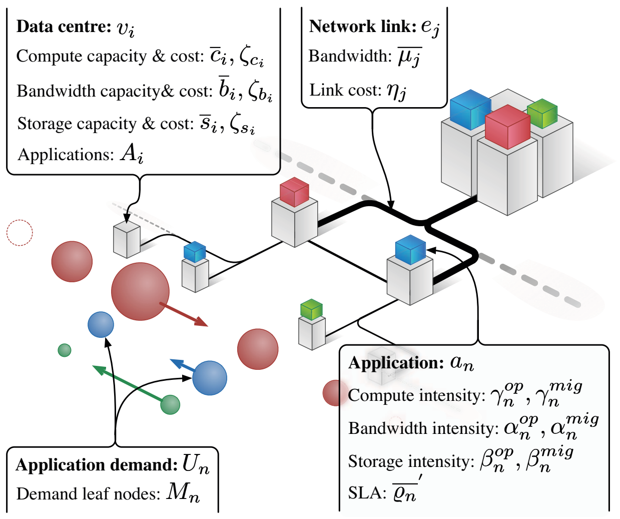
A. Topology

An MCN infrastructure is modelled as an undirected graph where the vertices are DCs and the edges are network links, each with a set of finite resources. Applications admitted to the MCN are hosted in DCs and are subject to a demand through the network links, originating at the graph's leaf, i.e. vertices with degree one. Thus, let the graph $G = (\mathcal{V}, \mathcal{E})$ denote an MCN topology, where

$$\mathcal{V} = \{v_i \mid i = 1, 2, ..., I\}, \tag{1}$$

$$\mathcal{E} = \{e_j \mid j = 1, 2, ..., J\}, \tag{2}$$

See Figure 1 for an illustration of the system's topology.



Data centre: v_i
Compute capacity & cost: $\overline{c}_i, \zeta_{c_i}$
Bandwidth capacity& cost: $\overline{b}_i, \zeta_{b_i}$
Storage capacity & cost: $\overline{s}_i, \zeta_{s_i}$
Applications: A_i
Network link: e_j
Bandwidth: $\overline{\mu_j}$
Link cost: η_j
Application: a_n
Compute intensity: $\gamma_n^{op}, \gamma_n^{mig}$
Bandwidth intensity: $\alpha_n^{op}, \alpha_n^{mig}$
Storage intensity: $\beta_n^{op}, \beta_n^{mig}$
SLA: $\overline{\varrho_n}'$
Application demand: U_n
Demand leaf nodes: M_n

Fig. 1: Model overview with entities and their properties

B. Data centre model

In an MCN, traditional centralised DCs are supplemented by a large set of geographically dispersed DCs that are embedded into a Mobile Network Operator (MNO)'s infrastructure. The DCs within an MCN are both capacity- and cost-heterogeneous.

A vertex v_i in the graph has the following capacities, expressed as real positive numbers: compute capacity $\overline{c_i}$, storage capacity $\overline{s_i}$, and bandwidth $\overline{b_i}$. The resource requests of a DC are aggregated into resource units. These units can be seen as VMs or containers. A resource unit is defined by a compute capacity $\overline{c}_{VM}$, a storage capacity $\overline{s}_{VM}$, and a bandwidth $\overline{b}_{VM}$, expressed as real numbers, where $\overline{c}_{VM} \ll \overline{c_i}$, $\overline{s}_{VM} \ll \overline{s_i}$, $\overline{b}_{VM} \ll \overline{b_i}$. The momentary utilisations of these resources are expressed as real numbers; compute utilisation c_i, storage utilisation s_i, and bandwidth utilisation b_i. A DC is assumed to be able to accommodate any set of applications that aggregately do not exceed its capacity.

Additionally, a vertex v_i is associated with an operational cost per resource and time unit. These operational costs are defined by the following real number functions of utilisation: compute cost ζ_{c_i}, storage cost ζ_{s_i}, and bandwidth cost ζ_{b_i}.

C. Network model

In an MCN, the links that join the DCs have different cost and capacity depending on the depth they are at in the network, who own them, and their medium type. A link in the network is modelled as an edge $e_j, j \in \{1, 2, ..., J\}$ in G and has a non-directional capacity expressed as a bandwidth $\overline{\mu_j}$. Additionally, a network resource e_j has a link cost η_j, which is a function of throughput that returns the link's running cost per time unit.

D. Application model

The set of applications hosted by an MCN, $\mathcal{A} = \{a_n \mid n = 1, \ldots, N\}$, are assumed to be wholly managed by the

MCN. The resident applications' owners are therefore agnostic to where and how their applications are being executing.

Each application a_n, where $n \in \{1, 2, \ldots, N\}$ is served by a DC, v_i. Each DC v_i hosts a set of applications $\mathcal{A}_i \subseteq \mathcal{A}$. An application a_n is defined by the following increasing functions of the demand for the application's operational compute intensity γ_n^{op}, operational storage intensity α_n^{op}, and operational bandwidth intensity β_n^{op}.

Migrating an application between two DCs incurs additional resource usage for both the recipient and the host, and is defined by a migration compute intensity γ_n^{mig}, migration storage intensity α_n^{mig}, and migration bandwidth intensity β_n^{mig}, all functions of to the application's aggregate demand.

1) Demand: The applications' end users subject the applications to a quantitatively and spatially time-variant demand. An application a_n is subject to an aggregate demand from a set of demand sources $\mathcal{U}_n = \{u_{n,m} | m \in \mathcal{M}_n\}$ where $\mathcal{M}_n \subset \mathcal{V}$ is the set of leafs from which the demand originates. Each source of demand $u_{n,m}$ is represented by a function of time that returns a real number specifying the demand for application a_n at time t from leaf m.

2) Performance requirements: Application owners can impose a set of performance requirements per application that the operator of an MCN is obliged to accommodate, a Service Level Agreement (SLA). In this work, an application's SLA is conventionally expressed as the maximum of the 95th-percentile of the network delay distribution [28]. Furthermore, network delay is proportional to the number of links separating an application's end user from the current hosting DC. Thus, an application's SLA $\overline{\varrho_n}'$ is defined as the upper limit of the 95th percentile of the mean network distances between its set of sources of demand U_n and the DC it is hosted v_i. The set of network distances for application a_n is defined as:

$$l_n = \{|\sigma_{\mathcal{V}}(v_i, v_m)| \mid m \in \mathcal{M}_n\} \tag{3}$$

where $|\cdot|$ denotes the cardinality of a set, $\sigma_{\mathcal{V}} : \mathcal{V} \times \mathcal{V} \to \mathfrak{P}(\mathcal{V})$, with $\mathfrak{P}(\cdot)$ being the power set operator, is a function that determines the minimum path between two nodes. See Figure 1 for an illustration of the relationship between the applications and their demand.

III. Distributed resource management algorithm

The presented algorithm scalably solves the challenge of where to place a set of highly heterogeneous applications in a cost- and capacity- heterogeneous distributed cloud infrastructure while meeting both the DCs' operational cost and the infrastucture's resident applications' performance objectives.

Central to the algorithm are two types of reactive agents, a DC-agent and an application-agent. The agents represent the objectives of the two primary stakeholders in the system, namely DCs and applications. The agents act independently based on the performance of their respective objectives, namely operational cost and application SLA.

To achieve the objectives in a tractable manner, each agent reacts on an objective violation by re-evaluating the placement of a set of applications over a subset of DCs in a neighbourhood. The neighbourhood of depth at most k for v_i is defined as the set:

$$\mathcal{N}_i^k := \{|\sigma_{\mathcal{V}}(v_i, v_j)| \leq k+1 \mid j = 1, \ldots, I, j \neq i\}. \tag{4}$$

The resulting placement decision is reached using a common heuristic objective function $\mathcal{R}$ that is formalised in Section III-A. The two agent types react to objective violations by re-evaluating the common heuristic objective function $\mathcal{R}$ over a subset of the system's resources and applications. To meet their individual objectives the common heuristic objective function is applied differently for each agent. The fundamental properties of the algorithm are illustrated in Figure 2.

Strict caps on resource utilization and costs do not accommodate variations demand across the system and might either put the system in an instable state or require a much finer granularity of evaluation, at a significant cost. Therefore, in this algorithm, a budget for each DC is adopted to represent its desired resource utilisation or cost level, over time. The long-term objective of the algorithm is to maximise the mean budget surplus across the system. A DC's budget surplus or deficit history is distributed amongst its peers and is used to evaluate its suitability when re-evaluating the neighbourhood's application's placements. More on the budget-mechanism in Section III-B.

A. Common objective function

In a distributed heterogeneous system, such as in fog computing, an application can incur very different loads and costs in different DCs in a neighbourhood. Similarly, applications' SLAs might be accommodated with varying success amongst a set of neighbouring DCs. Thus, migrating a set of applications from one DC to mitigate its budget violation may violate the applications' SLAs and/or incur budget violations in the recipient DCs. They in turn might incur additional violations and application placement oscillations due to subsequent mitigation actions.

Due to additional resource usage, application migrations are preferably avoided. Additionally, because any migration incurs a cost, the migration should preferably be long-lasting. Thus, the objective function should take into account both budget constraints, SLA constraints, and the additive cost of link usage in a holistic manner.

The common objective function $\mathcal{R}$ is formulated for DC $v_q \in \mathcal{V}$ and the running applications $\overline{\mathcal{A}}$ at time t as,

$$\begin{aligned} R(q, \overline{\mathcal{A}}, t) := \sum_{i \in \mathcal{N}_q^k} \sum_{n \in \overline{\mathcal{A}}} \mathcal{P}_{i,n}^q(\\ + \phi_b \frac{1 - (\vartheta_{n,i}^{\text{op}}(t) + \vartheta_{n,i}^{\text{mig}}(t))}{\xi_i(t)} \\ + \phi_s \frac{1 - \varrho_{n,i}^{95th}}{\overline{\varrho_n}'} \\ + \phi_l \mathcal{L}) \end{aligned} \tag{5}$$

where $\vartheta^{\text{op}}_{n,i}$ is the momentary operational cost for each application in DC i,

$$\vartheta^{\text{op}}_{n,i}(t) = \psi_n(t)\zeta^{VM}_i, \tag{6}$$

with $\psi_n(t)$ being the unitary resource allocation cost of application n in any DC that is defined as:

$$\psi_n(t) = \left\lceil \max\left(\frac{\gamma_n(t)}{\bar{c}_{VM}}, \frac{\alpha_n(t)}{\bar{s}_{VM}}, \frac{\beta_n(t)}{\bar{b}_{VM}}\right)\right\rceil, \tag{7}$$

where ζ^{VM}_i is the cost of each resource unit in DC i and is defined as:

$$\zeta^{VM}_i = \bar{c}_{VM}\zeta_{c_i} + \bar{s}_{VM}\zeta_{s_i} + \bar{b}_{VM}\zeta_{b_i}. \tag{8}$$

where $\mathcal{L}$ is the aggregate system-wide link and is formally defined as,

$$\mathcal{L} = \sum_{i\in V}\sum_{n\in\mathcal{A}_i}\sum_{m\in\mathcal{U}_n}\eta_m\beta^{op}_n u_{n,m}(t) \tag{9}$$

and $\mathcal{P}^q_{i,n}$ being the elements of a binary matrix $\mathcal{P}^q$ of size $|\mathcal{A}|\times|\mathcal{N}^k_q|$, here called *application placement decision matrix.* In particular, $\mathcal{P}^q_{i,n}$ is equal to 1 if and only if application $a_i \in \overline{\mathcal{A}}$ is placed in node $v_n \in \mathcal{N}^k_q$, 0 otherwise. Note that, by construction, each row sums to 1, since an application cannot be placed in more than one node. Finally, $\{\phi_b, \phi_s, \phi_l\}$ are weights in the interval $[0,1]$. The i^{th} row of the matrix represents the placement of the i^{th} application $a_i \in \bar{\mathcal{A}}$ amongst the DCs in the neighbourhood $\mathcal{N}^k_q$, represented by the columns.

The placement decision is expressed as follows,

$$\mathcal{P}^{q,\star}(t) := \underset{\mathcal{P}^q}{\arg\max}\,\mathcal{R}(q,\overline{\mathcal{A}},t) \tag{10}$$

constrained by each evaluated DC's budget surplus ξ_i and the operational cost ϵ_i, formalised as,

$$\sum_{n\in A_q}\mathcal{P}^q_{i,n}\vartheta^{\text{op}}_{n,i}(t)t_e \le \xi_i(t) \tag{11}$$

$$\sum_{n\in A_q}\mathcal{P}^q_{q,n}\vartheta^{\text{op}}_{n,q}(t)t_e \le \epsilon_i \tag{12}$$

Because the algorithm is applied iteratively and is not evaluated over the entire network, hard constraints cannot be applied on individual application's performance. In worst case, an application might have to traverse a set of DCs where its SLA will be violated to reach a DC where it can run in compliance with its SLA.

When maximising the objective function $\mathcal{R}(q,\bar{\mathcal{A}},t)$, the remaining budget across all its neighbours reduces the number of additional violations in the neighbourhood and the process also takes into account any pending SLA violations, when the system is in a stable state. By normalising each component in the objective function with their individual quantitative targets, the algorithm can indiscriminately evaluate the placement of $\bar{\mathcal{A}}$ across a highly heterogeneous set of DCs and applications. As the incurred link cost is not accommodated in a DC's budget, the link cost is treated independently. The above detailed mechanisms of the algorithm and its parameters are illustrated in Figure 2.

The use of a budget to represent the state of a DC and the limited evaluation domain imposed by the neighbourhood decouples the algorithm allowing in to be implemented in a distributed manner. Additional information and state granularity would require significantly more synchronisation between agents and states and information passing, marking the implementation intractable.

B. Data Centre agent

Each DC in the network is governed by a DC agent. The objective of a DC agent is to contain the operational cost of a DC and is realised by the budget monitor process which is continuously run in each DC $v_i \in \mathcal{V}$.

Essential to the algorithm and the DC agent is a budget that is assigned to each DC. A DC's budget is the maximum allowed operational expenditure over a period and is a heuristic for a DC's capacity and desired maximum utilisation over that period. In practice, the budget allows the operator of an MCN to set coarse-grained holistic objectives for the system's resources that do not interfere with the internal management of each DC. Additionally, the budget also allows the algorithm to integrate temporary costs, over time, such as migration overheads and smaller workload variations, within the confines of the budget over an epoch.

1) Budget monitor process: The budget monitor process in each DC is assigned a budget ϵ_i for its operational cost over a period of time, referred to as an epoch Δt_e. The operational cost of a DC is defined as,

$$\zeta_i(t_1,t_2) = \int_{t_1}^{t_2}\sum_{n\in\mathcal{A}_i}\vartheta^{\text{op}}_n(t)\,\mathrm{d}t, \tag{13}$$

In runtime, the operational cost ζ_i of each DC is evaluated over an epoch of length Δt_e. If the budget is violated before the end of the epoch, i.e., $\zeta_i(h\Delta t_e, t_2) \ge \epsilon_i$, with $h \in \mathbb{N}$, and $h\Delta t_e \le t_2$, the placement of the resident applications $\mathcal{A}_i$ is evaluated over the neighbourhood $\mathcal{N}^k_i$ using the objective defined in Section III-A.

When a budget is violated or when an epoch expires without a budget violation, the budget is renewed for another epoch. For each such event, the budget surplus $\xi_i(t) = \epsilon_i - \zeta_i(h\Delta t_e,(h+1)\Delta t_e)$ of v_i is passed to all its neighbours $\mathcal{N}^k_i$. The resulting vector of the last reported neighbours' budget surpluses for DC i is denoted as,

$$\mathcal{B}_i(t) = \{\xi_q(t) \mid q \in \mathcal{N}^k_i\} \tag{14}$$

The budget monitor process is summarised in Algorithm 1.

2) State: The state of a DC agent is defined by its budget surplus ξ_i, its resource unit cost ζ^{VM}_i, its resident applications $\mathcal{A}_q$, and the budget surplus of its neighbours.

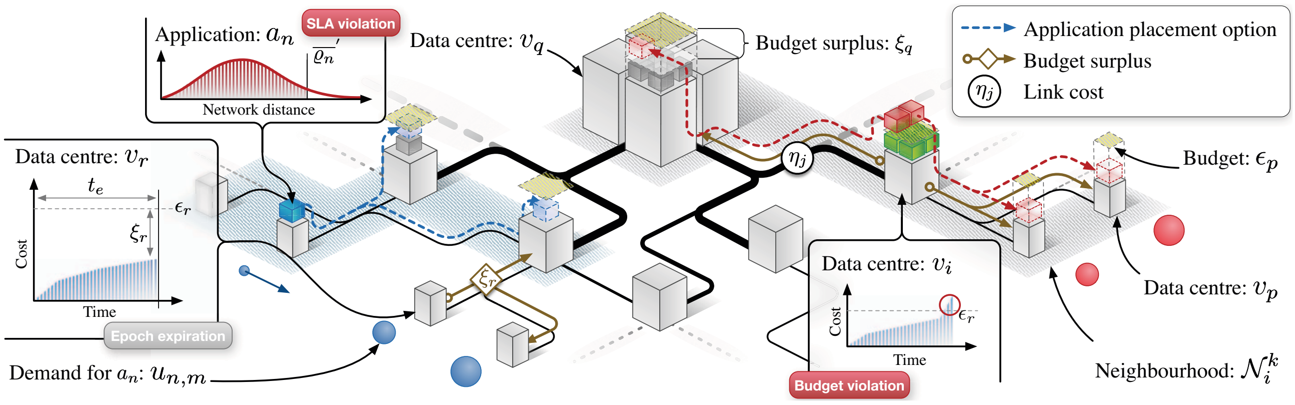


Fig. 2: The algorithm's mechanism; agents, actions, and evaluation domains.

Algorithm 1 Budget monitor process for DC v_i, for each epoch.

```
1:  Input : Budget $\epsilon_i$, application set $\mathcal{A}_i$, current placement
    $\mathcal{P}^i$, and budgets in neighbourhood $\mathcal{B}_i$,
2:  Output : Budget surplus $\xi_i$,
3:  application placement matrix $\mathcal{P}^{i,\star}$
4:  $t \leftarrow 0,\ \zeta_i^{'} \leftarrow 0$
5:  $\mathcal{P}^{i,\star} \leftarrow \mathcal{P}^i$
6:  while $t < \Delta t_e$ do
7:      $\zeta_i^{'} \leftarrow \zeta_i^{'} + \zeta_i(t, t + \Delta t)$
8:      if $\zeta_i^{'} \geq \epsilon_i$ then
9:          $\xi_i \leftarrow 0$
10:         $\mathcal{P}^{i,\star} \leftarrow \arg\max_{\mathcal{P}^i} \mathcal{R}(i, \mathcal{A}_i, t)$
11:         break
12:     end if
13:     $\xi_i \leftarrow \epsilon_i - \zeta_i^{'}$
14:     $t \leftarrow t + \Delta t$
15: end while
16: return $\{\xi_i, \mathcal{P}^{i,\star}\}$
```

C. Application agent

The performance of each resident application in the infrastucture is monitored by an application agent. The objective of an application agent is to ensure that the observed application meets its SLA. This is realised by an SLA monitoring process which is continuously run in parallel to each application a_n in each v_i.

1) SLA monitoring process: An application's performance is measured in terms of its SLA, $\overline{\varrho_n}'$. An application's placement is re-evaluated when its SLA is violated, $\overline{\varrho_{i,n}} \geq \overline{\varrho_n}'$. The SLA monitoring process is summarised in Algorithm 2.

Note that in the case of an SLA violation, only one application is evaluated, i.e. $\bar{\mathcal{A}} = a_n$.

2) State: The state of an application is defined by its demand's location and quantity $\hat{\mathcal{U}}_n$ and current latency performance $\varrho_{i,n}^{95th}$.

Algorithm 2 SLA monitor process for application n.

```
1:  Input : Hosting DC $v_i$, SLA $\overline{\varrho_n}$
2:  Output : Application placement matrix $\mathcal{P}^{i,\star}$
3:  while true do
4:      if $\overline{\varrho_{i,n}} \geq \overline{\varrho_n}'$ then
5:          $\mathcal{P}^{i,\star} \leftarrow \arg\max_{\mathcal{P}^i} \mathcal{R}(i, \{a_n\}, t)$
6:          break
7:      end if
8:      $t \leftarrow t + \Delta t$
9:  end while
10: return $\mathcal{P}^{i,\star}$
```

IV. Experiments

The experiments detailed below are designed to examine the viability of the algorithm as a tractable holistic MCN resource management approach. Given the distributed nature of the algorithm, the evaluation is primarily focused on stability and on how closely it performs to optimal, as defined in [30]. The experiments are designed to do so by determine the algorithm's convergence time from a random state as well as its step response. To add contrast to the experiments, both a random method and a naïve method. Additionally, to evaluate how the distributed algorithm performs in both current and forthcoming network topologies, the experiments employ both fat-tree and random graph network topologies. The models in the experiments are designed based on the findings in our previous work [21].

As there are no MCNs yet in existence, the experiments are conducted in a simulated environment. The simulator was design in python[1] around a time driven core using MILP solvers from PuLP [23] to represent and solve the objective function.

A. Infrastructure

An MCN infrastructure is represented by a set DCs and links in a network, as defined in Section II. To add cost- and

[1]Code and experiments available at: gitlab.com:eit-wit/mcn_placement_simulator.git

TABLE I: DC categories, their capacity and costs.

	Small	**Medium**	**Large**	**Huge**
Capacity ψ_n	250	500	1000	2000
Unit cost [2]	150%	125%	112.5%	100%

TABLE II: Link categories, their capacity and costs.

	Small	**Medium**	**Large**
Capacity $\overline{\mu_j}$	3000	5000	80000
Unit cost [3]	77%	87%	100%

capacity-heterogeneity to the infrastructure, a set of categories for each resource component are defined. Each category has a unique capacity and cost, reflective of their position in the infrastructure, these are specified below.

B. Data Centres

A DC's resources are partitioned into and provisioned as discrete units. DCs are categorised as either Small, Medium, Large, or Huge. The DC capacity is halved for each succeeding category, proportional to its depth in the network, while the operational cost grows linearly with depth. For example, a Huge DC is 8 times larger than a Small DC and cost 44% less to operate per resource unit. The properties of each DC category are summarised in Table I. The DCs are assigned a budget ϵ_i that is proportional to 80% of the total cost of all resources over an epoch, as advised by [26].

C. Links

The links are categorised by capacity $\overline{\mu_j}$ as either Small, Medium, or Large. A summary of the each link category's properties can be found in Table II.

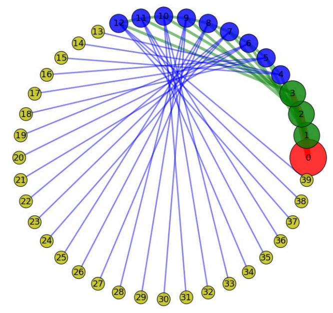
D. Topology

The DCs and links specified above are situated in a network. In this paper, a fat tree and an Erdös-Rényi random graph are used to evaluate the performance of the distributed algorithm, representing a current and a forthcoming network topology, respectively. The topology used in the evaluation consists of 40 nodes ($I = 40$), which corresponds to the typical size of a regional MCN. The topology types have an equivalent depth and total DC capacity.

1) Fat tree: Mobile core and access networks often take the shape of fat trees, with the middle tiers having the greatest amount of interconnectivity [3]. The network is assigned a Huge DC in the root node, Small DCs are assigned in the leaf nodes and remaining nodes are assigned a DC category per its depth in the network. Figure 3a illustrates the structure and resource assignment of the fat-tree topology.

2) Random graph: Access and core networks are becoming more and more interconnected, through multiple carriers and with the addition of new disaggregated network technologies [7]. To imitate this type of topology, an Erdös-Rényi random graph of $I = 40$ nodes is used. The graph is generated using a branching probability of 1.1.

[2]The costs are relative to the DC category: Huge
[3]The costs are relative to the Link category: Large


(a) Fat-tree with depth 4 and branching factor 3.

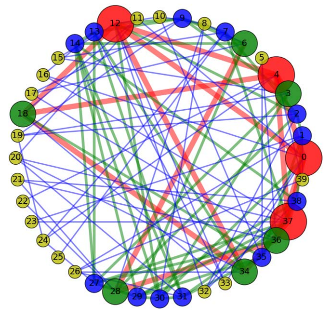

(b) Erdös-Rényi random graph with branching probability of 1.1.

Fig. 3: Network topologies used in experiments, each with 40 nodes. DC assignment: Huge, Large, Medium, Small. Link assignment: Large, Medium, Small.

The nodes in the network are assigned to a DC category based on their number of connections. The tier of nodes with the fewest connections are assigned a Small DC. The top 10% of the nodes with the highest number of branches are assigned a Huge DC. Intermediate nodes are assigned either Large and Medium DC in proportion to their network distance to one of the Huge DCs. The topology's structure and resource assignment are illustrated in Figure 3b.

A key difference between the two topologies is that a random graph is more heterogeneous than a fat-tree in the sense that a fat-tree is symmetric and that the depth of the network strongly correlates with the mean distance to the demand, DC capacity, and degree of connectivity. Furthermore, in a random graph, a set of neighbours do not have to be of similar capacity and with very different degrees of connectivity.

E. Workload and applications

To model the spatial-, and quantitative-heterogeneity of the applications and users in a MCN the system is subjected to a workload that is composed of a set of applications and their respective demand, as defined in Section II. In this paper, 400 applications heterogeneous are hosted in the infrastructure. The applications' aggregate requested resource needs equate to a time-average of 50% of the systems resources. A system load of 50% is reasonable for this type of system, yet high enough to cause resource contention.

An application is defined by three properties; its demand, its performance requirements (SLA), and its resource usage profile. They are as defined below.

1) Demand spread and quantity: The demand of an application is spread over a set of leaf nodes. In this paper, the spread of an applications demand is categorised as either local, regional, or global. The demand spread of an application is linked to the branching factor of the network's DCs. The spread of demand is at most the number of leafs that can be reached from a DC at the minimum network distance to any leaf node in the network from that any DC of that type. Local demand is associated with small DCs, regional demand

TABLE III: Application SLA range as the maximum of the 95th percentile of the distance distribution

	SLA range
Local	$[1, 2]$
Regional	$[2, 3]$
Global	$[3, 5]$

TABLE IV: Application resource utilisation characteristic types with utilisation intensities.

	Compute	**Storage**	**I/O**
CPU Intensive	0.95	0.5	0.05
I/O Intensive	0.05	0.75	0.95
Symmetric	0.5	0.5	0.75

is associated with medium and large DCs, and global demand is associated with huge DCs. The demand spread types are uniformly distributed across the 400 applications. Furthermore, the quantity of demand is proportional to the capacity of the DC type, which is associated with their demand spread.

2) Performance requirements: The performance requirements or SLA for an application is a real number upper limit of the 95th percentile of the network distance distribution of the number of hops from all users of an application to where the location of the DC in which the application is hosted. An application's SLA is associated with the type of DC that the application's demand spread is associated with. The SLA is uniformly chosen from a range from the minimum to the mean network distance between all leafs nodes to all DCs of the corresponding type. The ranges are specified in Table III.

3) Resource usage: An applications' resource usage profile is classified as either compute, storage, or I/O intensive. For example, a compute intensive application is characterised as using relatively more compute resources than storage and I/O resources, in proportion to its total demand. The resource usage intensity classes used in this paper are detailed in Table IV. In this paper, the resource usage profile types are uniformly distributed over the 400 applications, and assigned independently of the application's SLA and demand spread.

F. Comparison methods

In the experiments, the performance of the proposed algorithm is compared with an optimal, a random, and a naïve placement method.

1) Random selection and placement: This method utilises the fundamental change agents of the presented algorithm, but the decision is applied in a random manner. To be more precise, if the budget is violated in DC v_i, one random application out of $\mathcal{A}_i$ is selected and migrated to a random DC in $\mathcal{N}_i^k$ with a recorded budget surplus greater than 0. Similarly, if the SLA of an application $a_n \in \mathcal{A}_i$ is violated, it is migrated to a random DC in $\mathcal{N}_i^k$ with a reported budget surplus greater than 0. From now on this method is referred to as the random method.

2) naïve - maximum improvement worst-fit mitigation: This method utilises the fundamental change agents of the presented algorithm but the decision is applied in a maximum improvement worst-fit approach. The reasoning here is to locally minimise the additional operational cost and load incurred by an application placement change. If the budget is violated in DC v_i, a set of applications are selected for expulsion, based on the cost they incur if they are migrated in relation to how much the application contributes to the aggregate operational cost of the hosting DC, as given by:

$$\overline{a_i} = \underset{a_n \in \mathcal{N}_i^k}{\arg\max} \frac{\hat{\vartheta}_i^{\text{op}}(t_0, t_0 + \Delta t_e)}{\hat{\vartheta}_i^{\text{mig}}} \tag{15}$$

The applications are then placed in the DC in $\mathcal{N}_i^k$ with the largest budget surplus. SLA violations are mitigated by placing the application in the $v_k \in \mathcal{N}_i^k$ where the mean distance to the demand is minimised, per:

$$\underset{i \in \mathcal{N}_q^k}{\text{minimize}} \quad \varrho_{i,n}^{95th} \tag{16}$$

$$\text{subject to} \quad \vartheta_{n,i}^{\text{op}}(t) t_e + \vartheta_{n,i}^{\text{mig}} \leq \hat{\xi}_i(t) \tag{17}$$

The method is naïve in the sense that it acts locally without and independently of the system's other objectives.

3) Centralized optimal placement: To provide an upper performance bound for the presented algorithm, a centralised optimal placement method is also included. The approach is as described in our previous work [30]. All applications are placed where they incur the least amount of cost, meet their individual performance requirements, given that they do not aggregately exceed any individual DC's desired allocation level.

To increase the potential total utilisation level of the system given a highly heterogeneous workload this method does not have a global load balancing objective. A soft load balancing constraint would contradict the soft cost minimisation constraint. This is contrary to the presented distributed algorithm where a uniform load across a neighbourhood is actively pursued, as it is essential for the algorithm to iteratively permutate successfully to find a steady state.

G. Evaluation metrics

The algorithm is evaluated on its ability to meet the system's management objectives using the following metrics.

Total system cost The total momentary cost of all resources at time t, defined as:

$$\kappa(t) := \sum_{i \in \mathcal{V}} \sum_{n \in \mathcal{A}_i} (\vartheta_n^{\text{op}}(t) + \vartheta_n^{\text{mig}}(t)) + \mathcal{L} \tag{18}$$

The system's management objectives seek to minimise the total momentary cost, which means that a low value is desired.

Number of budget violations by any resource, at each point in time. A low number is desired.

Number of SLA violations by any application, at each point in time. A low number is desired.

Resource allocation distribution is defined as the standard deviation of the distribution of DC allocation levels across the infrastructure. The metric shows how well the load is balances across the system. A low standard deviation is desired.

V. Results

In this section the results from the experiments detailed in Section IV are presented and analysed. This section begins with observing the algorithm's convergence time to a steady state from a random state followed by their step responses and resource utilisation distributions.

A. Convergence

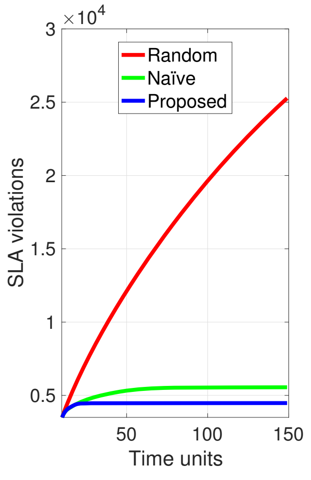
From the onset, at time $t = 0$, all applications are placed randomly in the network and 50% of the system's DC capacity is requested. Thus, on average, 65% of the applications violate their SLAs, and 50% of the DCs violate their budgets. Below, the convergence time of the algorithm is evaluated for both its agents' performance objectives, SLA and budget. The convergence time from a random state is representative of how quickly, if at all, an algorithm can reach a steady state.

Note that, when an agent evaluates the objective function the agent uses the last reported budget surplus values from its neighbours. Therefore, neither method begin to act until the first budget surpluses ξ_i are communicated, namely at the end of the fist epoch $t = 10$. Furthermore, as the optimal method is already in a steady state, its convergence time is naturally not considered.

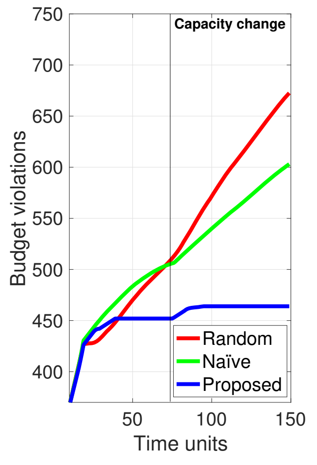
1) SLA: Starting with the traditional fat-tree topology, as illustrated by Figure 4a, the distributed algorithm can meet all resident applications' SLAs after 20 time steps. The naïve method does not do so until $t = 70$. This is due to the naïve method's competing actions, the budget violation and SLA processes. To this effect, up until this point, the naïve method has retarded 19% of the applications' SLA deficits while working towards meeting all DCs' budgets. In the random case, the SLA violation process does not converge within the time-frame of the experiment.

The random graph scenario leads to a different outcome. Due to the higher degree of connectivity, the distributed algorithm's SLA process is now able to converge after 12 time steps, see Figure 5a. The naïve and random methods fail to converge because they can at this point no longer be propelled by the differential between the heterogeneous layers in the fat tree topology. Instead, the naïve method permanently deposits 9% of the applications that violate their SLA's in DCs from which it cannot find more suitable hosts. Interesting to note is that the random method is well suited to handle this degree of heterogeneity. Although it does not converge until $t = 140$, the random method can reach a steady state.

2) Budget: The distributed algorithm's budget violation process converges after 40 time-steps when deployed in the fat-tree topology scenario, see Figure 4b. The naïve method converges at $t = 70$, at the expense of an additional 100 budget violations. When considering the SLA deficit/surplus of the applications, none of the applications with a small SLA surplus are migrated up until the point the SLA process converges at

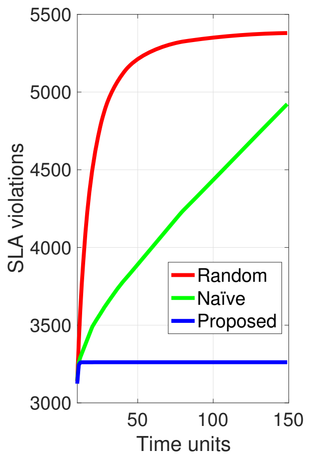


(a) Cumulated SLA violations.

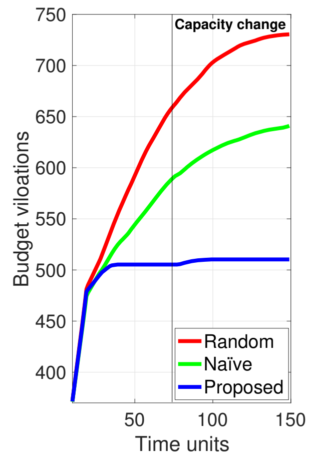


(b) Cumulated budget violations.

Fig. 4: Objective violations in a fat-tree topology.



(a) Cumulated SLA violations.



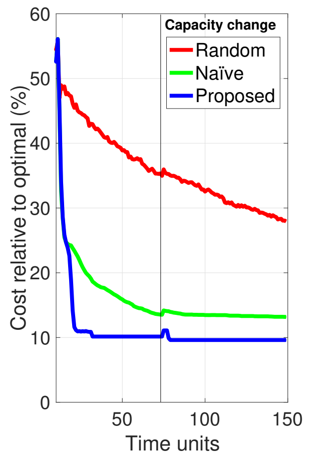
(b) Cumulated budget violations.

Fig. 5: Objective violations in a random graph topology.

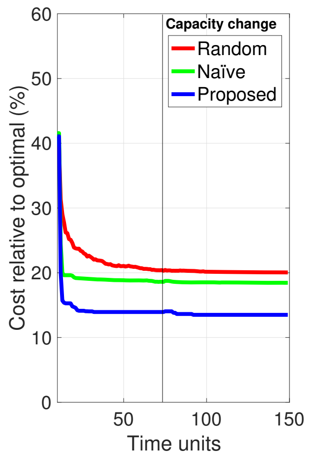
$t = 20$. Again, the random method does not converge within the time frame of the experiment.

A similar outcome can be found in the random graph topology, see Figure 5b. Again, the algorithm is assisted by the higher degree of connectivity, and now converges at $t = 35$.

3) Operational cost: Starting with the fat-tree topology; once converged, the distributed algorithm incurs a total system cost within 9% of the operational cost achieved by the optimal method, see Figure 6a. The method's ability to approach the optimal cost point reflects the system load and/or the budget. A smaller budget forces the methods to find a lower cost point but at the cost of ability to permutate. Despite failing to meet all DCs' budgets the naïve method incurred cost converges to 13% of the optimal. As with the previous scenarios, the random method fails to converge.

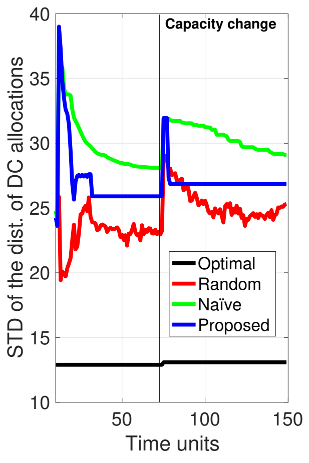


(a) Fat-tree topology.

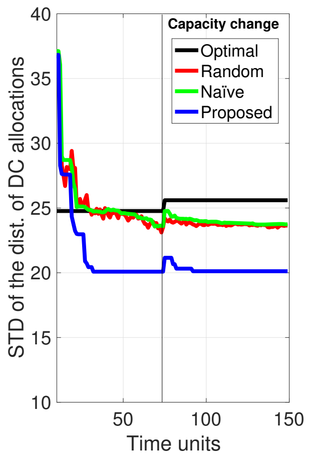


(b) Random graph topology.

Fig. 6: Total operational cost relative to the cost incurred by the optimal approach.



(a) Fat-tree topology.



(b) Random graph topology.

Fig. 7: Standard deviation of the distribution of DC allocation levels across the system.

The outcome for the random graph topology is illustrated in Figure 6b. The total system cost achieved by the distributed algorithm when employed in the random graph topology converges to 13% within the cost incurred by the optimal approach. In this case, both the naïve and random methods converge to an incurred system cost of 18% and 20% from the optimal, respectively.

B. Step response

Observing the algorithms' step responses reveal how well they can respond to changes from a steady state. To subject the system to a change, the capacity of a random medium-sized DC in the network is instantly halved. The budget of that DC is also adjusted accordingly.

In the fat-tree topology, the capacity change is done at $t = 75$. The distributed algorithm can spread the affected DC's excess demand to its neighbours, who then propagate any excess to their neighbours while attempting to balance the load throughout the system. Because the objective function considers the SLA deficit/surplus of the applications, only applications that would reduce the net load in the neighbourhood or improve its SLA deficit, are likely to be affected. The distributed algorithm's budget process thus reaches a new steady state after 7 time steps. The naïve method on the other hand fails to spread the excess load over DC 2's neighbours, and instead creates a bottleneck in the middle of the network. From this point on, the naïve and random methods' budget process diverges, and therefore fails to load balance the system, which will inhibit the system to handle any forthcoming changes in load or capacity.

In the random topology, the capacity is also changed at $t = 75$. With a larger number of neighbours per DC, the distributed algorithm can converge to a new steady state with only 20% of the violations compared to the fat-tree topology. Due to the increased interconnectivity of the random graph it can handle a large change in capacity. Furthermore, the naïve and the random methods have still not converged at $t = 150$, with a significant number of both SLA and budget violations.

C. Allocation distribution

Although the allocation of the resources is not one of the system objectives, as explained earlier, it does provide an idea of the state to which the algorithms converge to. As a result, the optimal solution does not have this objective. A wide allocation distribution across the system's resources implies that certain resources are less able to permutate in the event of a change in capacity and/or demand. In these experiments the load distribution imposed on the system by the applications' demands is uniform, thus a narrow distribution is desired.

Figure 7a shows the standard deviation of the distribution of DC allocation levels across a fat-tree topology. The figure reveals that all the non-optimal methods achieve a very similar level of allocation distribution. Note that each algorithm converges to its previous level despite a significant reallocation of resources.

For the random graph topology, presented Figure 7b, all the non-optimal solutions achieve a lower allocation variance than the optimal solution. Additionally, in the random graph topology, all non-optimal methods converge significantly faster and are less disrupted by the change in the capacity at $t = 75$ than the fat-tree topology. This can be attributed to the greater variety of resources available to any given node in the random graph topology.

VI. Conclusions

This paper presents a distributed algorithm to holistically manage a large set of heterogeneous DCs and applications with different objectives. The main challenge has been to reach a steady system state and while accommodating a set of entities with heterogeneous objectives hosted in a cost- and capacity-heterogeneous network. The distributed algorithm was evaluated over two different types of topologies with varying degrees of heterogeneity and compared to both a centralised optimal solution, and two naïve methods. The results reveal that the distributed algorithm presented in this paper can quickly and consistently converge despite a high degree of heterogeneity in the system. The evaluations also reveal some of the properties in a heterogeneous topology that can be used to extend this work.

A possible investigative extension of this work is a thorough investigation of the distributed algorithm's convergence performance under a transient workload and resources with time-variant capacity and cost. Possible extensions to the algorithm include elastic horizontal scaling of applications and multi component applications.

VII. Acknowledgements

This work is funded in part by the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control. Maria Kihl and William Tärneberg are members of the Lund Center for Control of Complex Engineering Systems (LCCC) funded by the Swedish Research Council (VR) and the Excellence Center Linköping - Lund in Information Technology (ELLIIT). William Tärneberg is also funded by the Mobile and Pervasive Computing Institute Lund University (MAPCI). While producing this work, William Tärneberg was hosted by the University of Virginia. Alessandro V. Papadopoulos is funded by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032.

References

[1] G. H. Ballantyne. Robotic surgery, telerobotic surgery, telepresence, and telementoring. *Surgical Endoscopy and Other Interventional Techniques*, 16(10):1389–1402, 2002.

[2] S. K. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, pages 35–46, New York, NY, USA, 2010. ACM.

[3] P. Bedell. *Cellular Networks: Design and Operation: a Real World Perspective*. 2014.

[4] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[5] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[6] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. A stable network-aware vm placement for cloud systems. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 498–506. IEEE Computer Society, 2012.

[7] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski. Five disruptive technology directions for 5g. *IEEE Communications Magazine*, 52(2):74–80, 2014.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[9] S. Borst, V. Gupt, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[10] P. Bosch, A. Duminuco, F. Pianese, and T. L. Wood. Telco clouds and virtual telco: Consolidation, convergence, and beyond. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 982–988. IEEE, 2011.

[11] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann. Cloud ran for mobile networks—a technology overview. *Communications Surveys & Tutorials, IEEE*, 17(1):405–426, 2015.

[12] H.-L. Chi, S.-C. Kang, and X. Wang. Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. *Automation in construction*, 33:116–122, 2013.

[13] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[14] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[15] S. Höst, W. Tärneberg, P. Ödling, M. Kihl, M. Savi, and M. Tornatore. Network requirements for latency-critical services in a full cloud deployment, 2016.

[16] S. Islam and J.-C. Grégoire. Giving users an edge: A flexible cloud model and its application for multimedia. *Future Generation Computer Systems*, 28(6):823–832, 2012.

[17] G. Karagiannis, A. Jamakovic, A. Edmonds, C. Parada, T. Metsch, D. Pichon, M. Corici, S. Ruffino, A. Gomes, P. S. Crosta, and T. M. Bohnert. Mobile cloud networking: Virtualisation of cellular networks. In *21st International Conference on Telecommunications (ICT), 2014*, pages 410–415, May 2014.

[18] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[19] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. *ACM SIGCOMM Computer Communication Review*, 41(4):74–85, 2011.

[20] Z. A. Mann. Allocation of virtual machines in cloud data centers&mdash;a survey of problem models and optimization algorithms. *ACM Comput. Surv.*, 48(1):11:1–11:34, Aug. 2015.

[21] A. Mehta, W. Tärneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth. How beneficial are intermediate layer data centers in mobile edge networks? In *FAS* Foundations and Applications of Self* Systems University of Augsburg, Augsburg, Germany, 12-16 September 2016*, pages 222–229, 2016.

[22] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[23] S. Mitchell, M. OSullivan, and I. Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.

[24] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 58:1–58:11, New York, NY, USA, 2011. ACM.

[25] A. V. Papadopoulos and M. Maggio. Virtual machine migration in cloud infrastructures: Problem formalization and policies proposal. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6698–6705. IEEE, 2015.

[26] T. Püschel, N. Borissov, M. Macías, D. Neumann, J. Guitart, and J. Torres. Economically enhanced resource management for internet service utilities. In *Web Information Systems Engineering–WISE 2007*, pages 335–348. Springer, 2007.

[27] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[28] J. Sommers, P. Barford, N. Duffield, and A. Ron. Accurate and efficient sla compliance monitoring. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 109–120. ACM, 2007.

[29] B. Spinnewyn, B. Braem, and S. Latre. Fault-tolerant application placement in heterogeneous cloud environments. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 192–200. IEEE, 2015.

[30] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, and E. Elmroth. Dynamic application placement in the mobile cloud network. *Future Generation Computer Systems*, 2016.