Algorithms for Network-Aware Application Component Placement for Cloud Resource Allocation

Maryam Barshan, Hendrik Moens, Steven Latré, Bruno Volckaert, and Filip De Turck

Abstract: **Due to the soaring popularity of cloud-based services over the last years, the size and the complexity of cloud environments has been growing quickly. In the context of cloud systems, mapping a number of application components to a set of physical servers and assigning cloud resources to those components is challenging. Traditional resource allocation systems which rely on a centralized management paradigm suffer from scalability issues, making them inappropriate for large-scale cloud environments. Therefore, there is a need for providing new management solutions that scale well to large size cloud systems. In this article, we present optimal and heuristic solutions for network-aware placement of multi-component applications with differing component characteristics. The optimal integer linear programming (ILP)-based solution minimizes the application rejection rate and the cost of mapping while respecting application component requirements and physical network limitations. As the execution time of the optimal model scales exponentially, we also offer scalable heuristic solutions for centralized and hierarchical application placement, which are thoroughly explained and evaluated and compared to the optimal solution. Our evaluations show that while the proposed centralized heuristic is near-optimal, the hierarchical approach is much faster and offers higher scalability compared to a centralized approach, e.g., mapping 2.7 million application components onto 512k servers. Moreover, the percentage of servers used and fully placed applications remain close to that of the centralized and optimal solutions.**

Index Terms: **Application placement, cloud management, hierarchical management system, optimization, scalability.**

I. INTRODUCTION

CLOUD computing has emerged as a powerful paradigm which has revolutionized the way in which computing infrastructures are used. Elasticity and on demand services are the main characteristics which make these computing infrastructures appealing. Nowadays many companies make use of cloud technologies to reduce costs, increase flexibility and to respond faster to customer needs. Although the benefits of cloud systems are considerable, numerous challenges remain, among them, effective supervision of resource usage, scalability and in particular resource allocation to the applications. The application placement refers to the act of deciding where on the clusters of servers, applications are placed [1].

The initial placement policy used to map applications onto physical servers has important effects in terms of application performance and resource efficiency, and making a suitable initial decision is essential to reduce the future need for migrations. In literature, most efforts have been directed towards optimizing the usage of CPU, memory and disk resources, and reducing the energy consumption of physical servers. According to [2], however, there has been a drastic increase in the amount of data generated and consumed by each application. Thus, resource allocation methods have to expand and take into account this growing focus on data. Inappropriate placement of application components with heavy communication requirements could lead to the saturation of certain network links, with subsequent negative impact on applications, e.g., slow response or execution times.

Cloud-based applications often consist of multiple interacting components with differing requirements. While some components may consist of high CPU intensive tasks, requiring powerful computational servers, others may deal with large volumes of data, making servers tailored to data-throughput more appropriate for such components. In order to offer an efficient placement service, different requirements of application components should be taken into account in deciding on where to deploy application components.

In order to address this problem of network-aware application placement in cloud environments, in this article, we first introduce a centralized integer linear programming (ILP)-based optimal model. The main objective is maximizing the percentage of mapped applications while taking the cost of application mapping into account. In our proposed approach, we deal with multi-component applications with multiple component types. The interdependence among application components implies that either the entire application or none of the application components are mapped. This is known as the full deployment placement constraint [3]. Therefore, a mapped application is an application for which all components are successfully allocated. Moreover, due to the characteristics of our applications, in order to have deterministic performance and for security reasons we have made a distinction between different types of application components. Each pair of application components may either

Manuscript received June 9, 2016; approved for publication by YU Hua, Division III Editor, June 10, 2017.

The computational resources (Stevin Supercomputer Infrastructure) and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by Ghent University, the Hercules Foundation and the Flemish Government - department EWI. The work is also partly supported by the iMinds DMS2 project and the FP7 NoE FLAMINGO project.

M. Barshan, H. Moens, B. Volckaert, and F. D. Turck are with the Department of Information Technology, Ghent University – IMEC, Technologiepark-Zwijnaarde 15, 9052 Gent, Belgium, email: {maryam.barshan, hendrik.moens, bruno.volckaert, filip.deturck}@ugent.be.

S. Latré is with the Department of Mathematics and Computer Science, University of Antwerp – IMEC, Middelheimlaan 1, 2020 Antwerpen, Belgium, email: steven.Latre@uantwerpen.be.

M. Barshan is the corresponding author.

Digital object identifier: 10.1109/JCN.2017.000081

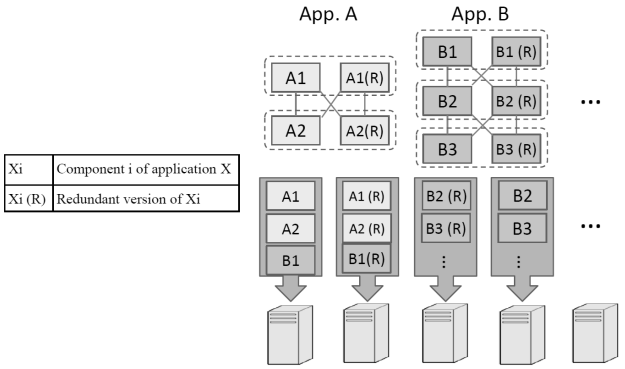

Fig. 1. The process of application component placement with the anti-collection placement requirement. Redundant components are not allowed to be allocated on the same host as non-redundant components.

be allowed to share a hardware resource or not. This isolation of component types can be modeled as an anti-collocation placement constraint [3]. In our approach, the anti-collocation constraint implies that different component types are not allowed to be placed onto the same servers. We achieve this by ensuring that each server is only allowed to place one type of application component. It should be noted that these constraints can be applied to a wider range of generic problems, such as multi-tenant applications with a strong focus on data security (banking, insurance, etc.) or anti-collocation of redundant components for increased reliability and fault tolerance purposes. Fig. 1 shows an example of a reliable application component placement onto a small cluster of cloud servers. As illustrated in this figure, each application component and its redundant version must be mapped onto two different physical servers.

Due to the NP-hardness of the problem [4]–[6] and limited scalability of the optimal model, we also propose an approximate centralized heuristic as well as a hierarchical approach for large-scale cloud environments with the following design goals: scalability and performance. While centralized approaches are omniscient in nature and can make better placement decisions, our hierarchical solution has been designed in such a way that component placement optimality is nearing that of the centralized heuristic approach. Also, the proposed hierarchical algorithm executes faster as each management cluster maintains a partial view of the network. In this article, we will prove that the resource allocation process is scalable both in number of cloud servers (e.g., 512 k servers) and the number of application components (2.7 million application components) needing to be placed onto the cloud servers. Furthermore, as part of our presented approach an application placement policy, prioritizing local deployment is taken into account for each administrative domain. This partial solution also tries to minimize the number of servers used within each administrative domain. This feature, known as server consolidation, is mostly effective in reducing the power consumption of large-scale datacenters [7].

In the context of modern cloud platforms, the application placement process consists of placing the application's components in a set of virtual machines (VMs) and deploying these to the physical infrastructure [8]–[11]. In this article, we assume that the components are already encapsulated in VMs or (micro-service) containers, and the application component placement decides on where to place these VMs on the available physical servers, taking network demands between the VMs into account.

The rest of the article is organized as follows. In Section II, the related work is discussed. Section III describes the architecture for our distributed approach to network-aware application placement for large-scale cloud datacenters. In Section IV, the formulation of the ILP-based model is presented in detail. In Section V the proposed algorithms are extensively discussed, followed by an evaluation of the proposed algorithms in Section VI. Finally in Section VII, we sum up our contribution and conclude the article.

II. RELATED WORK

Recently many different approaches for application placement and cloud resource allocation have been proposed [12], each focusing on different aspects of the problem. While many approaches such as [10], [13]–[16] rely on centralized approaches which suffer in term of scalability, [17] offers a distributed protocol in order to design a resource management middleware. However, their solution is different as interaction between application components is ignored and there is no global overview of system states for the network administrators. Authors in [18] also focus on resource allocation in IaaS clouds. Nevertheless, the main contribution of this paper is maximizing resource utilization and request acceptance rate. Another work [19], clarifies the definition of distributed cloud and the challenges of resource allocation on distributed clouds.

The authors in [20] have made a discussion and categorized the VM placement schemes into resource-aware, power-aware, cost-aware and network-aware. Network-aware approaches, such as [21]–[25], try to reduce traffic related issues or avoid congestion and in general, VMs make use of network either to communicate with each other or to access the required data from storage components. However, none of these approaches take into consideration the anti-collocation requirement of VMs. On the other hand, deployment of VMs under placement constraints has been investigated in [3], [26], [27]. Shi *et al.* [3], [26] focus on different VM placement constraints, e.g., full deployment, anti-collocation, security, etc. and Breitgand *et al.* [27] study the SLA compliant placement of multi-VM elastic services under the anti-collocation placement constraints. However, these approaches do not take network demands between the VMs into account.

The model defined in this article has similarity with [4] that describes a linear application placement model and [28] which offers a cost-aware algorithm. Nevertheless, these approaches work at the application level, contrary to our component-level application modeling policy where we make a distinction between different component types. In addition, based on the definition of "the best-fit placement" in [4], our centralized heuristic solution follows the same rule, which is finding a feasible server whose residual capacity is minimal. Nevertheless, it differs from our approach as it is centralized and not network-aware. Many other application placement approaches [13], [29]–[31] also focus on placing a set of independent applications, and do not take the underlying network into account. These approaches do

not provide a guaranteed quality of communication between individual interacting components due to potential contention of network resources [12]. In order to mitigate the effect of this risk, network-aware solutions have recently been proposed. Authors in [32] pay special attention to time varying nature of traffic demand and dynamic routing capabilities for medium size data centers. Jiang *et al.* [33] focus on a multi-path routing scheme and live migration. Incorporation of various network functions has been studied in [34]. While these network-aware approaches only focus on network congestion or minimizing network traffic, we explicitly take both network capacity and delay requirements of applications into account in our formulation as QoS constraints. Among the other network-aware approaches which also take both requirements into account, we refer to [10], [35]–[38] that are centralized and [39] which specifically focus on geo-distributed clouds.

According to [12], while most of the existing application mapping solutions focus on centralized systems, only 11.5% of approaches, including [40]–[44], use hierarchical control schemes and one of those approaches [45] takes network constraints into account. However, their hierarchical approach is not related to their management system but instead it is related to their placement process.

This article is related to our previous work on hierarchical cloud resource management [46], [47]. In [46] the underlying network was however not taken into account, while this work specifically focuses on network-aware management of multi-tier interactive applications. [47] does take the underlying network into account, but does not make a distinction between different component types. In this article we however make a distinction between multiple component types, taking their requirements and characteristics into account during the placement decision. In addition, this article presents much larger scale evaluations.

This work is the extension of our previous work [48], [49]. In [48], we presented an optimal ILP-based solution which offers limited scalability, making it only suitable for small datacenters. This optimal solution is also useful for benchmarking real-time heuristic algorithms. In [49], a decentralized algorithm was designed and evaluated which can be applied to large-scale cloud environments. In this article, we have updated the ILP-based model in 3 ways. First, the model has been generalized to support arbitrary numbers of component types as it was previously limited to two different component types. Secondly, in the previous version, a feasible solution was reached only when all the applications have been mapped. The model has now been extended to incorporate an application mapping failure rate, which penalizes failures of mapping applications, but allows finding solutions when otherwise no feasible solution would be possible. Finally, the main objective has been altered to minimize the application mapping failure rate in addition to the cost of application mapping. Our evaluations have also been extended to be more comprehensive.

Generally, what distinguishes our method from other approaches is the combination of the following: 1) Our solution is network-aware, decentralized and scalable; 2) application modeling is component-based with different types, and interaction between those components affects the placement process; 3) SLA agreements and the properties of underlying network, bandwidth and delay, are respected; 4) anti-collocation placement constraints are defined, based on which multiple but same-type components can be placed onto a single physical node; and 5) our method minimizes the number of servers used while respecting application requirements.

III. MODELING OF A LARGE-SCALE CLOUD ENVIRONMENT

The model for the proposed large-scale algorithm can be divided into three parts: the physical cloud system, the management plane model and the application model.

Physical cloud system model: In literature, several new topologies have been proposed for future cloud-based environments, such as Jellyfish [50], Dcell [51], etc. Nevertheless, tree-based topologies are still dominant in the existing operational cloud datacenters [52], [53]. Although our proposed approach is not limited to the type of network topologies, in this work the physical cloud system is considered a hierarchical tree topology, which is common in modern data centers. As application components of different types can not be mapped onto the same server in our proposed approach, each server has different responsibility and provides specific functionality.

Management plane model: The management plane relies on multi-layered hierarchical architecture in which three types of managers are defined: Low level manager (LLM), mid level manager (MLM) and root level manager (RLM). The LLMs are located in the lowest level of the management hierarchy, the RLM in the top level and MLMs in middle ones. LLMs directly deal with physical servers and are responsible for mapping application components onto physical servers. MLMs manage several LLMs and have the authority to choose the current active LLM, which has to take the responsibility of mapping new application components. RLMs have the general overview of the cloud management systems. In multi-domain cloud environments, RLMs can communicate with other domains if the need arises. The management plane specifications are shown in Table 1. The number of management levels ($|ML|$) and the number of supported servers ($|SS|$) for each LLM are taken and the branch factor of each tier (μ) is calculated for each management domain. In addition, the number of supported servers and the number of levels determine the number of LLMs. By calculating the level branch factor the number of MLMs can be achieved as follows.

$$\mu = \sqrt[|ML|-1]{|LLM|} \tag{1}$$

$$|LLM| = \lceil |S|/|SS| \rceil \tag{2}$$

$$|MLM| = \sum_{level=1}^{|ML|-2} \mu^{((|ML|-1)-level)} \tag{3}$$

An example of a physical cloud system and how this maps to its management plane is illustrated in Fig. 2. In each administrative domain different servers are chosen as default servers for different component types.

Application model: In this article, the architecture of the applications is service oriented, meaning applications can be represented as a service graph and the application topology is a

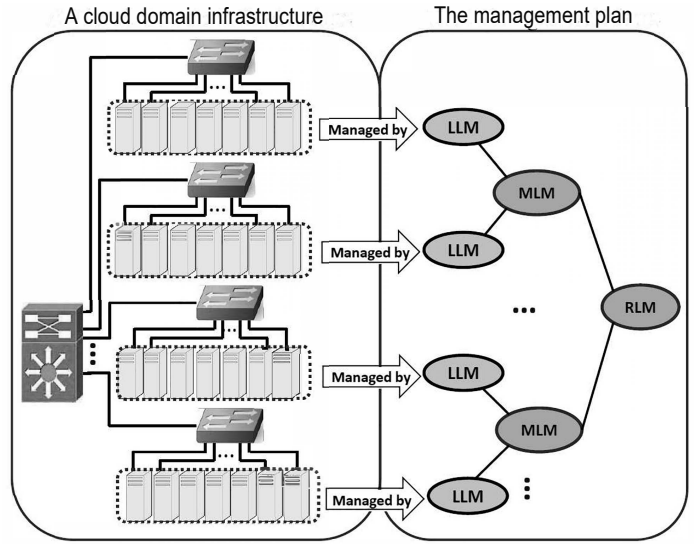

Fig. 2. The architecture of physical infrastructure and the management plane.

Table 1. The management plane parameters.

Parameter	Description
$\|LLM\| \in \mathbb{N}^+$	Number of LLMs.
$\|MLM\| \in \mathbb{N}^0$	Number of MLMs.
$\|RLM\| \in \mathbb{N}^0$	Number of RLMs.
$\|SS\| \in \mathbb{N}^+$	Number of supported servers for each LLM.
$\|ML\| \in \mathbb{N}^+$	Number of management levels.
$\mu \in \mathbb{N}^+$	Management level branch factor.

graph. Although an arbitrary number of application component types can be supported by our approach, we focus on two component types in our evaluations: Database (e.g., data sources, data stores) and computational (e.g., application business logic or user interfaces) components. Database components store and manage data and are more storage intensive, whereas computational components are more CPU intensive. The application database and computational components are the nodes and connections between these components form the directed links of the application graph. Each application component requires a specific number of data sources, CPU power, memory and storage, etc. and the storage demand of database components is much higher than their CPU/memory demand, whereas for logic components, power of CPU is of the highest demand. In our model,maximum allowed delay and bandwidth requirements are defined for application links, which need to be satisfied as well.

IV. FORMAL ILP-BASED PROBLEM FORMULATION

A. Introduction to the Model

We first present a formal model for application component placement for cloud resource allocation. In this model the substrate network is considered as an undirected graph and the application network as a directed graph due to interdependencies between different components. The parameters of the physical network graph and application network and their descriptions are listed in Table 2. Both infrastructures consist of nodes (N)

Table 2. Symbols and notations used in the formal model.

Physical cloud-based infrastructure parameters	
Variable	**Description**
G_{ph}	Physical graph, $G_{ph} = (N_{ph}, L_{ph})$.
N_{ph}	Physical nodes set in G_{ph}.
L_{ph}	Physical links set in G_{ph}.
$S_u \in N^+$	Available storage capacity of node u.
$M_u \in N^+$	Available memory capacity of node u.
$C_u \in N^+$	Available CPU capacity of node u.
$D_{e_{uv}} \in N^+$	Delay of link *euv*.
$BW_{e_{uv}} \in N^+$	Bandwidth capacity of link *euv*.
$type_{e_{uv}} \in \{0, 1\}$	Whether link *euv* is a LAN or WAN link.
$Ccost_u \in N^+$	Cost of each CPU unit of node u.
$Mcost_u \in N^+$	Cost of each memory unit of node u.
$Scost_u \in N^+$	Cost of each storage unit of node u.
$BWcost_{e_{uv}} \in N^+$	Cost of each BW unit of link *euv*.
$fcost_u \in N^+$	The fixed cost of using node u.
$fcost_{e_{uv}} \in N^+$	The fixed cost of using link *euv*.
Component-based application parameters	
Variable	**Description**
G_{app}	App. graph, $G_{app} = \{a \| a = (N_a, L_a)\}$.
$AppNo$	Number of applications.
$CompNo_a$	Number of components of application a.
S_{type}	Set of types of application components.
$ai \in N_a$	Component i of application a.
$\gamma^t_{ai} \in [0, 1]$	Has value 1 if *ai* is of type t.
$c_{ai} \in N^+$	Computation demand of app.a, comp. i.
$s_{ai} \in N^+$	Storage demand of app. a, comp. i.
$m_{ai} \in N^+$	Memory demand of app. a, comp. i.
$e_{aij} \in L_a$	Link between comp. i and j of app. a.
$bw_{e_{aij}} \in N^+$	Bandwidth demand of link e_{aij}.
$d_{e_{aij}} \in N^+$	Max. allowed delay of link e_{aij}.

and links (L). In this context, application links refer to the connections between application components with certain demands that need to be met. Nodes in substrate graph ($u \in N_{ph}$) have specific properties such as data storage capacity (S_u), CPU power capacity (C_u) and memory capacity (M_u). Physical links ($e_{uv} \in L_{ph}$) can be either LAN or WAN link and this is determined by a binary variable, $type_{e_{uv}}$ in which 0 refers to the LAN and 1 refers to the WAN links. We made this distinction because WAN links cost more than LAN links. Each link has delay ($D_{e_{uv}}$) and bandwidth capacity ($BW_{e_{uv}}$) properties. It has to be noted that the physical network resource capacities are residual capacities, considering the previous mappings. As we aim to minimize the cost of mapping applications onto cloud resources, the general cost of physical nodes and links as well as cost of using each unit of CPU, memory, storage and link capacity has been taken into account.

Similarly, each application has been considered as a workflow, consisting of multiple components and links between those components form a directed weighted graph. In the application network, *ai* refers to the component i of application a with specific computational (c_{ai}), storage (s_{ai}) and memory (m_{ai}) requirements and e_{aij} refers to the link between component i and j of application a with specified bandwidth ($bw_{e_{aij}}$) and maximum allowed delay ($d_{e_{aij}}$) demands. Different component types are collected in S_{type} and γ^t_{ai} is a binary input variable which indicates whether or not ai is of type t.

B. Decision Variables

Seven decision variables have been defined in this ILP model and all variables are binary. First, x_u^{ai} shows the accomplished mapping between component i of application a and physical node u, regardless of the type of component. It has to be noted that this variable is equal to 0 in two states, either when due to limitations there is no possibility to have a mapping between nodes or when physical node x is not chosen for the mapping although it was possible. Next, $f_{e_{uv}}^{e_{aij}}$ indicates success of mapping between physical link e_{uv} and the link between components i and j of application a (e_{aij}).

As we assume that each physical node is exclusively used for components of the same type, variable T_u^t is defined for the purpose of determining whether node u is used to host components of type t. Multiple components of the same type can be mapped onto the same physical server.

Furthermore, two other variables are defined: B_u is a binary variable to show whether physical node u is used, either as a routing node or a used server in the entire mapping. $B_{e_{uv}}$ is another binary variable to indicate whether physical link e_{uv} is used in the mapping scheme or not. Finally, M_a has been defined to indicate whether the application a is fully mapped or not.

$$
\begin{array}{ll}
x_u^{a,i} \in [0,1] & \forall u \in N_{ph}, \forall a \in G_{app}, \forall i \in N_a \\
f_{e_{uv}}^{e_{aij}} \in [0,1] & \forall e_{uv} \in L_{ph}, \forall a \in G_{app}, \forall e_{aij} \in L_a \\
T_u^t \in [0,1] & \forall u \in N_{ph}, \forall t \in S_{type} \\
B_u \in [0,1] & \forall u \in N_{ph} \\
B_{e_{uv}} \in [0,1] & \forall e_{uv} \in L_{ph} \\
M_a \in [0,1] & \forall a \in G_{app}
\end{array}
$$

C. Objective Function

Guaranteeing the quality of service and taking physical constraints into account, application placement services have to be performed with minimum rejection of application placement requests. To achieve this, the sum of M_a variables should be maximized. Minimizing cost of mapping should always be considered the second optimization objective. The cost of physical servers can be determined by combining the individual costs of using each unit of CPU, memory and storage and the fixed cost of using each server. Moreover, since in multi-domain cloud networks the cost of LAN links are almost zero, for estimating the link cost, only the WAN links are taken into account.

The optimization objective function minimizes both the application rejection rate and the cost of mapping with lower and higher priorities respectively. Given α and β as higher (e.g., 10^6) and lower (e.g., 1) priority parameters respectively, ***FailureRate*** as application mapping failure rate and ***NodeMapCost*** as cost of physical node usage and ***LinkMapCost*** as cost of physical links usage, the objective function is defined as follows:

Minimize

$$\alpha \times FailureRate + \beta \times (NodeMapCost + LinkMapCost),$$

where

$$FailureRate = \left(AppNo - \sum_{\forall a \in G_{app}} M_a \right) / AppNo,$$

$$NodeMapCost = \sum_{\forall u \in N_{ph}} \left(fcost_u \times B_u + \sum_{\forall a \in G_{app}} \sum_{\forall i \in N_a} (c_{ai} \times Ccost_u + m_{ai} \times Mcost_u + s_{ai} \times Scost_u) \times x_u^{ai} \right),$$

$$LinkMapCost = \sum_{\forall e_{uv} \in L_{ph}} \left(fcost_{e_{uv}} \times B_{e_{uv}} \times type_{e_{uv}} + \sum_{\forall a \in G_{app}} \sum_{\forall e_{aij} \in L_a} (bw_{e_{aij}} \times BWcost_{e_{uv}} \times type_{e_{uv}}) \times f_{e_{uv}}^{e_{aij}} \right).$$

As a result of minimizing the cost of mapping, the objective of the presented model also minimizes the number of nodes on which the applications can be hosted while satisfying the following constraints and requirements.

D. Constraints

The defined constraints for application component mapping in cloud system have been organized into 7 sub-sections as follows:

D.1 Physical Node Limitations

Constraints (1), (2), and (3) are considered for physical network nodes and are related to computational, memory and storage limitations respectively. For all physical nodes, the common idea is that sum of all mapped requests' demand must not exceed their maximum available capacities.

$$\sum_{\forall a \in G_{app}} \sum_{\forall i \in N_a} c_{ai} \times x_u^{ai} \leq C_u \qquad \forall u \in N_{ph} \qquad (1)$$

$$\sum_{\forall a \in G_{app}} \sum_{\forall i \in N_a} m_{ai} \times x_u^{ai} \leq M_u \qquad \forall u \in N_{ph} \qquad (2)$$

$$\sum_{\forall a \in G_{app}} \sum_{\forall i \in N_a} s_{ai} \times x_u^{ai} \leq S_u \qquad \forall u \in N_{ph} \qquad (3)$$

D.2 Physical Link Limitations

A bandwidth constraint has to be considered for each physical link, regardless of being either WAN or LAN link. Constraint (4) represents that for each physical link, the sum of bandwidth demands of all applications must not exceed maximum available bandwidth.

$$\sum_{\forall a \in G_{app}} \sum_{\forall e_{aij} \in L_a} bw_{e_{aij}} \times f_{e_{uv}}^{e_{aij}} \leq BW_{e_{uv}} \quad \forall e_{uv} \in L_{ph} \qquad (4)$$

D.3 Quality of Service Requirements

For delay and bandwidth, Constraints (5) and (6) are defined for each application link e_{aij}. It has to be noted that the bandwidth constraint can be ignored as it will be satisfied with the physical link constraint (4).

$$\sum_{\forall e_{uv} \in L_{ph}} D_{e_{uv}} \times f_{e_{uv}}^{e_{aij}} \leq d_{e_{aij}} \quad \forall a \in G_{app}, \forall e_{aij} \in L_a \tag{5}$$

$$BW_{e_{uv}} \times f_{e_{uv}}^{e_{aij}} \geq bw_{e_{aij}} \quad \forall e_{uv} \in L_{ph}, \forall a \in G_{app}, \forall e_{aij} \in L_a \tag{6}$$

D.4 Well-Connected Mapping Constraints

Constraint (7) makes sure that when 2 adjacent application components cannot be physically mapped next to each other, a chain of continuous physical links is used to map each application link. This assures that a closed path is considered to map an application link. As can be observed from this equation, for each physical node u, the subtraction of the sum of all incoming and outgoing f values should be equal to the subtraction of X values between target and source of each application link e_{aij}.

$$\sum_{\forall u \in N_{ph}} f_{e_{uv}}^{e_{aij}} - \sum_{\forall u \in N_{ph}} f_{e_{uv}}^{e_{aij}} = x_u^{aj} - x_u^{ai} \tag{7}$$

$$\forall a \in G_{app}, \forall e_{aij} \in L_a, \forall e_{uv} \in L_{ph}, \forall u \in N_{ph}$$

D.5 Full Deployment Constraints

The statements below, Constraints (8) and (9), ensure that if an application is mapped each individual component of application a has to reside in exactly one server in order to have a successful mapping. Constraint (10) indicates that either all or none of application components have to be mapped.

$$\sum_{\forall u \in N_{ph}} x_u^{ai} = M_a \qquad \forall a \in G_{app}, \forall i \in N_a \tag{8}$$

$$\sum_{\forall u \in N_{ph}} x_u^{ai} \leq 1 \qquad \forall a \in G_{app}, \forall i \in N_a \tag{9}$$

$$\sum_{\forall i \in N_a} \sum_{\forall u \in N_{ph}} x_u^{ai} = CompNo_a \times M_a \qquad \forall a \in G_{app} \tag{10}$$

D.6 Anti-Collocation Constraints

We also need other constraints between X and T values to ensure that each physical node is only used for components of the same type. Constraint (11) and (12) are defined to ensure that the type of the application component and the physical node on which this component is mapped are identical. Since mapping of components of different types is not feasible in the proposed approach, constraint (13) ensures that for each physical node sum of all T_u^t values for all component types should be less than or equal to 1.

$$\gamma_{ai}^t \times x_u^{ai} \leq T_u^t \qquad \forall u \in N_{ph}, \forall a \in G_{app}, \forall i \in N_a, \forall t \in S_{type} \tag{11}$$

$$x_u^{ai} \leq \sum_{\forall t \in S_{type}} T_u^t \quad \forall u \in N_{ph}, \forall a \in G_{app}, \forall i \in N_a \tag{12}$$

$$\sum_{\forall t \in S_{type}} T_u^t \leq 1 \qquad \forall u \in N_{ph} \tag{13}$$

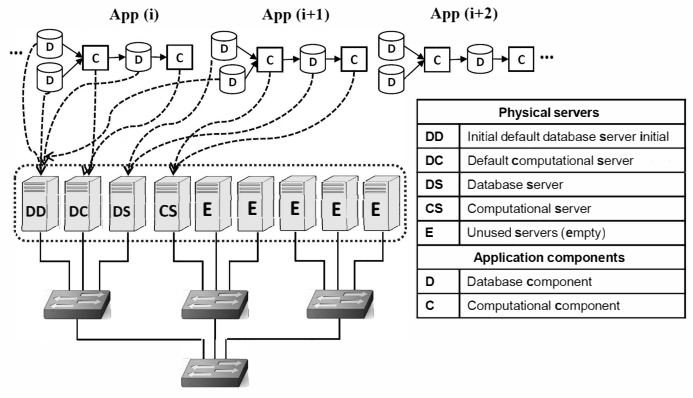

Fig. 3. The process of application component placement onto a cluster of cloud servers for two types of components (database server and computational server).

D.7 Additional Constraints

Constraints (14) and (15) are needed to make logical correlations between physical resources and their usage. K in both constraints is a large number. In constraint (14) its value has to be larger than the sum of all possible X values and in a same way larger than all possible f values in constraint (15). In constraint (16), B_u for each physical node shows that whether node u is used to host any type of component or not.

$$\sum_{\forall a \in G_{app}} \sum_{\forall i \in N_a} x_u^{ai} \leq K \times B_u \qquad \forall u \in N_{ph} \tag{14}$$

$$\sum_{\forall a \in G_{app}} \sum_{\forall e_{aij} \in l_a} f_{e_{uv}}^{e_{aij}} \leq K \times B_{e_{uv}} \qquad \forall e_{uv} \in L_{ph} \tag{15}$$

$$B_u = \sum_{\forall t \in S_{type}} T_u^t \qquad \forall u \in N_{ph} \tag{16}$$

V. ALGORITHM DESCRIPTIONS

A. ILP-based Algorithm

This algorithm implements the optimal ILP-based model which was extensively explained in Section III. This ILP-based algorithm is solved using IBM ILOG CPLEX Optimization Studio [54] which is a tool to build efficient optimization models. The objective function minimizes both the application mapping failure rate and the cost of mapping, taking the constraints into account.

B. Heuristic Algorithm

As has been shown in [4]–[6], the problem of application placement onto a network with bandwidth constrained links is NP-hard. Based on computational complexity theory, in large scale environments, an optimal solution for an NP-hard problem is too expensive to be used in practice; instead a near-optimal solution is desired.

In this section a centralized and a hierarchical approach is discussed which we refer to as the centralized cloud mapping algorithm (CCMA) and the hierarchical cloud mapping algorithm (HCMA), respectively. These algorithms are executed within the management plane.

The centralized CCMA algorithm is proposed as a near-optimal alternative to the centralized ILP based approach. This centralized approach can be deployed independently and efficiently up to the scale of medium-size networks. We will show that the centralized approach always outperforms the hierarchical solution in terms of number of fully mapped applications. Comparing the quality and complexity, the use of the hierarchical approach is only recommended for large scale environments, where the CCMA can not be practically used due to high complexity. The HCMA has made use of the CCMA algorithm, in combination to the global cloud mapping algorithm (GCMA). The GCMA is introduced to have interactions between different managers within the hierarchical management plane.

B.1 CCMA

This algorithm first arbitrarily chooses different nodes as the default servers for different component types. For each application the algorithm, shown in Algorithm 1, goes through all the components and tries to allocate resources to each component. An illustrative example of this placement for two types of components is shown in Fig. 3. In order to have minimal bandwidth overhead, the algorithm uses the Dijkstra shortest path algorithm [55] for mapping the application links. However, there are two situations in which the application component cannot be placed onto the default server, either because of physical node limitations or due to physical link limitations. Node limitation occurs when there is not enough residual CPU, memory or storage capacity in one of default servers. In the latter case, again, there are two situations in which the link limitation leads to unsuccessful placement. First, the application components cannot be connected because there are no physical links to connect application components located on different servers. Second, placement can be unsuccessful if bandwidth or delay requirements cannot be resolved.

No matter what is causing unsuccessful application component placements and what the type of component is, the next server selection (NSS) process should be followed to choose another server as a default server. In the NSS process, a breadth first search (BFS) algorithm [55] is run with the current default server as the start vertex to initialize the next server selection. We use the BFS because this algorithm finds the nearest server with minimal path length which ensures there is a minimal communication overhead between the new and the previous servers. However, when link limitation occurs, first another server is chosen temporarily by the NSS process and then the algorithm checks the path availability and SLA fulfillment, and sets it as a default server provided that choosing this server satisfies both conditions. Otherwise, the placement is not successful. In this case the algorithm must remove all placed components of the application and backtrack to the state before the placement. This state is saved before placement of each application in order to backtrack when the need arises. After backtracking, the placement starts again with new default servers. This process will be continued until the NSS process is unable to find a new server. If this occurs, placement of the application is not possible.

Algorithm 1. CCMA, run on the management plane in the centralized approach and on each LLM in the hierarchical approach.

```
input: applications
for (c ∈ application components) do
    while (Map (c, defaultServer) = false) do
        if (Due to node limitations) then
            New defaultServer ← NSS(default Server);
            if (New defaultServer = Null) then
                Mapped_app ← false;
                return false;
            end
        else if (Due to link limitations) then
            Temp Server ← NSS (defaultServer);
            if (CheckLinks (Temp Server)) then
                New defaultServer ←Temp Server;
            else
                Mapped_app ← false;
            end
        end
        if (one of default servers = null) then
            Mapped_app ← false;
            return false;
        end
    end
end
```

Algorithm 2. HCMA, run on the hierarchical management planes.

```
input: Applications,
       Physical infrastructure (servers, links),
       Management plane (|S|, |SS|, |ML|);
for (app ∈ applications) do
    manager← current active LLM ;
    GCMA(app, manager, null, "new request");
end
```

B.2 HCMA

The HCMA algorithm is shown in Algorithm 2. Based on this algorithm, all placement requests are sent to the current *active LLM*, using the GCMA algorithm. The GCMA is designed in order to have interactions between different managers within the management plane. The GCMA is run on every manager. In the management hierarchy, each LLM is in charge of its own administrative domain and the current *active LLM* is the one which is active in mapping the application components. The current active LLM is determined arbitrarily when the algorithm starts. Each manager has two states: "full" and "not full". A manager is "full" when all its managed servers get fully occupied. The active LLM will be replaced when its state changes to "full". The next active LLM is chosen by the parent of the current active LLM, i.e. an MLM or the RLM. For each newly arriving application, the HCMA invokes the GCMA with the current active LLM and a "new request" message. In the GCMA three types of messages are defined: "new request", "from the parent node", and "full". This is illustrated in Fig. 4. Next, the GCMA sends the application request to the current active LLM by calling the

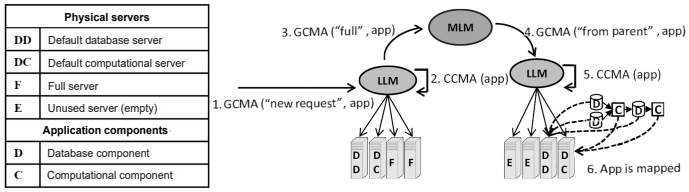

Fig. 4. Different messages for interacting between the managers in GCMA.

CCMA, which was presented earlier. In hierarchical approach the CCMA is run on every LLM. If this default administrative domain is not able to place the entire application components, the status of the current active LLM changes to "full". Afterward, this LLM calls the GCMA with a "full" message to its parent, indicating that the application cannot be placed onto this cluster. In this step, interaction between different management entities starts.

B.3 GCMA

The GCMA is a hierarchical algorithm, listed in Algorithm 3. Based on this algorithm when a request is received, three cases can be distinguished:

1. A request is received by the highest RLM: The request will be forwarded to the next unvisited domain with a "from the parent node" message. If all domains are full and the request is rejected the cloud system is not able to place this application.
2. A request is received by the MLM: The request will be forwarded by applying the same policy to one dedicated lower-level manager with a "from the parent node" message until the target LLM located at the lowest level is reached. Provided that all domains are full, the status of this manager turns to "full" and the request with a "full" message will be forwarded to the parent which can be either another MLM or the RLM.
3. A request is received by the LLM: At this level all request messages will be either "new request" or "from the parent node". No matter who is the request sender, the manager executes the CCMA algorithm. In a saturation case when placement of new applications is not possible, the LLM has to send the newly arriving requests to its parent and introduce itself as a "full" manager.

VI. EVALUATION DETAILS

The implemented physical cloud system is a tree-based multi-tier infrastructure, similar to current datacenter topologies [11], [12], [56], consisting of server nodes and links which we assume to be homogeneous. This means that all the servers have similar configuration of CPU, memory, storage and transmission medium (in terms of bandwidth and delay). In our evaluations, we make use of two component types. Each server can be either a database or a computational server. No backup servers are assumed. Servers are located in the lowest tier (level $= 0$) and the other levels consist of intermediate devices such as switches. In order to design the physical infrastructure, the number of server nodes ($|S|$) and the number of levels ($|L|$) are taken as inputs. In order to have the desired scale, these variables can be tuned.

Algorithm 3. GCMA, run on every manager in the hierarchical approach.

input: Application a, manager m, requestSender r, message s
Impossibility$_a$ $\leftarrow$ false;
Currentstate $\leftarrow$ save the current system state;
while *(Mapped$_a$ = false & Impossibility$_a$ = false)* **do**
  Set current system state to currentstate;
  if *(m_{type} = LLM &s = ("new request" OR "from the parent node"))* **then**
    if *(one of the default servers=null)* **then**
      full$_m$ $\leftarrow$ true;
      GCMA(a, parent$_m$, m, *"full"*);
    else
      CCMA(*m,a*);
    end
  end
  if *($m_{type} \neq$ LLM & s = ("full" OR "from the parent node"))* **then**
    for *($ch \in children_m$)* **do**
      if *($full_{ch}$ = false & $ch \neq r$)* **then**
        GCMA(*a, ch, m, "from the parent node"*);
        return;
      end
    end
    full$_m$ $\leftarrow$ true;
    if *(m_{type} = MLM)* **then**
      GCMA (*a,parent$_m$, m, "full"*) ;
    else
      Impossibility$_a$ $\leftarrow$true;
    end
  end
  if *(Impossibility$_a$ = true)* **then**
    Set current system state to currentstate;
  end
end

This physical cloud environment is a complete *N-ary* tree in which the *N* is the calculated branch factor (β). In addition, the number of switch ports and the number of tiers determine the number of network switches. The branch factor of each tier and the number of intermediate switches ($|IS|$) are calculated as follows. The variables defined to describe physical cloud system are listed in Table 3.

$$\beta = \sqrt[|L|-1]{|S|} \tag{1}$$

$$|IS| = \sum_{level=1}^{|L|-1} \beta^{((|L|-1)-level)} \tag{2}$$

In our evaluations, three types of applications, shown in Table 6, are implemented. Type 1 refers to the 5-component applications with 3 database and 2 computational components, type 2 refers to the 10-component applications with 7 database and 3 computational components and type 3 refers to 20-component applications that consisting of 14 database and 6 computational components. These types of applications have been provided by

Table 3. The physical network parameters.

Variable	Description
$\|S\| \in \mathbb{N}^+$	Number of physical servers.
$\|IS\| \in \mathbb{N}^+$	Number of intermediate switches.
$\|L\| \in \mathbb{N}^+$	Number of physical switching levels.
$\beta \in \mathbb{N}^+$	physical level branch factor.

Table 4. The physical infrastructure specifications.

Physical infrastructure specifications				
Case study	$\|S\|$	$\|IS\|$	$\|L\|$	$\|SP\|$
1	1000	111	4	10
2	4096	273	4	16
Physical server specifications			**Physical Link Specifications**	
CPU	**Storage**	**Memory**	**bandwidth**	**Delay**
3 GHz	200 GB	16 GB	400 Mbps	3 ms

Table 5. Management plane infrastructure.

Case study	type	$\|ML\|$	$\|SS\|$	$\|LLM\|$	$\|MLM\|$	$\|RLM\|$	μ
1	CCMA	1	1000	1	0	0	-
	HCMA	3	10	100	10	1	10
	HCMA	3	40	25	5	1	5
	HCMA	2	100	10	0	1	10
2	CCMA	1	4096	1	0	0	-
	HCMA	3	16	256	10	1	16
	HCMA	3	64	64	5	1	8
	HCMA	2	256	16	0	1	16

Table 6. Application specifications.

Type	# component	# link	# database	# computational	
1	5	4	3	2	
2	10	9	7	3	
3	20	19	14	6	
Type	**Component demands (random)**			**Link demands (random)**	
	CPU	**Storage**	**Memory**	**Delay**	**BW**
1	(1–1000) MHz	(1–20000) MB	(1–2000) MB	1 s	(1–50) Mbps
2	(100–500) MHz	(100–20000) MB	(100–1000) MB	1 s	(1–50) Mbps
3	(1–200) MHz	(1–10000) MB	(1–300) MB	1 s	(1–20) Mbps

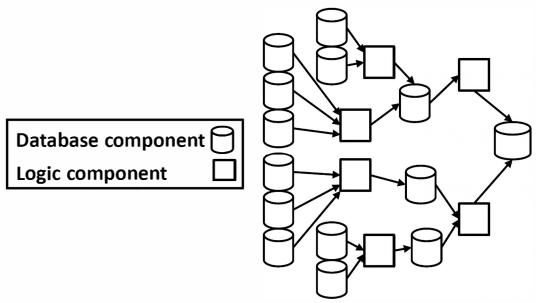

Fig. 5. An illustrative 20−component application (type 3).

our industrial partners based on realistic applications with deterministic characteristics, which implies that the structure of the applications is always known beforehand. To illustrate the used applications, a 20-component application is shown as a sample in Fig. 5. Throughout this section, the number of X-component applications refers to the number of applications, submitted for a possible placement to the cloud network management system. We assume that the application are either rejected or placed in full with all X components.

This section is divided into four parts. Our proposed CCMA approach combines a set of requirements including network awareness, anti-collocation and full deployment placement constraints, which are not supported by the state-of-the-art solutions presented in literature. This makes it difficult to accurately compare our results to existing methods as the alternative network-aware solutions focus on different aspects, such as migration [25], investigation of traffic pattern [38], energy efficiency [57], SLA-awareness [58], etc. Therefore, we compare the performance of CCMA to a generic network-aware method, in which anti-collocation characteristics of applications is ignored. In the evaluation cases, we refer to this solution as (anti-collocation unaware, network-aware) ACUNA. Then, to provide an accurate validation, an evaluation of the CCMA is provided by comparing to the ILP-based optimal solution which takes all these requirements into account. Next, we evaluate the HCMA by comparing its performance with the CCMA. Finally, we will end the section with a large-scale evaluation of the HCMA.

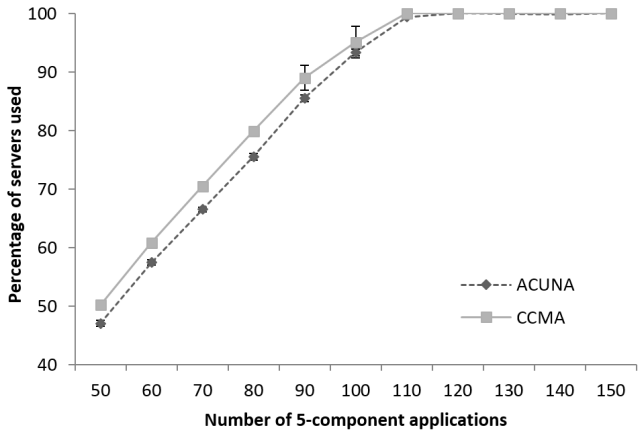

Fig. 6. Comparing the percentage of servers used in the CCMA and the ACUNA algorithm as a function of number of application placement requests (20 iterations).

The simulations are performed using the Stevin Supercomputer Infrastructure at Ghent University, containing quad core Intel Xeon L5420 servers with 16 GB RAM.

A. Comparing CCMA to the State-of-the-Art Solutions

A.1 Evaluation Set Up

For this evaluation, we consider small 5-component applications and a 3-tier network architecture consisting of 100 servers and 10 intermediate nodes. The number of applications varies from 50 up to 150. The experiments are iterated 20 times and the percentages of serves used, mapped application and anti-collocation constraint fulfillment are captured.

A.2 Evaluation Results

Our evaluation in Figs. 6, 7, and 8 shows that although generic network-aware approach is able to map up to 5.75% more applications and up to 4.35% lower number of servers, at least in 66.8% of evaluation cases the anti-collocation requirement of mapped applications are violated.

B. Comparing the CCMA to the ILP-based Algorithm

B.1 Evaluation Set Up

The optimal model and the CCMA are evaluated with a configuration of 6 servers arranged in a star topology. The specification of servers and links can be seen in Table 4.

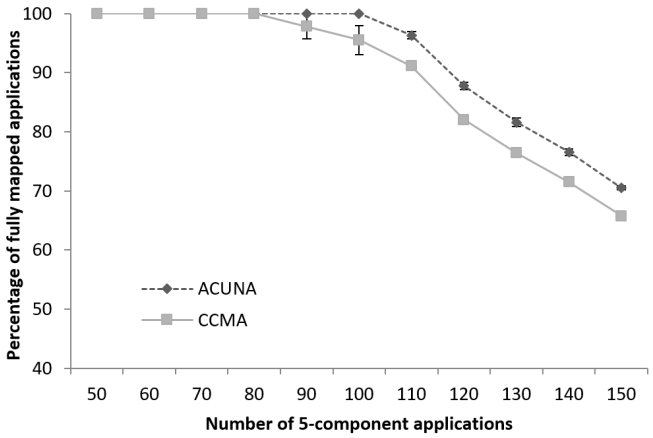

Fig. 7. Comparing the percentage of fully mapped applications in the CCMA and the ACUNA algorithm (20 iterations).

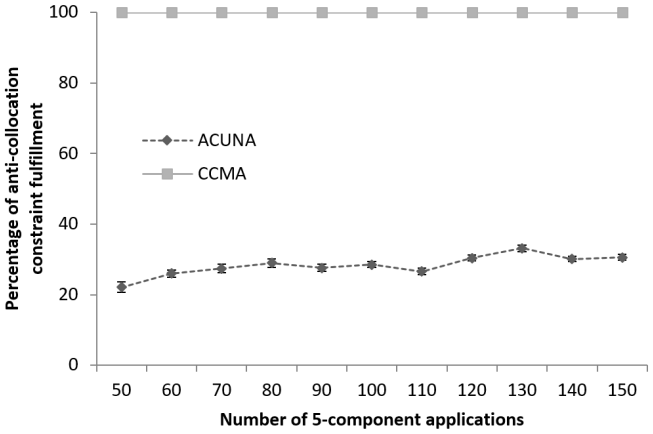

Fig. 8. Comparing the percentage of anti-collocation application placement fulfillment in the CCMA and the ACUNA algorithm (20 iterations).

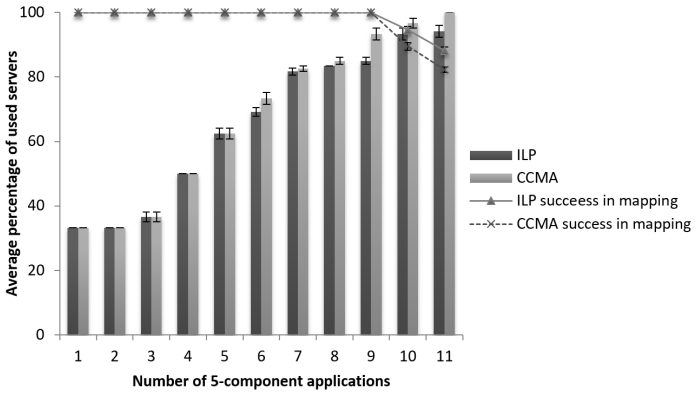

Fig. 9. Comparing the number of servers used (as bar charts) and the application mapping success rate (as line charts) in the CCMA to the ILP-based algorithm for 5-component applications (20 iterations).

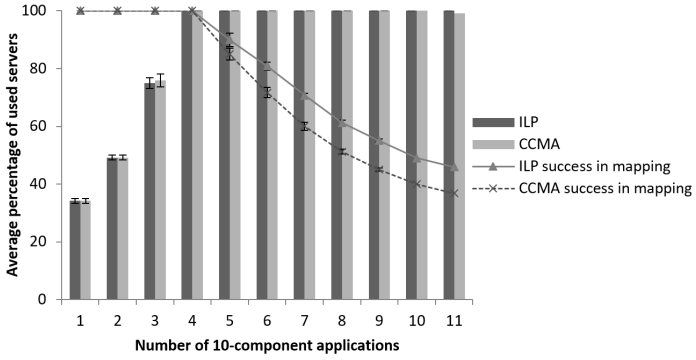

Fig. 10. Comparing the number of servers used (as bar charts) and the application mapping success rate (as line charts) in the CCMA to the ILP-based algorithm for 10-component applications (20 iterations).

Type 1 and type 2 applications are used to compare the performance of the proposed algorithms under light and heavy network load conditions. The number of scenarios varies from 1 up to 11. The average percentage of used servers and percentage of algorithm success in mapping all the applications are evaluated.

B.2 Evaluation Results

In Figs. 9 and 10 the CCMA is compared to the ILP-based optimal approach. The percentage of servers used is depicted in bars and the percentage of algorithm success in mapping all the application components are shown in lines.

As can be seen in both figures, when it comes to the physical resources usage, the CCMA provides a near-optimal solution compared to the ILP-based algorithm in this scenario. This can be clearly observed from Fig. 9 as the network is not saturated. This figure show that in 5 out of 11 experiments the numbers of used servers are equal in the CCMA and the ILP-based approach and the CCMA uses at most 8.33 more number of servers when the number of applications is 9. In Fig. 10, the percentage of algorithm success in mapping all the applications is more interesting. This figure reveals that when the network is saturated the capability of the CCMA in mapping application components stays within 10.7% of the optimal approach. This figure also shows that when both algorithms succeed in mapping all the application components (from 1 up to 4 number of applications), the CCMA uses almost the same number of servers as the optimal ILP-based approach. These results show the performance of the CCMA is close to that of the optimal algorithm.

The execution times of the CCMA and the ILP-based approaches are compared in Fig. 11 for type 1 applications. As can be seen, the execution time of the ILP-based model is exponentially increasing by adding more applications, which makes it inappropriate for larger evaluations. As such, the reminder of this section is devoted to the comparison of the CCMA and the HCMA algorithms. Throughout the next subsections, the HCMA (*XX,YY*) refers to a three-tier management plane of *XX* LLMs and *YY* MLMs and the HCMA (*XX*) refers to a two-tier management plane of *XX* LLMs. In a two-tier management plane no MLM is involved and one RLM is taken into account in all experiments.

C. Comparing the Hierarchical Algorithm to the Centralized Approach

We study three case studies. In the first case, 5-component applications are placed on a cloud system with 1,000 servers and the second case considers a larger scenario with 4,096 servers

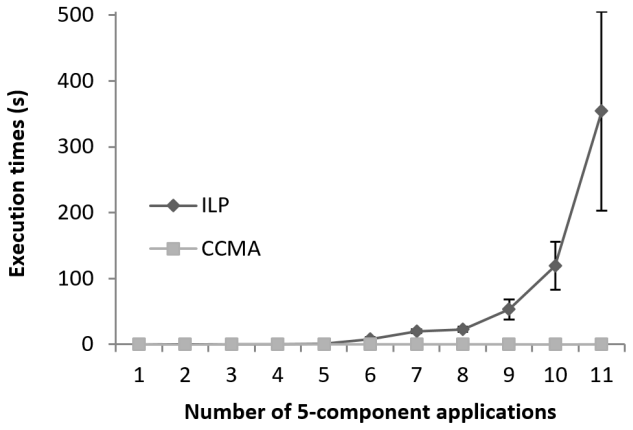

Fig. 11. Comparing the execution times of the CCMA to the ILP-based algorithm for 5-component applications (20 iterations).

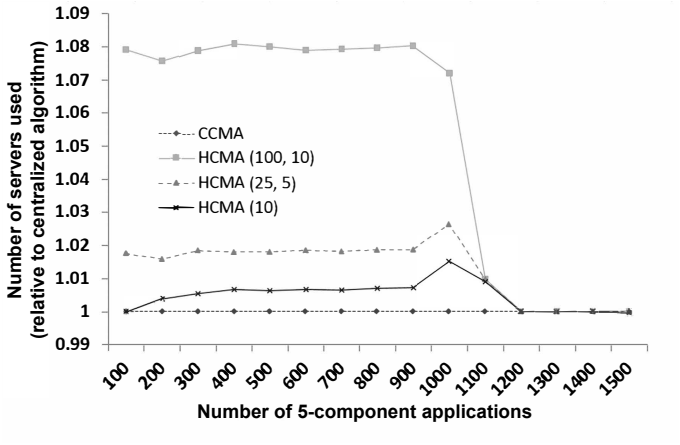

Fig. 13. The relative percentage of used servers, compared to CCMA (case study 1 with 1,000 servers and 20 iterations).

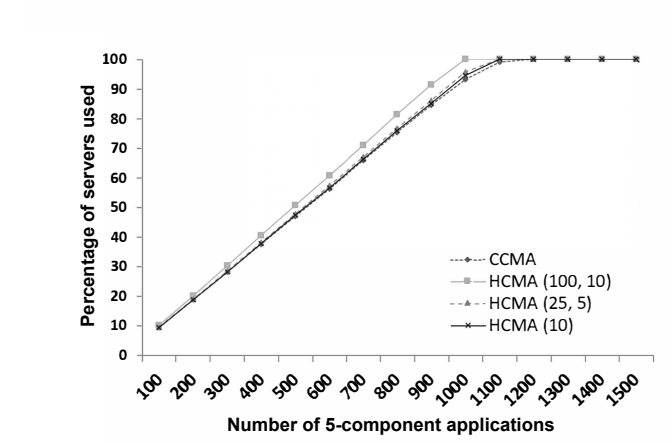

Fig. 12. The percentage of servers used (case study 1 with 1,000 servers and 20 iterations).

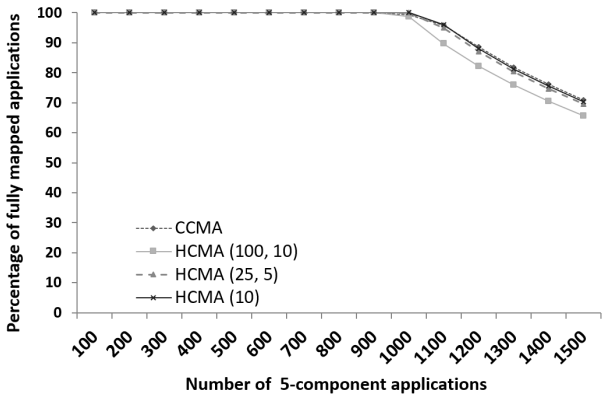

Fig. 14. The percentage of fully placed applications (case study 1 with 1,000 servers and 20 iterations).

and 20-component applications. In the experiments, we measure the percentage of servers used, the percentage of mapped applications and the execution times per application. Afterward, the impact of different physical infrastructures on the average number of fully mapped applications and the execution time for 1,000 up to 4,096 servers are analyzed. Due to negligible standard errors for the reminder of evaluations, standard error bars are left out.

C.1 Evaluation Set Up

The configuration of the simulated network, the management plane and the application structure are as follows. For the evaluation, the configuration of physical infrastructure is considered to be a 4-tier hierarchical tree topology. For the first scenario, the physical cloud system consists of 1,000 servers (resp., 4,096 for case study 2) in the lowest tier. The number of ports in each intermediate device is 10 (resp. 16) which results in $1+10+100$ (resp. $1 + 16 + 256$) switches in the first three tiers. Consequently, the number of physical nodes is 1,111 (resp. 4369) in the entire cloud system. The specifications of the physical cloud resources are shown in Table 4.

To make a better comparison apart from the central management system, three different hierarchical management planes are generated. The hierarchical management planes are defined as follows and are listed in Table 5.

- A 3-tier management plane with 100 (resp. 256) LLMs, 10 (resp. 16) MLMs and 1 RLM. Each LLM supports 10 (resp. 16) servers in this case.
- A 3-tier management plane with 25 (resp. 64) LLMs, 5 (resp. 8) MLMs and 1 RLM. In this scenario 40 (resp. 64) servers are supported by each LLM.
- A 2-tier management plane with 10 (resp. 16) LLMs, no MLM and 1 RLM. Each administrative domain consists of 100 (resp. 256) servers here.

The implemented applications are of type 1 (resp. type 3), the number of which varies from 100 up to 1,500 (resp. 400 up to 4,000). Each application component has different CPU, memory, storage and QoS demands which are randomly taken within a predefined interval, provided in Table 6.

C.2 Evaluation Results

Figs. 12 and 13 show the percentage of used servers for different management planes. As can be observed from Fig. 12, the number of used servers grows linearly with the number of applications until all the resources are completely occupied. Among

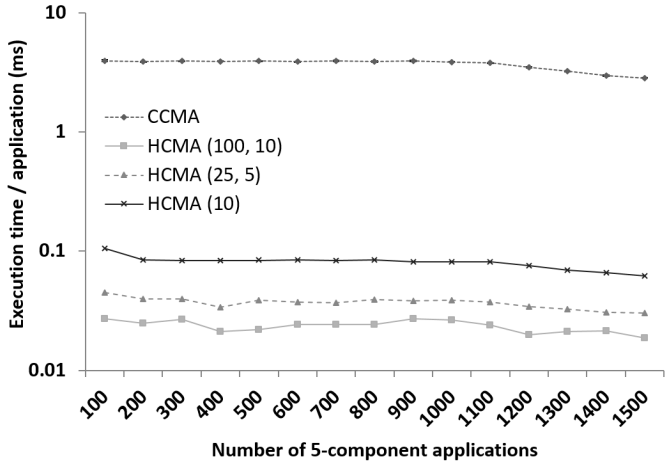

Fig. 15. Comparing the execution times per application (case study 1 with 1,000 servers and 20 iterations).

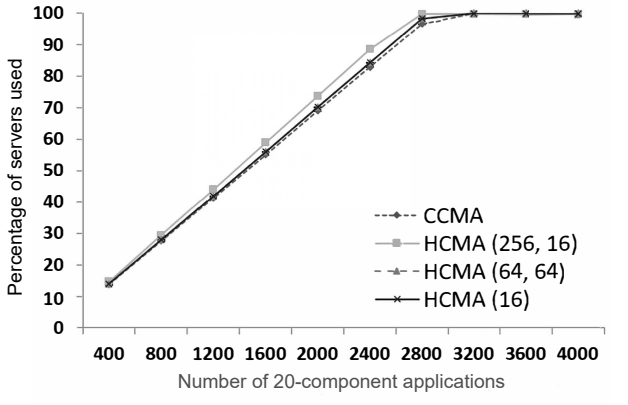

Fig. 16. The percentage of servers used (case study 2 with 4,096 servers and 20 iterations).

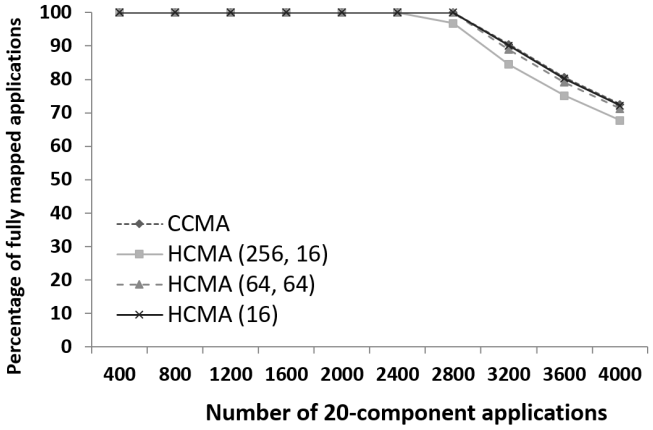

Fig. 17. The percentage of fully placed applications (case study 2 with 4,096 servers and 20 iterations).

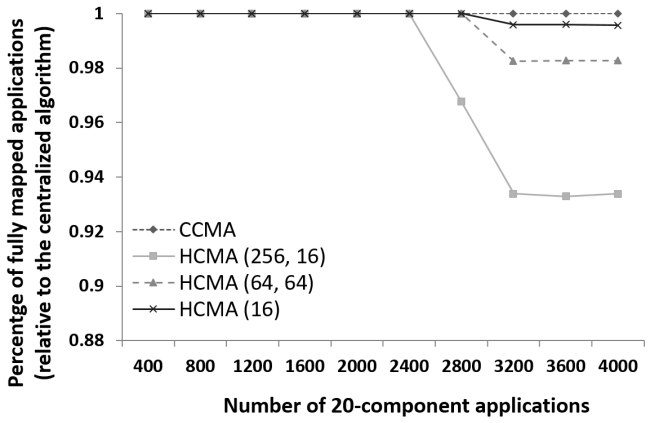

Fig. 18. The relative percentage of fully placed applications, compared to CCMA (case study 2 with 4,096 servers).

all, the CCMA uses the fewest and the HCMA with higher numbers of LLMs, uses the highest percentage of servers. In the worst case the hierarchical scenario with 100 LLMs uses 6.7% more servers. Moreover, the average standard error is 0.025% for the CCMA and 0.031% on average for the HCMAs.

In Fig. 14, the percentage of placed applications is depicted. As the results show, the CCMA offers the best performance and the HCMA with 100 LLMs the worst. Additionally, application placement failures are expected due to the fixed number of servers and resource saturation after 1,000 applications. Nonetheless, in both figures even in the worst case, the result is within 8% of the best result.

The execution time of the hierarchical approaches is promising. As can be clearly seen in Fig. 15, the time in which an application is placed in the CCMA is much higher than the hierarchical approaches, especially in the hierarchical management plane with more LLMs.

In Figs. 16–19 the percentage of servers used, the percentage of mapped applications, the percentage of mapped applications relative to the centralized approach and the average execution time per application are depicted respectively for the second case study. As can be observed from Fig. 16, the percentage of used servers increases by adding more applications up to when the servers are fully occupied. Afterwards, the percentages of mapped applications declines as the newly arriving applications are immediately rejected due to saturated resources. Although the CCMA shows better performance, the hierarchical management planes use at most 5.6% more resources. Fig. 17 and Fig. 18 compare the percentage of mapped applications in the hierarchical approaches to the centralized solution. As can be seen, in the worst case the result of the hierarchical management planes is within 7% of the best result. Also, Fig. 19 indicates that the execution time of the CCMA is high compared to the hierarchical scenarios.

We also evaluated the execution time and the number of fully mapped applications in different physical cloud systems with different numbers of servers and different numbers of switch ports. The applications are of type 1 based on Table 6. The number of servers and the number of intermediate switches are provided in Table 7 and the implemented management planes are presented in Table 8.

In Fig. 20 the number of fully mapped applications is depicted. As the branch factor (β) and consequently the number of servers increases, the number of mapped applications grows.

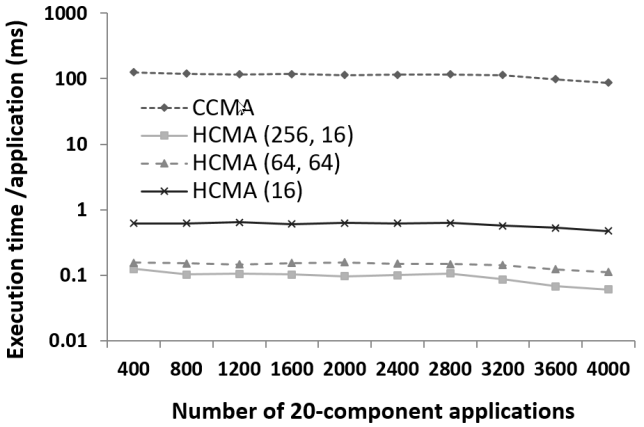

Fig. 19. Comparing the execution times per application (case study 2 with 4,096 servers and 20 iterations).

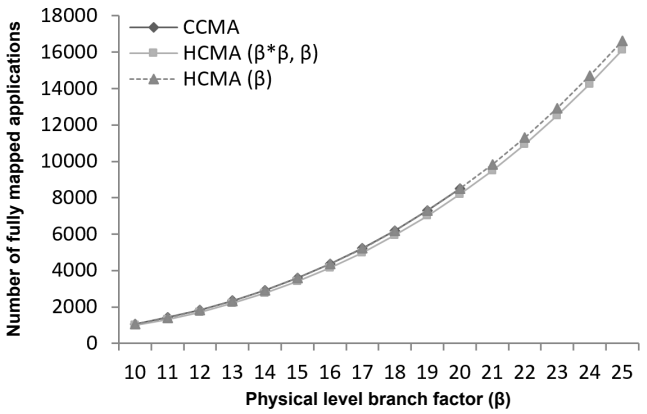

Fig. 20. The percentage of fully placed applications (20 iterations). Number of physical servers $= \beta^3$.

Table 7. The number of physical devices based on different β values.

β	**10**	**11**	**12**	**13**	**14**	**...**	**25**
$\|S\|$	1,000	1,331	1,728	2,197	2,744	...	15,625
$\|IS\|$	101	122	145	170	197	...	626
β	**20**	**30**	**40**	**50**	**60**	**70**	**80**
$\|S\|$	8 K	27 K	64 K	125 K	216 K	343 K	512 K
$\|IS\|$	401	901	1,601	2,501	3,601	4,901	6,401

Table 8. The management plane parameters based on different β values.

Type	$\|ML\|$	$\|SS\|$	$\|LLM\|$	$\|MLM\|$	$\|RLM\|$
CCMA	1	β^3	1	0	0
HCMA (β)	3	β^2	β	0	1
HCMA ($\beta * \beta, \beta$)	3	β	β^2	β	1

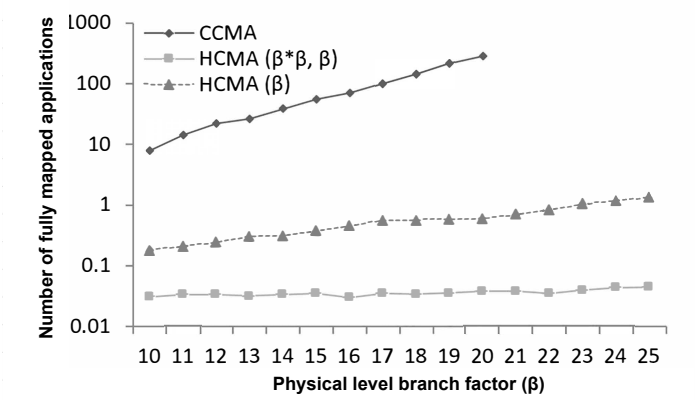

Fig. 21. The execution time per application (20 iterations). Number of physical servers $= \beta^3$.

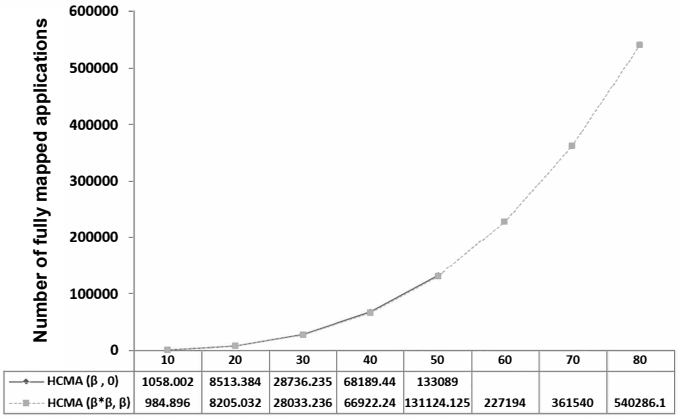

Fig. 22. The number of fully placed applications (10 iterations). Number of physical servers $= \beta^3$.

This evaluation shows that the HCMA with β LLMs is able to achieve the same performance of the CCMA, in terms of the number of fully mapped applications (with only 9 fewer applications on average). However, comparing to the HCMA with β^2 LLMs, the CCMA is able to map on average 6.2% more applications. In Fig. 21, the execution time of the different approaches is shown. As can be seen the execution time of the CCMA dramatically grows when the number of servers increases which makes the centralized algorithms inefficient in large scale cloud systems. Due to the increasing execution duration, we stop executing the CCMA once $\beta = 20$, indicating that the CCMA approach is not appropriate for a network larger than 8,000 servers. Instead in this evaluation, the HCMA with β number of low level managers has made a desired trade-off between the quality of application mapping and the execution time.

D. Large Scale Scenarios

In this phase, we focus on the scalability of the presented algorithms. We extend the scale of the experiments up to 512,000 servers and more than 540,000 5-component applications. In these experiments, the number of fully mapped applications is evaluated and the execution time per application is captured. The results are the average value of 10 experiments.

D.1 Evaluation Set Up

The experiments are conducted for an increasing number of servers from 1,000 up to 512,000 servers. The assumptions of the applications, of the physical networks and of the management planes are provided in Table 6, Table 7, and Table 8 respectively.

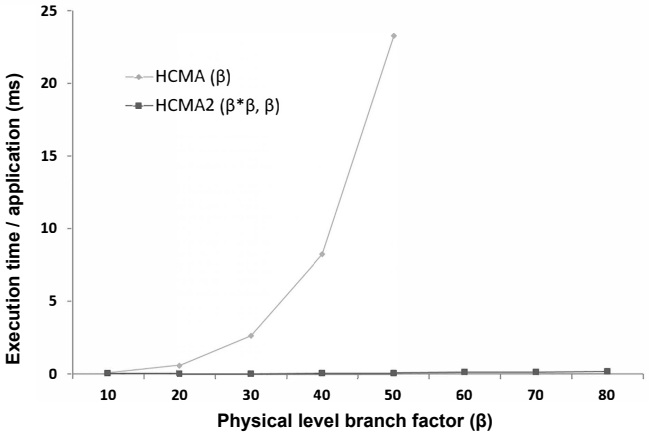

Fig. 23. The execution time per application (10 iterations). Number of physical servers $= \beta^3$.

D.2 Evaluation Results

Fig. 22 compares the number of fully mapped applications for two different hierarchical management plane architectures. The numbers of successfully mapped applications are close, but the management plane with a larger number of supported servers in each administrative domain allocates on average 3.4% more applications. Nonetheless, while the execution time of this approach grows exponentially, the HCMA with more LLMs shows better performance, as can be clearly seen in Fig. 23. As a result, for experiments larger than 125,000 servers, only the second hierarchical architecture is evaluated.

E. Evaluation Discussion

We have extensively assessed the CCMA and HCMA approaches. Our evaluation studies show that the best performance is constantly achieved by the centralized CCMA approach, compared to the hierarchical management planes, in terms of percentage of application placement and network resource usage. However, the execution time of CCMA dramatically grows when the number of servers increases. This makes the centralized algorithms inefficient in large scale cloud systems. While the CCMA approach is not appropriate for a network larger than 8 k servers (enough capacity to fully map 8,512 small applications), HCMA with β number of LLMs has made a desired trade-off between the quality of application mapping and the execution time. Moreover, a larger scale evaluation reveals that although the HCMA with β LLMs is able to achieve the same performance of the CCMA, in terms of the number of fully mapped applications, this hierarchical architecture shows limited scalability up to 125 k servers with 133 k fully mapped applications. Our large-scale evaluation case studies indicate that the management architecture with β^2 LLMs is the most appropriate management plane for very large datacenters ($512\ k$ servers and more than 2.7 million application components).

VII. CONCLUSIONS

This article focused on the problem of component-level application placement in large-scale cloud environments. Our approach takes the characteristic of the underlying network into account and works with multi-component applications, taking into account the application workflow with a distinction between application component types. To offer an optimal solution, we first presented an ILP-based model and to have a scalable solution, a near-optimal centralized approach was proposed and compared to the optimal solution. Due to limited scalability of the centralized approaches, a hierarchical heuristic was also designed to be deployed in large-scale cloud management systems. The experimental results showed that in large-scale clouds our proposed approach works efficiently compared to a centralized and optimal management systems in terms of resource usage and quality of application placement. The percentage of nodes used and the percentage of mapped applications remain close to that of the centralized algorithm, in the worst case within 6.7% and 8% respectively.

REFERENCES

[1] Y. Li *et al.*, "Self-adaptive resource management for large-scale shared clusters," *J. Comput. Sci. Technol.*, vol. 25, no. 5, pp. 945–957, 2010.
[2] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Inf. Sci.*, vol. 275, pp. 314–347, 2014.
[3] L. Shi, B. Butler, R. Wang, D. Botvich, and B. Jennings, "Optimal placement of virtual machines with different placement constraints in iaas clouds," in *Proc. IET CIICT*, 2012, pp. 35–40.
[4] B. Urgaonkar, A. L. Rosenberg, and P. Shenoy, "Application placement on a cluster of servers," *J. Foundations Comput. Sci.*, vol. 18, no. 05, pp. 1023–1041, 2007.
[5] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," vol. 41, no. 4, pp. 242–253, 2011.
[6] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 199–210, 2012.
[7] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. USENIX ATC*, 2009, pp. 28–28.
[8] Z. Usmani and S. Singh, "A survey of virtual machine placement techniques in a cloud data center," *Procedia Comput. Sci.*, vol. 78, pp. 491–498, 2016.
[9] C. Pham, N. H. Tran, M. N. Nguyen, J. H. Son, and C. S. Hong, "Hosting virtual machines on distributed datacenters," in *Proc. ACM IMCOM*, 2016, p. 85.
[10] R. P. Esteves, L. Z. Granville, H. Bannazadeh, and R. Boutabai, "Paradigm-based adaptive provisioning in virtualized data centers," in *Proc. IFIP/IEEE IM*, 2013, pp. 169–176.
[11] M. F. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2013.
[12] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *J. Netw. Syst. Manage.*, vol. 23, no. 3, pp. 567–619, 2015.
[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. WWW*, 2007, pp. 331–340.
[14] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic application placement under service and memory constraints," in *Proc. WEA*, 2005, pp. 391–402.
[15] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé, "Utility-based placement of dynamic web applications with fairness goals," in *Proc. IEEE NOMS*, 2008, pp. 9–16.
[16] Z. Zhou, Z. Hu, and K. Li, "Virtual machine placement algorithm for both energy-awareness and SLA violation reduction in cloud data centers," *Scientific Programming*, vol. 2016, 2016.
[17] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based resource management for cloud environments (long version)," *KTH Royal Institute of Technology, Tech. Rep*, 2010.
[18] A. Nathani, S. Chaudhary, and G. Somani, "Policy based resource allocation in iaas cloud," *Future Generation Comput. Syst.*, vol. 28, no. 1, pp. 94–103, 2012.
[19] P. T. Endo *et al.*, "Resource allocation for distributed cloud: concepts and research challenges," *IEEE Network*, vol. 25, no. 4, pp. 42–46, 2011.

[20] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *J. Netw. Comput. Appl.*, vol. 66, pp. 106–127, 2016.
[21] Q. Zhang, M. Li, and X. Hu, "Network traffic-aware virtual machine placement with availability guarantees based on shadows," in *Proc. IEEE CCGrid*, 2014, pp. 542–543.
[22] Z. Zhuang and C. Guo, "Ocpa: An algorithm for fast and effective virtual machine placement and assignment in large scale cloud environments," in *Proc. IEEE CloudCom-Asia*, 2013, pp. 254–259.
[23] K.-y. Chen, Y. Xu, K. Xi, and H. J. Chao, "Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems," in *Proc. IEEE ICC*, 2013, pp. 3498–3503.
[24] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *Proc. ACM SC*, 2011, p. 22.
[25] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Proc. IEEE GCC*, 2010, pp. 87–92.
[26] L. Shi, B. Butler, D. Botvich, and B. Jennings, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds," in *Proc. IFIP/IEEE IM*, 2013, pp. 499–505.
[27] D. Breitgand and A. Epstein, "Sla-aware placement of multi-virtual machine elastic services in compute clouds," in *Proc. IFIP/IEEE IM*, 2011, pp. 161–168.
[28] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM GREENCOM-CPSCOM*, 2010, pp. 179–188.
[29] C. Adam and R. Stadler, "Service middleware for self-managing large-scale systems," *IEEE Trans. Netw. Service Manage.*, vol. 4, no. 3, pp. 50–64, 2007.
[30] M. Korupolu, A. Singh, and B. Bamba, "Coupled placement in modern data centers," in *Proc. IEEE IPDPS*, 2009, pp. 1–12.
[31] G. Foster, G. Keller, M. Tighe, H. Lutfiyya, and M. Bauer, "The right tool for the job: Switching data centre management strategies at runtime," in *Proc. IFIP/IEEE IM*, 2013, pp. 151–159.
[32] O. Biran *et al.*, "A stable network-aware vm placement for cloud systems," in *Proc. IEEE/ACM CCGRID*, 2012, pp. 498–506.
[33] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, 2012, pp. 2876–2880.
[34] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "CloudNaaS: A cloud networking platform for enterprise applications," in *Proc. ACM SoCC*, 2011, p. 8.
[35] L. Hu, K. D. Ryu, D. Da Silva, and K. Schwan, "v-bundle: Flexible group resource offerings in clouds," in *Proc. IEEE ICDCS*, 2012, pp. 406–415.
[36] G. Koslovski, S. Soudan, P. Gonçalves, and P. Vicat-Blanc, "Locating virtual infrastructures: Users and inp perspectives," in *Proc. IFIP/IEEE IM*, 2011, pp. 153–160.
[37] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba, "Vdc planner: Dynamic migration-aware virtual data center embedding for clouds," in *Proc. IFIP/IEEE IM*, 2013, pp. 18–25.
[38] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.
[39] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Proc. IEEE INFOCOM*, 2012, pp. 963–971.
[40] X. Zhu *et al.*, "1000 islands: An integrated approach to resource management for virtualized data centers," *Cluster Comput.*, vol. 12, no. 1, pp. 45–57, 2009.
[41] L. Parolini, N. Tolia, B. Sinopoli, and B. H. Krogh, "A cyber-physical systems approach to energy management in data centers," in *Proc. ACM/IEEE ICCPS*, 2010, pp. 168–177.
[42] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proc. IEEE ICDCS*, 2010, pp. 62–73.
[43] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. IEEE/ACM CCGRID*, 2010, pp. 826–831.
[44] B. Viswanathan, A. Verma, and S. Dutta, "Cloudmap: Workload-aware placement in private heterogeneous clouds," in *Proc. IEEE NOMS*, 2012, pp. 9–16.
[45] D. Jayasinghe *et al.*, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Proc. SCC*, 2011, pp. 72–79.
[46] H. Moens, J. Famaey, S. Latre, B. Dhoedt, and F. De Turck, "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," in *Proc. IFIP/IEEE IM*, 2011, pp. 137–144.
[47] *Hierarchical Network-Aware Placement of Service Oriented Applications in Clouds*, 2014.
[48] M. Barshan, H. Moens, S. Latre, and F. De Turck, "Algorithms for efficient data management of component-based applications in cloud environments," in *Proc. IEEE NOMS*, 2014, pp. 1–8.
[49] M. Barshan, H. Moens, and F. De Turck, "Design and evaluation of a scalable hierarchical application component placement algorithm for cloud resource allocation," in *Proc. CNSM*, 2014, pp. 175–180.
[50] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. USENIX NSDI*, 2012, pp. 225–238.
[51] C. Guo *et al.*, "Dcell: A scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, 2008.
[52] J. Lee *et al.*, "Application-driven bandwidth guarantees in datacenters," vol. 44, no. 4, pp. 467–478, 2014.
[53] L. Yu and H. Shen, "Bandwidth guarantee under demand uncertainty in multi-tenant clouds," in *Proc. IEEE ICDCS*, 2014, pp. 258–267.
[54] I. AMPL, *CPLEX software*. [Online]. Available: http://www.ilog.com/products/cplex
[55] T. Cormen, *Introduction to Algorithms*. MIT Press, 2009. [Online]. Available: http://books.google.be/books?id=Jwr8jwEACAAJ
[56] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," *Emerging Research Cloud Distrib. Comput. Syst.*, vol. 42, 2015.
[57] 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.
[58] J. Zhang *et al.*, "SLA aware cost efficient virtual machines placement in cloud computing," in *Proc. IEEE IPCCC*, 2014, pp. 1–8.

Maryam Barshan is a Ph.D. student in Computer Science Engineering at the Information Technology (INTEC) Department, the Internet based communication networks and services (IBCN) research group of Ghent University in Belgium. She obtained a M.Sc. degree in computer system architecture from Iran University of Science and Technology (IUST), Tehran, Iran. Her research interest includes cloud computing, network and service management and optimization.

Hendrik Moens received a M.S. degree in computer science from Ghent University in 2010, and received a Ph.D. on the management of customizable Software-as- a-Service from the same university in 2015. Now, he works at the Department of Information Technology (INTEC) at Ghent University - imec, researching cloud and network resource management and optimization.

Steven Latré is a Professor at the University of Antwerp and IMEC, where he is leading the IDLab Antwerp research group. He received a M.S. degree in Computer Science from Ghent University, Belgium and a Ph.D. in Computer Science Engineering from the same university. His research expertise focuses on management and control of large-scale and/or heterogeneous Internet of Things networks, both networking and computing applications. He has also been involved in and coordinated several national and European research projects. He is author or co-author of more than 80 papers published in international journals or in the proceedings of international conferences. He is the Recipient of the IEEE COMSOC award for best Ph.D. in network and service management 2012, the IEEE COMSOC Young Professional award 2015 and is a Member of the Young Academy Belgium.

Bruno Volckaert is Professor Advanced Programming and Software Engineering in the Department of Information Technology (INTEC) at Ghent University and senior researcher at imec. He obtained his Master of Computer Science degree in 2001 from Ghent University, after which he worked on his Ph.D. at Ghent University on data intensive scheduling and service management for Grid computing, which he obtained in 2006. His current research deals with reliable and high performance distributed software systems for City-of-Things (IoT for Smart Cities), distributed decision support systems for UAVs, intelligent railway transportation applications, and autonomous optimization of cloud-based applications. He has worked on over 35 national and international research projects and is author or co-author of more than 80 papers published in international journals and conference proceedings.

Filip De Turck leads the network and service management research group at the Department of Information Technology of the Ghent University, Belgium and IMEC. He (co-) authored over 450 peer reviewed papers and his research interests include telecommunication network and service management, and design of efficient virtualized network and cloud systems. In this research area, he is involved in several research projects with industry and academia, serves as Chair of the IEEE Technical Committee on Network Operations and Management (CNOM), and is on the TPC of many network and service management conferences and workshops. He serves as Associate Editor in Chief of IEEE Transactions of Network and Service Management (TNSM), and steering committee member of the IEEE Conference on Network Softwarization (IEEE NetSoft).