Hierarchical Network-Aware Placement of Service Oriented Applications in Clouds

Hendrik Moens, Brecht Hanssens, Bart Dhoedt and Filip De Turck
Ghent University – iMinds, Department of Information Technology
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium
e-mail: hendrik.moens@intec.ugent.be

***Abstract*—In cloud environments, resources can be requested on-demand when they are needed. A cloud management system is responsible for determining which physical machines are responsible for processing the requests. The problem of determining which servers are used for which services is referred to as the Cloud Application Placement Problem (CAPP), and multiple criteria such as cost and number of migrations must be taken into account. When applications are constructed as a collection of communicating services, such as in Service-Oriented Architectures, it becomes important to take the underlying network properties into account when these placement decisions are made. In this paper, we propose an Integer Linear Programming (ILP) formulation for the CAPP, which optimizes multiple criteria such as cost, latency and number of migrations between subsequent invocations by using multiple optimization criteria. We also present hierarchical algorithms based on particle swarm optimization and genetic algorithms to solve the CAPP. These algorithms are be executed within a management hierarchy, which reduces the amount of information needed for the algorithms to function, increasing scalability of the management system. Finally, we evaluate the hierarchical algorithms by comparing them to an optimal algorithm based on the ILP formulation.**

I. Introduction

The Cloud Application Placement Problem (CAPP) is used to determine on which machines applications and services are instantiated within clouds. This optimization problem is subject to many constraints, such as CPU, memory, bandwidth and management policies, and can greatly influence the number of requests that can be accepted, making it an important problem for cloud providers. While it is possible to determine optimal solutions for the CAPP, these solutions scale badly as the problem is NP-hard. This makes the calculation very time-consuming, and hardly scalable when the number of requests, applications, or resources changes over time.

We consider that applications that are composed of multiple communicating components, such as applications that are designed using a Service-Oriented Architecture (SOA). In such an architecture, applications are constructed by combining a set of communicating services. Each of these services can be provided by a virtual machine, and a service may be used by multiple tenants. When such an application is instantiated within a cloud, the services of each application are allocated on resources in the network, and may be distributed within the network. If this is the case, communication is required between these different resources, and the impact on the underlying network should therefore not be abstracted in the problem definition. Apart from bandwidth use, various other factors, such as latency, cost and number of service migrations between invocations in a dynamic scenario should also be taken into account.

978-1-4799-0913-1/14/$31.00

As the scale of cloud environments increases, the scalability of management applications becomes more important. Hierarchical management approaches [1] scale well, while still maintaining a good overview of the global system state: at lower levels of the hierarchy, the management system has a detailed view of a small part of the system state, while at higher levels in the hierarchy, a coarser, less detailed view of a larger part of the entire system is stored.

In this paper, we address three research questions: (1) How can the requirements, optimality, network-awareness, and migration-awareness, be represented formally and how can the various objectives be prioritized? (2) How can heuristic hierarchical algorithms, using limited local information to solve the CAPP, be designed? and (3) How do the heuristic algorithms perform compared to an optimal, ILP-based algorithm, both in terms of placement quality and execution speed? To achieve this, we propose a formal model for the network-aware CAPP which takes the network infrastructure of a cloud provider into account, that is designed to deal with applications built conforming a SOA. We also present two hierarchical algorithms, based on Particle Swarm Optimization (PSO) [2] and Genetic Algorithms (GAs) [3], that were designed to calculate scalable near-optimal solutions for this problem. The presented model and algorithms optimize multiple objectives: the number of accepted requests, and the network demand satisfaction are maximized, while the number of compute nodes used, the number of service migrations and the hop count between communicating services are minimized.

In the next Section we discuss related work. Subsequently, in Section III, we formally describe the network-aware CAPP. In Section IV the designed management algorithms are discussed. In Section V we describe the evaluation setup, which is followed by the evaluation in Section VI. Finally, in Section VII we state our conclusions.

II. Related Work

In this paper we study the application placement problem [4]. Multiple centralized solutions to solve this problem have been proposed [5], [6], [7], [8]. Fully distributed solutions, that lack a global system overview, have also been proposed in literature. These approaches work using peer-to-peer communication [9], [10] and economic approaches where requests are traded between nodes [11], [12].

In our previous work, we have designed hierarchical application placement algorithms [13] that were not network-aware as they focused on placing self-contained applications. Now, by contrast, we focus on managing applications consisting of communicating services, where the underlying network can not be abstracted. Additionally, in [14] and [15], we describe algorithms for placing highly variable applications on clouds. These algorithms were however not designed hierarchically, and while we focused on managing SOA applications, the underlying network was not taken into account. In this paper we, by contrast, do take the underlying network into account, and the algorithms are designed hierarchically, ensuring they scale better. We use the principles described in [16], where we discussed the network impact of adding and removing services within a cloud environment, to formally define the communication between services. Our previous work focused solely on determining the network impact of service changes, and not on allocating the services.

Our management approach incorporates multiple optimization objectives. While previously, work has been done in various areas such as maximal demand satisfaction [5], network-awareness [12], minimization of the number of migrations [7], [6], [5], and minimization of the hop count between communicating services [12], our work combines the various optimizations in a single model. Additionally, we describe scalable hierarchical algorithms to determine approximate solutions.

In [17], a PSO algorithm capable of scheduling workflows on servers is presented. The algorithm takes both the network and computation cost into account during its execution, but is centralized. Our approach, by contrast, is distributed using a management hierarchy, making it more scalable. Additionally, our approach differs in that it focuses on allocating server resources for longer-running applications, rather than scheduling short-running workflows, which is why the algorithms described in this paper also take the number of migrations into account.

III. Problem Model

The CAPP takes into account a set of applications that are constructed by combining multiple communicating services, and a set of resources on which the services are instantiated. Based on this information, a placement must be determined where the services are allocated on the resources, subject to a collection of constraints:

- Resources have limited CPU and memory capacities.
- Some services cannot be instantiated on every resource, due to the requirement of specific hardware, software, or because of business or security policies.
- Communication links between resources in the network have limited bandwidth capacities.

The application services are assumed to be multi-tenant, which implies that one service instance can be used by multiple applications. Thus, a single service instance, having a fixed memory requirement, can be used by multiple applications.

The problem is optimized according to multiple objectives. This is achieved by sequentially executing different optimizations, where the results from previous optimizations are added as constraints for the next solution. As the second objective must also satisfy the optimal condition of the first problem, the size of the solution space continuously decreases. This also implies that the first objectives are deemed more important, as they take priority over the other objectives. The optimization objectives are (from most to least important):

1) First, the number of accepted application requests is maximized. Applications are weighted based on their CPU needs, ensuring the maximum amount of CPU utilization is achieved. Alternative weighting approaches, such as one based on priory or utility can also be used.
2) Subsequently, the satisfaction of the bandwidth requirement of all services is maximized, ensuring the communication requirements between services are guaranteed as best as possible.
3) Next, the total number of computation nodes used is minimized, making it possible to shut unused nodes down, improving energy-efficiency and reducing the cost of the placement.
4) Then, the number of service migrations between subsequent invocations of the algorithm invocations is minimized.
5) Finally, the hop count between nodes is minimized. This places communicating services as closely together as possible in order to reduce communication latency.

Multiple optimization objectives are considered within the model. To achieve this, the optimization is executed iteratively, with a different objective in every iteration. In every iteration, additional constraints and variables that are needed to formulate the new optimization objective are added to the model. Additionally the previous optimization objective and its value are then also added as additional constraints, imposing an upper or

Symbol	Description
$\mathcal{N}$	The set containing all nodes on which services can be executed.
$\mathcal{E}$	The set containing all network edges that connect the nodes.
$\mathcal{A}$	The set of all applications. Every application consists of a set of communicating services.
$\mathcal{R}_a$	The set containing all requests for an application $a \in \mathcal{A}$.
$\mathcal{S}$	The set containing all of the services out of which the applications are composed.
D_a	The total number of requests for an application $a \in \mathcal{A}$.
Ω_n	The CPU capacity of the node $n \in \mathcal{N}$ (in GHz).
Γ_n	The memory capacity of the node $n \in \mathcal{N}$ (in GB).
ω_s	The CPU requirement of a service $s \in \mathcal{S}$ (in GHz).
γ_s	The memory requirement of a service $s \in \mathcal{S}$ (in GB).
$I_{a,s}$	The instance-matrix, indicating which services are part of which application. If $I_{a,s} = 1$, the service s is part of application a, and must be instantiated when the application is instantiated. If $I_{a,s} = 0$, the service s is not part of application a.
$R_{s,n}$	The relation matrix is used to indicate which services may be executed on which node. If $R_{s,n} = 1$, the service s can be executed on node n. If $R_{s,n} = 0$, no instances of service s may be allocated on node n.
C_{s_1,s_2}	The communication matrix is used to determine the amount of bandwidth (in Mbit/s) that is needed between services for an application. C_{s_1,s_2} represents the amount of bandwidth needed between services s_1 en s_2.
B_{n_1,n_2}	The bandwidth matrix, where B_{n_1,n_2} contains the capacity available between nodes n_1 and n_2 (in Mbit/s).
H_{n_1,n_2}	The hop count matrix, where H_{n_1,n_2} indicates the number of devices, such as routers, are present between the nodes n_1 and n_2.

TABLE I: Input variables of the CAPP.

Symbol	Description
$G_{a,r}$	The acceptance matrix. $G_{a,r} = 1$ indicates that the rth request for application a could be accepted. $G_{a,r} = 0$ if the request is not accepted.
$P_{s,n}^{a,r}$	The placement matrix. $P_{s,n}^{a,r} = 1$ indicates that an instance of service s is executed on node n for the rth request of application a. Otherwise, $P_{s,n}^{a,r} = 0$.
$U_{s,n}$	The execution matrix. If $U_{s,n} = 1$, an instance of service s is executed on node n. Otherwise, $U_{s,n} = 0$.
U_n	The node utilization matrix. $U_n = 1$ indicates that n is being used by at least a single service. If $\vec{U}_n = 0$ the service is unused.
$F_{s_1,s_2}^{a,r}(n_1, n_2)$	The flow matrix. $F_{s_1,s_2}^{a,r}(n_1, n_2)$ contains the amount of bandwidth (in Mbit/s), belonging to the rth request for application a, that is used for communication between services s_1, executed on node n_1, and s_2 executed on node n_2.
$z_{s_1,s_2}^{a,r}$	The percentage of requested service bandwidth that is guaranteed for every flow between services s_1 and s_2 for the rth request of application a.

TABLE II: The decision variables of the CAPP.

lower bound for its value. Thus, every subsequent optimization extends the model defined in the previous optimization. This approach ensures that every new iteration further refines the previous result, improving it based on additional criteria. The first optimization objective, *maximization of the number of accepted requests* thus defines the bulk of the formal model, while subsequent objectives only add limited numbers of decision variables to the defined model. An overview of the used symbols is shown in Table I and Table II.

A. Maximizing the Number of Accepted Requests

Given are a set of applications $\mathcal{A}$, and a set $\mathcal{R}_a$ containing all requests for an application $a \in \mathcal{A}$. A discrete input variable D_a represents the total number of requests for an application a. Every application a consists of a collection of communicating services, which are contained in the set $\mathcal{S}$. Every service $s \in \mathcal{S}$ has a given CPU and memory demand, respectively represented by ω_s (in GHz) and γ_s (in GB). A binary instance matrix I indicates whether service s is part of application a, which is the case if $I_{a,s} = 1$. The services are to be placed on computation nodes that are contained in the set $\mathcal{N}$. Every node $n \in \mathcal{N}$ has a CPU capacity, Ω_n (in GHz) and a memory capacity Γ_n (in GB).

Additionally, we define a binary relation matrix R, that indicates which services may be executed on which nodes. If $R_{s,n} = 1$, the service s can be executed on node n. Otherwise, the service may not be executed on the chosen node. This could, for example, occur because specific hardware is needed for executing a service, due to security policies, or due to software licensing restrictions.

A binary acceptance matrix G is used to indicate which requests $r \in \mathcal{R}_a$ can be accepted. $G_{a,r} = 1$ indicates that the rth request for an application a can be placed. Using these parameters, the maximization of the number of accepted request can be represented as shown in Equation (1). This equation maximizes the CPU utilization of the datacenter.

$$\max \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}_a} G_{a,r} \times \left(\sum_{s \in \mathcal{S}} I_{a,s} \times \omega_s \right) \tag{1}$$

The optimization objective in Equation (1) is subject to multiple constraints. To represent these constraints, additional decision variables are needed.

First, we define the placement matrix P, which is used to represent the final service placement. If $P_{s,n}^{a,r} = 1$, this implies that a service s is executed on node n for the rth request of application a; otherwise, if $P_{s,n}^{a,r} = 0$, this is not the case. For an application a to be active, however, each of its services s from which is composed must also be executed within the network. Equation (2) ensures that if a request r for application a can not be satisfied, none of its services are placed within the network. Equation (3) ensures a service request can be executed on at most one server. Finally, Equation (4) states that all of the services s of an application a must be executed in the network before the application is placed.

$$\forall a \in \mathcal{A}, r \in \mathcal{R}_a, s \in \mathcal{S}, n \in \mathcal{N} : P_{s,n}^{a,r} \leq G_{a,r} \tag{2}$$

$$\forall a \in \mathcal{A}, r \in \mathcal{R}_a, s \in \mathcal{S} : \sum_{n \in \mathcal{N}} P_{s,n}^{a,r} \leq 1 \tag{3}$$

$$\forall a \in \mathcal{A}, r \in \mathcal{R}_a : G_{a,r} \times \sum_{s \in \mathcal{S}} I_{a,s} = \sum_{s \in \mathcal{S}} \sum_{n \in \mathcal{N}} P_{s,n}^{a,r} \tag{4}$$

Additionally, the CPU and memory limits of the applications must be respected when services are executed. For this, we first define the execution matrix U. The binary decision variable $U_{s,n} = 1$ indicates that an instance of service s is executed on node n. With this, we can express that the sum of the CPU of all services s running on a node n must be less than the available CPU capacity of the node; this is shown in Equation (5). Similarly, in Equation (6), a memory limit is defined, but as memory can be shared between multiple application instances, this memory cost is only added once if the service is used on the server, resulting in a different formulation.

$$\forall n \in \mathcal{N} : \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}_a} \sum_{s \in \mathcal{S}} P_{s,n}^{a,r} \times \omega_s \leq \Omega_n \tag{5}$$

$$\forall n \in \mathcal{N} : \sum_{s \in \mathcal{S}} U_{s,n} \times \gamma_s \leq \Gamma_n \tag{6}$$

$$U_{s,n} \in \{0, 1\} \tag{7}$$

Equation (8) ensures $U_{s,n}$ can only take on value 0 if the service is not used for any application request: the left part of the inequality represents the total number of services s that are executed on node n, while the right part of the inequality is a binary variable indicating whether at least one instance of the service s is executed on node n which is multiplied by an upper limit to the number of services that can be executed. Equation (9) ensures that the constraints defined in the relation matrix R, indicating whether or not a service may execute on a node, are respected.

$$\forall s \in \mathcal{S}, n \in \mathcal{N} : \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}_a} P_{s,n}^{a,r} \leq U_{s,n} \times \left(\sum_{a \in \mathcal{A}} D_a \times I_{a,s} \right) \tag{8}$$

$$\forall s \in \mathcal{S}, n \in \mathcal{N} : U_{s,n} \leq R_{s,n} \tag{9}$$

In the network, bandwidth is allocated between every pair of communicating service s_1 and s_2 that are part of the same request r. As not all requests are continuously active however, it is not necessary for the entire demand to be available at all times. Therefore, a request is permitted on the network if 80% of its bandwidth requirement can be guaranteed at all times. In a subsequent optimization, an effort is made to increase the amount of bandwidth that is guaranteed. This two-stage approach to allocating bandwidth is similar to the one we described in [16].

The CAPP is extended by incorporating every communication link $e \in \mathcal{E}$ in the network, and defining the bandwidth matrix B, where B_{n_1,n_2} indicates the amount of bandwidth (in Mbit/s) available between the pair of nodes n_1 and n_2. To model the amount of bandwidth needed between services, the communication matrix C is defined, where C_{s_1,s_2} indicates the amount of bandwidth needed between services s_1, which is the source of the communication, and s_2, which is the sink of the network flow. The flow passing over the edges is stored in the flow matrix F. $F^{a,r}_{s_1,s_2}(n_1,n_2)$ contains the amount of bandwidth used by services s_1 and s_2 for the rth request of application a when the services are executed on nodes n_1 and n_2.

The additional network constraints are similar to those introduced in [16]. Two additional constraints are needed to correctly represent network flows: *edge capacity limitations*, expressed in Equation (10), limiting the amount of bandwidth of network edges, and *flow conservation*, which ensures no flow is lost within the network, which is expressed in Equation (11) and Equation (12). In this formulation, $z^{a,r}_{s_1,s_2}$ is a decision variable that contains the percentage of the requested bandwidth between services s_1 and s_2 that is guaranteed for the rth request of application a.

$$\forall n_1, n_2 \in \mathcal{N} : \sum_{a\in\mathcal{A}} \sum_{r\in\mathcal{R}_a} \sum_{s_1,s_2\in\mathcal{S}} F^{a,r}_{s_1,s_2}(n_1,n_2) \leq B_{n_1,n_2} \tag{10}$$

$$\begin{aligned} &\forall a \in \mathcal{A}, r \in \mathcal{R}_a, s_1, s_2 \in \mathcal{S}, n \in \mathcal{N} : \\ &F^{a,r}_{s_1,s_2}(n) = \sum_{(m,n)\in\mathcal{E}} F^{a,r}_{s_1,s_2}(m,n) - \sum_{(n,m)\in\mathcal{E}} F^{a,r}_{s_1,s_2}(n,m) \end{aligned} \tag{11}$$

$$F^{a,r}_{s_1,s_2}(n) = \begin{cases} -z^{a,r}_{s_1,s_2} \times C_{s_1,s_2} & \text{if } n \text{ is the } \textit{source} \text{ of } s_1 \\ +z^{a,r}_{s_1,s_2} \times C_{s_1,s_2} & \text{if } n \text{ is the } \textit{sink} \text{ of } s_2 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

In the expression of the flow conservation constraint in Equation (11), an additional decision variable $F^{a,r}_{s_1,s_2}(n)$ is introduced, which represents the net flow on every node n, and which can be computed using the flow matrix F. The sum of incoming and outgoing flows on a node should equal this net flow value, the value of which is defined in Equation (12): the source has a negative net flow, the sink has a positive net flow, and for other nodes this flow must be 0.

As mentioned previously, we only consider an application to be placed if at least 80% of its net flow is achieved, even if its CPU and memory requirements can be achieved. This is expressed in Equation (13), the final constraint of the accepted requests optimization.

$$\forall a \in \mathcal{A}, r \in \mathcal{R}_a, s_1, s_2 \in \mathcal{S} : G_{a,r} \leq z^{a,r}_{s_1,s_2} + 0.2 \tag{13}$$

B. Maximizing the Satisfied Service Bandwidth Demand

Within the previous model, bandwidth and the underlying network are already incorporated, as 80% of the required bandwidth must be guaranteed. The objective of this second optimization is to further increase the allocated bandwidth, ensuring as much capacity as possible is allocated to the application services. This can be achieved by maximizing the sum of the z values defined in the previous section. This maximization is expressed in Equation (14).

$$\max \sum_{a\in\mathcal{A}} \sum_{r\in\mathcal{A}_r} \sum_{s_1,s_2\in\mathcal{S}} z^{a,r}_{s_1,s_2} \tag{14}$$

C. Minimizing the Number of Used Computation Nodes

A third optimization objective is the number of computation nodes used in a placement, as reducing the number of server used can result in cost and energy savings. To achieve this an additional binary decision variable U_n is defined, where $U_n = 1$ indicates a node n is used in the placement. This results in the optimization objective shown in Equation (15), where the number of used servers is minimized.

$$\min \sum_{n\in\mathcal{N}} U_n \tag{15}$$

The U_n decision variables are subject to additional constraints, ensuring they can only take on value 1 if there are no active applications. This is expressed in Equation (16).

$$\forall n \in \mathcal{N} : \qquad \sum_{s\in\mathcal{S}} U_{s,n} \leq U_n \times |\mathcal{S}| \tag{16}$$

$$U_n \in \{0,1\} \tag{17}$$

D. Minimizing the Number of Migrations Between Algorithm Invocations

In a dynamic management scenario, services can be migrated from one node to another node. Despite this, it is preferable to select a solution where the amount of resource migrations is kept at a minimum. To minimize this information, an addition input variable, U^{t-1}, the execution matrix resulting from the previous invocation of the placement algorithm, is used. This matrix corresponds to the execution matrix of the current iteration U, but while U contains decision variables, U^{t-1} contains constant values. Minimizing the number of migrations compared to the previous invocation can be expressed as shown in Equation (18).

$$\min \sum_{s\in\mathcal{S}} \sum_{n\in\mathcal{N}} |U_{s,n} - U^{i-1}_{s,n}| \tag{18}$$

E. Minimizing the Hop-Count Between Service Instances

A final objective is to minimize the network latency of the communication between services. We achieve this by minimizing the hop count between the services, which we use as a metric to define the distance between nodes. This information is an additional input variable which is stored in the hop count matrix H. The variable H_{n_1,n_2} contains the number of hops (e.g. routers) between nodes n_1 and n_2. We use a hop count matrix as it is easier to determine these values than it is to accurately estimate the network latency: the latter can vary throughout the execution, so it would have to be continuously updated. For increased accuracy, but at the cost of additional management overhead, the hop count matrix can be replaced by a latency matrix without any other changes to the model.

We introduce a measure, referred to as the Bandwidth-Hopcount Distance (BHD), indicating the efficiency of the network routing, which is determined using the total network flow between nodes and the hop count between the nodes. The BHD will have a lower value if smaller hop counts are used but this is weighted by the network bandwidth: if a low amount of bandwith is needed, the penalty of using nodes that are further away lessens. Using the flow matrix F this can be stated as shown in Equation (19). This objective ensures the largest amount network communication occurs between nodes close to each other.

$$\min \sum_{n_1,n_2 \in \mathcal{N}} H_{n_1,n_2} \times \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}_a} \sum_{s_1,s_2 \in \mathcal{S}} F^{a,r}_{s_1,s_2}(n_1,n_2) \quad (19)$$

F. Integer Linear Programming (ILP) Algorithm

The formal problem description, discussed in this section was used to define an Integer Linear Programming (ILP)-based algorithm which was implemented in Java using the CPLEX [18] ILP solver. This solver can, for small problem sizes, determine the optimal solution of the CAPP using Simplex and Branch and Bound algorithms. As noted, the optimization is executed five times for the different objectives, where the result of the previous iterations are added as additional restrictions to the model for the subsequent optimizations.

IV. Hierarchical CAPP Algoritms

An ILP-based algorithm for solving the CAPP was described in the previous section. In this section, we describe two heuristic algorithms, based on PSO and GAs, for solving the CAPP. These algorithms are executed within a hiearchic management system, which is also explained in this section. Both approaches are based on meta-heuristics, and are entirely defined by how the specific subfunctions are defined. Therefore we will focus on how these specific functions are defined.

A. Management Hierarchy

The structure of the management system is based on that proposed in [13] and is structured hierarchically. Leaf nodes are referred to as execution nodes, and inner nodes are referred to as management nodes. The execution nodes are responsible for executing the cloud applications, while the management nodes are responsible for executing the CAPP algorithm. For the algorithms we use four layers to manage the network, organized as in Figure 1. Initially, all requests are passed to the lowest management nodes, whose children are execution nodes. Out of all these lowest level nodes, the one that has most available resources is chosen, which is the cluster with the lowest ratio between used CPU capacity and total available CPU capacity. If this resource is unable to process the request, it is passed on to higher level management nodes. Using this approach, the lowest management nodes only have a limited view of the network and are used to process the majority of the requests. Higher level management nodes have a broader system overview, and thus take longer to execute the CAPP algorithms, but also process less requests.

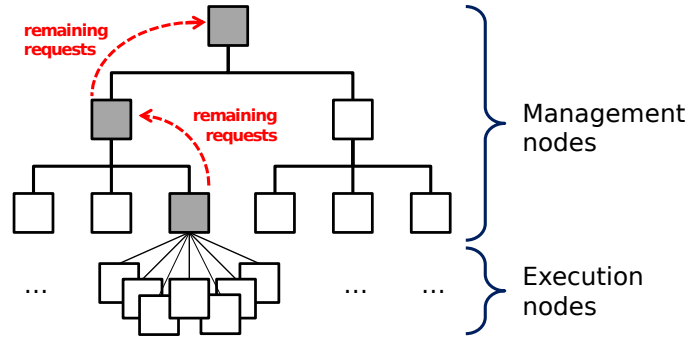

Fig. 1: The hierarchical approach to CAPP. When placement requests cannot be handled by lower level management nodes they are passed on to higher-level management nodes.

B. Particle Swarm Optimization (PSO)

PSO [2] is a strategy for defining heuristics based on simulating the behavior of a swarm of particles. Every particle determines its best position, and the entire swarm is aware of current best position. The speed of particles is dependent on the local best position and the global best position, ensuring every particle searches in its own vicinity while still moving towards a known good position. To build a PSO-based algorithm, multiple factors must be determined: (1) first a particle representation must be determined, making it possible to map particles to a position in a multi-dimensional space; (2) next, a measure of the quality of a solution must be determined, making it possible to find out which solutions are better than others; and finally (3) a velocity is used to determine how particles move throughout the solution space during subsequent iterations.

1) Particle Representation: Every particle represent a solution of the CAPP. Every application, service and node in the system is assigned a number. We define a solution as a 3-dimensional array of elements. The first dimension is the application, the second dimension is the request number of the application, and finally, the third dimension are the services. Contained in the array is the number corresponding to the node on which the service is executed, or -1 if the services are not instantiated. This is illustrated in Figure 2.

2) Particle Quality Evaluation: Every particle in the swarm is assigned a quality based on a quality function. This objective function is shown in Equation (20). The various parameters are

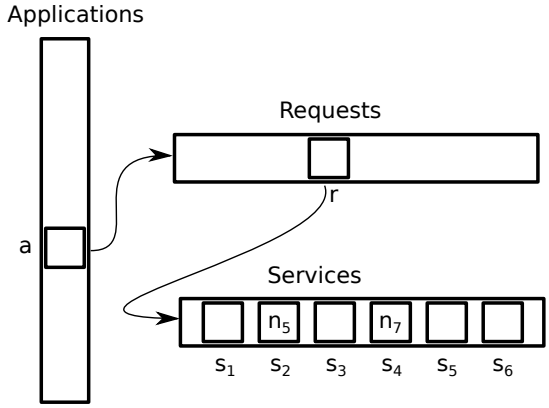

Fig. 2: Particles are represented as a multi-dimensional array, where the dimensions are application, request, and service. The value of the array indicates the server on which a given service is executed. In this example, service s_2 for request r of application a is executed on node n_5.

TABLE III: Parameters used in the PSO objective function.

Symbol	Description
$placedCPU$	The total CPU that is placed for the applications.
$totalCPU$	The total CPU that is requested.
$totalRequests$	The number of application requests.
$placedRequests$	The number of placed application requests.
$\|\mathcal{N}\|$	The number of nodes available within the network.
$usedNodes$	The number of nodes used by the placement.
$migrationCount$	The number of migrations for this placement.
$\|\mathcal{S}\|$	The number of services.

explained in Table III. If a particle violates any of the model constraints, its quality is reduced to 0.

$$\begin{aligned} obj &= \left(\frac{placedCPU}{requestedCPU}\right)^2 \\ &\times \left(\frac{placedRequests}{totalRequests}\right) \\ &\times \left(1 - \frac{usedNodes}{|\mathcal{N}|+1}\right) \\ &\times \left(1 - \frac{migrationCount}{|\mathcal{N}| \times |\mathcal{S}|+1}\right) \end{aligned} \tag{20}$$

3) Particle Velocity: The positions of particles are modified iteratively. During every iteration, the speed vector of a particle, which is used to determine its next position within the solution space, is modified based on the quality of the particle. Every particle retains the best position it has encountered, $pbest$. Additionally, the entire swarm is aware of the global best position $gbest$. The speed vector $v_{p,d}$ of a particle p for a dimension d is changed using Equation (21). The factors r_1 and r_2 are random numbers chosen in the interval $\{0, 1\}$ using a uniform distribution. The parameters c_1 and c_2 modify the weight of both parameters; if $c_1 < c_2$, a solution closer to the global best value will be chosen, while if $c_2 > c_1$ a solution closer to the personal best will be chosen. We choose $c_1 = 2$ and $c_2 = 1$ based on empirical evaluations.

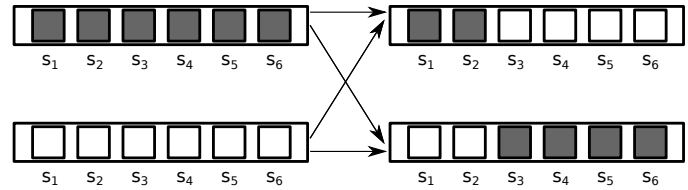

Fig. 3: An illustration of crossover.

$$\begin{aligned} v_{p,d} &\leftarrow v_{p,d} \\ &+ \frac{c_1 \times r_1 \left(pbest_d - x_{p,d}\right) + c_2 \times r_2 \left(gbest_d - x_{p,d}\right)}{c_1 + c_2} \end{aligned} \tag{21}$$

Equation (21) describes how the new speed of particles is determined. Based on the resulting speed vector v_p, the new position x_p of all particles can be determined. Determining these new positions is done using Equation (22), where the new position of a particle p for a given dimension d is shown.

$$x_{p,d} \leftarrow x_{p,d} + v_{p,d} \tag{22}$$

C. Genetic Algorithm (GA)

A second heuristic for solving the CAPP is constructed using GAs. GAs, like PSO, simulate natural processes for solving problems. The process is based on natural selection. A population of solutions is maintained, and new solutions are created by combining two existing solutions. Between different iterations, natural selection occurs based on the solution *fitness*, where better solutions have a higher chance of survival. To create a GA, three functions must be defined: (1) a solution must be represented using an identifying string of characters, similar to a chromosome for living creatures, (2) a method must be defined to combine two solutions into a new solution that has properties of both parent solutions, and (3) a fitness function, used to determine the quality of a solution must be defined.

We re-use both the solution representation and the quality function used for the PSO algorithm. Creating new solutions is done using one-point crossover, where at the start of a first solution is combined with the end of another solution, resulting in two new solutions. The point where crossover occurs is chosen uniformly and at random. This process is illustrated in Figure 3. Additionally there is a 10% chance for mutation to be applied to a solution, where values are randomly changed to a different value. This process of mutation is used to add additional randomness, and to prevent the algorithm from getting stuck in a local optimum.

V. Evaluation Setup

To evaluate the algorithms, we consider generated scenarios where long-running service workflows are active within a datacenter. When determining problem inputs to compare the ILP, PSO and GA algorithms, very small problem sizes must be used, as the ILP scales very badly. For this reason, we use a small collection of 3 randomly generated applications that are composed of 4 services. The probability of a service belonging

to a given application is 60%. The CPU demand of a service is randomly chosen from the set $\{0.5, 0.6, 0.7, 0.8, 0.9, 1GHz\}$. Memory demand of a service is randomly chosen from the set $\{0.5, 0.7, 0.9, 1.1, 1.3, 1.5GB\}$. Every node of the management system manages two nodes, resulting in a hierarchy with 8 execution servers and 7 managing servers. The CPU capacity of a server[1] is randomly chosen from the set $\{9, 12, 15, 18, 21GHz\}$, and the memory capacity of servers is chosen from the set $\{6, 8, 10, 12, 14GB\}$. The communication demand between services is a random number in the range $[0.02, 0.04]$ Mbit/s. The hop count between pairs of execution servers are chosen at random between 1 and 10, while the available bandwidth between the servers is randomly chosen in the range $[6, 14]$ Mbit/s.

A second problem setup is used to compare the PSO and GA algorithms for larger problem configurations. In this second configuration, 10 applications composed of 13 services are considered, where there is a 40% chance that a service belongs to an application. The CPU and memory demand and capacities are chosen like in the small problem setup. A larger management hierarchy is used, with 160 execution servers and 21 management servers (every management either manages 4 other management servers or 10 execution servers). The communication demand between services is normally distributed with $\mu = 2.5$ and $\sigma = 0.20$. The hop count between servers is also normally distributed with $\mu = 7$ and $\sigma = 2$, while there is a 5% chance of there being a direct link between two nodes with a capacity which is normally distributed with $\mu = 20$ and $\sigma = 1$.

The complexity of these generated problems can vary greatly. To better characterize the difficulty of solving a problem, we define the CPU Load Factor (CLF), a factor to determine the CPU intensity of a given problem. This factor, shown in Equation (23), is computed by summing the total CPU demand of all requests, and dividing the result by the total available CPU within the network. The symbols used within the equation are shown in Table I.

$$CLF = \frac{\sum_{a \in \mathcal{A}} D_a \times \left(\sum_{s \in \mathcal{S}} I_{a,s} \times \omega_s\right)}{\sum_{n \in \mathcal{N}} \Omega_n} \tag{23}$$

All measurements are repeated 100 times.

VI. Evaluation Results

A. ILP, PSO and GA, small evaluation setup

First, the ILP, PSO and GA algorithms are compared using the small problem setup presented in Section V. In Figure 4 we compare the number of accepted requests for the various algorithms. We observe that, for lower CLF values, 100% of all requests can be fulfilled. As the CLF value increases, and the problems thus become harder, the number of accepted requests decreases. The PSO and GA algorithms both achieve results close to the optimal value, with the ILP algorithm placing $\pm 8\%$ more requests for an overloaded datacenter with CLF 1. For this scenario, there is no significant difference in the behavior of the PSO and GA algorithms.

[1]These values represent the processing power per core times the number of cores.

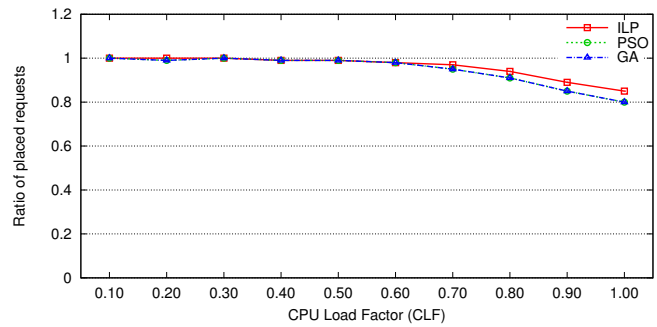

Fig. 4: Ratio of placed requests versus received requests for varying datacenter CPU loads using the small evaluation setup.

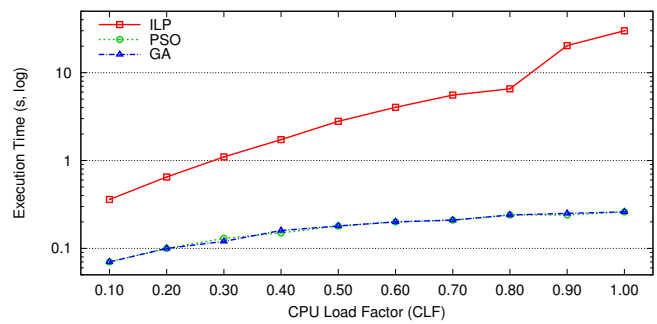

Fig. 5: Comparison of the execution speed of the algorithms.

The execution time of the ILP, PSO and GA algorithms is shown in Figure 5. The ILP requires significantly more time to calculate a solution compared to the PSO and GA algorithms. As the CLF increases, the gap between the optimal and heuristic algorithms further increases, with the execution speed of the PSO and GA algorithms remaining less than 0.3 seconds while the ILP algorithm requires 30 seconds to run.

B. PSO and GA, large evaluation setup

Using the larger problem setup, we compare the PSO and GA algorithms. In this scenario, network capacity becomes a bottleneck, and some applications can not be placed as it is not always possible to guarantee 80% of the requested network bandwidth. In Figure 6, the average network demand satisfaction of accepted applications is shown. For lower CLF values, nearly 100% demand satisfaction is achieved, but as the CLF approaches 1, the achieved network demand decreases until it is slightly over 80%, the minimal value.

In Figure 7, the ratio of placed requests for applications to the number of received requests is illustrated. We can see that the placement of applications indeed decreases from a CLF of 0.40. This corresponds to the decrease in bandwidth satisfaction observed in Figure 6.

The PSO and GA algorithms have a similar performance, with only minor differences. In Figure 6 we observe that the GA algorithm achieves a slightly higher average network demand satisfaction. This can be explained as the PSO algorithm accepts a little more requests. Because of this, the PSO algorithm has more active applications which in turn need more

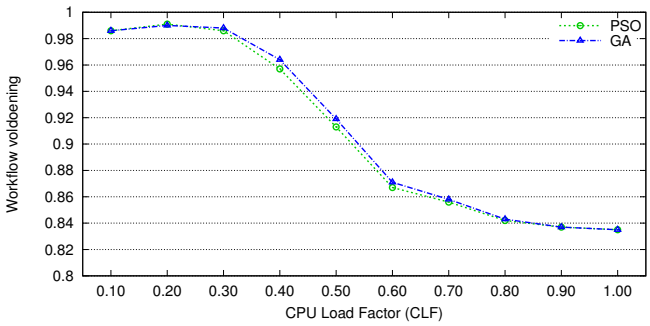

Fig. 6: Network demand satisfaction for varying datacenter CPU loads using the large evaluation setup.

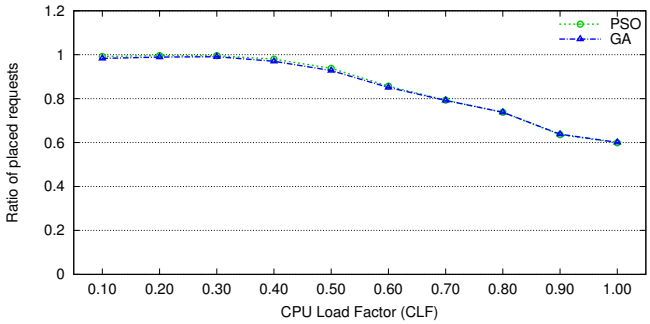

Fig. 7: Ratio of placed requests to received requests for varying datacenter CPU loads using the large evaluation setup.

bandwidth, increasing the total network demand and lowering the achieved demand. Thus, for this scenario, the PSO-based algorithm slightly outperforms the GA-based algorithm.

VII. Conclusion

In this paper, a generic model for the network-aware CAPP was presented that takes the network infrastructure of a cloud provider into account. The model optimizes multiple parameters such as achieved requests, achieved network demand, server use, the number of service migration and latency. We then presented a hierarchical cloud management architecture and two hierarchical algorithms based on PSO and GA to solve the network-aware CAPP. We evaluated the PSO and GA algorithms, and compared their performance with an optimal ILP algorithm. We found that the optimal algorithm manages to accept 8% more requests than the hierarchical PSO and GA algorithms, but the optimal algorithm can only computed for very small problem sizes as the computation time needed to determine the optimal value quickly becomes prohibitively large.

Acknowledgment

Hendrik Moens is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). This work was partly funded by Flamingo, an EU FP7 NoE project (318488). This work was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government – department EWI.

References

[1] J. Famaey, S. Latré, J. Strassner, and F. De Turck, "A hierarchical approach to autonomic network management," *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pp. 225–232, Apr. 2010.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, 1995, pp. 1942–1948 vol.4.

[3] M. Mitchell, *An introduction to genetic algorithms*. MIT Press, 1998.

[4] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating enterprise application placement in resource utilities," in *Self-Managing Distributed Systems: 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2003)*. Springer, 2004, pp. 118–129.

[5] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 331–340.

[6] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic application placement under service and memory constraints," in *Proceedings of the 4th international conference on Experimental and Efficient Algorithms*, Apr. 2005, pp. 391–402.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 595–604.

[8] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé, "Utility-based placement of dynamic web applications with fairness goals," in *Proceedings of the 11th Network Operations and Management Symposium (NOMS 2008)*. IEEE, 2008, pp. 9–16.

[9] C. Adam and R. Stadler, "Service Middleware for Self-Managing Large-Scale Systems," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 50–64, Dec. 2007.

[10] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based Resource Management for Cloud Environments," in *Proceedings of the 6th International Conference on Network and Service Management (CNSM 2010)*, 2010, pp. 1–8.

[11] Y. Li, F.-H. Chen, X. Sun, M.-H. Zhou, W.-P. Jiao, D.-G. Cao, and H. Mei, "Self-Adaptive Resource Management for Large-Scale Shared Clusters," *Science And Technology*, vol. 25, no. 2009, pp. 945–957, 2010.

[12] C. Low, "Decentralised Application Placement," *Future Generation Computer Systems*, vol. 21, no. 2, pp. 281–290, 2005.

[13] H. Moens, J. Famaey, S. Latré, B. Dhoedt, and F. De Turck, "Design and Evaluation of a Hierarchical Application Placement Algorithm in Large Scale Clouds," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, 2011, pp. 137–144.

[14] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, "Feature Placement Algorithms for High-Variability Applications in Cloud Environments," in *Proceedings of the 13th Network Operations and Management Symposium (NOMS 2012)*, 2012, pp. 17–24.

[15] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, "Cost-Effective Feature Placement of Customizable Multi-Tenant Applications in the Cloud," *Journal of Network and Systems Management*, Feb. 2013.

[16] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, "Network-Aware Impact Determination Algorithms for Service Workflow Deployment in Hybrid Clouds," in *Proceedings of the 8th International Conference on Network and Service Management (CNSM 2012)*, 2012, pp. 28–36.

[17] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in *24th IEEE International Conference on Advanced Information Networking and Applications*. Ieee, 2010, pp. 400–407.

[18] (2013) IBM ILOG CPLEX 12.4. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer