Algorithms for Efficient Data Management of Component-Based Applications in Cloud Environments

Maryam Barshan[1], Hendrik Moens[1], Steven Latré[2], Filip De Turck[1]

[1]*Ghent University - iMinds, Department of Information Technology*
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium

[2]*University of Antwerp – iMinds, Department of Mathematics and Computer Science,*
Middelheimlaan 1, B-2020, Antwerp, Belgium

e-mail: maryam.barshan@intec.ugent.be

Abstract— ***Cloud environments face a growing demand for application hosting, and applications consisting of multiple data-sources and storage components. The need to ensure service level agreements for these types of applications creates important challenges for cloud infrastructure providers. The main contribution of this paper is an optimal cost-effective model and two algorithms to map component-based data oriented applications to cloud platforms. The first algorithm is based on an Integer Linear Programming formulation and minimizes an objective function, taking into account the capacities of the available nodes and links, as well as the customer requirements. This algorithm is able to obtain the optimal solution, but shows a limited scalability. For this reason a heuristic algorithm is designed to solve the scalability issue. The experimental results thoroughly compare the execution times and obtained node usage for both algorithms.***

Keywords— **Data management, resource allocation, application placement, cloud management.**

I. INTRODUCTION

Traditionally, businesses had to build and maintain infrastructure to run their applications. The rise of cloud computing, which delivers reliable, scalable, and cost-effective computing resources to host the applications, has in recent years offered an alternative. As a result, the tendency of application providers to use these environments is on a rise. Nevertheless, many management challenges remain. In order to handle all requests for application execution, an intelligent management system is needed to place the applications onto physical infrastructure in an efficient way taking into account complex placement constraints.

In this paper, we present two algorithms for application placement of component-based applications in Clouds. Each application consists of sub-components that can be categorized into two different building blocks: data components (e.g. data sources, data stores) and computational components (e.g. application business logic or user interfaces). We made a distinction between these component types as their requirements are different. Data components store and manage data and are more storage intensive, whereas computational components are more CPU intensive. We consider each application as a workflow, consisting of computational components and data components.

In this paper we focus on an IaaS (Infrastructure as a Service) cloud where services are allocated using Virtual Machines (VMs). We specifically focus on supporting services to efficiently deploy applications, described as workflows. In this scenario, a *customer* refers to an application provider and *manager* refers to the infrastructure provider. Every customer or manager in the cloud environment has access to the application components through user interfaces and each cloud customer has specific Quality of Service requirements such as minimum bandwidth demand, throughput or maximum allowed latency. In general, it is better for each application component to have access to the data components, which are the nearest in distance (e.g., in the same LAN, if it is possible). However, this might be challenging due to capacity limitations of the cloud infrastructure and cost considerations.

Figure 1 shows an illustrative example of a cloud system, consisting of multiple domains and WAN links in between: the applications can be mapped over all domains of such a multi-domain cloud network, not just a single domain depending on the resource availability.

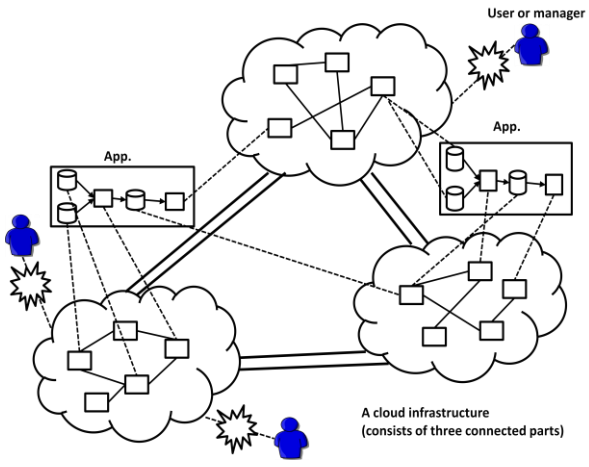

Fig 1. Application mapping in a cloud system.

978-1-4799-0913-1/14/$31.00 ©2014 IEEE

The objective is to map the component-based applications to the available physical nodes in an efficient way. For this reason, an optimization method based on an Integer Linear Programming (ILP) model is presented in this paper. Two different algorithmic approaches were studied and compared to each other.

This paper is organized as follows. In Section II related work is discussed. Section III describes the proposed method and the related constraints. In Section IV the implemented algorithms are discussed. The evaluation results are presented in Section V and finally, in Section VI, the conclusions are presented.

II. RELATED WORK

The use of virtualization techniques for cloud-based platforms and mapping multiple virtual networks over a single physical infrastructure, gained a lot of attention recently [1, 2, 3, 4, 5]. This paper does not focus on mapping of virtual networks on substrate networks, but concentrates on mapping of application components to cloud resources. Multiple application components can be mapped to the same physical resource, which is not the case in virtual network mapping algorithms.

Within cloud environments, application mapping and resource allocation are similar concepts that are both used to determine where applications are allocated. These terms are used to determine which node in physical part of cloud system is chosen to host the application. In recent years much work has been done relating this problem, each focusing on different aspects. An approach to autonomic resource management is proposed in [6] which aims at defining a cloud management system architecture and optimizing a utility function with regard to SLA requirements and management cost. The authors in [7] in their placement technique take Service Level Agreements into account.

Many application placement approaches [8, 9, 10, 11] focus on placing a set of independent applications, and do not take the underlying network into account. We, however, focus on applications that consist of multiple components, specifically focusing on the underlying network. Additionally, compared to previous work we also make a distinction between two different types of application components: data and analyzing components.

In addition, the model defined in this paper has similarity with [12] that described a linear application placement algorithm and [13] which is cost-aware and their mapping system works in application level. Our approach further works differently as it offers a component-level modeling and the component is characterized as being either a database or an application logic as has been explained.

In our previous work [14], algorithms for determining the impact of the underlying network on the performance of component-based applications were studied. The focus was specifically on determining the impact of a given service mapping, whereas this work focuses on determining the service mapping itself.

We have also previously described a method for hierarchically managing a cloud management system [15, 16]. In contrast, this previous work however focused specifically on addressing scalability and performance of cloud management systems, and did not take the underlying network into account. The techniques presented in [15, 16] can be used to improve the scalability of the algorithms described in this paper.

III. FORMAL PROBLEM FORMULATION

A. *Introduction to the model*

The model consists of two parts which can be represented as two different graphs, one for physical infrastructure and another for applications. The substrate network is considered as an undirected graph, and the application networks as directed graphs because of dependency between different components. Each infrastructure consists of nodes (N) and links (L). Each node has properties such as storage capacity (S), CPU capacity (C) and memory capacity (M). Each link has also delay (D) and bandwidth (BW) capacities.

It has to be noted that the physical network resources capacities are the residual capacities, considering the previous mappings.

The parameters of the physical network graph and their descriptions are listed in Table I.

$$G_{ph} = (N_{ph}, L_{ph})$$

$$\forall\, u \in \mathrm{N_{ph}} \quad \mathrm{C_u}, \mathrm{M_u}, \mathrm{S_u}, Ccost_u, Mcost_u,$$

$$Scost_u, \mathrm{BWcost_{e_{uv}}}, \; fixedcost_u \in \mathbb{N}^+$$

$$\forall\, e_{uv} \in \mathrm{L_{ph}} \qquad \mathrm{D_{e_{uv}}}, \mathrm{BW_{e_{uv}}}, \mathrm{fixedcost_{e_{uv}}} \in \mathbb{N}^+$$

$$, \mathrm{type_{e_{uv}}} \in \{0,1\}$$

TABLE I

THE PHYSICAL NETWORK AS A GRAPH

Variable	Description
G_{ph}	Physical Graph
N_{ph}	Physical nodes set in G_{ph}
L_{ph}	Physical links set in G_{ph}
C_u	Available computation capacity of physical node u
S_u	Available storage capacity of physical node u
M_u	Available memory capacity of physical node u
$D_{e_{uv}}$	Delay of physical link e_{uv}
$BW_{e_{uv}}$	Available Bandwidth (throughput) capacity of physical link e_{uv}
$type_{e_{uv}}$	Binary Variable. Phy. link e_{uv} is a LAN or WAN link
$Ccost_u$	Cost of each CPU unit of physical node u
$Mcost_u$	Cost of each memory unit of physical node u
$Scost_u$	Cost of each storage unit of physical node u
$BWcost_{e_{uv}}$	Cost of each BW unit of physical link e_{uv}
fixed $cost_u$	The fixed cost of using physical node u.
fixed $cost_{e_{uv}}$	The fixed cost of using physical link e_{uv}.

The binary variable $type_{e_{uv}}$ indicates type of physical link e_{uv} which this link is a WAN or LAN link.

$$type_{e_{uv}} = \begin{cases} 0, & Physical\ link\ e_{uv}\ is\ a\ LAN\ link \\ 1, & Physical\ link\ e_{uv}\ is\ a\ WAN\ link \end{cases}$$

Similarly, in the application network, application components (database components and computational component) and links between these components form a directed weighted graph. Each application component requires a specific amount of data sources, CPU power, memory and storage, etc. The definition of the parameters mentioned below and the explanation is listed in Table II.

$$G_{app} = (N_{app}, L_{app})$$

$$\forall\ ai\ \in N_{app} \qquad c_{ai}, m_{ai}, s_{ai} \in \mathbb{N}^{+},$$

$$\text{Database}_{\text{ai}}, Logic_{ai} \in \{0,1\}$$

$$\forall\ e_{aij} \in L_{app} \qquad d_{e_{aij}}, bw_{e_{aij}} \in \mathbb{N}^{+}$$

TABLE II

THE APPLICATION NETWORK AS A GRAPH

Variable	Description
G_{app}	application graph
N_{app}	Application component set in G_{app} (N_{app_d} or N_{app_l})
L_{app}	Set of links between application components in G_{app}
c_{ai}	Computation demand of app. *a*, comp. *i*
s_{ai}	storage demand of app. *a*, comp. *i*
m_{ai}	Memory demand of app. *a*, comp. *i*
$d_{e_{aij}}$	Max. allowed delay of link e_{aij} (link between comp. i and j of app. *a*)
$bw_{e_{aij}}$	Bandwidth demand of link e_{aij}
Database_{ai}	Binary variable has value 1 iff component *i* of application *a* is a database component
$Logic_{ai}$	Binary variable has value 1 iff component *i* of application *a* is a logic component

When we mention application component (N_{app}) we either refer to the data components or logic components. As mentioned and stated in Table III, we assume that the storage demand of data component is much higher than logic component and it is completely vice versa for CPU demand.

TABLE III

COMPONENT SETS OF AN APPLICATION

database component (N_{app_d})		logic component (N_{app_l})	
C_{appN_d}	CPU demand	C_{appN_l}	CPU demand
S_{appN_d}	storage demand	S_{appN_l}	storage demand
M_{appN_d}	memory demand	M_{appN_l}	memory demand
$S_{appN_d} \gg S_{appN_l}$		$C_{appN_d} \ll C_{appN_l}$	

B. *Decision variables*

The decision variables which have been presented in the ILP formula are described in this section, and are shown in Table IV. First, $\boldsymbol{x_u^{ai}}$ shows the accomplished map between application *a*, component ***i*** 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 there is no possibility to have a mapping between nodes or when the mapping has not been done although it was possible.

Next, the binary variable $\boldsymbol{f_{e_{uv}}^{e_{aij}}}$ indicates success of mapping between physical link $\boldsymbol{e_{uv}}$ and the link between components ***i*** and ***j*** of application ***a*** ($\boldsymbol{e_{aij}}$).

Moreover, we assume that each physical node is exclusively used for data components, or exclusively for execution of computational components. Binary variables T_u^l and T_u^d are defined for this purpose (Table V).

Furthermore, two other binary variables are defined: $\boldsymbol{B_u}$ is a binary variable to show whether physical node ***u*** is used, either as a routing node or a mapped one in the entire mapping, $\boldsymbol{B_{e_{uv}}}$ is another binary variable to indicate whether physical link $\boldsymbol{e_{uv}}$ is used in our mapping or not.

TABLE IV

DESCRIPTION OF EACH BINARY VARIABLE IN THE ILP MODEL

Decision variables	Description
$\boldsymbol{x_u^{ai}}$	Binary variable for mapping between nodes. 0 iff App.a, comp I is not mapped to phy. node u or mapping is not possible.
	$\boldsymbol{x_u^{ai}} \in \{0,1\} \quad \forall\ (u, ai) \in (N_{ph} \times N_{app})$
$\boldsymbol{f_{e_{uv}}^{e_{aij}}}$	Binary variable for mapping between links. 0 iff $\boldsymbol{e_{aij}}$ is not mapped onto $\boldsymbol{e_{uv}}$.
	$\boldsymbol{f_{e_{uv}}^{e_{aij}}} \in \{0,1\}\ \forall\ (e_{uv}, e_{aij}) \in (L_{ph} \times L_{app})$
$\boldsymbol{T_u^l}$	Binary variable, whether phy. node ***u*** is used for mapping application logic components. 0 iff no logic comp. is mapped onto node u.
	$\boldsymbol{T_u^l} \in \{0,1\} \qquad \forall\ u \in N_{ph}$
$\boldsymbol{T_u^d}$	Binary variable, whether phy. node ***u*** is used for mapping application database components. 0 iff no database comp. is mapped onto node u.
	$\boldsymbol{T_u^d} \in \{0,1\} \qquad \forall\ u \in N_{ph}$
$\boldsymbol{B_u}$	Binary variable, whether physical node ***u*** is used in mapping. 0 iff u is not used in the entire mapping.
	$\boldsymbol{B_u} \in \{0,1\} \qquad \forall\ u \in N_{ph}$
$\boldsymbol{B_{e_{uv}}}$	Binary variable, whether physical link $\mathbf{e_{uv}}$ is used in mapping. 0 iff $\mathbf{e_{uv}}$ is not used in the entire mapping.
	$\boldsymbol{B_{e_{uv}}} \in \{0,1\} \quad \forall\ e_{uv} \in L_{ph}$

TABLE V

DESCRIPTION OF DIFFERENT STATES OF $\mathbf{T_u^l}$ AND $\mathbf{T_u^d}$

T_u^l	T_u^d	**Description**
0	0	Physical node u is used neither in mapping database nor for logic component.
0	1	Physical node u is used in mapping database component.
1	0	Physical node u is used in mapping logic component.
1	1	It is not possible in our assumption.

C. *Objective function*

Guaranteeing the quality of service and taking other physical constraints into account, application placement has to be done in a way that delivers minimum cost mapping services. The optimization objective is minimizing the cost[1]. Moreover, since in multi-domain cloud networks the cost of LAN links are almost zero, for estimating the link cost, just the WAN links have to be taken into account. Consequently, the objective function can be assumed as the following. All the variables in this function are defined in the previous Section.

Minimize:

$$\sum_{\forall u \in N_{ph}} \sum_{\forall i \in N_{app}} cost(ai,u) + \sum_{\forall e_{uv} \in l_{ph}} \sum_{\forall e_{ij} \in l_{app}} cost(e_{aij}, e_{uv})$$

In more detail, this is represented as :

Minimize:

$$\sum_{\forall u \in N_{ph}} \left((\text{fixed cost}_u \times B_u) + \sum_{\forall i \in N_{app}} (c_{ai} \times Ccost_u + m_{ai} \times Mcost_u + s_{ai} \times Scost_u) \times x_u^{ai} \right)$$
$$+ \sum_{\forall e_{uv} \in l_{ph}} \left((\text{fixed cost}_{e_{uv}} \times B_{e_{uv}} \times \text{type}_{e_{uv}}) + \sum_{\forall e_{aij} \in l_v} \left(bw_{e_{aij}} \times BWcost_{e_{uv}} \times f_{e_{uv}}^{e_{aij}} \times \text{type}_{e_{uv}} \right) \right)$$

D. *Constraints*

All limitations and constraints in the entire cloud system have been organized into 5 sub-sections: for physical nodes, physical links, Quality of Service (QoS) requirements, well-connected mapping and other needed constraints.

1) *Physical node limitations:*

The constraints considered in this paper are computational memory and storage for network nodes. For every physical node, the sum of all requests must not exceed their capacity. So we have the following node Constraints (1), (2) and (3) to make sure that these parameters do not exceed the maximum amounts

$$\sum_{\forall ai \in N_{app}} c_{ai} x_u^{ai} \leq C_u \quad \forall u \in N_{ph} \tag{1}$$

$$\sum_{\forall ai \in N_{app}} s_{ai} x_u^{ai} \leq S_u \quad \forall u \in N_{ph} \tag{2}$$

$$\sum_{\forall ai \in N_{app}} m_{ai} x_u^{ai} \leq M_u \quad \forall u \in N_{ph} \tag{3}$$

2) *Physical link limitation:*

Bandwidth constraint has to be considered for each physical link, regardless of link type which is WAN or LAN link. Therefore, Constraint (4) represents that for each physical link, the sum of bandwidth demands of all application must not exceed available bandwidth.

$$\sum_{\forall e_{aij} \in l_v} bw_{e_{aij}} f_{e_{uv}}^{e_{aij}} \leq BW_{e_{uv}} \quad e_{uv} \in L_{ph} \tag{4}$$

3) *Quality of service requirements*

In term of QoS requirements, the following 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).

$$\sum_{\forall e_{uv} \in l_{ph}} D_{e_{uv}} f_{e_{uv}}^{e_{aij}} \leq d_{e_{aij}} \quad \forall e_{aij} \in L_{app} \tag{5}$$

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

4) *Well-connected mapping Constraints*

The Constraint (7) makes sure when 2 adjacent application components could not be mapped next to each other physically, a chain of continuous physical links is used to map each application link. In fact, it assures a closed path to be used to map an application link.

As it can be observed from this equation, it shows that for each physical node *u*, the subtract of sum of all incoming and outgoing ***f*** values should be equal to the subtracts of *x* values between target and source of each application link e_{aij}.

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

[1] The cost related to each physical node, separately refers to CPU usage, memory usage, storage usage and the fixed cost which is listed in Table I.

5) Additional constraints

The statement below (8) indicates that every application component has to be mapped exactly once to have a successful mapping.

$$\sum_{\forall u \in N_{ph}} x_u^{ai} = 1 \qquad \forall ai \in N_{app} \qquad (8)$$

Also we need some other constraints between *X* and *T* values to make sure that each physical node is just used either for database or logic component (Constraint (9), (10), (11) and (12)).

$$database_{ai} \times x_u^{ai} \leq T_u^d \quad \forall (u, ai) \in (N_{ph} \times N_{app}) \qquad (9)$$

$$logic_{ai} \times x_u^{ai} \leq T_u^l \qquad \forall (u, ai) \in (N_{ph} \times N_{app}) \qquad (10)$$

$$x_u^{ai} \leq T_u^d + T_u^l \qquad \forall (u, ai) \in (N_{ph} \times N_{app}) \qquad (11)$$

$$T_u^d + T_u^l \leq 1 \qquad \forall u \in N_{ph} \qquad (12)$$

The following constraints are also required. The variable K in Constraints (13) and (14) is a large number comparable to the maximum value of a double. In Constraint (13) 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 (14). In Constraint (15), B_u for each physical node shows that whether node u is used or not either as a database or a computational node.

$$\sum_{\forall ai \in N_{app}} x_u^{ai} \leq K \times B_u \qquad \forall u \in N_{ph} \qquad (13)$$

$$\sum_{\forall e_{aij} \in l_{app}} f_{e_{uv}}^{e_{aij}} \leq K \times B_{e_{uv}} \qquad \forall e_{uv} \in l_{ph} \qquad (14)$$

$$B_u = T_u^d + T_u^l \qquad \forall u \in N_{ph} \qquad (15)$$

IV. Algorithm Descriptions

In this Section the two implemented algorithms to evaluate the presented model are detailed. One of the algorithms refers to the proposed approach.

A. *ILP-based algorithm*

This algorithm implements the optimal ILP based model which extensively was explained in Section III. The proposed cost-aware mapping algorithm which we refer to as ILP-based algorithm is solved using IBM ILOG CPLEX Optimization Studio [16] which is a tool that provides a way to build efficient optimization models. In this work the objective function minimizes the cost based on the constraints and physical limitations. As a result, the objective of the presented model also minimizes the number of nodes on which the applications can be hosted while satisfying their resource requirements. For this reason the percentage of physical nodes used and the execution time have been measured and are reported upon in the next section.

B. *Sequential cloud mapping algorithm : SCMA*

As ILP-based algorithms have limited scalability, a heuristic algorithm is designed which we refer to as the Sequential Cloud Mapping Algorithm (SCMA). This algorithm iterates over applications and the individual application components and based on the type of component starts mapping from two default nodes, separately for computational and database components. It moves to the neighbours when there is not enough capacity or when the quality of service cannot be met. The SCMA aims to place all application components one by one while offering a backtracking phase to the previous state if placement of entire application fails. This is achieved by saving the current state of entire problem before starting to map each application. In the case SCMA tries to start mapping again from a neighbour if there is an unvisited one, otherwise another unused node will be chosen instead. Based on this algorithm, this process is continued until there is no unvisited node. The algorithm will be terminated when there is no appropriate unvisited node in the whole physical infrastructure. The SCMA can be used in a real-time setting, thanks to its very short execution times. However, this algorithm is not optimal, as in some cases it is not able to map all applications.

The pseudo code of SCMA can be seen in Algorithm 1. The SCMA is designed to map the applications to the computation nodes. First, the mapping impossibilities of all applications are set to false to show that, by default, the system is able to map all the applications. The current state of the entire cloud system (physical server and links configurations and the application and its components status) is also saved. This ensures the state can be rolled back when placing an application fails. The algorithm then goes through all of the applications and tries to map them by invoking the **ApplicationMapping** function.

```
SCMA
FOR (a ∈ applications)
   ImpossibilityOfMapping(a) ← false;
   CurrentState ←Save the current system state;
   WHILE (IsMapped(a) = false
            AND ImpossibilityOfMapping(a) = false)
         Set current system state to CurrentState;
         IF one of the default servers is null THEN
            ImpossibilityOfMapping (a) ← true;
         ELSE
            ApplicationMapping(a);
         ENDIF
   ENDWHILE
ENDFOR
ENDSCMA
```

Algorithm 1. The SCMA pseudo code.

The **ApplicationMapping** function, shown in Algorithm 2, iterates over each component of an application. It examines whether the default server is still the appropriate choice for this component by calling the **Map** function, otherwise a new neighbor server is chosen and processed again.

```
ApplicationMapping(a)
FOR (c ∈ all the components of a)
  IF (c = a logic component) THEN
       default server= logic default server;
  ELSE
       default server= database default server;
  ENDIF
  WHILE (Map(c , default server)=false)
     default server ← Find New Server by calling BFS algorithm
     IF (default Server = null) THEN
          IsMapped(a) ← false;
          RETURN false;
     ENDIF
   ENDWHILE
ENDFOR
IsMapped(a) ← true;
ENDFUNCTION
```

Algorithm 2. The ApplicationMapping function called by the SCMA.

The **Map** function, shown in Algorithm 3, tests the designated physical server resources like CPU, memory and storage capacities and, if they are adequate and meet the component demands, check the connected links using the **CheckLinks** function.

```
Map (component, default server)
IF there is not enough CPU, memory or storage capacity THEN
    RETURN false;
ENDIF
IF (Checklinks( component , default server)= true)
    mapbetween(component , default server ) ← true;
    CPU_default server −= CPU demand_component ;
    Memory_default server −= Memory demand_component ;
    Storage_default server −= Storage demand_component ;
ENDIF
ENDFUNCTION
```

Algorithm 3.The Map function called by the ApplicationMapping function.

Algorithm 4 shows the **Checklinks** function, which determines whether mapping of a component to the default server can cause link connection problems. This function checks for the path connectivity between the servers which are used to map the connected components. It iterates over all of the application links connected to the given component and if the component on the other side of the link was mapped, check for the path connectivity between these two servers. The shortest path is checked using the Dijkstra shortest path algorithm. Afterward, once a path is found, the **CheckBWandDelay** function is used to evaluate the path QoS.

As outlined in Algorithm 5, the **CheckBWandDelay** function checks whether the bandwidth and delay demands of a given path are met. If every step works well, the application can be mapped. Afterward, the next application mapping will continue until either there are no more applications left or there is not enough remaining capacity in the cloud.

```
CheckLinks(component, default server)
FOR (all links connected to component)
   c1= the component on the other side of the link
   IF (c1 is mapped) THEN
     Do
     n= the node which is used to map c1;
     Path= Shortest path between default server and n;
     WHILE (CheckBWandDelay(Path)=false
        AND Path ≠null)
       IF (Path = null) THEN
          IsMapped(a) ← false;
       ENDIF
   ENDIF
 ENDFOR
ENDFUNCTION
```

Algorithm 4. The CheckLinks function called by the Map function.

```
CheckBWandDelay(path)
FOR (all the physical links in the path)
   EndtoEndDelay+= link delay;
   IF (BW demand_applicationLink > BW_physicalLink) THEN
      Remove physical link from the temporary graph;
      RETURN false;
   ELSEIF (EndtoEndDelay does not exceed constraints) THEN
      FOR (all physical links in this path)
         BW_physicalLink −= BW demand_applicationLink ;
      ENDFOR
   ENDIF
ENDFOR
ENDFUNCTION
```

Algorithm 5. The CheckBWandDelay function called by Checklinks.

V. Evaluation Results

In this section we first explain the evaluation setup. Afterwards, we present the evaluation results.

A. Evaluation Set-up

The ILP-based and the SCMA algorithms are evaluated with two different simulation setups. Based on the configurations of physical network and application network, a physical network with 5 nodes (Table VI) and two types of data-oriented applications with 5 and 10 components are defined (table VII). In this paper just deterministic application has been taken into account in which the structure of the application is always known beforehand.To illustrate the used applications, a 10-component application is shown as a sample in Figure 2.

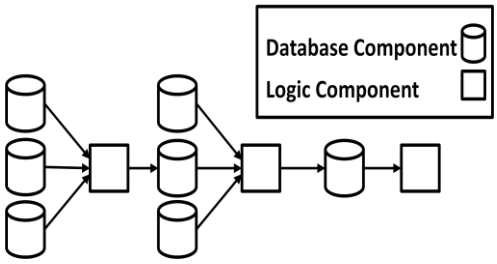

Fig 2 . Sample 10-component application.

TABLE VI

PHYSICAL NETWORK ASSUMPTIONS.

Physical infrastructures						
No. of nodes	No. of links		No. of WAN links		No. of LAN links	
5	6		1		5	
Physical node specifications						
Capacity CPU	Capacity Storage	Capacity Memory	CPU cost	Strg. cost	Mem. cost	Fix cost
3 GHZ	200 GB	16GB	1	1	1	100
Physical link specifications						
Link type	Bandwidth		delay		Fix cost	
LAN	400 Mb/s		3 ms		1	
WAN	100 Mb/s		60 ms		20	

TABLE VII

APPLICATION NETWORK ASSUMPTIONS.

Different Application specifications					
App. type	No. of apps	No. of comp /app	No. of app.link /app	No. of database comp/app	No. of logic comp/app
Type1	1..20	5	4	3	2
Type2	1..10	10	9	7	3
Application component specifications					
CPU demand	Storage demand	Memory demand	Logic	Database	
Random (1-100)MHZ	Random (1-10000)MB	Random (50-500) MB	Binary	Binary	
Application link specifications					
Max allowed delay		BW demand			
1000 ms		Random (1-50) Mb/s			

Combining the mentioned assumptions along with different number of applications has made two different scenarios which have been used for evaluation: 1 up to 20 type 1 applications is considered for the first scenario and 1 up to 10 type 2 applications for the second scenario. To make a comparison, for each case, the average value and the standard error of execution time, the percentage of used nodes and the percentage of successful mappings in the SCMA algorithm are calculated. Each scenario is iterated 20.

Both algorithms are evaluated using the Stevin Supercomputer Infrastructure at Gent University. Each node on which the algorithms were executed contains dual-socket 2.5 GHz quad core Intel Xeon L5420 processors, thus having 8 cores and 16GB memory.

B. Simulation results

Figures 3 and 4 refer to the case with 5-node substrate network and applications type 1 which are 5-component data oriented applications (consist of three database and two logic component) and number of same applications is 20. As it can be observed from Figure 3 the execution time of ILP-based method is much higher than the SCMA. As it was explained before, the ILP-based method consist of 16 linear equations (15 constraints and the objective function) with more than 1000 decision variables in the implemented cases (3 out of 6 variables are three dimensional data structures and the others have one dimension). Increasing the number of physical nodes and applications and components of each application, it takes longer to find the most efficient solution. In Figure 4 the percentage of used nodes in each case and the percentage of successfully mapped applications (just in SCMA) are depicted. As it can be observed from this figure the number of used node in ILP-based method is less as it leads to an optimal solution. In addition, when the ILP-based algorithm provides a feasible solution which means mapping 100% of applications, the curve in Figure 4 shows that just up to around 10 applications, SCMA is able to map all applications. For more applications, this method is not able to map all applications.

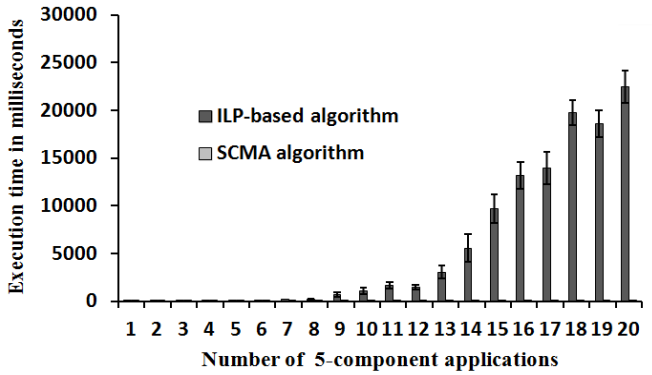

Fig 3. The execution times of the ILP-based algorithm and the SCMA for 5-component applications.

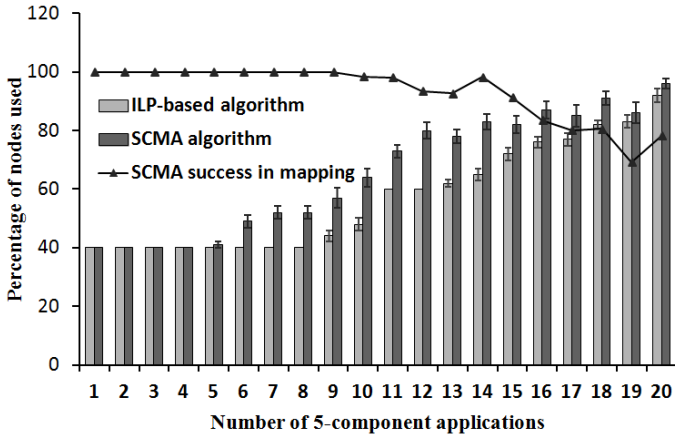

Fig 4. The percentage of used nodes of the ILP-based algorithm and the SCMA for 5-component applications together with the percentage of successfully mapped applications in the SCMA.

In a similar way, Figure 5 and Figure 6 assume a 5-node infrastructure with applications type 2 which are 10-component applications (7 database and 3 logic components). As it can be seen in Figure 5 the execution time is increased exponentially by adding more applications. Since the number of components is twice more than the previous scenario and the number of application is a half, the results are almost in the same range. Also in Figure 6 there is a same trend as Figure 4, but for 10 applications. In this scenario the number of used nodes in SCMA algorithm is higher and this method is

not able to map all applications.

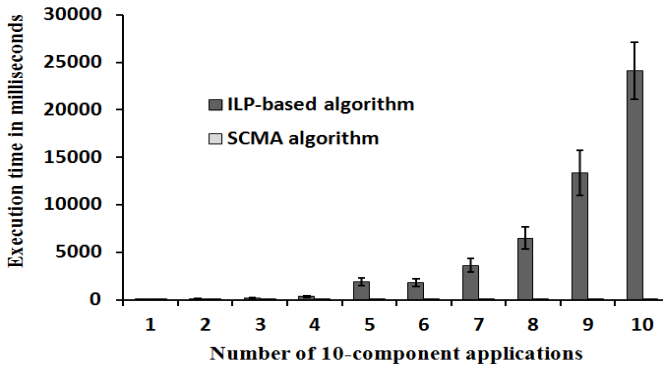

Fig 5. The execution times of the ILP-based and SCMA algorithms for 10-component applications.

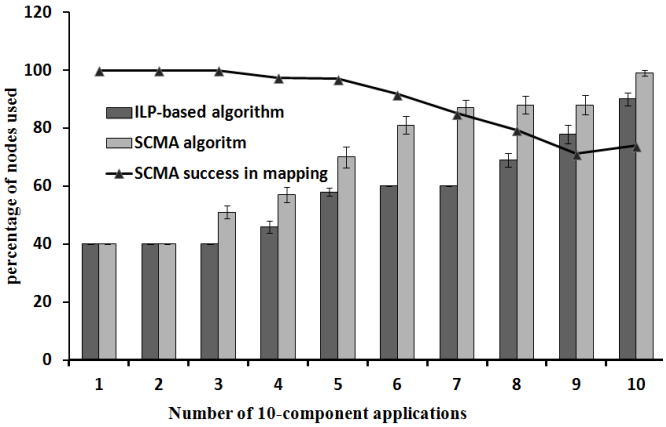

Fig 6. The percentage of used nodes of both algorithms for 10-component applications together with the percentage of successfully mapped applications in SCMA.

VI. Conclusion

This paper focuses on cost-effective management of application placement in a cloud environment, taking into account the application workflow and the distinction between data components and computational components. An ILP-based model was proposed and an algorithm based on this model was implemented. What makes this work different from other related work from one hand is focusing on component-based applications, and from another hand considering multi domain network-aware approach. In order to evaluate the ILP-based approach, another algorithm, referred to as SCMA, was designed. The experimental results showed that in application mapping, the ILP-based approach can be highly efficient in terms of number of used nodes and the cost of mapping. The ILP-based approach is very useful for benchmarking real-time heuristic algorithms, such as the presented SCMA algorithm.

For future work, we intend to focus on the scalability and dynamic versions of the presented algorithms.

ACKNOWLEDGEMENT

This research is carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules and Flemish Government, department EWI. The work is also partly supported by the iMinds DMS^2 project.

REFERENCES

[1] Tianyi, X.; Xuan L.; Chun-Jen Ch.; Wada, A.; Ata, S.; Dijiang Huang; Medhi, D.,"Constructing a virtual networking environment in a Geo-distributed programmable layer-2 networking environment (G-PLaNE)," 2012 IEEE International Conference on Communications (ICC). IEEE, 2012.

[2] Kim-Khoa, N.; Cheriet, M.; Lemay, M., "Enabling infrastructure as a service (IaaS) on IP networks: from distributed to virtualized control plane," IEEE Communications Magazine 51.1 (2013): 136-144.

[3] Chowdhury, N. M., and Boutaba, R.; "A survey of network virtualization," Computer Networks 54.5 (2010): 862-876.

[4] Feamster, N.; Gao, L.; and Rexford, J., "How to lease the Internet in your spare time," ACM SIGCOMM Computer Communication Review 37.1 (2007): 61-64.

[5] Pereira Esteves, R.; Zambenedetti Granville, L.; Bannazadeh, H.; Boutaba, R., "Paradigm-based adaptive provisioning in virtualized data centers," Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on , vol., no., pp.169,176, 27-31 May 2013.

[6] Hien Nguyen Van; Tran, F.D.; Menaud, J.-M., "Autonomic virtual resource management for service hosting platforms," ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. pp. 1-8, May 2009. doi: 10.1109/CLOUD.2009.5071526

[7] Breitgand, D.; Epstein, A., "SLA-aware placement of multi-virtual machine elastic services in compute clouds," 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp.161-168, May 2011.doi: 10.1109/INM.2011.5990687

[8] Adam, C.; Stadler, R., "Service Middleware for Self-Managing Large-Scale Systems," IEEE Transactions on Network and Service Management, vol.4, no.3, pp.50,64, Dec. 2007 doi: 10.1109/TNSM.2007.021103

[9] Tang, C.; Steinder, M.; Spreitzer, M.; and Pacifici, G., "A scalable application placement controller for enterprise data centers," Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

[10] Korupolu, M.; Singh, A.; Bamba, B., "Coupled placement in modern data centers," IEEE International Symposium on Parallel & Distributed Processing, 2009. IPDPS 2009. pp. 1-12, May 2009. doi: 10.1109/IPDPS.2009.5161067

[11] Biran, O.; Corradi, A.; Fanelli, M.; Foschini, L.; Nus, A.; Raz, D.; Silvera, E., "A Stable Network-Aware VM Placement for Cloud Systems," 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012 , pp.498-506, May 2012. doi: 10.1109/CCGrid.2012.119

[12] Urgaonkar, B.; Rosenberg, A.L.; Shenoy, P., "Application placement on a cluster of servers," Int.J. Found. Comput. Sci. 18(05), 1023 (2007). doi:10.1142/S012905410700511X

[13] Jing, X.; Fortes, J. A B, "Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments," 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom) , Green Computing and Communications (GreenCom), pp.179-188, Dec. 2010

[14] Moens, H.; Truyen, E.; Walraven S.; Joosen W.; Dhoedt B.; De Turck, F., "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), pp. 28–36, 2012.

[15] Moens, H.; De Turck, F., "A Scalable Approach for Structuring Large-Scale Hierarchical Cloud Management Systems," 9th International conerence on network and service management, October 2013.

[16] Moens, H.; Famaey, J.; Latre, S.; Dhoedt, B.; De Turck, F., "Design and evaluation of a hierarchical application placement algorithm in large scale clouds," 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp.137-144, May 2011. doi: 10.1109/INM.2011.5990684.

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