WCCI 2010 IEEE World Congress on Computational Intelligence
July, 18-23, 2010 - CCIB, Barcelona, Spain CEC IEEE

A Penalty-based Genetic Algorithm for the Composite SaaS Placement Problem in the Cloud

Zeratul Izzah Mohd Yusoh and Maolin Tang, *Senior Member, IEEE*

Abstract— **Cloud computing is a latest new computing paradigm where applications, data and IT services are provided over the Internet. Cloud computing has become a main medium for Software as a Service (SaaS) providers to host their SaaS as it can provide the scalability a SaaS requires. The challenges in the composite SaaS placement process rely on several factors including the large size of the Cloud network, SaaS competing resource requirements, SaaS interactions between its components and SaaS interactions with its data components. However, existing applications' placement methods in data centres are not concerned with the placement of the component's data. In addition, a Cloud network is much larger than data center networks that have been discussed in existing studies. This paper proposes a penalty-based genetic algorithm (GA) to the composite SaaS placement problem in the Cloud. We believe this is the first attempt to the SaaS placement with its data in Cloud provider's servers. Experimental results demonstrate the feasibility and the scalability of the GA.**

I. Introduction

Cloud computing [1] is an emerging computing paradigm in which applications, data and IT resources are provided as a service to users over the Internet. One kind of services that can be offered through the Cloud is Software as a Service or SaaS [1], [2], [3]. Although SaaS can be delivered without using the Cloud computing infrastructure, with increasing demands for SaaS each year [4], SaaS vendors have to find a solution to cope with these increasing requests. Cloud providers can take this as an advantage as Cloud infrastructure is by far the best option for supporting SaaS as Cloud computing provides scalability that is much needed by SaaS [4]. To date, however, little research has been done on the SaaS placement problem. This research aims to fill the gap by focusing on the initial placement of SaaS in Cloud infrastructure.

A SaaS deployed in a Cloud is usually composed of several components, where each of the components represents a business function of the SaaS that is being delivered [5]. Some of these components might be dependent on other components, and some of the components may need to access data files that are located in Cloud storage servers. For SaaS placement in the Cloud, the problem relates to how a composite SaaS should be placed in a Cloud by the Cloud's providers such that its performance is optimal based on its estimated execution time. As we know, Cloud providers have many clusters of servers located across the globe. For example, Amazon has its storage servers in the United States and Europe, while Nirvanix placed its servers in the United States, Germany and Singapore [6]. The placement of SaaS components and their related data components in the provider's servers that are located in geographically dispersed locations needs to be done strategically, as the placement can directly affect the resource usage as well as the SaaS performance.

The challenges in the SaaS placement process rely on several factors, including the size of the Cloud network, SaaS competing resource requirements, SaaS interactions between its components and SaaS interactions with data components. Interactions with data components play a significant role in SaaS execution time as the data are located at the Cloud servers instead of local machines [7][8]. Existing SaaS placement methods were not designed for the Cloud. The methods mostly focus on the resource consumption by the components and are not concerned with the placement of the component's data [9]-[13]. This research will address this problem by proposing a genetic algorithm for the composite SaaS placement problem in the Cloud, which considers not only the placement of the software components of a SaaS, but the placement of data of the SaaS as well. To the best of our knowledge, this is the first attempt to handle the SaaS placement problem in the Cloud.

The remaining paper is organized as follows. Section II discusses related work. The problem formulation is described in Section III. Section IV presents the GA. Then Section V is about the evaluation that has been carried out. The concluding remark is presented in Section VI.

II. Related Work

The problem of placing the components of a SaaS and their related data in the Cloud is referred to as SaaS Placement Problem (SPP). SPP shares some similar characteristics with an existing problem called Component Placement Problem (CPP). CPP is concerned with the placement of an application's components among available servers [14]. Based on existing literature on CPP, the problem can be further divided into two categories: 1) offline CPP and, 2) online CPP, where in the online CPP the placement of the components is made during the run-time. SPP is more similar to the offline CPP as the placement of the components is carried out at the initial stage. This section

Z. Yusoh is with the Faculty of Information, Communication and Technology, Universiti Teknikal Malaysia Melaka and currently a PhD student with the Faculty of Science and Technology, Queensland University of Technology, 2 George Street, Brisbane Australia (phone: 61-7-31389519; email: zeratul.mohdyusoh@student.qut.edu.au)

M. Tang is with the Faculty of Science and Technology, Queensland University of Technology, 2 George Street, Brisbane Australia (phone: 61-7-31385225; email: m.tang@qut.edu.au)

discusses the existing solutions for both types of CPP that are relevant to SPP.

Several existing studies for the online CPP are concerned with allocation of resources to an application's components. Among the resources that are commonly considered are computation capacity [9]-[11][15], memory [9][15], network bandwidth [10][16] and storage capacity [11]. The main aim for the online CPP, usually, is to optimize the resource usage by the components, and at the same time to minimize the total execution time of the application [9][11][12]. Several online CPPs have included an offline CPP at the beginning of the placement. Although CPP shares the same aims with SPP, none of these existing problems considered the component's communication with its data, and the placement of the data in the network, whereby in SPP this is the major concern since both components and their data reside in the Cloud.

Reference [13] proposed a solution to an offline application placement problem, namely Application Component Placement (ACP), in a data center. In ACP, the location of the data components is considered as one of the decision factors. ACP also considers the application's processing communication and storage requirements in making the decision. The communication here refers to the amount of traffic that is transferred between the components and between the components and the servers where the application's data are located. ACP is similar with SPP, but, it is assumed that the location of the data in ACP is already known prior to the placement process. The data location is treated as an input to the placement problem, while in SPP the data components are being placed together with SaaS components.

A placement that is concerned with SaaS is presented in [11]. That work considers a placement problem for SaaS components in a multi-tenant architecture. The placement of the components is made within a set of available servers, and the main objective is to optimize the resource usage in each server. The principal rule used by the placement approach is rather straightforward, that is, a new instance of a component should be deployed in a server with the smallest residual resource left after having the instance. This is to reserve servers that have larger residual resources for instances that have a higher resource demand. That work also proposes a resource computation model to calculate the resource demand for an upcoming tenant, including the storage requirement. Although that work is concerned with SaaS placement, it is more focused on the multi-tenant resource model and does not take into consideration the SaaS's data placement in the network.

To sum up, none of researches has considered the placement of the application's data together with the application's components. This is probably because the application's data reside at local machines instead of in the data centre. In the SaaS placement case, the data as well as the components will be located at the Cloud provider's servers. As such, the Cloud providers must apply a strategic placement method in order to ensure the components and the data are well placed and the SaaS performance is optimized. This paper will propose a placement solution to address this gap.

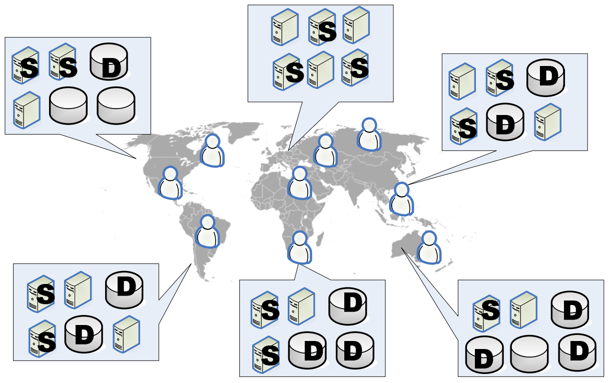

Fig. 1. A high level illustration of SaaS Placement Problem (SPP)

III. SaaS Placement Problem

Fig. 1 shows a high level illustration of SPP. The objective of the problem is to decide which server should host the SaaS components such that the requirements are satisfied and the SaaS performance is optimal based on its estimated execution time. The SPP inputs are:

- a component-based SaaS with its requirement for computation and memory (denoted as 'S' in Fig. 1),
- data components that the SaaS needs to access (denoted as 'D' in Fig. 1), and
- a Cloud's network topology with its computing servers and storage servers. The servers' geographical locations are widely spread across the globe.

The decision of SPP will be based on estimated SaaS execution time on a set of selected Cloud servers. The calculation of the estimated execution time is mainly based on the computing resources and data transfer time between the servers that have a SaaS component, and the storage servers that have the data of the SaaS. Other factors such as geographical locations of users or the location of users' data are not considered at this stage as it is an initial placement of a SaaS where such information is not yet available.

The following sections describe SPP problem formulation, numerical attributes for the estimated execution time and critical path method applied in the calculation process.

A. SPP Problem Formulation

1) SaaS Modelling: The SaaS modelling is made general enough to represent a SaaS. Table I summarizes the SaaS components' requirements, workflows and its communication. Followings are further descriptions on SaaS requirement and its dependencies:

- *SaaS Computing Requirements:* Each component has requirements for the servers on which it can be hosted. For this problem, only the main requirements will be considered. The requirements are the computing capacity, memory size and the storage size.

- *Storage Access Requirements:* Some components may need to access data that are located in a Cloud storage server. Based on Table I, the term 'data chunk' represents a set of data that may be accessed by the component.
- *Component Workflows:* Dependencies of components to components and components to data chunks are depicted through the workflows. A SaaS may have multiple workflows and each workflow has a weighting. This value indicates the significance of a workflow to the SaaS. These workflows are presented by directed acyclic graphs (DAG). An example of a SaaS multiple workflows is illustrated in Fig. 2.

2) Cloud Resource Modelling: This Cloud consists of computing servers and storage servers. Each server has its own attributes that are relevant to SaaS requirements such as processing capacity, memory size, secondary storage capacity and storage capacity. Table II summarizes the resource sets and attributes.

3) Cloud Network Modelling: The Cloud network is represented by an undirected graph, $G=\{V,E\}$. $V=\langle P\cup D\rangle$ is the sets of vertices including computation servers and storage servers, $e\in E$ is the set of undirected edges connecting the vertices, if and only if there exists a physical link transmitting information from v_i to v_j, where $v_i,v_j\in V$. $B_{v_i,v_j}:E\rightarrow R^+$ and $L_{v_i,v_j}:E\rightarrow R^+$ is a bandwidth and latency functions of the link from v_i to v_j respectively.

B. SPP Attributes

The objective of SPP is to place SaaS components and their data in a set of computing servers and storage servers in the Cloud such that the execution time of the SaaS is minimized. The problem involves selecting the suitable server to serve a component. To determine a suitable server for a component, four numerical attributes will be used. These attributes define the estimated execution time (in seconds) of a component in a selected server and a total estimated execution time of the SaaS that is being deployed. Followings are the descriptions of the attributes.

1) Estimated Data Transfer Time (EDTT): EDTT is the estimated time taken for transferring data between storage servers and computing servers. As mentioned in a previous section, some of the components may require an access to data in storage servers. Given a current placement for a component sc_i, EDTT is calculated based on AD_{sc_i} the total bytes of amount of read/write task of sc_i, B_{p_i,d_j}, bandwidth of the links involved, and L_{p_i,d_j}, the latency that may occur in those links.

$$EDTT(sc_k)=\sum_{v_i,v_j\in E}\frac{AD_{sc_k}\times 8}{B_{p_i,d_j}}+L_{p_i,d_j} \qquad (1)$$

2) Estimated Processing Time (EPT): EPT is the estimated processing time for a component in a selected computing server. It is based on the task size of a component, TS_{sc_i}, the processing capability of the selected server PC_{p_i}, and the value of EDTT (refer Equation 1) if there is any. If a component accesses more that one storage servers, only the maximum value of EDTT will be considered.

$$EPT(sc_k)=\frac{TS_{sc_k}}{PC_{p_i}}+\max(EDTT) \qquad (2)$$

TABLE I
SETS, PARAMETERS AND REQUIREMENTS OF SAAS

SaaS Modelling	Description
$sc_i\in SC$	Set of n SaaS components, and $1\le i\le n$
$dc_i\in DC$	Set of m data chunks that may be accessed by SC, and $1\le i\le m$
$wf_i\in WF$	Set of q SaaS workflows where $WF\subseteq SC$, and $1\le i\le q$
TS_{sc_i}	Task size of sc_i
M_{sc_i}	Memory requirement of sc_i
S_{sc_i}	Size of sc_i
AD_{sc_i}	Amount of read/write task of sc_i
CDC_{sc_i}	Set of dc_i that may be accessed by sc_i, and $CDC\subseteq DC$
S_{dc_i}	Size of dc_i
W_{wf_i}	Weighting for wf_i

TABLE II
SETS AND ATTRIBUTES OF CLOUD RESOURCE

Cloud resources	Description
$p_i\in P$	Set of k computing servers, and $1\le i\le k$
$d_i\in D$	Set of r storage servers, and $1\le i\le r$
PC_{p_i}	Processing capacity of p_i
MC_{p_i}	Memory of p_i
SS_{p_i}	Secondary storage of p_i
SP_{d_i}	Storage capacity of d_i

3) Estimated Execution Time (EET): Equation 2 defines the Estimated Processing Time (EPT) for a single component in a SaaS. Some of the components may be dependent on other components as depicted by workflows in Fig. 2. Thus, a new computation is designed to calculate the Estimated Execution Time (EET) for a path in a workflow. It is based on the sum of EPT of each component in a path. The EET is defined as:

$$EET(path_k)=\sum_{sc\in path_k}EPT(sc_i) \qquad (3)$$

4) Estimated Total Execution Time (ETET): ETET is the estimated total execution time for the whole SaaS that is being deployed in the Cloud. From Equation 3, the Estimated Execution Time (EET) for all paths in the SaaS workflows is obtained. These values are used to determine the critical path of a workflow if the workflow has more than one path. Then, the Estimated Total Execution Time (ETET) is defined by the sum of the Estimated Execution Time (EET) of the critical path of each workflow multiplied by its weighing as shown in Equation 4.

$$ETET(SaaS) = \sum_{i=1}^{q} EET(critical\_path) \times W_{wfi} \qquad (4)$$

Where q is the number of SaaS workflows.

C. *Critical Path*

As mentioned in a previous section, a SaaS may have multiple workflows. For example, a SaaS for photos editing like Adobe Photoshop online [17], may have a workflow for basic editing function, a workflow for graphical effects and another workflow for uploading photos. These workflows consist of several components, each of which represents a specific task for a particular function. A workflow may have multiple paths. And these paths may run in parallel. Fig. 2 shows an example of a SaaS that has two workflows. Based on the example, workflow 1 has more than one path from the 'Start' component to the 'Finish' component. In this case, a critical path algorithm will be applied.

In Fig. 2, a circular node sc_k represents a SaaS component where $1 \le k \le n$, and n is the number of components. The time consumed to execute sc_k in a particular computing server, p_i is denoted at the lower half of the circle. This is calculated based on the task size of sc_k and the processing capacity of p_i. A square node with an arrow to a component represents a data chunk, dc_r, at a storage server, d_j. The time taken for data transfer from p_i to d_j is depicted at the edge. The calculation for the time taken is defined in Equation 1 (Estimated Data Transfer Time, EDDT). A undirected edge $e_{k,j}$ from sc_k to sc_j means that sc_k has to be executed first before sc_j can be executed.

To determine the critical path, the graphs in Fig. 2 will be converted to super graphs illustrated in Fig. 3. The super graphs combine a circular node in Fig. 2 with its corresponding square node. If there is more than one square nodes involved, the maximum value will be selected. This combination indicates the component's Estimated Processing Time (EPT) as described in Equation 2. The value in the lower half of the circle depicts the value of EPTs. The super graphs for Fig. 2 are shown in Fig. 3.

For each path in the super graphs, the estimated execution time of a path (EET) is computed based on Equation 3. This value will be used to determine the critical path of a workflow in order to obtain the EET of each SaaS workflow.

The Estimated Total Execution Time (ETET) for the whole SaaS will then be computed by Equation 4. It is based on the EET for critical paths and its weighting. This value will be used in the placement algorithm to determine the optimal placement solution for SaaS components.

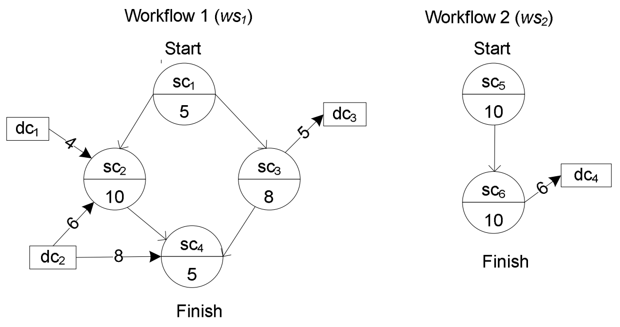

Fig. 2. Directed Acyclic Graphs for SaaS workflows

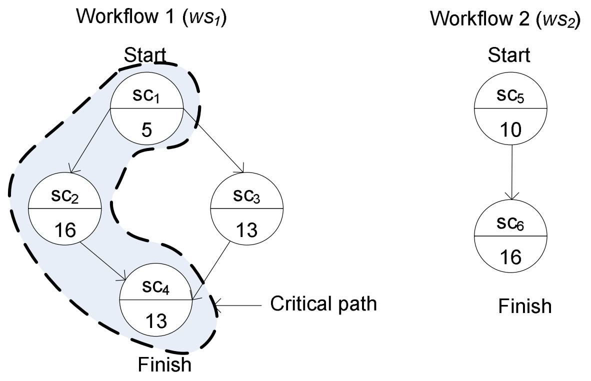

Fig. 3. Directed Acyclic Super Graphs for SaaS workflows

IV. A Genetic Algorithm For SPP

From the computational point of view, SPP is a large-scale and complex combinatorial optimization problem with constraints, and a GA would be suitable for it. Thus, we have developed a GA for SPP. In the following we discuss the design of the GA in details.

A. *Chromosome Representation*

A chromosome in the GA represents a placement plan for the SaaS. It consists of two compartments. The first compartment contains *n* genes, each of which corresponds to a software component, representing the computation server where the software component would be placed in the placement plan of the SaaS, where *n* is the total number of software components in the SaaS. The second compartment holds *m* genes, each of which corresponds to a data chunk that is used in the SaaS, standing for the storage server where the data chunk would be stored in the placement of the SaaS. Each gene in the chromosome is represented in a triple <*C*, *R*, *S*>, where *C*, *R* and *S* are the IDs for continent, region and server, respectively. Fig. 4 shows a gene encoding and an instance of the chromosome representation, where the total number of software components in the SaaS is 10 and the total number of data chunks is 10 as well.

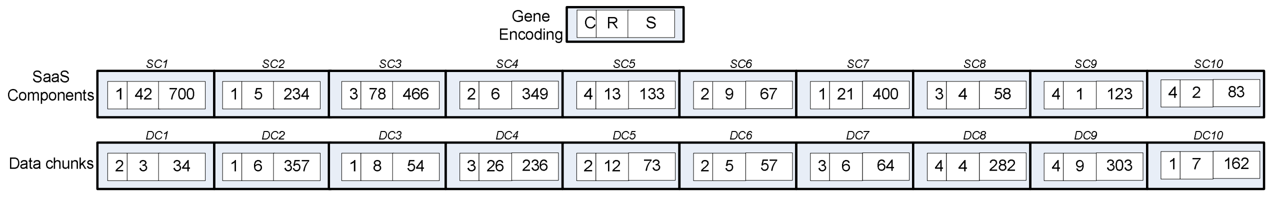

Fig. 4. : An example of gene and chromosome encoding scheme with 10 SaaS components and 10 data chunks

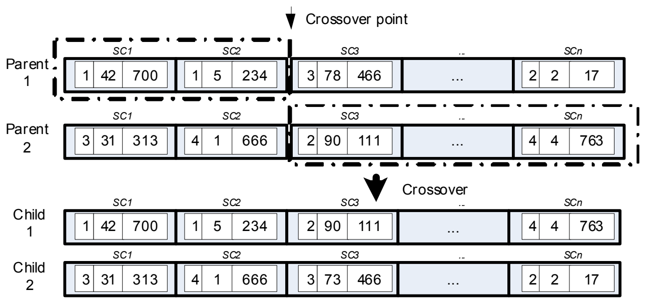

Fig. 5. : An example of crossover operation for SaaS computing servers

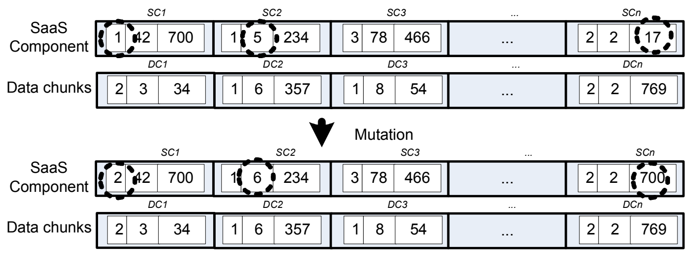

Fig. 6. : An example of mutation operation

B. *Infeasible Encoding Problem*

The representation naturally maps a SaaS placement plan into a chromosome of the GA. However, it has a deficiency, which is the individuals generated randomly in the initial population and individuals generated by the genetic operators, which will be discussed in the following, may not be feasible. For example, on continent one there are only 10 regions, the genetic operator may produce an individual <1, 11, 1>, indicating the corresponding software component or data chunk would be located at the 11th region of the 1st continent, which does not exist. Thus, in order to handle the infeasible encoding problem, we developed an infeasible encoding repairing technique.

The repairing technique performs a simple check in each gene to find any invalid codes for servers. Each gene should contain a set of three valid integers that represent an existing server in the Cloud. If any of these integers is invalid, that is, an invalid code for continent, region or server, another integer will be randomly generated based on a correct range of the codes.

C. *Genetic Operators*

Crossover: The crossover operation is a classical one point crossover. The point of crossover is between the segments of genes in a chromosome. The crossover operation combines segments from two selected parents and produces two children. The top two fittest among the parents and children are selected into the next generation. Fig. 5 illustrates the crossover operation.

Mutation: To promote further exploration in the search space, a mutation operator is used in order to keep the diversity of the genes in the population. The mutation operator is a knowledge-based one that changes the computation server of a particular component to another computation server such that the new computation server located closely to the storage server that has the component's data. By doing this, the data transfer time between the storage servers to the computation servers can be minimized; hence can reduce the SaaS' total execution time.

Mutation operation is applied to each gene in a selected chromosome. As described in a previous section, the gene represents the location of a computation server that holds a particular SaaS component. The operation will be conducted by randomly selecting any integer in a gene and replacing it with another server that is closer to the data. Based on the SaaS requirement in Table I, a component may access one or more data chunks. As such, the data chunk that has the highest EDTT (Equation 1) is selected as the component's new location. This mutation may change the SaaS component location to a computation server that is located in the same continent or same region as the storage server that has the component's data; hence it has a better bandwidth value. Fig. 6 illustrates the mutation operation.

D. *Fitness Function*

As the aim of the placement is to achieve the optimal performance of the SaaS that is being deployed, the main attribute for the fitness function evaluation is the *Estimate Total Execution Time* (*ETET*) for the SaaS components. The calculation for ETET is defined in Equation 4.

The requirements of each SaaS component are treated as the constraints of this placement problem. There are two main constraints that need to be fulfilled by the solutions 1) C_1, component's memory requirement and 2) C_2, component's secondary storage requirement.

Chromosomes that were generated may violate these constraints, making the placement infeasible. However, to fully discard these chromosomes from the population is unaffordable, as the chromosome may contain some useful building blocks in its genes. This is important in order to produce fitter children for the next generations. Therefore, to handle this situation, the following strategy is applied to these infeasible chromosomes. A penalty will be imposed to the fitness value of chromosomes that have violated C_1 and/or C_2 such that any infeasible chromosome will always have a lower fitness value than any feasible chromosome. The fitness function is defined as follow:

$$Fitness(X) = \begin{cases} 0.5 + \dfrac{ETET^*(Pop)}{ETET(X) \times 2} & if\ C(X) = 0 \\ \dfrac{ETET^*(Pop)}{ETET(X) \times 2} \times \dfrac{1}{\sum_{i=1}^{2} C_i(X)} & otherwise \end{cases} \quad (5)$$

where $ETET^*(Pop)$ is the minimum value of *Estimate Total Execution Time* in a population, Pop , and it is defined as:

$$ETET^*(Pop) = \min\left(ETET(X)\right) \quad (6)$$

where X is a solution in the population, $X \in Pop$.

Equation 5 guarantees that feasible chromosomes always have a fitness value greater 0.5 and infeasible chromosomes always have a fitness value less than 0.5, and that for infeasible chromosomes the more constraints are not satisfied, the less their fitness values are.

V. Evaluation

The GA described above has been implemented and tested. The programming language used was Microsoft .NET Visual Studio C++ 6.0. The test was focused on the scalability of the GA. Two sets of experiments were done. One set of experiments was to look at how the computation time of the GA increases when the scale of the Cloud increases; another was to find the increasing trend of the computation time of the GA when the number of software components and data chunks increases.

The experiment was conducted on several randomly generated SaaS placement problems with five to 20 components and data chunks. The Cloud was randomly created as well. The attributes of the nodes, including computation servers and storage servers, were randomly generated using the models presented in [18] [19].

A. Experiments on the size of Cloud network

This experiment was to study the relationship between the computation time and the size of the cloud network. In this experiment, we tested the GA for a Cloud that contained from 200 to 1200 servers, with an increment of 200. The numbers of SaaS components and data chunks were both fixed at 10. The parameters setting for the GA is listed in Table III and the characteristics of the problem are shown in Table V.

The experiments were carried out on a desktop computer with 2.33 GHz Intel Core 2 Duo CPU and 2GB RAM. Table V shows the statistic of the experimental results. Considering the nature of the GA, each of the test cases was repeated 10 times. The test results include the minimal, maximal and average values of estimated total execution time (ETET) of the SaaS based on its placement, and the minimal, maximal, and average computation times spent on finding the placement solution for each of the configurations. Fig. 7 visualises the average computation time taken on finding the solution for each of the test cases. It can be seen that the computation time of the GA appears to grow close to linearly with the size of the Cloud.

B. Experiments on the number of SaaS components

This experiment was to observe the computation time when the number of SaaS components and data chunks increase. The numbers of SaaS components and data chunk were set from five to 20 with an increment of five. The cloud contained up to 600 computation servers and storage servers. The same server's capacities and network were used in each test case.

All the experiments were conducted in a desktop computer with 2.66 GHz Intel Core 2 Duo CPU and 3.23GB RAM. For this experiment, the parameter settings used for the GA are listed in Table IV. The test cases, its configurations, the statistical results of the solutions and computation times taken are shown in Table VI. Experiments for each test case were conducted for 10 times. Fig. 8 shows the average computation time taken on finding the solutions. The result shows that the computation time increased close to linearly with the number of components.

TABLE III
PARAMETER SETTINGS FOR EXPERIMENTS ON THE SIZE OF CLOUD NETWORK

Parameter	Value/Condition
Population size	150
Initial population	Randomly generated solutions
Crossover probability	0.95
Mutation probability	0.15
Maximum generation	200
Termination condition	Maximum generation

TABLE IV
PARAMETER SETTINGS FOR EXPERIMENTS ON THE NUMBER OF SAAS COMPONENTS

Parameter	Value/Condition
Population size	150
Initial population	Randomly generated solutions
Crossover probability	0.95
Mutation probability	0.15
Termination condition	No improvement for the best individual in ten consecutive generations

TABLE V
EXPERIMENTAL RESULTS OF THE GA ON THE SIZE OF CLOUD NETWORK

Test Case	Problem Size			ETET (s)				Computation Time (mins)			
	$P \cup D$	SC	DC	Min	Max	Ave	StDev	Min	Max	Ave	StDev
1	200	10	10	1298.68	1515.61	1414.01	63.548	12	12	12	0
2	400	10	10	1264.57	1554.34	1397.44	97.05	72	75	73.3	0.95
3	600	10	10	1234.64	1806.42	1375.32	171.57	149	155	151.1	2.01
4	800	10	10	1204.23	1383.04	1319.49	63.655	275	285	278.2	2.82
5	1000	10	10	1247.18	1538.13	1340.53	93.375	379	393	385	4.27
6	1200	10	10	1199.03	1376.68	1316.23	50.963	608	620	613.6	4.2

TABLE VI
EXPERIMENTAL RESULTS OF THE GA ON THE NUMBER OF SAAS COMPONENTS

Test Case	Problem Size			ETET (s)				Computation Time (mins)			
	$P \cup D$	SC	DC	Min	Max	Ave	StDev	Min	Max	Ave	StDev
1	600	5	5	659.486	727.438	684.811	20.02	10	24	14.1	4.56
2	600	10	10	861.877	1132.87	986.522	93.53	45	103	71.5	20.81
3	600	15	15	1322.49	1579.95	1407.13	90.366	66	166	100.3	34.0
4	600	20	20	1154.01	1534	1398.23	105.6	82	192	113.1	34.1

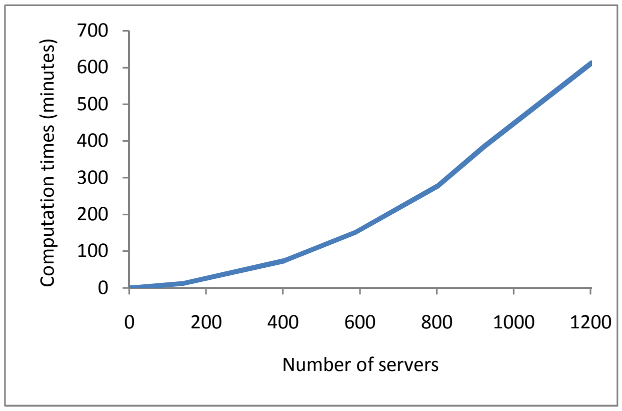

Fig. 7. : Experiment on the size of Cloud network

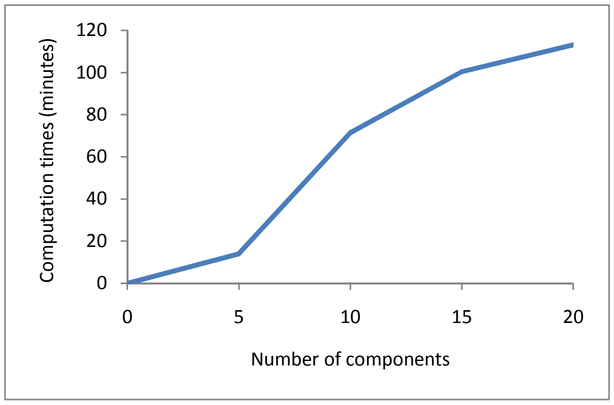

Fig. 8. : Experiment on the number of components

VI. Conclusion and Future Work

This paper has presented the problem formulation and modelling of a composite SaaS placement problem on clusters of Cloud physical servers. The problem can be classified as an optimisation problem where the aim is to optimise the performance of the SaaS based on its estimated execution time. A penalty-based GA is used, and the GA is designed in such way that it considers not only the placement of the software components of a SaaS, but the placement of data of the SaaS as well. We believe that our solution is the first attempt on the composite SaaS placement problem considering data storage in the Cloud. Based on the experiments conducted, the GA always produces a feasible solution for all the test problems in all the experiments. It also can be seen that the proposed GA is scalable. The computation time increases close to linearly when the size of the Cloud increases and when the number of components is increased.

Our work can be improved in a number of ways. First, although the GA has proved its scalability, the computation time taken in finding the solutions can be further improved. This can be done by implementing the GA in a parallel manner. The network can be decomposed into several segments, and the solution can be executed in parallel based on the segmentations.

Second, our technique is implemented in a centralised location. With large size of Cloud networks, better performance of the solution may be achieved if the technique is implemented in several decentralised locations based on Cloud providers' need. However, further research is needed to determine the suitability of the decentralised strategy with SaaS model in order to improve the performance of SaaS in Cloud.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments on this paper.

This research was carried out as part of the activities of, and funded by the Smart Services Cooperative Research

Centre (CRC) through the Australian Government's CRC Programme (Department of Innovation, Industry, Science and Research).

REFERENCES

[1] Foster, I., Yong, Z., Raicu, I., & Lu, S., *Cloud Computing and Grid Computing 360-Degree Compared*, in *Grid Computing Environments Workshop*. 2008, IEEE: Austin, Texas. p. 1-10.

[2] Aymerich, F.M., Fenu, G., & Surcis, S., *An approach to a Cloud Computing network*, in *Proceedings of IEEE Applications of Digital Information and Web Technologies*. 2008, IEEE: Ostrava, Czech Republic. p. 113-118.

[3] Vaquero, L.M., Rodero-Merino, L., Caceres, J., & Lindner, M., *A Break in the Clouds: Towards a Cloud Definition.* SIGCOMM Computer Communication Review, 2009. 39(1): p. 50-55.

[4] Candan, K.S., Li, W.S., Phan, T., & Zhou, M., *Frontiers in information and Software as Services*, in *Proceeding of the IEEE 25th International Conference on Data Engineering*. 2009, IEEE: Shanghai, China. p. 1761-1768.

[5] Laplante, P.A., Jia, Z., & Voas, J., *What's in a name? Distinguishing between SaaS and SOA.* IT Professional, 2008. 10(3): p. 46-50.

[6] Broberg, J., Buyya, R., & Tari, Z., *MetaCDN: Harnessing storage clouds for high performance content delivery*, in *Proceeding of the Sixth International Conference on Service-Oriented Computing*. 2008, ACM: Sydney, Australia.

[7] Gartner Inc (2007) *Introducing SaaS-enabled application platforms: Features, roles and futures*.

[8] Motahari-Nezhad, H.R., Stephenson, B., & Singhal, S., *Outsourcing Business to Cloud Computing Services: Opportunities and Challenges.* HP Laboratories–2009.

[9] Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., et al., *Dynamic placement for clustered web applications*, in *15th International Conference on World Wide Web*. 2006, ACM New York: Edinburgh, Scotland. p. 595-604.

[10] Kichkaylo, T., Ivan, A., & Karamcheti, V., *Constrained component deployment in wide-area networks using AI planning techniques*. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*. 2003.

[11] Kwok, T. & Mohindra, A., *Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications*, in *Sixth International Conference on Service-Oriented Computing*. 2008, Springer: Sydney, Australia. p. 633-648.

[12] Urgaonkar, B., Rosenberg, A.L., Shenoy, P. & Zomaya, A., *Application placement on a cluster of servers.* International Journal of Foundations of Computer Science, 2007. 18(5): p. 1023-1041.

[13] Zhu, X., Santos, C., Beyer, D., Ward, J., & Singhal, S., *Automated application component placement in data centers using mathematical programming.* International Journal of Network Management, 2008. 18(6): p. 467-483.

[14] Zimmerova, B. *Component placement in distributed environment wrt component interaction*. In *Proceedings of the Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*. Czech Republic: FIT VUT Brno, Czech Republic.

[15] Tang, C., Steinder, M., Spreitzer, M., & Pacifici, G., *A scalable application placement controller for enterprise data centers*. In *Proceedings of the 16th International World Wide Web Conference* 2007, Canada: ACM. p. 331-340.

[16] Low, C., *Decentralised application placement.* Future Generation Computer Systems, 2005. 21(2): p. 281-290.

[17] http://www.photoshop.com

[18] http://www-07.ibm.com/storage/au/

[19] http://h71028.www7.hp.com/enterprise/cache/418226-0-0-14-121.html