International Journal of Applied Information Systems (IJAIS) – ISSN: 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 12– No. 40, April 2023– www.ijais.org

A Framework for a Decision Support System to Optimize Cloud-hosted Services for Multitenancy Isolation

Laud Charles Ochei
Department of Computer Science
University of Port Harcourt

Rotimi Ogunsakin
Department of Computer Science
University of Port Harcourt

Nemitari Ajienka
Department of Computing and Technology
Nottingham Trent University

ABSTRACT
One of the challenges of optimizing the deployment of components of cloud-hosted services for guaranteeing multitenancy isolation is how to make optimal decisions that involve resolving the trade-off between a lower degree of isolation versus the possible interference that may occur between components or a higher degree of isolation versus the challenge of high resource consumption and the running cost of the components. Although, many cloud providers offer some functionality in the form of rule-based algorithms, such as Amazon's Auto-Scaling and Microsoft's Windows Azure Traffic Manager. These functionalities are deployed to configure the scaling function of the cloud-hosted services but do not implement the varying degrees of multitenancy isolation for individual components. The aim of this paper is to present a framework for developing a decision support system for optimizing the deployment of components of cloud-hosted services for guaranteeing multitenancy isolation. The framework comprises of a decision support model algorithm, a system architecture, and an algorithm for creating the input files for implementing the decision support system. Extensive experimental evaluation of the framework with a decision support model algorithm shows that it can be used by cloud providers and users to guarantee varying degrees of isolation between tenants.

Keywords
Framework, Decision Support System, Cloudhosted Service, Cloud Deployment, Optimization, Multitenancy, Tenant Isolation.

1. INTRODUCTION
Applications and services are increasingly being deployed to the cloud to be used by multiple tenants/ users, and there is therefore a need to isolate tenants, processes, and components, and thus implement multitenancy. Multitenancy architectures are typically used for deploying components of cloud-hosted services for multiple tenants/users. This is based on the assumption that when tenants share resources, it would lead to a reduction in resource consumption and running costs per tenant.

Multitenancy is a software architecture where one instance of a cloud offering is used to serve multiple tenants and/or components [1] [2]. Figure 1 (adopted from Fiaidhi et al. [3]) represents general architecture for multitenancy cloud environments employing customer integration in three layers: application, infrastructure and data-centre layer. A similar view is shared by Walraven et al. [4] who emphasized that multitenancy can be achieved at three levels: infrastructure level1, middleware level2 and at the application level3.

Two of the most important challenges to address when implementing multitenancy are: (i) how to ensure that there is isolation between multiple components of a cloud-hosted application when one of the components experiences high load; (ii) how to guarantee the varying degrees of isolation between tenants and components of the cloud-hosted services [1], [5]. Varying degrees of tenant isolation are possible, depending on the type of component being shared, the process supported by the component and the location of the component on the cloud application stack (i.e., application level, platform level, or infrastructure level) [6], [1].

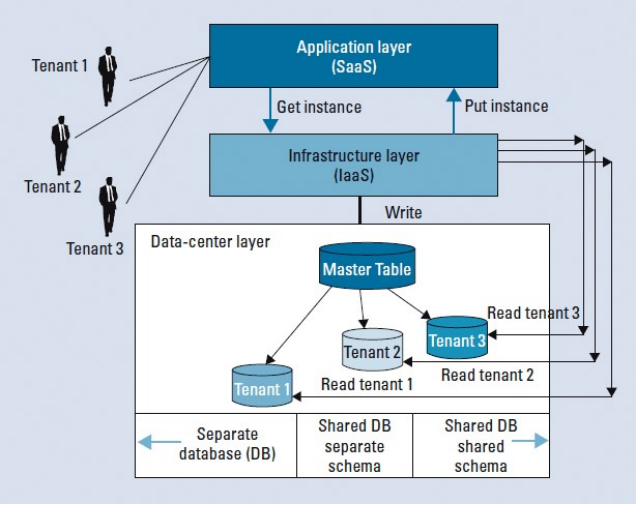

Fig. 1: Overview of a Generic Multitenancy Cloud Architec- ture, adopted from [3]

A high degree of isolation can be achieved by deploying an application component exclusively for one tenant. This would ensure that there is little or no performance interfer- ence between the components when the workload changes. However, because components are not shared (e.g., in a case where there are strict laws and regulations preventing them from being shared), it implies duplicating the components for each tenant, which leads to high resource consumption and running costs. Overall, this will limit the number of requests allowed to access the components. A low degree of isolation may also be required for a component, for example, to allow sharing of the component's functionality, data, and resources. This would reduce resource consumption and running costs, but the performance of other components may possibly be affected when one of the components experiences a change in workload [7].

This leads to an optimal decision-making issue with regard to the trade-off between a lower degree of isolation versus possible interference that may occur between components or a high degree of isolation versus the challenge of high resource consumption and the running cost of the component. In other words, this is a decision-making problem that requires an optimal decision to be taken in the presence of a trade-off between two or more conflicting objectives [8], [9].

Existing approaches to address this problem often look at it from the perspective of the cloud provider (i.e., SaaS, PaaS or IaaS). For example, many cloud providers offer some functionality in the form of rule-based algorithms, such as Amazon's Auto-Scaling and Microsoft's Windows Azure Traf- fic Manager. In addition, some optimization models have been proposed for use by SaaS providers such as Salesforce.com. Although these tools and models can be deployed to configure the scaling function of the cloud-hosted services, they do not implement the varying degrees of multitenancy isolation for individual components.

Motivated by this problem, this paper presents a framework for developing a decision support system for: (i) optimiz- ing the deployment of components of cloud-hosted services for guaranteeing multitenancy isolation; (ii) making optimal decisions when faced with the trade-off between a lower degree of isolation versus the possible interference that may occur between components or a high degree of isolation versus the challenge of high resource consumption and the running cost of the components. The framework comprises of a decision support model algorithm, a system architecture, and an algorithm for creating the input files for implementing the decision support system.

The framework has been extensively evaluated by com- paring the solutions obtained from our framework with the optimal results obtained from an exhaustive search of the entire solution space for a small problem. The main research question to address in this paper is: ***"How can we make optimal decisions that involve resolving trade-offs between conflicting objectives to support the deployment of components of a cloud-hosted service for guaranteeing multitenancy isola- tion?"***. To the best of our knowledge, this study is the first to present a framework for developing a decision support system for making optimal decisions for resolving trade-offs between conflicting objectives when deploying components of a cloud-hosted system to guarantee multitenancy isolation. This paper looks at the problem from the perspective of a tenant who knows in advance the number of tenants or users to grant access to its service or components, owns software components and is responsible for configuring them to design and deploy its cloud-hosted application on a shared cloud platform not controlled by the cloud provider. Many cloud providers offer runtime information of components and mon- itoring information, for example, information about network availability and utilisation of components deployed on their cloud infrastructure. It is the responsibility of the customer to extract, deduce and interpret these values and then provide important decisions regarding, for example, the availability of components, provisioning of required components and decommissioning of unused components.

This paper extends and expands on the previous work conducted by Ochei et al. [7]. We summarise the additions to the previous work as follows. Firstly, a framework is provided that can be used by cloud providers and users to develop a decision support system for optimizing the deploy- ment of components of cloud-hosted services for guaranteeing multitenancy isolation. Secondly, we modified the previous decision support model algorithm to include a module to further evaluate the results of the optimization model. Thirdly, the system architecture for Decision Support System has also been modified in line with the inclusion of the module to evaluate the decisions support systems. Fourthly, we include an algorithm for creating input files for running the decision support system. The instance file represents a typical structure of components of a cloud-hosted service. Fifthly, an extensive experimental evaluation of the decision support system with two variants of a metaheuristic which are based on a simulated annealing algorithm and hill climbing. Lastly, recommenda- tions and best practice guideline for using the decision support system has been provided.

The main contributions of this paper are:

1) Creating a novel decision support model algorithm called *optimalDSS*, together with a system architecture called *optimalDssArch* for making decisions regarding the deployment of components of a cloud-hosted service with guaranteed multitenancy isolation. The paper also includes a novel *evaluateDSS* algorithm for analysing optimal solutions and other related information for mak- ing the best decisions for the optimal deployment of components.

2) Presenting a novel algorithm for creating input files (i.e., instance file, service demand file and workload files) for running the decision support system. The algorithm can generate input files all at once. In particular, the instance file represents a typical structure of components of a cloud-hosted service.

3) Extensive experimental evaluation of the decision sup- port system with two variants of a metaheuristic which are based on a *simulated annealing* algorithm and *hill climbing*.

4) Presenting recommendations and implications for prac- titioners for using the decision support system to deploy optimal components to the cloud for guaranteeing mul- titenancy isolation.

The rest of the paper is structured as follows: Section

II reviews prior related studies to the work presented in the paper. Section IV described the guaranteed multitenancy decision support system framework while Section V described the Decision Support Model Algorithm called optimalDSS to assist decision makers (e.g., software architects and cloud developers). Section VI describes the dataset, and experimental

setup while Section VII describes the derived results from the experiments. The results are then discussed in Section VIII and the implications of the study for practitioners (e.g., cloud engineers) are discussed in Section IX. Finally, the study is summarised and concluded in Section XI with plans for further work outlined therein.

2. REVIEW OF RELATED WORK

In this section of the paper, we review some of the related work to this study. To the best of our knowledge, this study presents a novel decision support algorithm for cost-effective cloud component deployment while guaranteeing multitenancy isolation.

2.1 Related Work on Multitenancy Isolation

Multitenancy isolation has recorded considerable traction in the cloud computing domain in the past years. Multitenancy Isolation is paramount to ensure each user accessing a cloud-hosted service is insulated from the interference that may occur as a result of the activities of other cloud users, and that each cloud user's data are not visible to other cloud users.

Pathirage et al. [10] argued that multitenancy could en- able cloud middleware that maximizes sharing and supports near-zero costs for unused applications. According to Fiaidhi et al. [3], the three main ways of achieving multitenancy in cloud environments are: using databases, virtualization or physical separation. Walraven et al. [4] emphasized that multitenancy can be achieved at three levels: infrastructure level using virtualization; middleware level using shared OS and middleware; and at the application level where maximum cost efficiency is achieved by sharing the underlying infras-tructure, database, OS, middleware and application between different tenants. The authors explored the challenges of performance isolation in view of multitenant software-as-a-service (SaaS) cloud applications and proposed a middleware architecture prototype which enforces performance isolation based on tenant-specific SLAs using a tenant-aware profiler and scheduler. Differently from the study in [4], our decision support algorithm does not rely on the tenants' SLAs. Cai et al. [11] on the other hand proposed and developed a 3-step transparent approach to enable an existing web application to support multitenancy when migrated to the cloud. Calero et al. [12] proposed an authorization model for controlling access to resources in a multitenancy cloud environment. The basic authorization model that takes into cognizance a 3-tuple, which are the subject, its privilege, and the object of interest, was extended to enable multitenancy support.

Hence, Ochei et al. [6] resolve to analyze the degree of tenant isolation for cloud-hosted software services. The authors posit that guaranteeing multitenancy isolation requires making optimal decisions regarding the trade-off between a lower degree of isolation versus the possible interference that may occur between components or a high degree of isolation versus the challenge of high resource consumption and the running cost of the components.

2.2 Related Work on Optimal Deployment of cloud-hosted Services

Research work on optimal deployment and allocation of cloud resources on the cloud is quite significant. However, there has been little or no work on providing an optimal solu- tion for deploying components of a cloud-hosted application in a way that guarantees the required degree of multitenancy isolation. In [13], the authors used an evolutionary algorithm to minimize resource consumption for SaaS providers and improve execution time. The authors in [14] and [15] used a multitenant SaaS model to minimize the cost of cloud infrastructure. Heuristics were not used in this work. The authors in [16] developed a heuristic for capacity planning that is based on a utility model for the SaaS. This utility model mainly considers the business aspects related to offering a SaaS application with the aim of increasing profit.

The authors in [17] proposed a geography- aware task scheduling (GATS) approach by considering spatial variations in a cloud environment (e.g., a distributed green data center

- DGDCs) to maximize the total profit of the cloud provider by intelligently scheduling tasks of all applications. Also, [18] proposes a revenue-based workload admission control method to judiciously admit requests by considering factors including priority, revenue and the expected response time. Then, this paper presents a cost-aware workload scheduling method to jointly optimize the number of active servers in each CDC, and the selection of Internet service providers for the CDCs provider. In [19] a new approach to optimize the profit of VCDC based on the service-level agreements (SLAs) between service providers and customers. Heuristics based on simulated annealing and particle swamp optimization were used in this work.

In [20], the authors described how the optimal configuration of a virtual server can be determined, for example, the amount of memory to host an application through a set of tests. Fehling et al [21], considered how to evaluate the optimal distribution of application components among virtual servers. A closely related work to ours is that of Aldhalaan and Menasce [22], where the authors used a heuristic search technique based on hill climbing to minimize the SaaS cloud provider's cost of using VMs from an IaaS with response time SLA constraints. Related work on multitenancy isolation has largely focused on isolation at the data tier [23]. The main aspect of isolation is usually performance isolation. For example, the authors in

[24] mainly focus on performance isolation in a multitenant application in the cloud. The varying degrees of multitenancy isolation based on multitenancy patterns and the different aspects of isolation are described in [25].

Most work on optimal deployment and allocation of cloud resources on the cloud focuses on minimising the cost of using the cloud infrastructure resources [13]. Previous work concern-ing the optimization of cloud resources does not use heuristics at all, although a few use simple heuristics. For example, the authors in [26], [22] used a heuristic based on hill climbing for minimising the cost of SaaS cloud providers with response time SLAs constraints. This study, unlike others, focuses on providing an optimal solution for deploying components of a cloud-hosted application in a way that guarantees the required degree of multitenancy isolation.

2.3 Related work on Decision Support System for Optimal Deployment of Cloud-hosted services

There are several works on developing decision support systems for performing different kinds of cloud operations. For example, Sri and Balaji [27] developed a speculation-based decision support system for efficient resource provisioning in the cloud data centers. The decision support system guaranteed to dodge over/under utilization of resources and minimized the cost economically without compromising the Quality of Service.

Mathirajan et al [27] developed a cloud-based decision support system (C-DSS) for transport analytics. The C-DSS is based on an intelligent model on location of depots for opening new depots and/or closing a few existing depots and allocation of city-buses to depots. Andrikopoulos et al. developed decision support for application migration to the cloud [28]. Menzel and Ranjan developed CloudGenius, a decision support for web server cloud migration [29].

This study, unlike other decision support systems, focuses on providing a decision support system for providing an optimal solution for deploying components of a cloud-hosted service in a way that guarantees the required degree of multitenancy isolation.

3. PROBLEM FORMALIZATION AND NOTATION

Before we describe the framework, it is important to understand the formalisation of the problem that motivated the development of the framework. The reader is referred to the previous work conducted by Ochei et al. to understand the problem formalization and notation including the system model and description of the problem, the optimal function of the problem, and the mapping of the problem to a multi- choice multidimensional knapsack problem (MMKP) [7]. It is important to note that the model for the optimal component deployment problem assumes that CPU, RAM, Disk, and Bandwidth service demands are known or easily measured. The cloud provider can provide third-party tools to measure this, or the SaaS customer can extract and measure the data. These sections will not be repeated in this paper.

The problem that motivated the development of the framework has several application areas such as optimal allocation in a resource-constrained environment, monitoring runtime information of components, and controlling the provisioning and decommissioning of components in a cloud environment. As a specific example, consider a version control system (e.g., subversion) configure to record changes to a file or set of files (e.g., source code) over time so that you can recall specific versions later. In such a system, the shared component would be better for reducing resource consumption while the dedicated component would be better to avoid performance interference. However, this might not necessarily be so because as additional copies of the files are created in the repository, the disk space consumed continues to enlarge. Over time, performance begins to degrade as more time is spent searching across many files on the disk. This is a trade-off decision that required a decision support system to analyse the set of optimal solutions together with other related information to provide an optimal decision for deploying the components to the cloud.

4. A FRAMEWORK TO DEVELOP A DECISION SUPPORT SYSTEM FOR GUARANTEEING MULTITENANCY ISOLATION

The framework is first presented as part of an input- process-output (IPO) model (see Figure 2). This approach is widely used in systems analysis and software engineering for describing the basic structure of a service or process [30], [31]. In our case, the model represents a cloud-hosted service that can be designed to use or integrate with several components and/or other services. In using the IPO model, the framework receives inputs from a user (that is, a cloud provider or a cloud user), sends it to the decision support system which carries out some analysis and optimization, and then returns decisions regarding multitenancy isolation. Such decisions could be, for example, the optimal solutions regarding the required degree of isolation; optimal solutions that would allow a certain number of requests to access a component; or a whole system, or a notification when a certain threshold is reached.

Fig. 2: Framework for developing a decision support system to deploy cloud-hosted services for guaranteeing multitenancy isolation (as part of an Input-Process-Output Model)

The main architecture for the decision support system is presented in Figure 3. It is important to note that the main addition to the architecture in our previous work [7] is the decision analyzer. In the previous architecture, the main output was the optimal solutions. This can be likened to the usual monitoring information provided by cloud providers. Although, many providers offer monitoring information, for example, information about network availability and utilization of components deployed on their cloud infrastructure. However, it is the responsibility of the customer to extract, deduce and interpret these values and then provide information regarding the availability of components.

In our case, the optimal model will provide data regard- ing the optimal values and the associated optimal solutions. However, further analysis is required to provide information regarding the set of solutions that would guarantee a low degree of isolation and not just a high degree of isolation, or which optimal solution(s) would guarantee a certain level of resource consumption for certain components or the whole system.

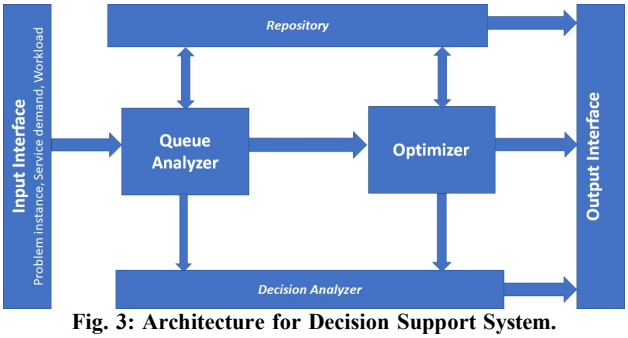

Fig. 3: Architecture for Decision Support System.

Finally, we present a sequence diagram in Figure 4 to illustrate the overall flow of tasks in using the components of the framework to develop a decision support system for deploying components of a cloud-hosted service for guaran- teeing multitenancy isolation.

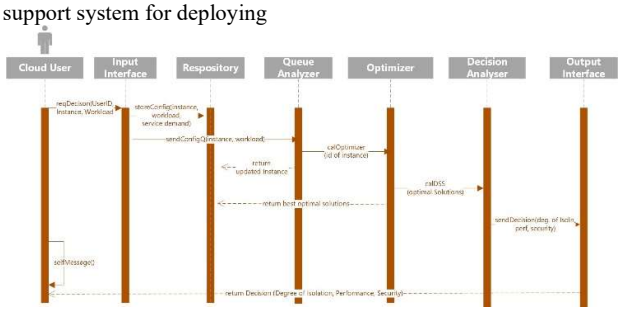

Fig. 4: Sequence Diagram for Implementing DSS

5. DECISION SUPPORT MODEL ALGORITHM FOR OPTIMAL DEPLOYMENT OF COMPONENTS

In this section, we describe the Decision Support Model algorithm, optimalDSS, to assist decision-makers (e.g., soft- ware architects, cloud developers) in analyzing different cloud deployment scenarios for deploying components of cloud-hosted software, to guarantee multitenancy isolation. In this paper, we provide the main algorithm which can be used to drive the decision support system (DSS). The DSS can be implemented in different ways such as a desktop application, web application or cloud-based hosted service, or embedded into other applications running on the cloud or distributed environment.

It is important to note that the DSS provided in this paper can be seen as an abstract format that allows the implemen- tation of a decision support system for optimal deployment of components to the cloud in various ways. It captures the essential properties required for the successful implementation of DSS for optimal deployment of components of a multitenant cloud-hosted service while leaving large degrees of freedom to cloud deployment architects depending on the required degree of isolation between components, and the deployment environment. Furthermore, our approach can be applied at different levels of the application or cloud stack as long as the components can be represented as described in the optimization model.

5.1 OptimalDSS: An algorithm for Decision Support Model

This section describes the OptimalDSS algorithm first pre- sented in our previous work [7]. This algorithm is repeated here for ease of reference, and clarity, and to show how this algorithm fits into the Decision Support System. The optimalDep model presented in our previous work [7] maps to the optimizer module in the architecture for the decision sup- port system shown in Figure 3. The goal of the optimizer is to address the problem of providing optimal solutions and other related information (e.g., the number of function evaluations to produce the optimal value, the magnitude of deviation of the optimal value from the target solution) for deployment to the cloud in such a way that meets the system requirements and also provides the best value for the optimal function. The OptimalDSS algorithm is an extension of the optimalDep. In this algorithm, we have added a module referred to as: evaluateDSS, which can be embedded within the optimalDSS or implemented as a separate module. In short, Algorithm 2 is an implementation of the evaluateDSS module in line 21 of Algorithm 1.

Algorithm 1 optimalDSS Algorithm

1: optimalDep (workloadFile, mmkpFile)
2: *optimalSoln* ← null
3: *optValue, isolValue, Req, Pen, optimalSoln* ← null
4: Accept workload from SaaS users
5: Load workloadFile, mmkPfile; populate global variables
6: **repeat**
7: /*Compute No. of req. using QN Model*/
8: **for** $i \leftarrow 1, NoGroups$ **do**
9: **for** $j \leftarrow 1$,GroupSize **do**
10: Calculate Utilization
11: Calculate No. of req.
12: Calculate Total No. of req.
13: Store fitValue, Isol, qLength of optimal soln.
14: **end for**
15: **end for**
16: Update the mmkpFile with qLength
17: /*Run Metaheuristic-SA(Greedy) or Hill(Greedy)*/
18: SA(GREEDY)( ) ← result
19: /*Encapsulate all result properties in Result class*/
20: /*and return an object of Result class
21: EVALUATEDSS(RESULT)( )
22: /*Display results from DSS*/
23: /*regarding optimal deployment*/
24: **until** no more workload
25: **Return** (DegreeIsoln, ResUtil, Perf)

Decision Analyser of the OptimalDSS Algorithm

The following example (Table III) shows the different ways of evaluating the results of the Decision Support System. In this paper, we look beyond the optimal value, which is an objective function that is optimized in the optimization model presented in our previous work [7]. That is, in addition to the optimal value, we show how the optimalDSS can be used to provide information regarding the degree of isolation of the components, resource consumption, performance, and security.

Algorithm 2 evaluateDSS Algorithm

1: evaluateDSS (optimalResult)
2: *worstOptVal, worstSoln* ← null
3: *worstIsolVal, worstReqNo, worstPen* ← null
4: *bestOptVal, bestSoln* ← null
5: *bestIsolVal, bestReqNo, bestPen* ← null

```
6: /*set worstOptVal, bestOptVal as first item in list*/
7: worstOptVal, bestOptVal ← first value in list
8: for i ← 1,NoSimulations do
9:     for j ← 1,resultSize do
10:    if currenOptVal < worstOptVal then
11:    /*replace the initial worst values by*/
12:    /*the current worst values*/
13:    worstOptVal ← currentOptVal
14:    worstSoln ← currentWorstSoln
15:    worstIsolVal ← currentIsolVal
16:    worstReqNo ← currentReqNo
17:    worstPen ← currentPen
18:    end if
19:    Store worstOptVal, worstSoln
20:    Store worstIsolVal, worstReqNo, worstPen
21:    if currentOptVal > bestOptVal then
22:    /*replace the initial best values by*/
23:    /*the current best values*/
24:    bestOptVal ← currentOptVal
25:    bestSoln ← currentWorstSoln
26:    bestIsolVal ← currentIsolVal
27:    bestReqNo ← currentReqNo
28:    bestPen ← currentPen
29:    end if
30:    Store bestOptVal, bestSoln
31:    Store bestIsolVal, bestReqNo, bestPen
32:    end for
33: end for
34: /*Display worst and best values of DSS*/
35: Return (all required results from DSS)
```

Every time there is a change in the workload, the op- timalDSS algorithm finds a new optimal solution for deploying components with the highest degree of isolation and the highest number of supported requests. This information is encapsulated in the optimal value.

In line 18, the metaheuristic that runs to produce the optimal algorithm returns a class that encapsulates the result properties of the decision support system. In line 20, this result is then analyzed in EvaluateDecision module to produce several important pieces of information that can be used to make optimal decisions regarding the optimal deployment of components of cloud-hosted services. Such information may include, may include but is not limited to: (i) the degree of isolation of the components; (ii) resource utilization and consumption of the components; (iii) performance (e.g., the time required to produce optimal solutions for deployment) computation time interference (iv) security interference (e.g., quality of optimal solutions, number and magnitude of penalty violations) ; (v) deployment rate (that is, the rate at which optimal components can be deployed).

The evaluateDSS algorithm works as follows: the input to the algorithm is a group of objects 4 class object that encapsulates all variables (e.g., number of simulations) and methods (e.g, best and worse optimal solution) required to access the information required for evaluation by the decision analyser.

After defining and initializing the variables and data struc- tures (line 2-5) to store the required values, set a loop to iterate (line 8-9) and search through the optimal results based on the conditions/rules provided to the decision analyser (line 10 and line 20). In line 10, if the current worst value is lower previous/initial worst optimal value, then replace the worst optimal value with the current value. Other rules could be set for the decision analyser. For example, set rules to specify that a new set of components be selected for deployment once the arrival rate of requests exceeds a defined threshold. The loop continues until all the solutions are traversed. The selected values are then stored and later displayed to the user.

We assume that the input of this algorithm (i.e., the optimal results) is stored in a linear data structure (e.g., array list) and the size of the data structure is N. The time complexity to find the worst or best optimal value in the data structure is linear O(N) and the space complexity is O(1).

6. EVALUATION

The decision support system is driven by the OptimalDSS algorithm shown in Algorithm 1. This algorithm combines an open Queuing Network model and a meta-heuristic to select a set of optimal solutions for deployment to the cloud in order to guarantee multitenancy isolation.

6.1 Dataset

The dataset used for simulation experiments on the opti- mization model was based on a simulation test bed. There are four types of datasets used in this study:

(i) *MMKP Instance file:* This file represents the components and their properties.See section 6.2 for an algorithm to gener- ate the MMKP instance.

(ii) *Workload file:* This file contains information about the varying workload that the system is exposed to.

(iii) *Service demand:* This file contains information about the service demands of the system.

(iv) *MMKP Instance file:* This file contains information about the updated instance based on workload changes.

The dataset was generated and tailored on the MMKP instances widely cited in the literature: (i) OR benchmark Library [32] and other standard MMKP benchmarks, and (ii) the new irregular benchmarks used by Shojaei et al. [33].

4In most high-level programming languages like Java, this is equivalent to a class which is used to represent a group of objects which have common properties. Several MMKP instances of various sizes and densities were randomly generated following a Poisson distribution. The MMKP problem instances represent a repository of compo- nents (e.g., database, a database table, a message queue, VM or docker container) that can be deployed to design (or integrate with) a cloud-hosted service. The weight values generated in the MMKP instance could be normalised (or transformed) to represent different resources units (gigabytes of memory) of the components.

6.2 createInstance: An algorithm for MMKP instance files

This paper also provides the createInstance algorithm (see Algorithm 3), for creating all the input files required to run the decision support system. The decision support system requires three types of input files, namely, instance file, service demand file and workload file. The algorithm shows how the three input files can be created all at the same time, and once these files have been created, they can be used as input into the decision support system.

The createinstance algorithm works as follows:

The algorithm starts by defining and initializing the vari- ables required for generating files for workload, service de- mand and problem instance (line 2-6). Thereafter a loop is set up to run from 0 to the number of workloads required (line 7-37). For each iteration, first, calculate the resource limit for each resource supporting the components (based on the specified rule), and for each component group, calculate values for isolation level, resource values, number of users, service demands, and arrival rate (based on a Poisson distribution as shown line 19), and write values to workload file and service demand file, an instance file.

6.3 Experimental Settings and procedure

Aim of the Experiment: The aim of the experiment is to evaluate the performance of the decision support system in terms of the quality of solutions obtained when there are varying workloads changes.

The instance-generating program and the algorithms were written using Java programming with Apache Netbeans IDE 11.3. All experiments have been carried out on the same computation platform, which is Windows 10 Pro running on a SAMSUNG Laptop with an Intel(R) CORE(TM) i7-3630QM at 2.40GHZ, with 8GB memory and 1TB swap space on the hard disk.

In the experiment, we carried out 1000 function evaluations and 20 runs. This implies that we take the best and worst optimal values out of 20000 solutions (1000 ˆ 20). This is a fairly large number of solutions with which to evaluate the metaheuristic that drives the decision support system. Table 2 shows the parameters used for the experiments.

This study is novel and there are no existing approaches that can be used to make a direct comparison with our approach in terms of the quality of optimal solutions. This study is novel in the sense that the optimalDep algorithm combines a Queuing Network model and metaheuristics to find optimal solutions for component deployment while guaranteeing the required degree of multitenancy isolation. Because of this, the solutions obtained from our approach were compared with the optimal solutions obtained from an exhaustive search of a small problem instance. Thereafter, the obtained solutions are also compared with the target solution obtained from different problem instances of varying sizes and densities.

Algorithm 3 createInstance Algorithm

1: createInstance ()
2: *NoWorkloadFiles, NoGroups, NoComp, NoConstr* ← 0
3: *CPUlimit, RAMlimit, Disklimit, BWlimit* ← 0
4: *maxCPU, maxRAM, maxDisk, maxBW* ← 0
5: Create new *instanceFile, worklaodFile, servicedemandFile*
6: $ARate \leftarrow 0$
7: **for** $i \leftarrow 1, NoWorkload$ **do**
8: Calculate Resource limit
9: Write instance, resource properties to workload file
10: Write instance, resource properties to serv. demand file
11: **for** $i \leftarrow 1, NoWorkload$ **do**
12: Write instance properties to instance file
13: Write Resource limits to file
14: **end for**
15: Write instance properties to service demand file
16: Write Resource limits to service demand file
17: **for** $i \leftarrow 1, NoGroups$ **do**
18: Write group size to instance file
19: Generate Arrival rate following Poisson distribution
20: **for** $i \leftarrow 1, NoWorkload$ **do**
21: Write group size to workload file
22: **end for**
23: Write group size to service demand file
24: **end for**
25: **for** $i \leftarrow 1, NoGroups$ **do**
26: Generate isolnLevel, NoUsers, CPUlimit,
27: RAMlimit, Disklimit, BWlimit
28: Write isolnLevel, NoUsers, CPUlimit,
29: RAMlimit, Disklimit, BWlimit to instance file 30: Create service demands for the components 31: Write service demands to file
32: **for** $i \leftarrow 1, NoWorkloadFiles$ **do**
33: Write Arrival Rate and Service Demand
34: to workload file
35: **end for**
36: **end for**
37: **end for**
38: Close all files
39: **Return** (instanceFile, servicedemandFile, workloadFile)

TABLE I: Experimental Parameters (based on Simulated An- nealing)

Open Multiclass QN Model	**Value**
λ (offered load)	[0,4]
Isolation Value	[1,2,3]
No. of Requests	[1,10]
Resource consumption	[1,10]
Service Demands	[0.15, 0.24]
Metaheuristic	

No. of Iterations	1000000
Population size	1000
No. of Runs	20
Temperature	$T0$ = st. dev of N randomly generated solutions (N=no. of groups)
Cooling Schedule	Ti`1 " $T0$ ` pA ´ $T0$q

7. RESULTS

This section presents the results of an extensive evaluation of the DSS to show how we can deduce more information regarding the optimal deployment of components of a cloud-hosted service. This section compliments the results about the quality of optimal solutions, robustness of solutions, and the computational effort required to produce optimal solutions for deploying components of cloud-hosted solutions discussed in [7]. Results are shown in Table I to Table IV. Column 1 shows the workload represented either as a single value for increasing(or decreasing) arrival rates or as a set of values for fluctuating(varying) arrival rates. Column 2 - 5 shows the results shown in the following format - (worst-case value/best-case value). The second column shows the worst and best set of optimal solutions selected for deployment. The third column shows the first optimal value associated with the worst or best solution.

7.1 Small Problem Instance

This section presents results for a small dataset that rep- resents a small problem instance. The small dataset has an instance file with the following dimensions - C(10, 20, 4), that is, the instance file has ten groups of components, 20 components per group and is supported by four resource types

CPU, memory, disk space and bandwidth. The dataset also has an accompanying service demand file and a workload file.

1) Components Experiencing the same workload: Table II shows the results of a decision support system for a workload (i.e., arrival rate) that increases (or may decrease over time). In this scenario, all the components in a particular group are exposed to the same arrival rate.

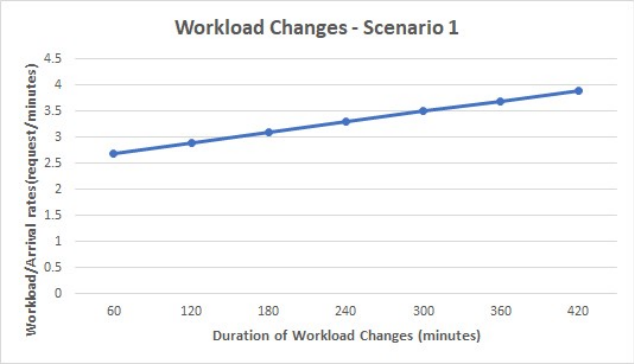

Fig. 5: Workload Changes for increasing (or decreasing) arrival rates

In order to evaluate the degree of isolation of each optimal solution we have to first obtain the solution with the lowest optimal value and the solution with the highest optimal value.

For the results of HC(greedy) in Table II, when the system experiences a workload of 3.9 (request/second) the lowest and highest optimal value is 3036.34 and 3132.70, respectively.

In this study, we assume there are three degrees of tenant isolation which also correspond to the three types of multitenancy patterns that can be used to deploy components to the cloud (shared pattern, tenant-isolated pattern and dedicated pattern). The shared pattern guarantees the lowest degree of isolation, and the dedicated guarantees the highest degree of isolation, while the tenant-isolated pattern is in the middle [34] [6].

The following computation shows how to deduce the required degree of isolation decision using the decision support system.

1. Calculate the magnitude of the optimal solution for the workload

$$mag = OptV\ alue_{highest} \_\ OptV\ alue_{lowest} \quad (1)$$

From the above equation, Mag = 3132.70 - 3036.34= 96.36

2. Compute the range of values for a low degree of isolation (i.e., shared pattern), a middle-level degree of isolation, and the highest degree of isolation (i.e., the dedicated pattern used for deployment). We assume that each of the three degrees of isolation contains one-third of the magnitude of the optimal values (that is, 33% of the values) from the lowest degree to the highest degree of isolation.

   i. Low degree of isolation:
   $$mag = OptV\ alue_{highest} \_\ OptV\ alue_{lowest} \quad (2)$$

   ii. Middle-level degree of isolation
   $$Isoln_{high} = [(Mag - (Mag * 0.33) - Mag] \quad (3)$$

iii. High degree of isolation:
$Isoln_{high} = [Mag * 0.33) - (Mag - (Mag * 0.33)]$ (4)

Assuming a cloud deployment architect wants to deploy components of a cloud-hosted service that guarantee a low degree of isolation, maybe to improve the sharing of resources between the components and the system as a whole. Then the cloud architect would choose an optimal value between 0 – 31.80. In this example, 31.80 (that is, 96.36 * 0.33) is the upper bound of the magnitude of the optimal values. Assuming again that this value is 25, then this will translate to an optimal value of 3061.34. The minimum optimal value for the required degree of isolation is therefore 3061.34 (that is, 3036.34 + 25) and the maximum optimal value for the required degree of isolation is 3068.14 (that 3036.34 + 31.80). These two values become an input to the decision support system, and it produces the following optimal solutions: [18, 18, 1, 8, 4, 7, 10, 9, 3, 10].

Assuming the required optimal value is L (that is, 3061.34) and the upper bound of the magnitude of the optimal values is U (that is, 3068.14). From an implementation point of view, this translates to inserting the following rule (Algorithm 4) in the EvaluteDecision module of the decision support system.

TABLE II: Results of DSS based on HC(Greedy) for workload with increasing(or decreasing) arrival rates

Workload	Optimal Solutions	Optimal value	Isolation value	No. Request	Penalty
2.7	[9, 3, 1, 8, 13, 19, 2, 15, 3, 4]/ [6, 3, 1, 6, 4, 7, 11, 15, 14, 10]	2937.20/3038.38	29/30	38.20/38.38	0/0
2.9	[6, 18, 1, 3, 2, 7, 10, 15, 1, 10]/ [6, 1, 1, 15, 12, 7, 18, 5, 14, 19]	2946.17/3044.76	29/30	44.76/48.17	0/0
3.1	[6, 1, 1, 15, 4, 7, 9, 15, 14, 10]/ [6, 1, 1, 15, 4, 7, 9, 15, 14, 10]	2949.90/3045.85	29/30	45.85/50.90	0/0
3.3	[6, 18, 1, 3, 12, 7, 18, 12, 14, 19]/ [6, 6, 1, 6, 12, 7, 2, 12, 3, 19]	2956.09/3056.24	29/30	56.24/58.09	0/0
3.5	[6, 1, 1, 15, 4, 7, 9, 15, 14, 10]/ [6, 1, 1, 15, 12, 7, 18, 5, 14, 19]	2971.89/3078.24	29/30	72.89/78.24	0/0
3.7	[11, 3, 1, 8, 4, 10, 11, 15, 18, 19]/ [6, 3, 1, 15, 7, 7, 18, 12, 1, 4]	3004.50/3101.91	29/30	101.91/106.50	0/0
3.9	[18, 18, 1, 8, 4, 7, 10, 9, 3, 10]/ [6, 6, 1, 6, 12, 7, 2, 12, 3, 19]	3036.34/3132.70	29/30	132.70/137.34	0/0

Algorithm 4 Algorithm for producing Optimal Solution for specified Optimal Value

1: **INPUT:** req. optimal value
2: **OUTPUT:** optSoln, degofIsolation
3: *optSoln* ← null
4: **if** optSoln ≥ L AND optSoln ≤ U **then**
5: *return current best optimal solution*
6: **end if**
7: **return** (optSoln, degofIsolation)

3. Component *Experiencing the Varying Workload:* Table III shows the results of the decision support system for a cloud- hosted service that experiences varying workload changes. In this scenario, different components in the group experience varying workloads. That is, the arrival rates for these components follow a Poisson distribution. Figure 5 and 6

represent the workload for the first workload file in Table III and V. In this workload file, the arrival rates to each component group are given in the following vector: **[1,2,3,2,2,4,4,4,3,1,3]**. We assume that each workload changes every 60 minutes. In some dynamic and real-time systems, these changes can be much faster, for example, in seconds and milliseconds.

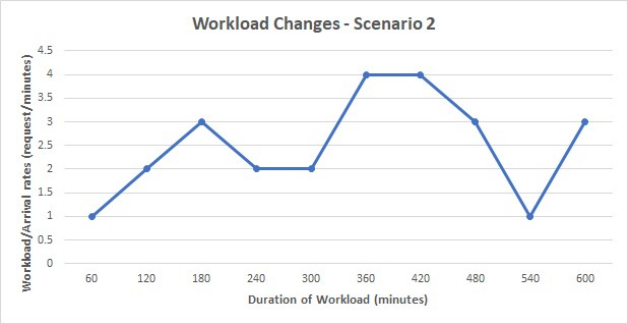

Fig. 6: Workload changes for fluctuating (or varying) arrival rates

7.2 Large Problem Instance

This section presents results for a large dataset that rep- resents a large problem instance. The large dataset has an instance file with the following dimensions - C(500, 20,4), that is, the instance file has 500 groups of components, 20 components per group and is supported by four resource types

TABLE III: Results of DSS based on HC(Greedy) for work- load with fluctuating(varying) arrival rates

Workload	Optimal Solutions	Optimal Value	Isol. value	No. Request	Pen.
[1,2,3,2,2,4,4,4,3,1,3]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [18 3 14 12 17 10 0 18 11 10]	3047.48/3052.48	30/30	47.48/52.48	0/0
[3,2,1,3,4,0,2,2,4,2]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [18, 3, 14, 12, 17, 10, 0, 18, 11, 10]	3077.68/3082.69	30/30	77.68/82.69	0/0
[3,1,2,4,4,2,1,3,4,2]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [18, 3, 14, 12, 17, 10, 0, 18, 11, 10]	3069.01/3074.01	30/30	69/01/74.01	0/0
[4,2,4,4,2,3,3,4,2,3]	[0, 3, 13, 12, 13, 10, 0, 18, 4, 4]/ [18, 3, 14, 12, 17, 10, 0, 18, 11, 10]	3061.04/3066.04	30/30	61.04/66.04	0/0
[2,3,4,0,0,2,2,1,2,3]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [18, 3, 14, 12, 17, 10, 0, 18, 11, 10]	3039.13/3043.13	30/30	39.13/43.13	0/0
[1,3,1,1,4,3,3,3,2,0]	[5, 3, 8, 19, 17, 10, 16, 18, 11, 10]/ [18 3 14 12 17 10 0 18 11 10]	3025.71/3030.71	30/30	30.25/30.71	0/0
[1,4,2,2,3,2,2,4,1,1]	[5, 6, 2, 12, 17, 16, 10, 18, 4, 4]/ [18, 3, 14, 12, 17, 10, 0, 18, 11, 10]	3039.90/3044.90	30/30	39.90/44.90	0/0

TABLE IV: Results of DSS based on SA(Greedy) for work- load with increasing(or decreasing) arrival rates

Workload	Optimal Solutions	Optimal Value	Isolation value	No. Request	Pen.
2.7	[6, 3, 1, 6, 4, 7, 11, 15, 14, 10]/ [6, 1, 1, 15, 4, 7, 9, 15, 14, 10]	2837.21/3040.40	28/30	38.21/40.40	0/0
2.9	[6, 3, 1, 6, 4, 7, 11, 15, 14, 10]/ [6, 3, 1, 15, 7, 7, 18, 12, 1, 4]	2945.17/3047	29/30	48.17/ 47	0/0
3.1	[18, 18, 1, 12, 12, 12, 11, 5, 14, 19]/ [6, 18, 1, 3, 2, 7, 10, 15, 1, 10]	2846.84/3050.53	28/30	46.84/50.53	0/0
3.3	[9, 3, 1, 3, 16, 7, 10, 5, 3, 10]/ [6, 3, 1, 15, 7, 7, 18, 12, 1, 4]	2956.09/3062.16	29/30	58.09/62.16	0/0
3.5	6, 3, 1, 6, 4, 7, 11, 15, 14, 10]/ [6, 1, 1, 15, 12, 7, 18, 5, 14, 19]	2881.41/3086	28/30	82.41/86	0/0
3.7	[6, 3, 1, 3, 12, 1, 2, 12, 14, 7]/ [6, 1, 1, 15, 4, 7, 9, 15, 14, 10]	2881.99/3101.91	28/30	81.98/101.91	0/0
3.9	[6, 1, 1, 15, 4, 7, 9, 15, 14, 10]/ [6, 18, 1, 3, 12, 7, 18, 12, 14, 19]	2945.65/3177.79	28/30	146.65/177.80	0/0

- CPU, memory, disk space and bandwidth. The dataset also has an accompanying service demand file and a workload file.

Components Experiencing the same workload: In this scenario, all components in the group experience the same workloads. The length of the array containing the best and worst optimal solutions is very large. As a result, these optimal solutions for the different workloads will not be represented in a tabular form, as shown in Table II to V. Therefore, we will only show the optimal value, isolation value, and the number of requests for one workload with an arrival rate of 2.7 requests per second.

Due to space limitation, we have shown sample optimal solutions (i.e., the best and worst optimal solutions) for a large problem instance (i.e., C(500,20,4)) when arrival rates in the workload file are the same (i.e., 2.7 requests per seconds) and also when arrival rates in the workload file vary is shown in Appendices B. The problem instance is applied on SA(Greedy) metaheuristics.

TABLE V: Results of DSS based on SA(Greedy) for workload with fluctuating(varying) arrival rates

Workload	Optimal Solutions	Optimal value	Isol. value	No. Request	Pen.
[1,2,3,2,2,4,4,3,1,3]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [0, 3, 4, 19, 17, 10, 0, 11, 12, 16]	2855.37/ 3057.87	28/30	56.37/57.87	0/0
[3,2,1,3,4,0,2,2,4,2]	[18, 3, 16, 19, 7, 16, 0, 7, 9, 12]/ [0, 12, 16, 19, 7, 16, 0,	2831.37/ 3083.4	28/30	34.37/83.46	0/0

	18, 9, 12]	6			
[3,1,2,4,4,2,1,3,4,2]	[5 6 2 12 17 16 10 18 4]/ [0, 6, 13, 19, 7, 16, 0, 18, 4, 4]	2858.87/ 3074.01	28/30	55.96/126.73	0/0
[4,2,4,4,2,3,3,4,2,3]	[7, 6, 10, 19, 17, 10, 0, 11, 11, 16]/ [0, 6, 13, 19, 7, 16, 0, 18, 4, 4]	2864.18/ 3076.99	28/30	65.18/76.99	0/0
[2,3,4,0,0,2,2,1,2,3]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [4, 6, 8, 12, 17, 10, 0, 18, 18, 0]	2822.46/ 3043.13	28/30	25.46/43.13	0/0
[1,3,1,1,4,3,3,3,2,0]	[0, 6, 13, 19, 7, 16, 0, 18, 4, 4]/ [0, 3, 16, 6, 7, 16, 0, 18, 4, 12]	2828.29/ 3032.61	28/30	30.29/32.61	0/0
[1,4,2,2,3,2,2,4,1,1]	[0, 3, 2, 19, 7, 16, 0, 7, 13, 12]/ [0, 6, 13, 2, 11, 16, 0, 18, 4, 10]	2842.71/ 3044.90	28/30	43.71/44.90	0/0

TABLE VI: Summary of results of DSS based on Large Dataset - C(500,20,4)

Meta-heuristic	Workload	Arrival rate	Optimal Soln	Optimal Value	Isolation value	No. of Request
HC(Greedy)	2.7 req/sec [Appendix A]	Same arrival rate for all components in all groups	Appendix B	151627.98/ 152295.57	1488/1494	2834.98/ 2895.57
SA(Greedy)	2.7req/sec [Appendix A]	Same arrival rate for all components in all groups	Appendix C	151175.40/ 151865.32	1490/1483	2876.40/ 2863.32

Component Experiencing the Varying Workload: In this scenario, different components in the group experience varying workloads. In the workload file that represents varying arrival rates to each component group, this can also be represented as a vector whose length is 500. The reader can inspect the attached workload file to see the order of the arrival rates. As usual, it is assumed that each workload changes every 60 minutes. The instance file is applied both to HC(Greedy) and SA(Greedy) metaheuristics.

8. DISCUSSION

In this section, the results derived from the evaluation of the decision support system are discussed.

8.1 Degree of Isolation

The isolation value of the DSS can be used to infer the degree of isolation of the systems. For the format used in computing the optimal function, we can know the number of isolation values. Best optimal values imply the highest combined value of isolation for the set of optimal solutions. The worst (minimum value) means the lowest combined value of isolation. This corresponds to a low degree of isolation which corresponds to a shared pattern. It has to be pointed out that this is not necessarily an undesirable solution, from a practical point of view. It simply means that the set of solutions allows resources to be shared more flexibly. This may be a valuable

option for a business that is not interested in the highest degree of isolation. These two options (i.e., low or high degree of isolation) are in some sense at opposite extremes of a spectrum. Between these extremes lies a third option, which translates to specifying an optimal value between a given range.

8.2 Resource Utilization of the Components

The number of requests that can be allowed to access a component is a very important parameter that can be used to study the behaviour of the cloud-hosted system and thus make a decision about the resources or capacity of the components and the resources.

The decision support system provides information about the number of requests that can access a component. This gives an indication of the level of utilization of the components and the system as a whole.

8.3 Performance Interference

The decision about the performance of the system can be inferred from the decision support system. This is important in a situation where there is a limitation in terms of time and resources required to produce a stable and robust optimal solution for deploying components of a large and complex cloud system. Another situation is in a case where there are limitations in the computation machine on which the decision support system is executed, or it is infeasible to obtain the optimal solution.

The number of function evaluations can be used to answer this kind of question: *Assuming there is a limitation in terms of time, what are the optimal solutions that can be provided for deployment to the cloud?*

8.4 Penalty violations

The magnitude of penalty violations by the solutions for violating the constraints is an indication of the quality of the solutions. Solutions with lower penalty values are regarded as better solutions because the solutions had fewer violations of the constraints of the problem. From a practical point of view, it means that the solutions with fewer violations translate to a lower cost of deployment of the components to the cloud.

TABLE VII: Summary of trade-offs, DSS parameters and effect on cloud-hosted service

SN	Trade-offs for considera-tion	DSS parameters	Effect on Cloud-hosted Service
1	tenant isolation versus re-source sharing	Isolation value	If the isolation value from the DSS is low, then it means that the cloud consumer has the ability to share resources.
2	tenant isolation versus the number of users	No. of requests	If the isolation value from the DSS is high [5] then it that the cloud consumer needs more scope of control of the cloud stack.
3	tenant isolation versus customizability	Isolation value	If the isolation value from the DSS is high then it will be more flexible to customize the cloud service
4	tenant isolation versus size of generated data	Isolation value	If the isolation value from the DSS is high then it means it will be more difficult to achieve a high degree of isolation if a large volume of data is generated by the cloud service.
5	tenant isolation versus scope of control	Isolation value	If the isolation value from the DSS is high (on the upper one-third (of the difference between the lowest and highest optimal value) then it that the cloud consumer needs more scope of control of the cloud stack. The reverse is true for a low isolation value.
6	tenant isolation	No. of	If penalty values are more, it

	versus business constraints	requests, Penalty values (security)	means that the component or system is more vulnerable to security breaches.

9. RECOMMENDATIONS AND CONSIDERATIONS FOR IMPLEMENTATION OF DECISION SUPPORT SYSTEM

This section discusses the recommendations and considerations for practitioners (e.g., cloud providers and users) to use the decision support system.

9.1 Optimal Decisions related to Trade-offs for achieving the required degree of tenant isolation

The DSS can be used to make trade-off decisions for consideration when deploying components of cloud-hosted service for guaranteeing multitenancy isolation. Table VII summarises some of the key trade-offs to consider, the parameters to tune on the DSS, and the effect on cloud-hosted service.

9.2 Evaluation of the Optimal Solutions for the deployment of cloud-hosted services

The DSS can be used to evaluate the optimal solutions to be deployed to the cloud. Specifically, the DSS can be used to evaluate the quality of the solutions, the robustness of the solutions, and the estimated computational time required to produce the required optimal solutions for deployment. This is particularly important to both a cloud provider as well as a cloud consumer. For example, in a resource-constrained environment, a cloud consumer would be interested in knowing the computation time (and resources) that would produce a particular optimal solution. The reader can refer to our previous work for details on evaluating key metrics for optimal deployment of services to the cloud.

9.3 Optimal Decisions related to the Size and Nature of Cloud- hosted Service

The size of the cloud-hosted service is also an important consideration when making optimal decisions. This size could be based on the number of existing components (i.e., a small or large number of components), or the number of data generated as a result of executing the cloud-hosted service. In the case of a large cloud-hosted service, the DSS will take on a large input dataset (for example, an instance file consisting of 500 groups of components, 20 components per group and supported by four resource types) together with an accompanying service demand file, and a workload file). Without bias, our DSS is able to produce the optimal value that corresponds to a suitable multitenancy pattern out of the three possible patterns, which also points to the associated degree of tenant isolation. Results presented in Section VII show that the decision support algorithm is scalable and can be applied to small or large decision-making problems.

9.4 Optimal Decisions related to Aspects of Isolation to im- prove in cloud-hosted services

There are considerations for practitioners regarding the na- ture of improvements to make to the cloud-hosted service us- ing the DSS. Previous experimental work shows that the components of cloud-hosted service deployed based on the shared component pattern changed significantly for performance-related parameters (e.g., response times, error%, and throughput), while the dedicated component pattern changed significantly for the system's resource-related parameters (e.g., CPU, memory, and disk I/O) [6]. Therefore, the DSS can be fine-tuned to make recommendations regarding the parameters that support the deployment of components with the shared pattern in order to improve the systems resource consumption and vice versa for the deployment of components with the dedicated pattern to improve the performance of the cloud-hosted service.

9.5 Optimal Decisions requiring multiple criteria

Our decision support system allows practitioners to perform some degree of parameter tuning depending on the required degree of isolation. For example, a cloud customer may want to deploy components experiencing frequent workload changes while at the same time enhancing the sharing of resources between the components (and the whole system). In this case, an optimal value between 0 and one-third of the difference between the lowest and highest optimal value can be specified as an input allowing the algorithm to produce an optimal solution for the required degree of isolation. A slight modification can be made to obtain the optimal solutions when the isolation value or an average number of requests is provided. Also, when a set of optimal solutions is given, it is also possible to use the decision support system to estimate the type of degree of isolation (and hence the type of multitenancy pattern to use) and the average number of requests that can be allowed to access the components or the whole systems.

10. SCOPE AND LIMITATIONS OF THE STUDY

The focus of this work was to provide a framework for developing a decision support system for providing an optimal solution for deploying components of a cloud-hosted application in a way that guarantees the required degree of multitenancy isolation. They can be adopted and implemented in a flexible way to suit the particular deployment environment that is required. The results in the previous work by Ochei et al. [7] focused on an extensive evaluation of the optimization model and a metaheuristic solution to provide a near-optimal solution for deploying components of a cloud-hosted application in a way that guarantees multitenancy isolation. Although the results presented in this paper reflect solutions evaluated by combining optimalDep with the Hill Climbing algorithm and Simulated Annealing algorithm to find an optimal solution to the optimization problem, other metaheuristics (e.g., genetic algorithm, particle swamp optimization) can also be used.

The results presented in this paper are a significant extension to the previous work by Ochei et al. [7] where the focus is on using the data provided by the optimization model to assist decision-makers in analyzing cloud deployment situations to guarantee the required degree of multitenancy isolation. The algorithm that drives the decision analyzer component of the framework has been included in the paper. This is followed by a simple example of the analyzer component provided in the result

section as part of experimentation.

The evaluation of results in our previous work focused on comparing the solutions obtained from our approach with the optimal results obtained first from an exhaustive search of the entire solution space for a small problem and secondly with the target solution. Although our previous work used hill climbing and simulated annealing algorithm for illustra- tion purposes, any recent and advanced meta heuristic (e.g, memetic algorithm, particle swarm Optimization Algorithm) can also be combined with optimalDep to produce optimal solutions as explained and so will not be repeated in this study. This current work focuses on how to further analyse the optimal solutions and other related information produced by the optimalDep algorithm using a decision support system. The paper describes a component which is fed with data obtained from realistic experiments conducted with real cloud- hosted software development tools. We previously conducted separate case studies to empirically evaluate the degree of tenant isolation in three real-life cloud-hosted software devel- opment process tools: continuous integration (with Hudson), version control (with File SCM Plugin and Subversion), and bug/issue tracking (with Bugzilla). The addition of modelling, simulation, and a framework for implementing a decision support system, constitutes a novel contribution to knowledge. [35], [36], [37].

11. CONCLUSION

In this paper, we have presented a framework that can be used to develop a decision support system for users (e.g., cloud deployment architects) to make decisions regarding the optimal deployment of components of a cloud-hosted service for guaranteeing multitenancy isolation. This work contributes to the literature on multitenancy isolation and optimization of the deployment of components of cloud-hosted services.

Multitenancy has become a key concept in the deployment of cloud-based services. The challenge of implementing mul- titenancy is not only how to ensure that there is isolation between multiple components of a cloud-hosted service when one of the components experiences high load, but also to guarantee that the required degree of isolation between tenants is attained [4], [6]. To address this challenge, this paper has presented a novel decision support model algorithm (i.e., optimalDSS), together with a system architecture (i.e., op- timalDssArch) for making decisions regarding the deployment of components of a cloud-hosted service with a guarantee for multitenancy isolation. In addition, a novel algorithm for creating input files (i.e., instance file, service demand file and workload files) for running the decision support system.

The study has revealed after extensive experimental evalu- ation of the framework that it can be used by cloud providers and users to make decisions regarding the optimal deployment of components of a cloud-hosted service. Such decisions include but are not limited to - the required degree of isolation (i.e., lowest degree or highest degree of isolation, or even a set of optimal solutions with a specific isolation value), resource utilization, of components or tenants accessing the system, performance issues (e.g., the best possible (optimal) solution that can be within a limited or specified time interval.

In future, we plan to develop a decision support model for predicting the availability of components of cloud-hosted services based on the required degree of multitenancy iso- lation. Availability can be used to indicate the uptime of a system (or components of a system) over a sufficiently long duration. In a cloud environment, the MTTR of different fault recovery techniques can be explored to compute how long it takes to bring the backup copy up to speed, and hence predict the availability of components. In our study, we applied our approach to optimise the deployment of components of cloud services. We plan to apply our approach to optimise the profit of virtualized cloud data centres (VCDCs) in a way that guarantees varying degrees of isolation between the various applications running in VCDCs. This is similar to the work of [17], [19] where the authors presented an approach to optimize the profit of VCDCs based on the service-level agreements (SLAs) between service providers and customers.

12. REFERENCES

[1] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud Computing Patterns, Springer, London, United Kingdom, 2014.

[2] K. Roche, J. Douglas, Beginning java google app engine, 1st Edition, Apress, New York, United States, 2009.

[3] J. Fiaidhi, I. Bojanova, J. Zhang, L.-J. Zhang, Enforcing multitenancy for cloud computing environments, IT professional (1) (2012) 16–18.

[4] S. Walraven, T. Monheim, E. Truyen, W. Joosen, Towards performance isolation in multi-tenant saas applications, in: Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing, 2012, pp. 1–6.

[5] E. Bauer, R. Adams, Reliability and availability of cloud computing, John Wiley & Sons, 2012.

[6] L. C. Ochei, J. M. Bass, A. Petrovski, Degrees of tenant isolation for cloud-hosted software services: a cross-case analysis, Journal of Cloud Computing 7 (1) (2018) 22.

[7] L. C. Ochei, A. Petrovski, J. M. Bass, Optimal deployment of compo-nents of cloud-hosted application for guaranteeing multitenancy isola-tion, Journal of Cloud Computing 8 (1) (2019) 1.

[8] S. Martello, P. Toth, Knapsack problems: algorithms and computer implementations, John Wiley & Sons, Inc., 1990.

[9] J. Legriel, C. Le Guernic, S. Cotton, O. Maler, Approximating the pareto front of multi-criteria optimization problems, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2010, pp. 69– 83.

[10] M. Pathirage, S. Perera, I. Kumara, S. Weerawarana, A multi-tenant architecture for business process executions, in: 2011 IEEE International Conference on Web Services, IEEE, 2011, pp. 121–128.

[11] H. Cai, N. Wang, M. J. Zhou, A transparent approach of enabling saas multi-tenancy in the cloud, Proceedings - 2010 6th World Congress on Services, Services-1 2010 (2010) 40–47doi:10.1109/SERVICES.2010.48.

[12] J. M. Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, Toward a multi-tenancy authorization

system for cloud services, IEEE Security and Privacy 8 (6) (2010) 48–55. doi:10.1109/MSP.2010.194.

[13] Z. I. M. Yusoh, M. Tang, Composite saas placement and resource optimization in cloud computing using evolutionary algorithms, in: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, IEEE, 2012, pp. 590–597.

[14] F. Shaikh, D. Patil, Multi-tenant e-commerce based on saas model to minimize it cost, in: Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on, IEEE, 2014, pp. 1–4.

[15] D. Westermann, C. Momm, Using software performance curves for dependable and cost-efficient service hosting, in: Proceedings of the 2nd International Workshop on the Quality of Service-Oriented Software Systems, ACM, 2010, p. 3.

[16] D. Candeia, R. A. Santos, R. Lopes, Business-driven long-term capacity planning for saas applications, IEEE Transactions on Cloud Computing 3 (3) (2015) 290–303.

[17] H. Yuan, J. Bi, M. Zhou, Geography-aware task scheduling for profit maximization in distributed green data centers, IEEE Transactions on Cloud Computing.

[18] H. Yuan, J. Bi, W. Tan, B. H. Li, Cawsac: Cost-aware workload scheduling and admission control for distributed cloud data centers, IEEE Transactions on Automation Science and Engineering 13 (2) (2015) 976–985.

[19] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, J. Li, Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center, IEEE Transactions on Automation Science and Engineering 14 (2) (2015) 1172–1184.

[20] M. L. Abbott, M. T. Fisher, The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise, Pearson Education, 2009.

[21] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, S. Dustdar, Moving applications to the cloud: an approach based on application model enrichment, International Journal of Cooperative Information Systems 20 (03) (2011) 307–356.

[22] A. Aldhalaan, D. A. Menascé, Near-optimal allocation of vms from iaas providers by saas providers, in: Cloud and Autonomic Computing (ICCAC), 2015 International Conference on, IEEE, 2015, pp. 228–231.

[23] T. Vanhove, J. Vandensteen, G. Van Seghbroeck, T. Wauters, F. De Turck, Kameleo: Design of a new platform-as-a-service for flexible data management, in: Network Operations and Management Symposium (NOMS), 2014 IEEE, IEEE, 2014, pp. 1–4.

[24] R. Krebs, Performance isolation in multi-tenant applications, Ph.D. thesis, Karlsruhe Institute of Technology (2015).

[25] R. Krebs, M. Loesch, Comparison of request admission based per-formance isolation approaches in multi-tenant saas applications., in: CLOSER, 2014, pp. 433–438.

[26] A. Aldhalaan, D. A. Menascé, Near-optimal allocation of vms form iaas providers by saas providers, Tech. rep., George Mason University (2015).

[27] R. L. Sri, N. Balaji, Speculation based decision support system for efficient resource provisioning in cloud data center, International Journal of Computational Intelligence Systems 10 (1) (2017) 363–374.

[28] V. Andrikopoulos, S. Strauch, F. Leymann, Decision support for ap-plication migration to the cloud, Proceedings of CLOSER 13 (2013) 149–155.

[29] M. Menzel, R. Ranjan, Cloudgenius: decision support for web servercloud migration, in: Proceedings of the 21st international conference on World Wide Web, 2012, pp. 979–988.

[30] J. O. Grady, System engineering planning and enterprise identity, Vol. 7, CRC Press, 1995.

[31] A. T. Bahill, A. M. Madni, et al., Tradeoff decisions in system design, Springer, 2017.

[32] J. E. Beasley, Or-library: distributing test problems by electronic mail, Journal of the operational research society 41 (11) (1990) 1069–1072.

[33] Z. Eckart, L. Marco, Test problems and test data for multi-objective optimizers, [Online: accessed in December, 2018 from https://sop.tik.ee.ethz.ch/download/supplementary/testProblemSuite/].

[34] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud computing patterns: fundamentals to design, build, and manage cloud applications, Springer, 2014.

[35] L. C. Ochei, J. Bass, A. Petrovski (a), Evaluating degrees of multitenancy isolation: A case study of cloud-hosted gsd tools, in: 2015 International Conference on Cloud and Autonomic Computing (ICCAC), IEEE, 2015, pp. 101–112.

[36] L. C. Ochei, A. Petrovski, J. Bass, Evaluating degrees of isolation be-tween tenants enabled by multitenancy patterns for cloud-hosted version control systems (vcs), International Journal of Intelligent Computing Research 6, Issue 3 (2015) 601 – 612.

[37] L. C. Ochei, J. Bass, A. Petrovski (b), Implementing the required degree of multitenancy isolation: A case study of cloud-hosted bug tracking system, in: 13th IEEE International Conference on Services Computing (SCC 2016), IEEE, 2016.

13. APPENDIX A SAMPLE DATASET

The sample dataset files used in the experiments have been made available online[6] to improve readability and re- producibility. The sample files included are - a small problem instance (i.e., C(10, 20,4) and a large problem instance file (i.e., C(500, 20,4). The service demand files and workload files associated with the instance files have also been included.

APPENDIX B - SAMPLE OPTIMAL SOLUTIONS FOR LARGE DATASET (C(500,20,4) WHEN ARRIVAL RATES IN THE WORKLOAD FILE ARE THE SAME. DATASET IS APPLIED ON SA(GREEDY) ALGORITHM

Best optimal solution for workload with an arrival rate of 2.7 per second applied to a large dataset defined as C(500,20,4) is shown below. The specification of this dataset means that there are 500 groups of components, 20 components per group and supported by 4 resource types -CPU, memory, disk space and bandwidth. The array shown below which contains 500 elements represents an optimal solution. The array is automatically created by the decision support system by selecting one component from each of the 500 groups for deployment in a way that meets the resource constraints of the system and maximises the optimal function G.

[2, 8, 7, 11, 11, 4, 17, 9, 8, 9, 13, 0, 12, 8, 3, 10, 16, 16, 9, 5, 14, 14, 7, 10, 17, 5, 9, 14, 0, 16, 17, 11, 12, 15, 8, 11, 15, 8, 9, 18, 8, 8, 19, 14, 15, 0, 15, 18, 5, 0, 8, 6, 17, 19, 18, 12,14, 0, 1, 3, 14, 15, 6, 11, 0, 13, 18, 6, 17, 19, 6, 13, 10, 9,19, 17, 12, 11, 11, 13, 3, 8, 13, 9, 18, 19, 7, 13, 17, 16, 9,15, 12, 1, 18, 5, 7, 8, 18, 13, 7, 7, 1, 1, 11, 10, 18, 4, 13, 18,19, 19, 13, 1, 9, 1, 18, 8, 9, 8, 16, 1, 17, 1, 5, 12, 12, 4, 17,11, 5, 1, 19, 15, 2, 8, 12, 14, 5, 6, 19, 13, 18, 17, 19, 14, 15,1, 4, 12, 2, 6, 19, 8, 19, 16, 13, 16, 3, 7, 19, 6, 18, 18, 19,17, 17, 5, 16, 18, 16, 19, 14, 14, 5, 10, 13, 13, 6, 15, 10, 16,19, 9, 8, 6, 13, 18, 12, 9, 1, 11, 17, 7, 3, 0, 10, 6, 6, 14, 13,3, 13, 7, 17, 9, 10, 19, 2, 19, 11, 3, 16, 5, 5, 2, 17, 7, 3, 1,3, 12, 2, 17, 15, 11, 12, 17, 11, 8, 17, 1, 5, 11, 12, 13, 5, 15,10, 17, 4, 6, 18, 7, 10, 19, 17, 10, 3, 8, 11, 6, 0, 18, 16, 6,14, 3, 5, 2, 0, 13, 13, 10, 12, 3, 7, 15, 1, 15, 14, 9, 5, 0, 10,8, 14, 17, 18, 1, 19, 10, 9, 4, 19, 16, 10, 9, 15, 5, 16, 4, 6, 0,6, 15, 8, 14, 17, 16, 14, 1, 18, 2, 2, 4, 18, 17, 1, 7, 18, 8, 11,11, 5, 8, 3, 6, 19, 16, 8, 17, 12, 10, 7, 13, 15, 1, 2, 12, 9, 3,17, 3, 15, 16, 16, 3, 14, 17, 7, 1, 17, 10, 10, 12, 0, 9, 10, 13,8, 10, 7, 3, 12, 19, 14, 2, 10, 0, 13, 4, 18, 3, 7, 9, 15, 16, 3,9, 18, 6, 5, 13, 18, 3, 19, 6, 3, 12, 9, 19, 10, 7, 11, 9, 13, 4,17, 13, 3, 8, 11, 8, 17, 13, 6, 4, 18, 1, 3, 0, 4, 3, 12, 11, 1,16, 5, 12, 15, 9, 11, 16, 6, 12, 9, 19, 7, 5, 3, 5, 11, 2, 17, 17,14, 10, 9, 17, 7, 7, 14, 2, 17, 16, 14, 10, 7, 10, 5, 7, 5, 14,12, 11, 18, 16, 18, 5, 13, 18, 19, 13, 18, 9, 13, 2, 6, 4, 17,13, 17, 8, 14, 9, 17, 13, 19, 8, 12, 9, 16, 6, 7, 19, 11, 3, 4,13, 0, 15, 16, 18, 12, 1, 4, 2, 6, 4, 10, 4, 6, 12, 16, 0, 12, 1,13, 7]

APPENDIX C

SAMPLE OPTIMAL SOLUTIONS FOR LARGE DATASET (C(500,20,4) WHEN ARRIVAL RATES IN THE WORKLOAD FILE ARE THE SAME. DATASET IS APPLIED ON SA(GREEDY) ALGORITHM

Worst optimal solution for workload with an arrival rate of 2.7 per second applied to a large dataset defined as C(500,20,4) is shown below. The specification of this dataset means that there are 500 groups of components, 20 components per group and supported by 4 resource types -CPU, memory, disk space and bandwidth. The array shown below which contains 500 elements represents an optimal solution. The array is automatically created by the decision support system by selecting one component from each of the 500 groups for deployment in a way that meets the resource constraints of the system and maximises the optimal function G.

[8, 8, 7, 11, 0, 4, 17, 9, 8, 9, 13, 0, 12, 8, 3, 10, 16, 16, 9, 5, 14, 14, 7, 17, 17, 5, 9, 14, 0, 16, 17, 11, 12, 15, 8, 11, 15, 8,9, 18, 8, 8, 19, 14, 15, 0, 13, 18, 5, 0, 8, 6, 17, 19, 18, 12,14, 0, 10, 3, 14, 15, 6, 11, 0, 13, 18, 6, 17, 19, 6, 13, 10, 9,19, 17, 12, 11, 11, 13, 3, 8, 13, 9, 7, 19, 7, 13, 17, 16, 9, 15,12, 1, 18, 5, 7, 8, 18, 13, 12, 2, 1, 1, 11, 3, 15, 4, 13, 18, 19,19, 13, 1, 9, 1, 18, 8, 14, 8, 16, 1, 17, 1, 5, 12, 7, 4, 17, 11,5, 1, 19, 15, 2, 8, 12, 14, 5, 6, 19, 13, 18, 17, 19, 14, 15, 1,4, 12, 2, 6, 19, 8, 19, 16, 13, 16, 3, 7, 19, 6, 18, 18, 19, 17,17, 5, 16, 18, 16, 19, 14, 14, 5, 10, 13, 13, 6, 15, 10, 16, 19,9, 8, 6, 12, 18, 12, 18, 1, 11, 17, 7, 3, 0, 18, 6, 6, 14, 13, 3,13, 7, 17, 9, 10, 19, 2, 5, 11, 3, 16, 5, 5, 2, 17, 7, 3, 1, 3, 12,2, 17, 15, 11, 12, 17, 11, 8, 2, 1, 5, 11, 3, 13, 19, 15, 10, 17,4, 6, 18, 7, 10, 19, 17, 10, 3, 8, 11, 6, 0, 3, 16, 6, 14, 3, 5, 2,0, 13, 13, 10, 12, 3, 7, 15, 18, 15, 14, 9, 5, 0, 10, 8, 14, 17,5, 1, 19, 10, 9, 12, 19, 16, 10, 9, 15, 13, 10, 4, 6, 0, 6, 15,8, 14, 12, 16, 15, 1, 18, 2, 2, 4, 13, 17, 11, 7, 18, 8, 11, 11,5, 8, 3, 11, 19, 16, 8, 9, 12, 10, 7, 13, 15, 1, 2, 12, 9, 3, 17,3, 15, 16, 16, 10, 14, 17, 7, 1, 17, 10, 10, 12, 0, 9, 10, 13, 8,10, 7, 3, 12, 15, 14, 2, 10, 0, 13, 4, 18, 3, 7, 9, 15, 16, 3, 13,18, 6, 5, 13, 18, 3, 19, 6, 3, 12, 9, 19, 10, 7, 11, 9, 13, 4, 17,13, 3, 8, 11, 8, 17, 13, 6, 4, 18, 1, 3, 0, 4, 3, 12, 11, 1, 16,5, 12, 15, 10, 11, 16, 6, 8, 9, 13, 7, 5, 3, 5, 11, 2, 17, 17, 14,17, 9, 17, 7, 7, 14, 2, 17, 16, 14, 10, 7, 10, 5, 7, 5, 14, 12,11, 18, 16, 18, 5, 13, 18, 19, 13, 18, 8, 13, 2, 6, 4, 17, 9, 17,8, 14, 9, 17, 13, 19, 6, 12, 9, 16, 6, 7, 19, 11, 3, 0, 13, 0, 15,16, 18, 12, 1, 4, 2, 6, 4, 15, 4, 6, 4, 16, 0, 12, 1, 13, 7]

Rotimi Ogunsakin holds an MSc in advanced computing from the school of computing at the University of Manchester, and a PhD in Information Systems from the Alliance Manchester Business School. His career journey has been in research and innovation at the intersection of artificial intelligence and business. Precisely in areas where changes are frequent and the nature of these changes, to a large extent, are unpredictable –

such as critical systems and businesses and technologies operating in un- predictable environments. Also, he has published research papers in peer-reviewed international conferences and journals.

Laud Ochei holds a PhD in Computing from Robert Gordon University, Aberdeen (UK). He has a broad range of research and software development expe- rience in various academic and industry collabora- tions. His research interests are in software engineer- ing, distributed systems, the internet of things, and cloud application architectures. He is also interested in developing novel approaches for deploying cloud-hosted services to guarantee multitenancy isolation. He has published several research papers in peer- reviewed international conferences and journals.

Nemitari Ajienka holds a doctorate in computer science from Brunel University London (UK) where he was a member of the Brunel Software Engineer- ing Lab (BSEL). He is currently a senior lecturer in computer science at Nottingham Trent University (UK) and a member of the computing and informat-ics research centre within the school of science and technology