Dynamic Service Assignment based on Proportional Ordering for the Adaptive Resource Management of Cloud Systems

Romeo Mark A. Mateo and Jaewan Lee
School of Electronics and Information Engineering, Kunsan National University
Daehak-Ro 558, Gunsan, Jeonbuk, South Korea 573-701
[e-mail: {rmmateo, jwlee}@kunsan.ac.kr]
*Corresponding author: Jaewan Lee

Received April 8, 2011; revised July 9, 2011; revised September 8, 2011; accepted November 19, 2011; published December 31, 2011

Abstract

The key issue in providing fast and reliable access on cloud services is the effective management of resources in a cloud system. However, the high variation in cloud service access rates affects the system performance considerably when there are no default routines to handle this type of occurrence. Adaptive techniques are used in resource management to support robust systems and maintain well-balanced loads within the servers. This paper presents an adaptive resource management for cloud systems which supports the integration of intelligent methods to promote quality of service (QoS) in provisioning of cloud services. A technique of dynamically assigning cloud services to a group of cloud servers is proposed for the adaptive resource management. Initially, cloud services are collected based on the excess cloud services load and then these are deployed to the assigned cloud servers. The assignment function uses the proposed proportional ordering which efficiently assigns cloud services based on its resource consumption. The difference in resource consumption rate in all nodes is analyzed periodically which decides the execution of service assignment. Performance evaluation showed that the proposed dynamic service assignment (DSA) performed best in throughput performance compared to other resource allocation algorithms.

***Keywords*: Cloud computing, resource allocation, migration, load balancing, adaptive system**

This research is supported by Industry and University Research Consortium of Small and Medium Business Administration (SMBA) of 2011.

DOI: 10.3837/ tiis.2011.12.002

1. Introduction

The Cloud has emerged recently as a new computing paradigm for provisioning of application services and computational resources via the Internet, in which, it offers an efficient way to share resources within organizations and promote business strategy. Application solutions in the Cloud, social network services [1] and cloud infrastructures [2] are popularly used as tools by business establishments to provide complete applications for employees and to improve collaboration with their business partners. Small business can also benefit from using cloud computing by not having the need of deploying physical infrastructure like file servers, e-mail servers, storage systems or computational resources. Currently, various application services and development trends in the Cloud are rapidly increasing [3], at the same time, improvements in the system infrastructure follow [4][5]. The first successful resource provider based on utility computing is the Amazon Elastic Compute Cloud (Amazon EC2) [2]. In response to the highly competitive trend, Google and Microsoft introduced the Google App Engine [6] and Microsoft Azure Services [7], respectively, which allowed application developers to program advance applications from existing provisioned web services. Research projects in [8][9] focus on large implementation of cloud systems. These and other cloud systems are required to be highly reliable, scalable, and autonomic to support ubiquitous access and dynamic discovery in able to operate successfully.

Services offered in the Cloud are expected to meet quality of service (QoS) requirements of customers. In the system architect view, the proper interaction from components of a cloud system is considered in providing QoS. Although there are various cloud architectures that demonstrate QoS, these designs share the same goal of virtualization, which is to provide users with simple access and reliable connection to resources or services. In [10][11], variables and methods for virtualization are studied which can be used to provide an efficient resource allocation. Resource management performs a significant role in hiding complex interaction of a distributed system while resources or services are transparently provisioned. Moreover, adaptive schemes in resource management are introduced by researches to improve resource utilization [12][13]. Intelligent distributed frameworks are proposed in [14][15] to integrate a large and complex distributed system with intelligent algorithms to be aware of service demand patterns and to determine the need for replication of services that leads to a responsive system [15]. The information from service demand patterns can be used to adjust the resource allocation of cloud servers by predicting the trends of client demands for services and producing more services to meet future demands. Therefore, the motivation of this study is the integration of intelligent algorithms in a cloud system to promote adaptive techniques in providing the QoS.

This paper presents a cloud system which utilizes intelligent methods in provisioning of software services and managing server resources. The proposed cloud architecture uses a SaaS model in delivering its cloud application services. The objectives of the proposed cloud architecture are as follows.

- To provide automation of service and resource provisioning using agents.
- To provide a classification method for search of cloud services and for efficient collaboration of cloud service providers.
- To provide adaptive schemes in the resource management of cloud servers.

In order to provide efficient resource allocation for responsive services, a dynamic service assignment is proposed. The proposed algorithm, which uses proportional ordering, handles the loads of cloud servers by assigning cloud services to cloud servers considering the equal resource consumption in all cloud servers. The service assignment is executed based on the variation of resource consumption rates in all nodes. Similar to a migration technique, the proposed algorithm relocates application services to the selected cloud servers. However, the service assignment is triggered by a periodic analysis of resource consumption rate variation, where the execution is not as frequent as in a migration scheme. To verify the efficiency of the proposed algorithm, conventional resource allocation techniques were compared by performances in message latencies and turn-around time of requests.

2. Related Work

2.1 Software-as-a-Service

Software-as-a-Service (SaaS) is a shift from software products to services [16], and nowadays, is popularly used to support business operations. SaaS is a delivery model for software applications using Internet as its medium which is designed for flexibility to both providers and customers [17]. On the customer side, SaaS eliminates complex configurations and time-consuming installations. A cloud service consumer only needs, at least, a standard web browser to access software applications. On the provider side, SaaS enables reuse of software applications and supports many clients using a common infrastructure. A study in [18] shows that open, modulated, and standardized software takes a significant market share compared to the traditional commercial off-the-shelf (COTS) solutions. With SaaS, providers can easily collect detailed information about defects, performance and usage trends to improve their services. Well-known research topics in SaaS are negotiations [19][20], search method [21] and load balancing [22][23]. Providing optimal negotiations in the Cloud promotes effective service provisioning. Developers and researchers are continuing to introduce more solutions for business such as using social networks for customer relationship management (CRM) [1], and extending their application solutions in the Platform-as-a-Service (PaaS) model. PaaS provides a complete development and hosting platform for applications delivered as a service like in [6][7] which enables the service providers or business owners to develop and implement their own solutions. Similar to SaaS, PaaS virtualizes the available resources to run in the Cloud. Consequently, the provisioning of software applications is dependent on the interactions of components in the resource management. Thus, the techniques in resource management for efficient and robust access to services are considered to provide QoS. Job requests will be delayed if there are many clients accessing a service in a cloud server. Dynamic replication of services is proposed in [22] to support the increase of service demands from cloud consumers. However, the replication scheme does not consider high variation of loads in cloud servers from a case where some nodes have very high access rate because of popular services and, in contrast, some nodes are not utilized because of less frequent accessed services.

2.2 Resource Allocation and Migration

Resource allocation and task management are key factors in providing a robust service system which are well researched topics of distributed systems. In [24], the measurements and system variables are defined that affect the system robustness in resource allocation. The values of task and system parameters are analyzed in [24] to provide a suitable design for resource allocation. In [25], the issue of assigning service tasks to the most appropriate nodes is studied. An optimal formulation solves the task assignment problem in [25] which is a two-phase approach. In [26], the advantages and drawbacks of using a simple approach which is a divisible load theory (DLT) and an optimal approach which is a genetic algorithm (GA) for the resource partitioning problem are discussed. Equal resource sharing is also considered by researches to efficiently distribute the loads in multiple resources. An equal sharing of loads in a group of object service providers offers a fast response to requests [27]. In [28], a peer assignment scheme is proposed to support equal sharing of loads in a P2P system which implements fairness in resource utilization among the participating servers. However, equal sharing of server loads in SaaS is hard to achieve when applications throughout the servers have high variation of access rate.

Migration methods are commonly used to transfer a process on a busy server to a less loaded server. Most of migration algorithms aim for transparency wherein the migration process does not significantly affect the system activities. In [29][30], mobile agents are used in load balancing techniques of a distributed system. The migration method is also effective in video-on-demand (VOD) systems [31]. However, the overhead in performing migration is the main drawback of a migration technique. The overheads in migration process, which are the additional network latency and increase of complexity and state, are discussed in [32]. A dynamic deployment and relocation of virtual machines is presented in [9] for the load balancing scheme but the measures and parameters to select the appropriate servers to migrate are not discussed. In this paper, the adaptive resource management uses a service assignment which considers equal resource consumption in a group of servers and migration overheads when performing the service migration.

3. Cloud System based on Adaptive Resource Management using a Dynamic Service Assignment

The proposed cloud system supports the integration of intelligent methods in the components of service provisioning and resource management. **Fig. 1** shows the proposed cloud architecture which is layered into two parts; cloud service layer and resource management layer. The basic components of a Cloud which are IT consumers, cloud service providers and resources are abstracted by the layered design. We refer a cloud service provider as the owner of cloud services hosted in a cloud server and a cloud service provider can have several cloud servers. Brokering of cloud services and virtualization of resources are supported by the proposed cloud system. In this paper, service provisioning is the automation of functions to acquire and deploy cloud application services like initializing, metering the use, finding, grouping, etc. These functions are mostly implemented by the components in the cloud service layer. The cloud service layer is composed of components that mainly implement the provisioning of cloud application services. A user is provided with a user agent to search for cloud services where the search engine is configured by users based on their preferences. A cloud service is defined as a service that is used by IT consumers for a certain application which, in our proposed cloud system, is classified as SaaS and PaaS. Also, a cloud service can be composed of several cloud services. Cloud service providers can have their own resources to host their services or rent resources. After a user agent finds the appropriate resource or service, the security service will process authentication before a user can access the resource or service. The resource management layer handles resource virtualization and dynamic service assignment in cloud servers. The local hardware of a cloud server is managed by a local resource manager (LRM). An independent resource provider has limited resources but this can be extended by the dynamic resource sharing of the proposed cloud system. A group of resource providers or data centers based on utility computing provides more storage and resources. The following describes the components of the proposed cloud architecture.

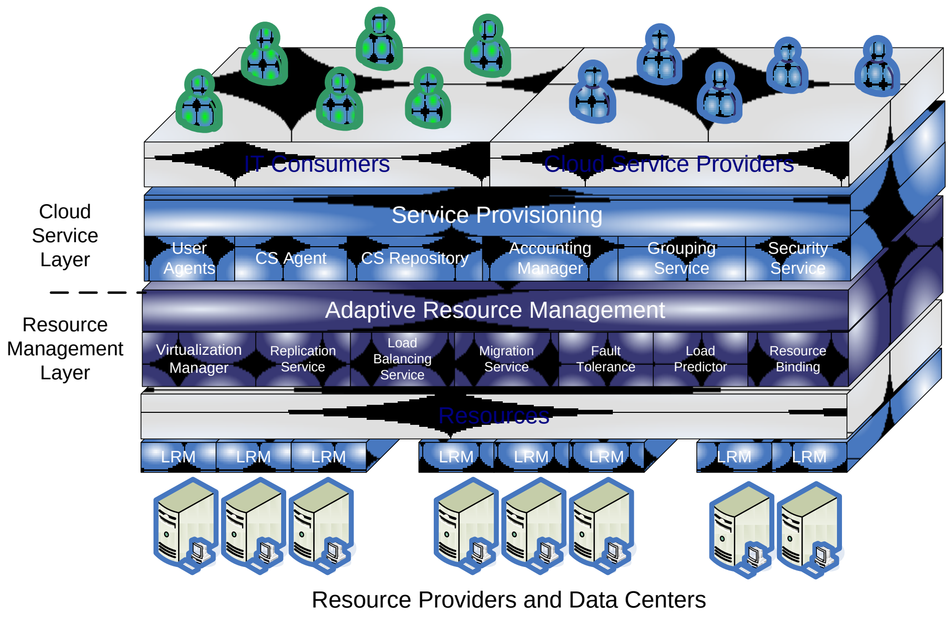


Fig. 1. The proposed cloud architecture separated into two layers. Most of the components are integrated with intelligent methods for service provisioning and adaptive resource management.

Cloud Service Layer

- User agent – used by a user to search for cloud services for a specified task. It communicates with CS agents to access and use cloud services.
- Cloud service (CS) agent – cloud service providers use this agent to interact with users, other cloud service providers, accounting manager, grouping service and virtualization manager.
- Cloud service (CS) repository – stores information of cloud services. Cloud services are registered to this repository that is accessed by a user agent to inquire for cloud services.
- Accounting manager – handles accounting procedures in using cloud services and resources. This is used by CS agents to meter resources and services usage.
- Grouping service – groups the cloud service providers according to cloud service properties.
- Security service – processes authentication and authorization in accessing cloud services or resources. This service interacts with user agents and CS agents to perform security procedures.

Resource Management Layer

- Virtualization manager (VM) – manages interactions from cloud users, service providers and resources.
- Replication service – handles replication of cloud services. VM analyzes current trends of cloud service access while replication service executes replication based on VM's analysis.
- Load balancing service – analyzes loads from cloud servers by gathering load information in every LRM.
- Migration service – migrates cloud services based on the analysis of load balancing

service. The VM also interacts with migration service to perform service assignment.

- Fault tolerance service – handles faults and disconnections of the cloud system. It also informs VM about the faulty cloud servers to replicate or relocate cloud services to other cloud servers.
- Load predictor – predicts system load trends to trigger the process of finding additional resources. VM interacts with load predictor to negotiate resource sharing.
- Resource binding – provides a method of binding resources for scalable resource sharing. The cloud service provider interacts with resource binding after joinning a group of cloud service providers.
- Local resource manager (LRM) –handles local resource management of a single cloud server.

In searching cloud services, a cloud user uses its agent to interact with cloud service providers. It is necessary to provide the right services to clients and this is considered by integrating user agents with intelligent algorithms. Some research studies utilized user profiles to filter and select resources in the Web as in [33]. This method can support information awareness of users in the cloud environment to promote user-centric cloud design [34] which is also suitable for collaboration methods. The proposed architecture integrates an intelligent algorithm which utilizes user preferences in selecting best cloud service for a request. **Fig. 2** shows the interaction of user agent to find cloud services.

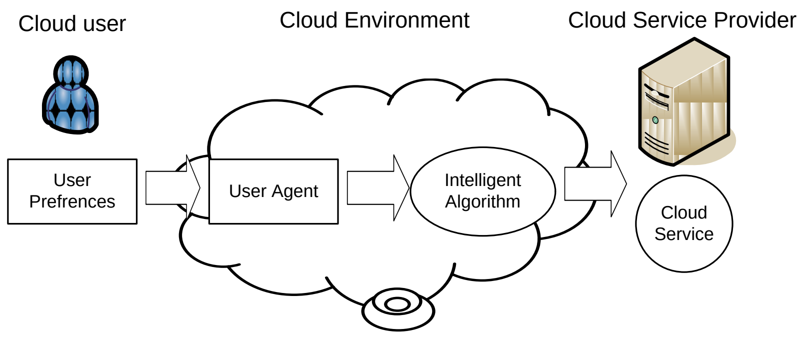


Fig. 2. The interaction of cloud user in searching appropriate cloud service using user preferences.

The interaction of finding a cloud service is illustrated in **Fig. 2**. Cloud users perform search for a cloud service using CS repositories and other tools in the Cloud wherein the complex interactions of components to execute the search are hidden. Grouping the cloud service providers offers more resources and promotes effective collaboration. The scalable grouping of objects in our previous work in [27] groups more objects for service availability and is supported by a load balancing technique to forward the incoming requests. A grouping method is also used in the proposed cloud system for scalable resources where it groups similar or related cloud service providers based on their service properties. Moreover, grouping cloud service providers supports PaaS where solutions are extracted on different nodes of a group. A solution of a specific application is a stack of services which can be on different nodes. A cloud user does not need to know where the services are located to acquire a solution. **Fig. 3** illustrates a solution composing of different services from node A, node B and node C. The complex methods of resource allocation in **Fig. 2** and **Fig. 3** are hidden from cloud users. The complex interactions are implemented in the adaptive resource management of the proposed cloud system and the details of integrating a dynamic assignment of cloud services is tackled by this paper. The dynamic service assignment uses the CPU utilization and access frequency of a cloud service as parameters for the resources consumption rate. These are used as load information to assign the cloud services within the cloud servers or nodes.

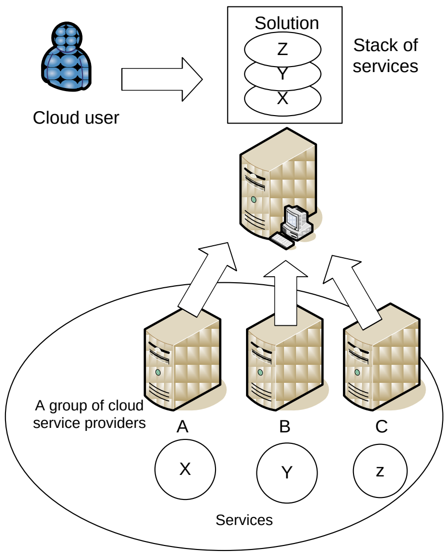


Fig. 3. Platform-as-a-Service model of the proposed cloud system where a platform or a solution is a stack of several services from different nodes in a group.

Typically, traditional load balancing approaches use current processes or queued jobs from nodes as loads in deciding the task forwarding and do not consider load prediction because of the complexity and overheads from forwarding procedure. These concerns are handled by the proposed method. The procedure of dynamic service assignment is summarized:

1. Calculate the mean resource consumption of all nodes.
2. Collect the cloud services based on the excess cloud services load of a node which are candidates for deployment.
3. Rank the cloud services and nodes based on the proportional ordering.
4. Deploy the cloud services to nodes using the assignment function.
5. Calculate the average difference of resource consumption rate in all nodes which will be compared to a threshold value of resource consumption rate variation. If the current resource consumption rate is higher than the threshold, then execute step 1.

3.1 Collecting Cloud Services for Deployment

In this paper, a cloud service provider has a single cloud server to host its cloud services. A singular view of accessing resources is achieved after a cloud service provider joins a group of cloud service providers. In the collection function, cloud services from a grouped cloud servers will be collected based on the *excess cloud services load*. The goal of the collection function is to select the cloud services for service assignment considering equal resource consumption from cloud servers. The *excess cloud services load* represents overload of a node which is the difference from current resource consumption of a cloud server and average resource consumption of all cloud server. v represents resource consumption of a cloud service, which is the processor consumption, and f represents access frequency of a cloud service. The value of v has a range of [0,100] from its equivalent percent value, e.g., 20 percent or 0.2 of CPU usage is v=20. The f is gathered based on the number of services accessed in a period of time represented by $f = count(F, pa)$ where $F=\{f_1,\ldots, f_s\}$, f_s is an access count in a specific time frame of F, and $pa = [pa_i, pa_f]$, pa_i is initial period and pa_f is ending period.

$$s = vf \qquad (1)$$

$$rc = \sum_{i=1}^{I} s_i \qquad (2)$$

The function *count()* will only include f_s that is specified in *pa*. In Equation 1, *s* is resource consumption rate of a cloud service where resource consumption (*v*) is multiplied by access frequency (*f*), and Equation 2 totals the resource consumption rate of all cloud services (*rc*) in node *n* where *I* is total number of services and *i* is service index. In **Table 1**, sample values for each s_i are given. The *rc* in each cloud server is used to calculate the mean resource consumption rate in all cloud servers (μ_{rc}).

Table 1. Resource consumption rate of services in each cloud server and mean resource consumption rate from all cloud servers. Each resource consumption rate of a service is calculated by $(v^*f)=s$.

CSP 1	CSP 2	CSP 3	CSP 4	Average
S1: (40.5x1000)=**40500**	S6: (59x1000)=**59000**	S9: (57x1000)=**57000**	S13: (30x100)=**3000**	
S2: (20x100)=**2000**	S7: (7.55x1000)=**7550**	S10: (35x1000)=**35000**	S14: (25.5x100)=**2550**	
S3: (10x100)=**1000**	S8: (90x100)=**9000**	S11: (9x100)=**900**	S15: (15x100)=**1500**	
S4: (7.5x100)=**750**		S12: (43x100)=**4300**	S2: (20x100)=**2000**	
S5: (7x100)=**700**			S3: (10x100)=**1000**	
44950	**75550**	**97200**	**10050**	**56937.5**

Equation 3 calculates μ_{rc} of nodes where *N* is the number of cloud service providers (CSP) and *n* is index of CSP. A value of 56937.5 is determined in processing Equation 3 using data in **Table 1**. Equation 3 is used to determine the *excess cloud services load* (Δ_n) from a single node *n*. In Equation 4, the calculation of Δ_n is shown where μ_{rc} is subtracted to rc_n. The Δ_n is used to collect cloud services in Equation 5 for the service assignment.

$$\mu_{rc} = \frac{1}{N}\sum_{n=1}^{N} rc_n \qquad (3)$$

$$\Delta_n = rc_n - \mu_{rc} \qquad (4)$$

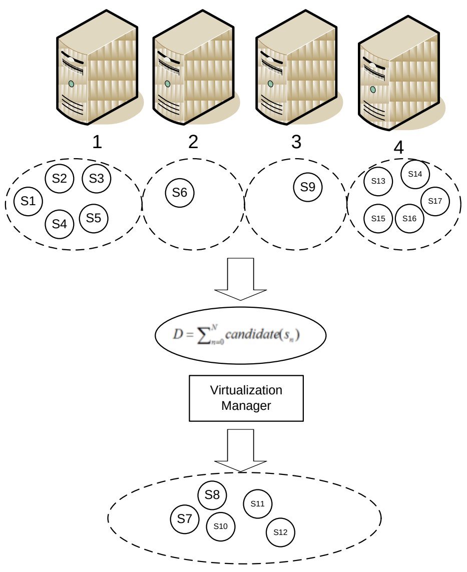
$$D = \sum_{i=0}^{I} candidate\ (s_i) \qquad (5)$$

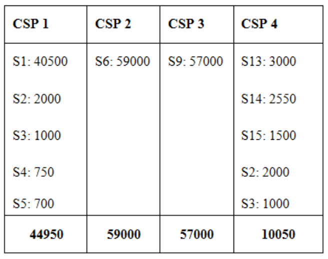
In Equation 5, *D* collects candidate cloud services in each node. If $rc_n < \mu_{rc}$ then the node is skipped by the collection because that node has already enough loads. Equation 5 uses the procedures of combination selection in Equations 6 and 7. The power set of cloud services $S=\{s_1, s_2, \ldots, s_i\}$ of CSP *n* is determined by $P(S)= \{(s_1)_1,(s_1,s_2)_2,(s_1,s_2,s_3)_3, \ldots, S_{kj}\}$ to generate the combination sets, where S_{kj} is a subset of $P(S)$, *k* is the number of items in a set and *j* is index of each subset. The value of *k*-combination set is the sum of *s* values in S_{kj} and this is denoted by C_j. The value of C_j is the total resource consumption rate of all combined cloud services. The complexity of this procedure is determined by adding the values from combination function in each element of *S*, $\{C(n, 1)+C(n, 2)+\ldots +C(n, r)|n=I \text{ and } r=k\}$, where time complexity is increased more than twice in every increment.

$$current\_combination = |C_j - \Delta_n| \qquad (6)$$

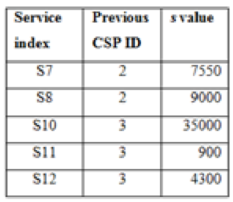
$$\begin{array}{l} if\ \ current\_combination < selected\_combination \\ then\ \ selected\_combination = current\_combination \end{array} \qquad (7)$$

In Equation 6, the difference of C_j and Δ is the value of current combination that will be compared to a selected combination. The selected combination represented by *selected_combination* is the S_{kj} that has the least combination value determined from previous selection. The *current_combination* represents the current S_{kj} that is used to compare to *selected_combination*. Equation 7 is a decision of selecting *current_combination* to be the new *selected_combination* value. After selecting the combination set, services in S_{kj} will be candidates for deployment. **Fig. 4** illustrates the collection of cloud services based on Δ_n determined in Equation 5. At right, the lists of current *rc* of nodes and collected *s* in *D* are shown.





CSP 1	CSP 2	CSP 3	CSP 4
S1: 40500 S2: 2000 S3: 1000 S4: 750 S5: 700	S6: 59000	S9: 57000	S13: 3000 S14: 2550 S15: 1500 S2: 2000 S3: 1000
44950	**59000**	**57000**	**10050**



Service index	Previous CSP ID	s value
S7	2	7550
S8	2	9000
S10	3	35000
S11	3	900
S12	3	4300

Fig. 4. Illustration of the cloud services collection (left), and list of resource consumption rates of services in each nodes (right top) and list of collected services (right bottom).

All nodes process the collection of cloud service in Equation 5 based on Δ. Also, service replicas, e.g. S3, are gathered in the collection function and these are handled in the assignment function of proportional ordering. After candidate cloud services are determined, labeled as $s_{\Delta i}$, cloud servers are selected to distribute the cloud services in *D* using proportional ordering.

3.2 Proportional Ordering and Deploying of Cloud Services to Nodes

The proportional ordering is defined as the ordering of sets of collections, services and resources to proportionate the resource consumption of services to the capacity or availability of resources in nodes. The current resource consumption rate of a node and the resource consumption rate of a service are the parameters used for proportional ordering. The approach of service assignment is different from [26] where the proposed method is executed before the cloud services are actually accessed by cloud consumers. The selection of cloud servers for the assignment function uses the current resource consumption rate for ranking.

$$R = \{rankLH(rc_{\Delta 1}, rc_{\Delta 2}, ..., rc_{\Delta n})\} \tag{8}$$

$$S = \{rankHL(s_{\Delta 1}, s_{\Delta 2}, ..., s_{\Delta i})\} \tag{9}$$

In the first run, all cloud servers are ranked from lowest to highest based on $rc_{\Delta n}$ values shown in Equation 8. $rc_{\Delta n}$ is the current resource consumption rate of a cloud server after processing the cloud service collection. On the other hand, the services are ranked from highest to lowest resource consumption rate in Equation 9. $s_{\Delta i}$ is the resource consumption rate of a cloud service from *D*. After ranking, the efficient assignment of cloud services is necessary. All cloud servers should have approximately equal values of resource consumption rate after deploying cloud services and this is considered by procedures in Equations 10 to 11.

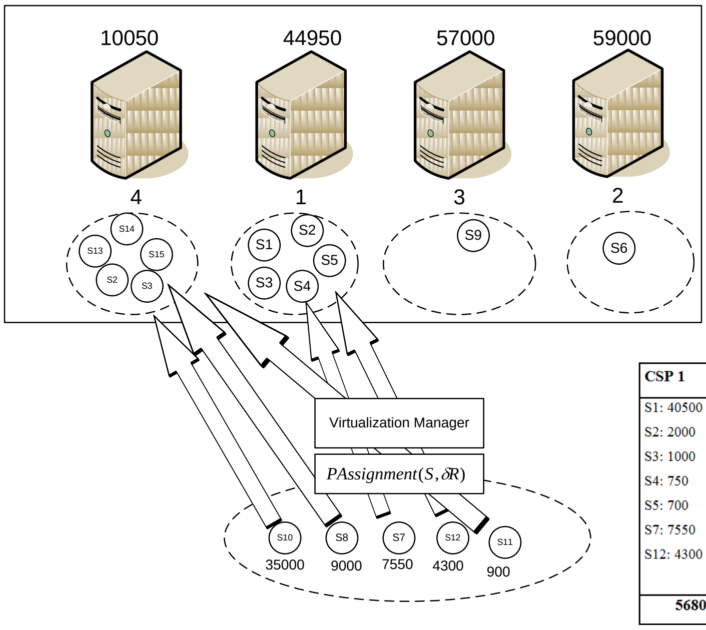
$$\delta_n = \mu_{rc} - rc_{\Delta n} \tag{10}$$

The current resource consumption rate of a cloud server *n* is subtracted from the mean resource consumption rate to get the value of δ_n in Equation 10. δ_n is used to select the collected cloud services in Equation 11 for deployment. If $rc_{\Delta n} > \mu_{rc}$ then the cloud server is skipped by the procedure because $rc_{\Delta n}$ is already high.

$$PAssignment(S, \delta R) = \{rc_{\Delta 1}(s_{\Delta\delta}), rc_{\Delta 2}(s_{\Delta\delta}), ..., rc_{\Delta n}(s_{\Delta\delta})\} \tag{11}$$

PAssignment in Equation 11 is the assignment function which assigns *S* to *R* based on δ_n. In the assignment function, $s_{\Delta\delta} = \{s_{\Delta 1}, s_{\Delta 2}, \ldots, s_{\Delta n}\}$ is a collection of $s_{\Delta i}$, which will be deployed to cloud server *n*. Cloud services in *S* are assigned to a cloud server in chronological order until the total value in $s_{\Delta\delta}$ is approximately equal to δ_n. The $s_{\Delta i}$ will be skipped if the total resource consumption of the assigned cloud services is greater than δ_n, {do $s_{\Delta\delta}= s_{\Delta 1}+s_{\Delta 2}+\ldots+s_{\Delta i}$ | skip $s_{\Delta i}$ if $s_{\Delta\delta}>\delta_n$ }. After the procedure reached the last index of *S*, the assignment function is processed to the next cloud server. If the procedure reached the last index of *R* and there are still $s_{\Delta i}$ needed to be assigned then the procedure recalculates the ranks of nodes and service assignment will start at the first index of new *R*. A tolerance value is added in δ_n which is the average resource consumption of the remaining cloud services ($\mu_{s\Delta}$) in *S* ($\delta_{n\Delta}=\delta_n+ \mu_{s\Delta}$) to have a high probability that all $s_{\Delta i}$ will be assigned on the next procedure. Same procedures are repeated until *S* is empty. An assignment of cloud service is skipped if the cloud server already has a same or replicated cloud service, and thus, will be assigned to the next node. After the assignment, the system performs the actual deployment. **Fig. 5** illustrates the service assignment based on proportional ordering and the list inside of **Fig. 5** shows the final values of service assignment. The figure shows that S10, S8 and S11 are assigned to CSP 4 while S7 and S12 are assigned to CSP 1. CSP 2 and 3 have only one cloud service, however, these services are frequently accessed by cloud consumers.

The algorithm tries to equally distribute cloud services to minimize the difference value of resource consumption rate in each cloud server since S6 and S7 cannot be reassigned. The cloud servers or cloud service providers are ordered from lowest to highest and the cloud services are ordered from highest to lowest before performing the proposed service assignment where cloud services are distributed efficiently to have equal resource consumption in all cloud servers. A round robin technique is used to distribute client requests in replicas and similar services. Also, whenever a service is reassigned to another node, the pending requests of that service will be transferred. The active process of a request in a node will continue until it is completed. The address of service will change so that incoming requests will be redirected to the new address of service.



CSP 1	CSP 2	CSP 3	CSP 4
S1: 40500 S2: 2000 S3: 1000 S4: 750 S5: 700 S7: 7550 S12: 4300	S6: 59000	S9: 57000	S13: 3000 S14: 2550 S15: 1500 S2: 2000 S3: 1000 S8: 9000 S10: 35000 S11: 900
56800	**59000**	**57000**	**54950**

Fig. 5. The service assignment of cloud services based on the proportional ordering.

3.3 Dynamic Execution of Service Assignment

It is ideal to perform migration frequently to minimize load variations, however, this also produces migration overheads. In a state where nodes have very low load variation, there is no need to perform migration because of migration overheads. Also, the decision of choosing a right state of load variation to perform the service assignment should be considered. The dynamic execution of service assignment is decided based on the average difference of resource consumption rates in all nodes which is compared to a threshold value. The same parameters from Section 3.1 are used in analyzing the resource consumption rate variation in the group of cloud servers. The difference of rc_n and μ_{rc} is calculated where s_i of rc_n is collected from a specified time period (p), $p = [p_i, p_f]$, p_i is the initial period and p_f is the ending period, $p < pa$. The average difference of resource consumption rates is represented by σ in Equation 12 where n is index of a node and N is the total number of nodes.

$$\sigma = \frac{1}{N}\sum_{n=1}^{N}\left|rc_n - \mu_{rc}\right| \tag{12}$$

The proposed algorithm tolerates σ to have a high value as much as μ_{rc}. We assume that the value of σ should not be greater than μ_{rc}. If $\sigma > \mu_{rc}$ then some nodes will have an additional resource consumption of more than μ_{rc} which will affect the service processes on those nodes to have longer processing time, and thus, migration is necessary. But also, migration overhead should be considered by the calculation. The threshold value represented by Φ in Equation 13 is calculated by μ_{rc} over λ in Equation 14.

$$\Phi_L = \frac{\mu_{rc}}{\lambda} \tag{13}$$

$$\lambda = \left(\frac{\mu_{pt}}{L}\right)\times\left(\frac{\sigma}{\max_{\sigma}}\right)\times M \tag{14}$$

Equation 14 determines the value of λ which is the product of mean processing time of cloud services (μ_{pt}) over the migration overhead (L), load variation ratio ($\sigma/\max_{\sigma}$), and number of services to be migrated (M). Migration overhead is the network latency of performing migration (L). L is determined by the mean size of cloud services over the speed of network (L=*service mean size*/*network speed*). If μ_{pt} is greater than L then migration can be performed which also means that migration is faster than processing time of a service. If L is greater than μ_{pt} then migration is not necessary. In the load variation ratio, $\max_{\sigma}$ is the value of maximum load variation that is calculated by overloading a single node with all *rc* while other nodes have no loads. M is determined in Equation 15 by adding the difference value of *rc* and μ in each node and divided by μ_s which will estimate the number of services to be migrated. Equation 16 is the function of skipping the value of node with $rc < \mu_{rc}$ by assigning it with 0 because there is no Δ in the node. The load variation ratio and M are important factors in service migration where a very small σ will not consider migration.

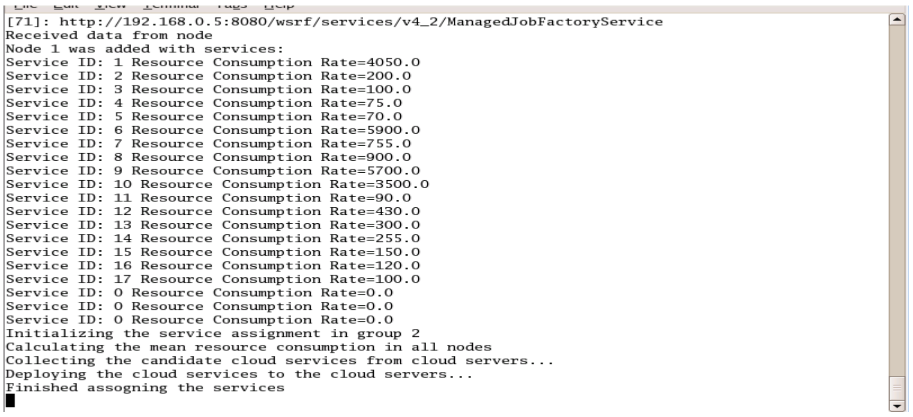
$$M = \frac{1}{\mu_s} \sum_{n=1}^{N} f(rc_n, \mu_{rc}) \tag{15}$$

$$f(rc_n, \mu_{rc}) = \begin{cases} 0, \ if \ \ rc_n < \mu_{rc} \\ rc_n - \mu_{rc}, \ if \ \ rc_n > \mu_{rc} \end{cases} \tag{16}$$

The value of Φ_L is adjusted based on the final value of λ. A value of lower than 1 from λ means that there is no need to perform migration while the opposite can trigger migration. The resource consumption analysis is performed by the load balancing service. The current σ in nodes is compared to Φ_L, and if $\sigma > \Phi_L$ then the virtualization manager executes service assignment. Also, because of f value in *rc*, the new assignments of services are efficient on serving the next requests if there is a similar trend of service access from previous service access trend of nodes, and if not the case then it is possible that the service assignments will be adjusted again. If $\sigma < \Phi_L$ then it means that there is no need to re-assign the services.

4. Simulation and Evaluation

The intelligent methods were developed using Java 2 SDK and these were integrated in the Globus toolkit [35] to implement the adaptive resource management. The cloud service providers performed grouping using the grouping service. In finding cloud service, a user agent was utilized as an interface for search. The complex interactions of finding appropriate cloud service and virtualization of resources were transparent to users. In managing resources, the load balancing service analyzed load variation in cloud servers and implemented a load balancing technique to forward requests to replicated services which were identified in a group of cloud service providers. The virtualization manager implemented the proposed dynamic service assignment (DSA) to support load distribution. Based on the analysis of resource consumption rate variation in all nodes, the dynamic execution of service assignment was operational. **Fig. 6** illustrates the interaction of the proposed dynamic service assignment where cloud services are collected from cloud service providers after the mean resource consumption rate and excess cloud services load are determined. After collecting cloud services, virtualization manager ranks the collected cloud services and nodes based on the proportional ordering, and then it deploys the cloud services.



```
[71]: http://192.168.0.5:8080/wsrf/services/v4_2/ManagedJobFactoryService
Received data from node
Node 1 was added with services:
Service ID: 1 Resource Consumption Rate=4050.0
Service ID: 2 Resource Consumption Rate=200.0
Service ID: 3 Resource Consumption Rate=100.0
Service ID: 4 Resource Consumption Rate=75.0
Service ID: 5 Resource Consumption Rate=70.0
Service ID: 6 Resource Consumption Rate=5900.0
Service ID: 7 Resource Consumption Rate=755.0
Service ID: 8 Resource Consumption Rate=900.0
Service ID: 9 Resource Consumption Rate=5700.0
Service ID: 10 Resource Consumption Rate=3500.0
Service ID: 11 Resource Consumption Rate=90.0
Service ID: 12 Resource Consumption Rate=430.0
Service ID: 13 Resource Consumption Rate=300.0
Service ID: 14 Resource Consumption Rate=255.0
Service ID: 15 Resource Consumption Rate=150.0
Service ID: 16 Resource Consumption Rate=120.0
Service ID: 17 Resource Consumption Rate=100.0
Service ID: 0 Resource Consumption Rate=0.0
Service ID: 0 Resource Consumption Rate=0.0
Service ID: 0 Resource Consumption Rate=0.0
Initializing the service assignment in group 2
Calculating the mean resource consumption in all nodes
Collecting the candidate cloud services from cloud servers...
Deploying the cloud services to the cloud servers...
Finished assogning the services
█
```

Fig. 6. Service assignment processed by the virtualization manager.

A simulation environment was configured to determine the network latency and throughput performance of the system using the proposed dynamic service assignment. Total message latencies and total turn-around time of requests were observed to determine the amount of network latency from exchanging messages and to determine the responsiveness of the system, respectively. These two performance measures impose a tradeoff in using conventional algorithms. An example of the tradeoff is in the case of using round robin and least load selection for load distribution. When there are fewer requests arriving in a period of time, the round robin has better performance in throughput and message overhead. However, if a large volume of requests arrived in a short period of time, then the least load selection procedure is more effective in providing responsive services compared to round robin. To set the configuration in measuring network latency and throughput performance, simVO [36] was used. The simVO implements message handlers for the communications of agents and scheduler to schedule events of message passing, task processing and migration of services which was used by DSA. Other simulators like in [37][38] have no support in handling message passing.

4.1 Simulation Environment

The simulation environment was consisted of 100 nodes, 20 services and 20 virtual groups. Virtual grouping of nodes was necessary to simulate the dynamic environment of a cloud system. The nodes were divided equally into 10 domains and identified by {A, B,…, J}. A ring topology was used to connect each domain where a domain has direct connection to two domains represented by {A↔B↔C↔D↔E↔F↔G↔H↔I↔J↔A} and each link has a latency of 10 milliseconds (ms). All nodes in a domain were connected to a router and their connections have a mean latency of 5 ms. Separated by domains, a node can connect with another node in a different domain by a virtual group. Services were replicated throughout the nodes where each node was deployed with different 5 services. The mean processing time of services was set to 550 ms. Each service has a size of 1 MB in migrating while the network speed is 100 Mbps. The resource consumption and processing time of each service in node 1 is shown in **Table 2**. Each *f* value from service was set to 1 and incremented by 1 after a service was invoked by a request.

The initial deployment of services and assignment of nodes in a group were random. Every virtual group was set with initial time and termination time where each has a 10000 ms period to execute and a time interval of 500 ms to start the next virtual group. The time period for analyzing the resource consumption rate variation as basis to execute the service assignment was set to 1000 ms. Client requests were generated to perform the simulation.

Table 2. Service resource consumption and processing time.

Service Index	CPU Consumption	Processing Time
1	45%	750 ms
2	33%	550 ms
3	30%	500 ms
4	28.5%	475 ms
5	27%	450 ms

A request contained tags of service, node identification of requester and arrival time of request. The services were application cloud services with properties of processing time and labeled by service tags. Each node was configured with only one processor that has a speed of 1 GHz. Loads were determined by total time of all waiting jobs in the queue of a node. The load distribution scheme and service assignment were implemented within the group. Also, the virtual groups were configured that the processing of tasks will continue even after a group is terminated.

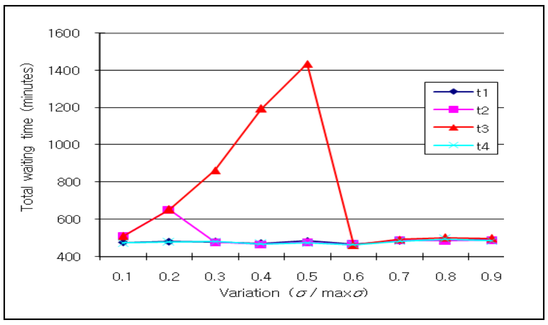
4.2 Performance Evaluation

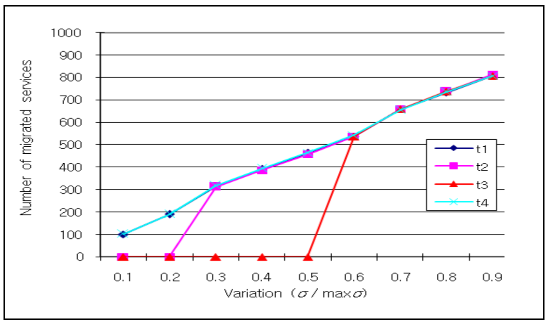
4.2.1 Threshold analysis for executing DSA

The threshold value in Equation 14 for the dynamic execution method was verified. Two measures were used which are the total turn-around time of requests and overhead from the number of migrated services. Different thresholds were compared to Equation 14 and these were; t1 = 0.0 which always executes migration process, t2 = $\mu_{rc}/2$ which is a value between t1 and t3, t3 = μ_{rc} which executes migration process when σ greater than μ_{rc} and t4= μ_{rc}/λ which is used for the dynamic execution method. We generated 1000 requests to perform the verification and distributed the requests to nodes based on a specified state of σ.

Fig. 7 shows the result of total turn-around time of requests and migration overhead using different load variation values represented in x axis of the graph. The load variation ratio (x) is calculated by $\sigma/\max_{\sigma}$. The x values were set from 0.1 to 0.9. Each case of x was repeated 10 times and calculated the average of results which was used as final value. In a high load variation, the waiting time of each request in a node is longer because of the high volume of pending requests to be processed. **Fig. 7-(a)** shows the total turn-around time of requests from each threshold in minutes and, noticeably, t3 took a longer time to finish each request from 0.2 to 0.5 values of x which is not efficient. t1 always executed migration but migration overhead was also high shown in **Fig. 7-(b)**. The turn-around time and migration overhead of t4 has similar trend as t1. The average results from **Fig. 7-(a)** (t1=480.95, t2=500.88, t3=734.43, t4=478.11) and from **Fig. 7-(b)** (t1=465.89, t2=433.11, t3=304.33, t4=467.44) shows that t4 has the shortest turn-around time compared to all algorithms and t4 is better in handling migration overheads than t1, respectively.

Moreover, the performance of each threshold in very low load variations was compared. x was set from 0.01 to 0.1 and generated 100 to 1000 requests, and then totaled the result of each case. In **Fig. 8**, the average of total results is shown where t1 has the longest total turn-around time because of migration overhead. t2 and t3 did not execute migration while t4 executed migration in x=0.1 and did not execute in x<0.1. The result from Figure 8 shows that migration is not necessary in a very low x.





(a) (b)

Fig. 7. Total turn-around time (a) and migration overhead (b) using different load variation ratio.

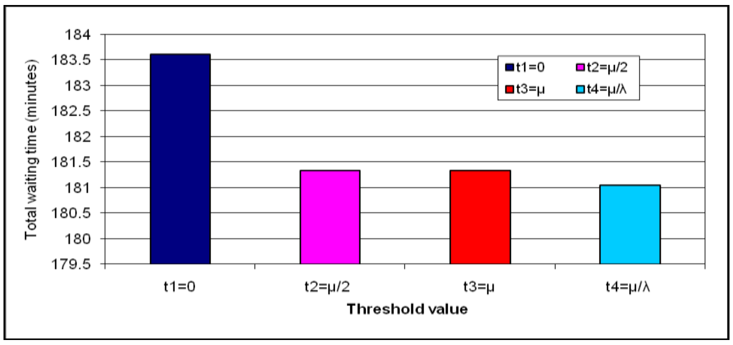
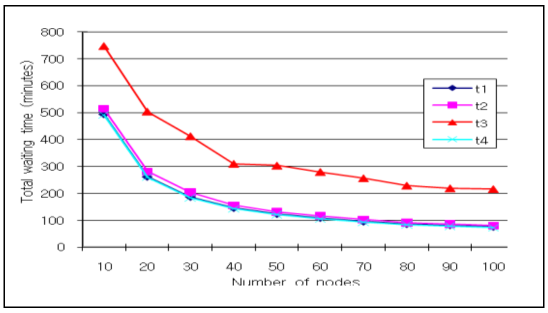
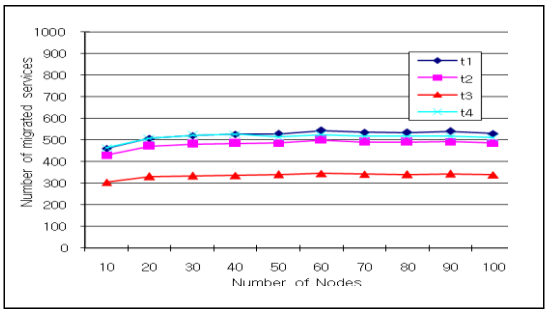


Fig. 8. Total turn-around time using load variation ratio ranging from 0.01 to 0.1.





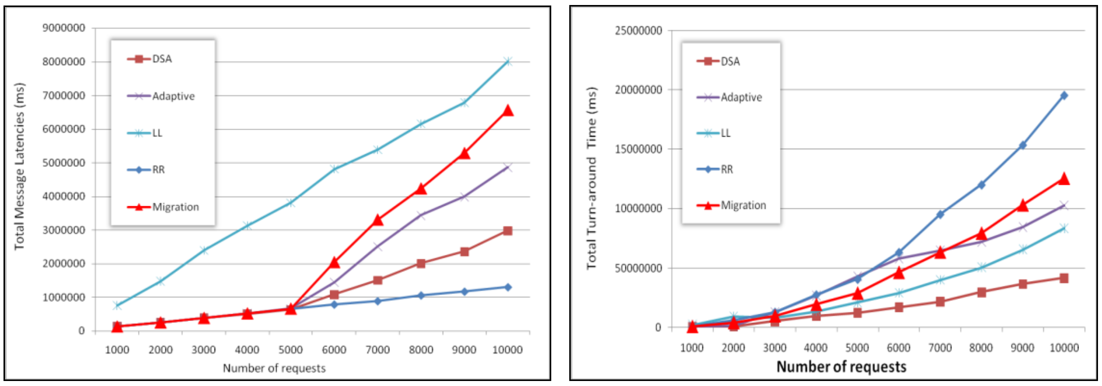
(a) (b)

Fig. 9. Total turn-around time (a) and migration overhead (b) in increasing the number of nodes.

In Fig. 9, the number of nodes in x axis represents the resource size in a group (*N*). The values used in Fig. 9 is the average result from 0.1 to 0.9 of *x*. Fig. 9-(a) shows that increasing *N* resulted to a shorter turn-around time of requests. This is because most of the loads were distributed throughout the nodes, and thus, the waiting time of a service was shortened. In Fig. 9-(b), the average of migration overhead is shown. Similar to Fig. 7-(b), t3 has the longest turn-around time but lowest in migration overhead. Fig. 9 shows that t4 is almost similar to t1 but is better in handling migration overhead when the nodes are increased. The average results from Fig. 9-(a) (t1=165.15, t2=176.11, t3=346.75, t4=163.46) and from Fig. 9-(b) (t1=522.38, t2=480.84, t3=335.62, t4=511.8556) shows that t4 has the shortest turn-around time and better in handling migration overhead, repectively.

4.2.2 Normal distribution result

There were two cases used in generating requests for the performance evaluation which are the normal distribution and exponential distribution. In the first case of simulation, requests were generated using a normal distribution. This determines the performance of an algorithm in handling the request arrivals in an exponential increasing and decreasing manner. The largest number of requests is observed in the middle of simulation period in this case. We used common load balancing techniques for the performance comparison. The round robin (RR), least load selection (LL), adaptive load distribution and migration scheme were used to compare the proposed dynamic service assignment (DSA) for message latencies and turn-around time performances. The LL, RR and adaptive distribution are not the same approach as the service assignment and migration scheme but these are used for efficient resource allocation and load balancing of distributed nodes [25][26][36]. RR forwarded the requests in an alternating manner while LL forwarded the requests to the least loaded node with the same service. The adaptive load distribution used both RR and LL alternately based on a load variation analysis which was already supported in simVO [36]. In the migration scheme, we used the method in [39]. Both DSA and migration scheme used RR technique to perform service migration, however, DSA analyzed the resource consumption rate variation while migration scheme analyzed the current difference on server loads, where the execution of DSA was not as frequent as in migration scheme. The total network latencies of messages and total turn-around time of requests used a volume of 1000 to 10000 requests for evaluation and the results are shown in **Fig. 10**.



(a) **(b)**

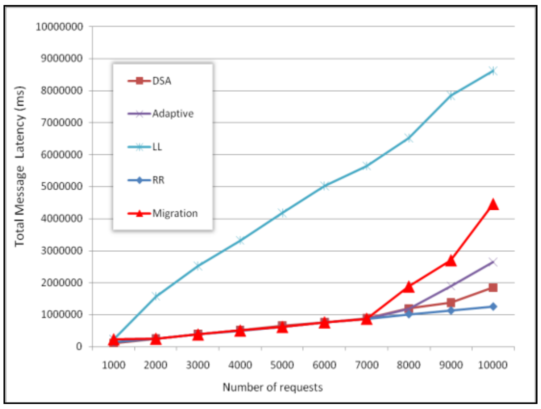
Fig. 10. Message latencies **(a)** and total turn-around time **(b)** using arrival time in normal distribution

The latencies of deciding for the least loaded node and migration processes were included in the message latencies. In **Fig. 10-(a)**, the message latencies of all algorithms are shown. It was observed that the RR produced the lowest message latencies and DSA was second to RR. The reason RR had the lowest result is because it did not produced latency of deciding in which node it will forward the request, which was the case in LL; nor did it produced latency in migrating cloud services, which was the case in migration scheme. It was also observed from the adaptive load distribution, that the migration scheme and DSA produced a significant increase in latencies starting from 6000 requests. In the adaptive load distribution, a frequent use of least load selection was occurring because of the high load variation. Similarly, migration scheme frequently executed migration of services because of the high load variation. Different from the two mentioned schemes, DSA was analyzing the resource consumption rate variance of nodes periodically to decide the execution of service assignment. In **Fig. 10-(b)**, the total turn-around time performance shows that DSA is the fastest to respond to requests.

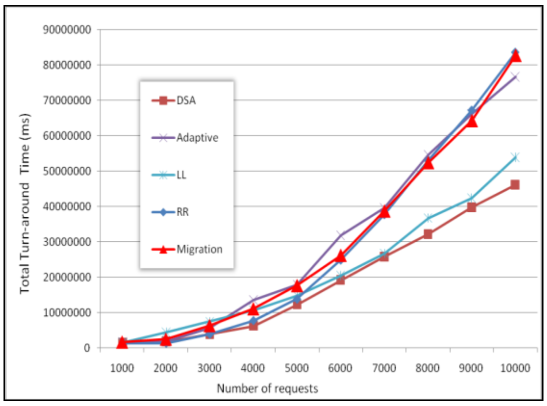
The results from LL in **Fig. 10-(b)** are opposite on those from **Fig. 10-(a)**. We observed that LL demonstrated better load distribution in **Fig. 10-(b)**, but has the highest message latencies compared to all algorithms. Moreover, LL was still outperformed by DSA in handling load distribution. RR was inefficient in handling the response time of requests because it cannot handle high load variation throughout the nodes which caused delays in processing the requests.

4.2.3 Exponential distribution result

The second case generated the requests using an exponential distribution. In this case, the performance of each algorithm in handling the request arrivals in an exponential increasing manner was determined. The largest number of requests is observed in the final period of simulation. At the start of simulation, few requests are arriving and then the number of requests is increased exponentially until the end of simulation. The same algorithms in Section 4.2.2 were used to compare the performance of the proposed algorithm in message latencies and total turn-around time of requests.



(a)



(b)

Fig. 11. Message latencies **(a)** and total turn-around time **(b)** using arrival time in exponential distribution.

In **Fig. 11-(a)**, it shows that the RR produces the lowest message latencies and the DSA is second to RR which has the same trend in a normal distribution in **Fig. 11-(a)**. The significant increase in latency from the adaptive load distribution, migration scheme and DSA was observed starting from 8000 requests. Also, it was observed that the message latencies from DSA, adaptive load distribution and migration scheme were lowered compared to **Fig. 10-(a)**. DSA is the fastest to respond to requests in both cases shown in **Fig. 10-(b)** and **Fig. 11-(b)**.

4.2.4 Average of normal and exponential distribution

The number of migrated services from migration technique and DSA in Sections 4.2.2 and 4.2.3 is summarized in **Fig. 12**. Also, the results from two cases were averaged and these are shown in **Fig. 13** which is a summary of performances of algorithms in message latencies and turn-around time of requests.

Fig. 12 shows that DSA is more efficient because of lower migration overhead compared to migration technique. Performing service migration produces network latencies and also affects the turn-around time of requests when these requests are waiting to be processed by a migrating service. However, in DSA, the services were assigned to nodes according to its proportioned resource consumption. The proposed method tried to provide equal resource consumption in all nodes so that a node can be responsive on the next requests. Fig. 13 shows the over-all performance from simulations. In Fig. 13-(a), RR is the lowest in producing message latencies while DSA is second to RR. LL has the highest message latencies because of high volume of message exchanges in the group of nodes. In Fig. 13-(b), DSA is the fastest in completing service requests while RR is the slowest. It is also shown that the adaptive load distribution and migration almost have same result.

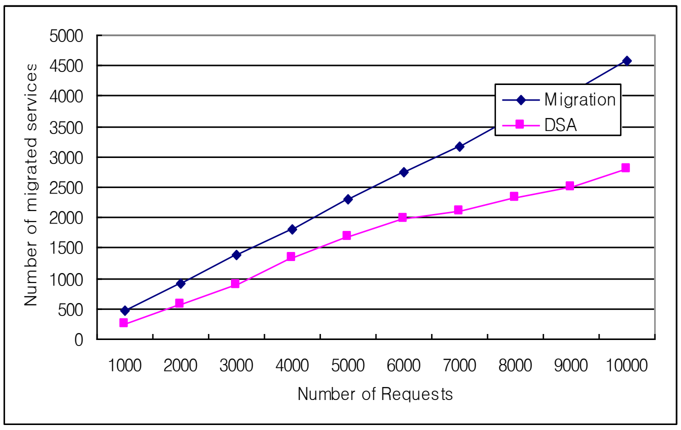
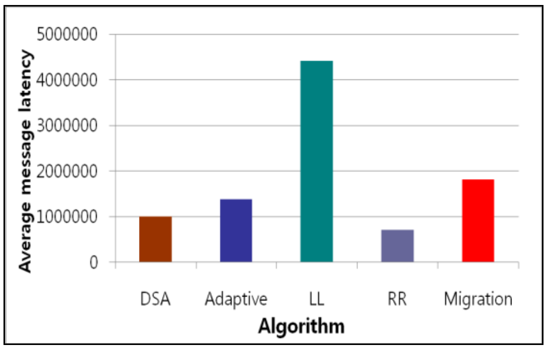
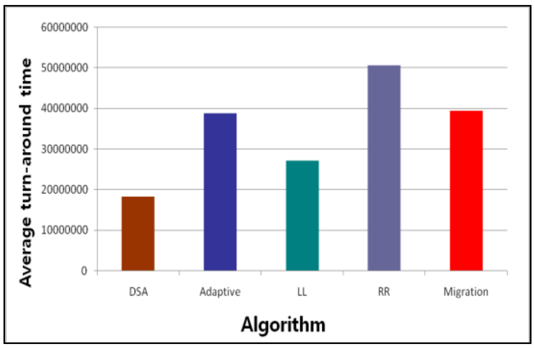


Fig. 12. Average number of migrated service of Migration and DSA.





(a) (b)

Fig. 13. Average message latency (a) and total turn-around time (b) from all cases.

We also calculated the ratio from the results of DSA to other algorithms to compare the efficiency in each algorithm, e.g., if the ratio of RR and DSA is 2:1 then DSA was twice efficient compared to RR. DSA was 1.4, 4.4 and 1.8 times better than adaptive scheme, LL and migration scheme, respectively, in message latencies. RR (0.7) had the lowest message latencies, however, it was the slowest in total turn-around time performance. DSA was 2.1, 1.4, 2.7 and 1.8 times better than adaptive load distribution, LL, RR and migration scheme, respectively, in total turn-around time performance.

5. Conclusion

Cloud computing is an emerging computing paradigm and, apparently, there will be a need for new technologies and innovations to exploit this new paradigm. The QoS in provisioning of application services to cloud users is very important and this is supported by proper interactions from components of resource management. A cloud system integrated with intelligent algorithms for adaptive methods in providing services and managing resources was presented in this paper. The adaptive resource management supported the service provisioning of the proposed cloud system. This paper mainly focused on the details of the proposed dynamic service assignment that supported the load distribution within cloud servers. The proposed technique identified excess cloud services based on the mean resource consumption, and then, assigned these to nodes considering equal resource consumption by the proportional ordering of services and nodes, finally, deployed the services to its assigned nodes. The variation of resource consumption rate was analyzed periodically to decide the service assignment execution.

A network topology for simulation was configured and the performance of DSA was evaluated in terms of message latencies and turn-around time. The result showed that DSA performed better in message latencies than adaptive scheme and LL because it did not produce the latency of service forwarding, and migration of services was not frequently executed compared to migration scheme. RR had the lowest message latencies but also had longer response time for service requests. DSA performed best in throughput performance because it provided the efficient assignment of services to nodes for the next requests and this resulted to the responsiveness of cloud services.

References

[1] Sales Force, "Service cloud and chatter," http://www.salesforce.com.

[2] Amazon Web Services, "Amazon EC2," http://aws.amazon.com/ec2.

[3] S. Zhang, S. Zhang, X. Chen and X. Hou, "Cloud Computing Research and Development Trend," in *Proc. of International Conference on Future Networks*, pp. 93-97, Jan. 2010. Article (CrossRef Link)

[4] S.S. Yadav and Z.W. Hua, "Cloud: A Computing Infrastructure on Demand," in *Proc. of Computer Engineering and Technology*, pp. 423-426, Apr. 2010. Article (CrossRef Link)

[5] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, Dec. 2009. Article (CrossRef Link)

[6] Google, "Google app eng," http://code.google.com/appengine.

[7] Microsoft Corporation, "Azure services platform," http://www.microsoft.com/windowsazure.

[8] NASA, "Nebula cloud computing platform," http://nebula.nasa.gov.

[9] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I.M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich and F. Galán, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing," *IBM Journal of Research and Development*, vol. 53, no. 4, July 2009. Article (CrossRef Link)

[10] J.P. Casazza, M. Greenfield and K. Shi, "Redefining Server Performance Characterization for Virtualization Benchmarking," *Intel Technology*, vol. 10, no. 3, Aug. 2006. Article (CrossRef Link)

[11] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao and D. Newell, "VM3: Measuring, Modeling and Managing VM Shared Resources," *Computer Networks*, vol. 53, no. 17, pp. 2873-2887, Apr. 2009. Article (CrossRef Link)

[12] Q. Li, Q. Hao, L. Xiao and Z. Li, "Adaptive Management of Virtualized Resources in Cloud Computing using Feedback Control," in *Proc. of IEEE International Conference on Information Science and Engineering*, pp. 99-102, Dec. 2009. Article (CrossRef Link)

[13] J. Li, M. Qiu, J.W. Niu, Y. Chen and Z. Ming, "Adaptive Resource Allocation for Preemptable Jobs in Cloud Systems," in *Proc. of 10th International Conference on Intelligent Systems Design and Applications*, pp. 31-36, Nov. 2010. Article (CrossRef Link)

[14] G. Ribeiro-Justo, A. Salehb and T. Karranb, "Intelligent Reconfiguration of Dynamic Distributed Components," *Electronic Notes in Theoretical Computer Science*, vol. 180, no. 2, pp. 91-106, Aug. 2007. Article (CrossRef Link)

[15] R.M.A. Mateo and J. Lee, "Data Mining Model based on Multi-agent for the Intelligent Distributed Framework," *International Journal of Intelligent Information and Database Systems*, vol. 4, no. 4, pp. 322-336, 2010. Article (CrossRef Link)

[16] M.A. Cusumano, "The Changing Software Business: Moving from Products to Services," *Computer*, vol. 41, no. 1, pp. 20-27, Jan. 2008. Article (CrossRef Link)

[17] D. Thomas, "Enabling Application Agility – Software as a Service, Cloud Computing and Dynamic Languages," *Journal of Object Technology*, vol. 7, no. 4, pp. 29-32, May-June 2008. Article (CrossRef Link)

[18] H. Liao, "SaaS Business Model for Software Enterprise," in *Proc. of International Conference on Information Management and Engineering*, pp. 604-607, 2009. Article (CrossRef Link)

[19] J.C. Chen, N.E. Gold, N. Mehandjiev and P.J. Layzell, "Managing Supply Chains of Software as a Service through Agent Negotiations," in *Proc. of IEEE 7th International Conference on E-Commerce Technology*, pp. 378-381, 2005. Article (CrossRef Link)

[20] A. Elfatatry and P. Layzell, "Software as a Service: A Negotiation Perspective," in *Proc. of Computer Software and Applications Conference*, pp. 501-506, Dec. 2002. Article (CrossRef Link)

[21] M. Godse and S. Mulik, "An Approach for Selecting Software-as-a-Service (SaaS) Product," in *Proc. of IEEE International Conference on Cloud Computing*, pp. 155-158, Sep. 2009. Article (CrossRef Link)

[22] W.T. Tsai, X. Sun, Q. Shao and G. Qi, "Two-tier Multi-tenancy Scaling and Load Balancing," in *Proc. of IEEE 7th International Conference on Business Engineering*, pp. 484-489, Nov. 2010. Article (CrossRef Link)

[23] L. Zhang, Y. Wen and Y. Han, "A Proactive Approach to Load Balancing of Workflow Execution in a SaaS Environment," in *Proc. of IEEE 5th International Symposium on Service Oriented System Engineering*, pp. 39-46, June 2010. Article (CrossRef Link)

[24] S. Ali, H.J. Siegel and A.A. Maciejewski, "The Robustness of Resource Allocation in Parallel and Distributed Computing Systems," in *Proc. of ISPDC/HeteroPar*, pp. 2-10, July 2004. Article (CrossRef Link)

[25] M. Louta and A. Michalas, "Efficient Service Provisioning through Dynamic Service Task Assignment in a Multi-domain Distributed Computing Environment," *International Journal of Internet Protocol Technology*, vol. 3, no.3, 2008. Article (CrossRef Link)

[26] B.Volckaert, P. Thysebaert, M. D. Leenheer, F. D. Turck, B. Dhoedt and P. Demeester, "Flexible Grid Service Management through Resource Partitioning," *Journal of Supercomputing*, vol. 38, 2006, pp. 275-305. Article (CrossRef Link)

[27] R.M.A. Mateo and J. Lee, "Proportional Load Balancing using Scalable Object Grouping based on Fuzzy Clustering," *Applications in Soft Computing, Advances in Intelligent and Soft Computing*, vol. 58, pp. 41-50, 2009. Article (CrossRef Link)

[28] C.L. Hu, D.Y. Chen, Y.H. Chang and Y.W. Chen, "Fair peer Assignment Scheme for Peer-to-Peer File Sharing," *KSII TIIS Journal*, vol. 4, no. 5, pp. 709-735, Oct. 2010. Article (CrossRef Link)

[29] H.A. Thant, K.M. San, K.M.L. Tun, T.T. Naing, N. Thein, "Mobile Agents based Load Balancing Method for Parallel Applications," in *Proc. of APSITT*, pp. 77–82, Feb. 2005. Article (CrossRef Link)

[30] Y. Yang, Y. Chen, X. Cao1 and J. Ju1, "Load Balancing using Mobile Agent and a Novel Algorithm for Updating Load Information Partially," in *Proc. of ICCNMC, LNCS*, vol. 3619, pp. 1243-1252, 2005. Article (CrossRef Link)

[31] Y.F. Huang and C.C. Fang, "Load Balancing for Clusters of VOD Servers", *Information Sciences*, vol. 164, no. 1-4, pp. 113-138, Aug. 2004. Article (CrossRef Link)

[32] L. Hughes, "Process Migration and its Influence on Interprocess Communication", *Computer Communications*, vol. 21, no. 9, pp. 781-792, July 1998. Article (CrossRef Link)

[33] J. Xu, Z. Zhu, X. Ren, Y. Tian and Y. Luo, "Personalized Web Search using User Profile," in *Proc. of International Conference on Computational Intelligence and Security*, pp. 222-227, Dec. 2007. Article (CrossRef Link)

[34] L. Ardissono, A. Goy, G. Petrone and M. Segnam, "From Service Clouds to User-Centric Personal Clouds," in *Proc. of IEEE International Conference on Cloud Computing*, *Cloud*, pp 1-8, Sep. 2009. Article (CrossRef Link)

[35] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997. Article (CrossRef Link)

[36] R.M.A. Mateo, H.H. Yang and J. Lee, "Managing Virtual Organizational Tasks using Simvo in Grid Environment," in *Proc. of ICONI & APIC-IST*, Dec. 2010, pp. 669-673.

[37] A. Sulistio, U. Cibej, S. Venugopal, B. Robic and R. Buyya, "A Toolkit for Modeling and Simulating Data Grids: an Extension to GridSim," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 13, pp. 1591-1609, 2008. Article (CrossRef Link)

[38] F. Howell and R. McNab, "Simjava: a Discrete Event Simulation Package for Java with Applications in Computer Systems Modeling", in *Proc. of International Conference on Web-based Modelling and Simulation*, 1998. Article (CrossRef Link)

[39] R.M.A. Mateo and J. Lee, "Load Balancing based on Migration in Cloud Environment," in *Proc. of ICONI & APIC-IST*, December 2010, pp. 675-679.

Romeo Mark A. Mateo received his B.S. degree in Information Technology from West Visayas State University, Philippines in 2004 and M. Eng. degree in Information and Telecommunications Engineering from Kunsan National University, South Korea in 2007. Currently, he is a Doctor of Engineering candidate in Information and Telecommunications and working as a research assistant at the Distributed Systems Laboratory (DSL). His research interests include distributed systems, mobile computing, wireless sensors, artificial intelligence and data mining.

Jaewan Lee received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Chung-Ang University in 1984, 1987, and 1992, respectively. Currently, he is a professor at the School of Electronic and Information Engineering in Kunsan National University, Kunsan City, South Korea. His research interests include distributed systems, database systems, data mining and computer networks.