RAS: Request Assignment Simulator for Cloud-Based Applications

R. Arokia Paul Rajan[1] and F. Sagayaraj Francis[2]
[1,2] Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India – 605 014
e-mail: [1]paulraajan@gmail.com, [2]fsfrancis@pec.edu
*Corresponding author: R. Arokia Paul Rajan

Received October 8, 2014; revised July 10, 2014; accepted May 26, 2015; published June 30, 2015

Abstract

Applications deployed in cloud receive a huge volume of requests from geographically distributed users whose satisfaction with the cloud service is directly proportional to the efficiency with which the requests are handled. Assignment of user requests based on appropriate load balancing principles significantly improves the performance of such cloud-based applications. To study the behavior of such systems, there is a need for simulation tools that will help the designer to set a test bed and evaluate the performance of the system by experimenting with different load balancing principles. In this paper, a novel architecture for cloud called Request Assignment Simulator (RAS) is proposed. It is a customizable, visual tool that simulates the request assignment process based on load balancing principles with a set of parameters that impact resource utilization. This simulator will help to ascertain the best possible resource allocation technique by facilitating the designer to apply and test different load balancing principles for a given scenario.

Keywords: Cloud computing, Cloud simulators, Request assignment, Load balancing

A preliminary work of this paper published in Advances in Intelligent Systems and Computing, Springer International Publishing, Volume 248, 2014.

http://dx.doi.org/10.3837/tiis.2015.06.004 **ISSN : 1976-7277**

1. Introduction

Cloud computing has become an inevitable impacting paradigm on recent technology evolutions in the IT industry. The cloud computing model turned out to be the way in which Internet applications were developed, delivered, and consumed [1]. In this model, all the resources and services are considered as a utility. These resources are largely distributed, highly scalable, and virtualized for its global users accessing through the Internet. Cloud computing gives the illusion that these resources are limitless and made available on demand. The appealing features from the consumer's viewpoint are that a cloud service does not require any cost upfront and is delivered with a pay as-you-go precept. Cloud services are offered on three levels [2]. At the lowest level, the physical resources are provisioned as services known as Infrastructure as a Service (IaaS). Amazon EC2, which offers its resources as a service, belongs to this category [3]. At the second level, services are offered on a software platform known as Platform as a Service (PaaS). Using this platform, application developers can develop and deploy their applications. Google App Engine provides such services [4]. At the last level, applications are offered as services to customers known as Software as a Service (SaaS). Rackspace offers many SaaS services [5].

Without suitable experimenting methodologies, designing such a computing and delivery model for these kinds of services may end up as a costly risk. Experimenting with the simulated environment and enumerating the alternative solutions will help the designer to make appropriate decisions. The main factors that influence the performance of large-scale distributed environment are the computing elements, network components, and the governing strategies through which they interact with each other. In cloud architectures, there are few policies that decide the way in which requests have been processed and delivered to the user. Service broker policy and load balancing principle in the request scheduling process are the two most important strategies that a cloud application architect has to design [6]. **Fig. 1** presents the locale of request assignment process in cloud architectures.

Based on the selection of service broker policy, the requests are routed to a data center. The objective of a service broker policy is to achieve the maximum rate of utilization of resources or service the maximum number of requests or to minimize response time and computing costs. The load balancing principle determines the best method to assign requests with the servers to achieve the objective policy of the service broker architecture [7]. In reality, it will be very expensive to set up a large-scale distributed platform to test and evaluate such an architectural design policy. Since the designer cannot have control or predict the varying nature of structural components in the real environment, a comprehensive study should be done on the impact of the choice of principle selection. Designing and adopting these strategies will enable higher resource utilization and meeting customers' expectations. It will also strive to achieve maximum service quality with minimum costs.

In this paper, we present a simulation tool called Request Assignment Simulator (RAS) that allows the learner to model, customize, test, and evaluate a use case in cloud architecture. It models the user base, data center controller, and resource manager layers of the cloud architecture. Using this tool, one can create user bases, generate cloudlets, configure the resource and their parameters, opt the load balancing principles, view requests assignment, compute performance, consolidate results, and eventually compare performances. This simulator is designed to represent a generic cloud model which could be extendable to any specific cloud model.

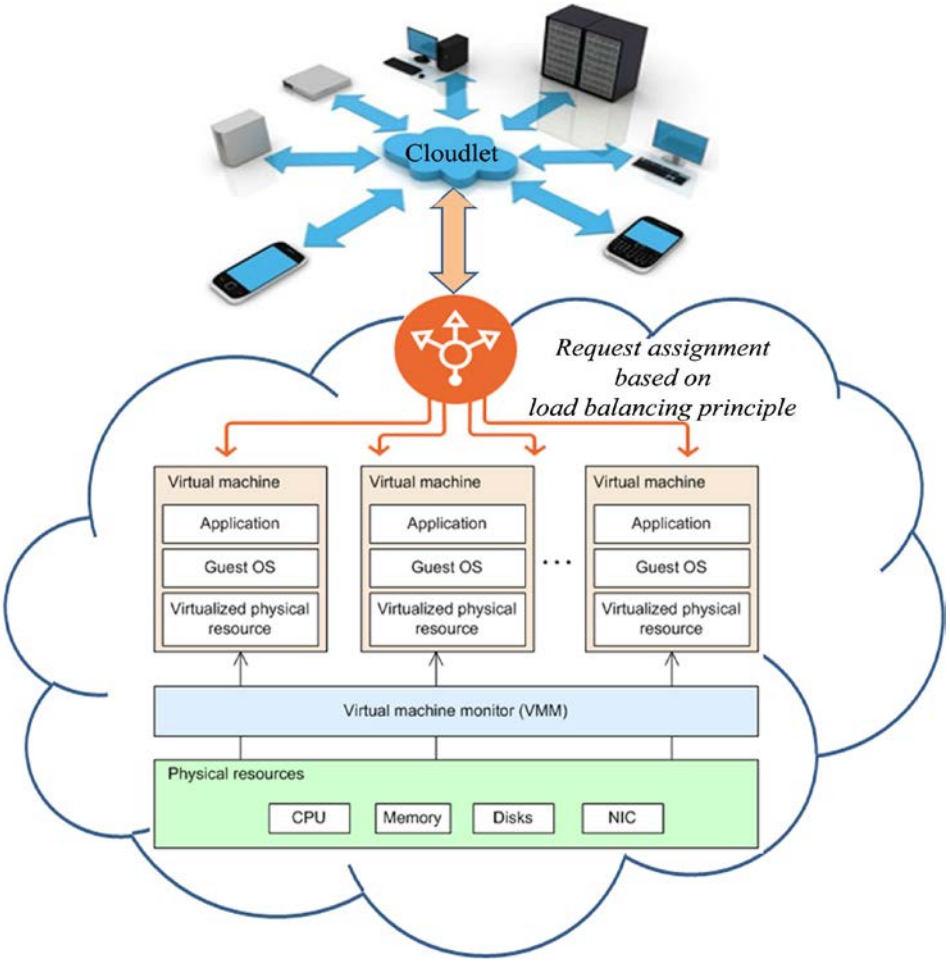

Fig. 1. Request assignment in cloud architecture

In this paper, we present a problem scenario and design a solution model and a suitable load balancing principle that effectively performs the requests assignment to compute resources. In order to compare a newly proposed load balancing principle with a few commonly adopted load balancing principles, we designed a visual tool which evolved into a simulator, namely, the Request Assignment Simulator (RAS). The proposed principle is more generic in nature. It can be extendable to any network architecture like p2p, cluster, grid, and cloud, provided the parameters are suitably personalized.

RAS complements the successful cloud simulator in the industry, namely, Cloud Analyst. The models, parameters, and metrics of RAS are compared and justified with Cloud Analyst. We concentrated on designing our tool as a test bed for evaluating some of the successful load balancing principles that are adopted in the cloud. We kept Cloud Analyst as our reference model for deriving the parameters and design principles. RAS computes the response time of the requests and the percentage of the rate of utilization of the compute nodes as the measure for the system's performance.

RAS is developed in .Net framework using C# and MS Access. It can be downloaded from the link: http://www.paulraajan.blogspot.in

The rest of this paper is organized as follows: Section 2 lists the related work. Section 3 introduces a problem scenario, architectural design of the components, used parameters, details about experimental setup, and significant features of our model. Section 4 concludes the paper and provides direction for future works.

2. Related Work

Until recently, there were only a few works in the realm of the development of simulators that focused on evaluating the behavior of large-scale distributed systems. A few such studies evolved with a handful of simulation tools for clusters, P2P systems, and grids, and these became the pioneering works that paved the path for the development of cloud simulators.

GridSim is a Java-based simulation toolkit designed to evaluate the performance of large-scale distributed systems typically fit for grid environments [8]. GridSim model request collection, creates a task pool, configures a resource panel, and schedules tasks with resources. CloudSim is a Java-based simulator that is built on top of GridSim toolkit's framework [9]. CloudSim is the pioneer in simulation platform, which supports modeling, simulation, and experimenting on cloud computing architectures. It is an effective simulator that provides better modeling features for large-scale computing environments. It includes configuring data center resources, virtual machine management, detailed network parameters, service broker and scheduling policies, tasks assignment between hosts and virtual machines, and a detailed performance evaluation of the system. CloudSim is a better fit for IaaS, PaaS, and SaaS of the service provider's infrastructure.

A cluster scheduler simulator can be used to model and equate different cluster scheduling strategies [10]. It simulates the generation of workloads from different clusters using empirical parameter distributions, scheduling and executing jobs using discrete event creators, and analyzing performance metrics. iCanCloud is a cloud simulator aimed to evaluate the trade-offs between cost and performance of a given scenario experimented with a specific infrastructure setup [11]. This simulator platform includes provisioning of cloud brokering policies, customizable virtual machines, configurations for a wide range of storage systems, and a user-friendly GUI for launching experiments extended with generating graphical reports.

GroudSim is an event-based simulator focused on IaaS [12]. It provides a comprehensive set of features like calculation of costs, job processing on computing resources, and load distribution on resources. TeachCloud is a modeling and simulation environment for cloud computing [13]. Learners can experiment with data centers, computing elements, networking, Service Level Agreement (SLA) constraints, and virtualization.

CloudAnalyst is a visual evaluation and modeling simulator that leverages the features of the CloudSim framework with an extension of some capabilities [14]. This simulator is developed to simulate large-scale distributed computing environments with the objective of studying the behavior of cloud applications under various deployment configurations. CloudAnalyst provides a visual platform for developers to understand more about configuring cloud infrastructures. Users can also set up a scenario with the goal of optimizing the application performance by suitably opting the service broker and load balancing strategies.

3. Request Assignment Simulator (RAS)

3.1 Request assignment in cloud

Request assignment involves job scheduling policies that control the order in which the tasks are allotted with the computing resources [15]. It is a process of allocation of required resources to the tasks under real constraints. This process plays a vital role in High-Performance Computing (HPC) and aims to achieve quality of service through the overall performance of a system as seen by users. A good request assignment policy makes a trade-off between customers' satisfaction and resource utilization at cheaper costs.

Load balancing principles are effectively used in the request assignment process through which the workload is distributed across amalgamated computing resources, such as computers, network links, central processing units, and storage elements [16]. There are different load balancing principles adopted to achieve objectives such as maximizing resource utilization and throughput, minimizing response time, and avoiding overload on any one of the resources.

3.2 Scenario

Consider a web service provider who offers different services through the Internet. Due to economic and administrative reasons, they want to shift their IT infrastructure to the cloud. Their IT infrastructure requirement mainly includes compute resources like processors, disks, and storage. The cloud infrastructure provider offers different configurations of computing resources with varying price plans. Therefore, an IT architect of the web service provider has to wisely choose the best infrastructure plan which should not be over or under provisioned as well as economical to the company. We illustrate this scenario in the following case study:

www.reverso.net/ is a famous online language solution provider offering services like language translations, grammar check, and spell check [17]. Reverso receives global requests for their web services. Each web service provided by Reverso is variable in size and processing time. The processing time is not the same for all services, but varies according to users' requests. For example, the processing time for a user who has requested a language conversion service for thousand words will be different from a user who requested the same service for ten thousand words. It is similar to comparing a service that sorts ten numbers and requires different processing time and memory to a service that sorts ten thousand numbers.

If Reverso wants to subscribe IT infrastructure from the cloud provider Amazon EC2, it would need to estimate the required virtual server reserved instances [18]. "How many virtual boxes are needed?" can be better answered if one knows "how much each virtual box can give."

3.3 Design principles

We attempt to answer the second part of the question with our proposed model. Our model is designed with the following principle: Each compute node alias virtual server instance is configured with processor(s), memory, the maximum number of connections it can support, bandwidth, operating system, etc. Based on the requests, the services are instantiated in the compute nodes. The instantiation of the requests is restricted by the load and storage limitations. Therefore, the assignment of requests to the compute nodes is subjective to the capacity constraints. When there is a restriction on the requests accommodation with the compute nodes, there is a need for prioritization of requests. Requests grouping may be based

on the requested service or weightage of the service or by the category of the users. The global requests need to be geographically grouped and assigned to the proximate data center.

We canonically describe the model as follows: There are services available as data items $I = \{i_1, i_2,\ldots.i_j\}$. Each data item i_j attains a value p_j based on its demand. There are compute nodes $N = \{n_1, n_2,\ldots...n_j\}$ that process the requests. Each node is configured with its maximum storage capacity Si and load capacity Lj, which indicates the maximum number of sessions that a node can serve simultaneously. There are requests $R = \{r_1, r_2,\ldots...r_i\}$ at time quantum t_i. Each request r_i seeks a particular data item i_j. The goal is to assign the requests to nodes with the objective of maximizing the total value earned by the nodes subject to the capacity constraints.

RAS supplements the model proposed in Cloud Analyst, a cloud test bed that has attracted the attention of many academicians and researchers in recent times. In our model, we used the parameters as they are meant to be in Cloud Analyst or with some alterations, ensuring that both the simulators yield the same result for a given scenario. The output metrics mainly focus on the request's response time and resource utilization factors [19].

3.4 Need for simulator

The described scenario in Section 3.2 has been a common and challenging issue for many mid-ranged companies in recent times. There are easy methods to measure the yielding of a system, but it is difficult to estimate which will yield favoring results. Setting up a distributed environment on the Internet and then performing experiments to test a research and development problem is a costly affair.

We implemented the proposed model described in Section 3.3 and a simulation tool, namely, Request Assignment Simulator (RAS), evolved. RAS is a customizable, user-friendly tool capable of simulating the process of request assignment on a large-scale distributed computing environment. RAS is modeled to fit into any cloud architecture, particularly the IaaS cloud model. This simulator can be a test bed tool for any web service provider planning to subscribe for IaaS. Assuming that the objective of the web service provider is to achieve maximization of resources, these kinds of simulators can help to derive a rough estimation of the performance in terms of resource utilization and response time.

3.5 Components of RAS

The following are the constituting components of RAS, which simulate cloud-based architectures:

User Base: Region classification and user profile maintenance are the two major objectives of this module. User base simulates the creation of the user, region, and their requesting service with time stamp. RAS categorizes the world into six zones with each continent representing a zone. Each request is generated along with the zone identification, using which the user is grouped. A single user can make any number of requests, but each request is considered as a separate job. The user base module derives consolidation of the requests with time quantum, averages the requests, enumerates requests' peak time, requests' peak region, ranking the services, and traffic pattern.

Cloudlet: Users' requests are grouped together, and this is termed as a Cloudlet [20]. Each cloudlet is consolidated for a time quantum. A Cloudlet maintains the details of the requesting service, arrival time of the request, size of the requested service, and user identification. Each request is generated with random process time. Requests are queued based on a first-come, first-served principle.

Cloud Dashboard (CD): This is the main configuration setup panel that simulates the distributed resource management controller in cloud architectures. It models the resources spread across data centers, networks, virtual machines, and physical hardware components in the cloud. CD includes the following structural elements:

(a) Data center profile: Each data center is characterized by its identification number, name, and bandwidth.

(b) Services: This component models the services available in the data store. Each service is configured with its size and unique identification. The data store is modeled as an object repository, and each object is instantiated whenever there is a request. A service is ranked based on the demand pattern. Demands for services are randomly generated, which automatically reconfigures its ranking.

(c) Virtual machine (VM) profile: This profile maintains virtual machines configuration summary mounted on each physical machine. It includes a number of VMs, VM ids, image sizes, memory capacities, architectures, operating systems, as well as a number of connections that it can handle in parallel.

(d) Network metric: This is modeled as a matrix representing connection costs between different zones.

CD serves as a master configuration panel. RAS lists the parameters that are pertinent to assess the performance of the system. The user has to assign the values for the parameters and set up the configuration. Load balancing principles are influenced by some of the impacting parameters arising from different aspects of cloud systems [21]. **Table 1** shows the list of the identified parameters and their description that are used in RAS. We have chosen only a few parameters that have a high impact on the system and that also suitably fit our proposed model.

Table 1. List of parameters used in RAS

Domain	Parameters	Description
User	UB_Prox	Geographical grouping of users with a data center
	Node_Policy	Neglecting a request by a data center
	User_Prefer	User's selection for a specific data center
Network	NW_Traffic	Rate of data transfer per time unit that changes from time to time
	NW_Conn_Cost	Transmission latency and data transfer delay between the nodes
	NW_BW	A hypothetical measure of the rate at which the data is transferred from one point to another in a given unit of time
Node	Load_Cap	Maximum number of sessions a node can accept
	Mem_Cap	Maximum memory size of a node
	Proc_Arc	Processor type
	Fail_Node	Event of failure of a node
	Suit_Proc	A service suitable for a processor to process
	Suit_OS	A service suitable for an OS to process
Requests	No_Reqs	Demand for a service at time *ti*
	Size	Size of a requested service
	Process_Time	Amount of time the node spends on a request
	Queue_Size	Number of requests waiting for the resource assignment

Strategy Assembly (SA): SA lists a set of load balancing principles, each favoring different objective functions. An objective function intends to achieve the goal in terms of performance metrics, whereas load balancing is the key principle that determines the

performance. In RAS, the user has to choose his preferred principles for an experiment.

RAS supplements the load balancing principles of Cloud Analyst and also accommodates other principles. Apart from the conventional load balancing principles listed in the strategy assembly, we included a new strategy, namely, the capacity proportioned load balancing principle. In essence, this principle assigns requests to each computing node based on its capacity. By selecting different principles for the same set of input and configuration, users can visualize how the performance varies.

The following list gives the category of load balancing principles experimented in RAS:

a. Random assignment:
    1. Random nodes with random requests
    2. Pre-ordered nodes with random requests
    3. Randomly pre-ordered nodes
b. Round robin:
    4. Ordered circular with max load capacity
    5. Ordered circular with min load capacity
    6. Ordered circular with max storage capacity
    7. Ordered circular with min storage capacity
c. Throttled load balancing:
    8. Max load capacity with random requests
    9. Max storage capacity with random requests
    10. Max requests with max load capacity
    11. Max requests with min load capacity
    12. Max requests with max storage capacity
    13. Max requests with min storage capacity
    14. Max load capacity
    15. Min load capacity
    16. Max storage capacity
    17. Min storage capacity
d. Equal requests split load balancing:
    18. Equal requests with pre-ordered nodes
    19. Equal requests with max load capacity
    20. Equal requests with min load capacity
    21. Equal requests with max storage capacity
    22. Equal requests with min storage capacity
e. Capacity proportioned load balancing:
    23. Equal load with pre-ordered nodes
    24. Equal load with max load capacity
    25. Equal load with min load capacity
    26. Equal load with max storage capacity
    27. Equal load with min storage capacity

Request Assignment Table (RAT): Once the cloud dashboard is configured with appropriate values, RAS executes the chosen load balancing principles. Successful execution results in the generation of the Request Assignment Table. This table assigns the job pool to VM pool subjective to capacity constraints.

RAT is not a job schedule for the compute nodes, but it keeps track of the assignment of compute node as soon a request has been allotted. The cloud resource manager is responsible

for resource allocation and actual execution of tasks. Task tracker, a component of CRM, monitors the execution of the tasks by the computing environment [22]. Whenever there is a change in the cloud infrastructure, CRM updates the Cloud Dashboard.

Performance Evaluator: Some of the objective functions of load balancing principles are to maximize resource utilization, minimize response time, and minimize cost. The RAS model strives to attain maximized resource utilization with a minimized response for prioritized services. The performance evaluator in RAS computes various metrics to assess the performance of the system for a given configuration setup. We consider each VM as a compute node and the current rank of the service as the value. Experiments were conducted repeatedly with the same setup and then the results were consolidated. **Table 2** shows the list of performance metrics calculated for each experiment. **Fig. 2** presents the sequence diagram that depicts the interaction of the components of RAS organized according to time.

Table 2. Parameters used for assessing the performance

Calculated parameter	Calculation
Response time	Request's service start time
Wait time	Request's service start time – Arrival time
Average wait time	∑ Wait time / Total number of requests
Turnaround time	Process time + Waiting time
Average turnaround time	∑ Turnaround time / Number of requests
Total value earned by a node	Service * value / Number of services
% of total value earned by a node	∑ Item * value * 100 / Number of requests
Total value earned by a node against $Item_n$	$Item_n$ * value / Number of requests
% of Total value earned by a node against $Item_n$	∑ $Item_n$ * value * 100 / Number of requests

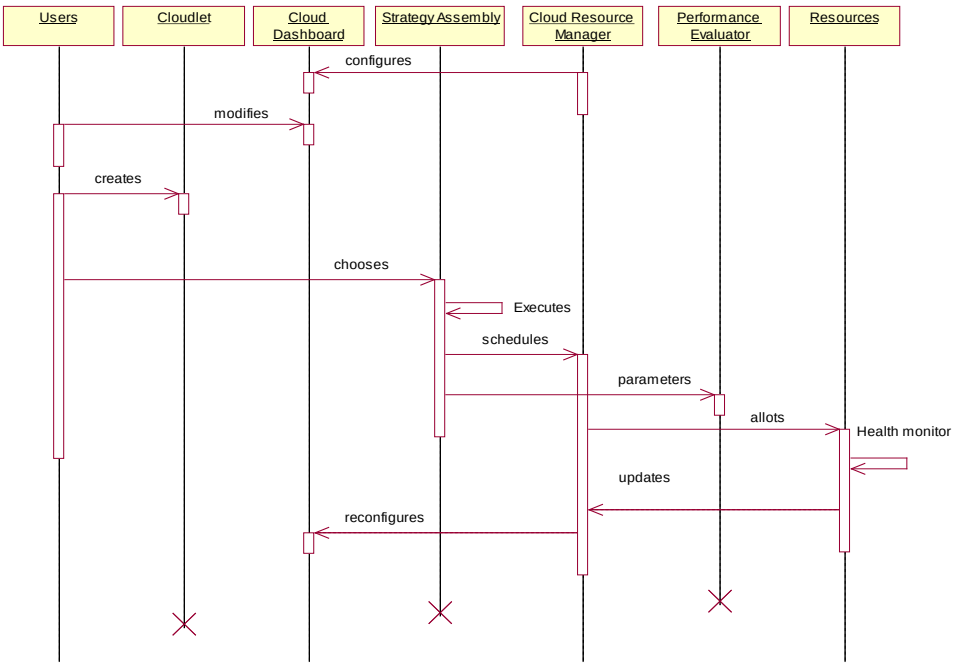

Fig. 2. Architecture of RAS components

3.6 Method of performance evaluation

The performance evaluation is carried out in two phases. Phase 1 quantifies the capacity of a node using the z-score method. The z-score is a statistical measurement of an observation's relationship to the mean in a group of observations [23]. Using this value, we calculate requests proportion for each compute node. For the performance comparison, we keep capacity proportioned principle as the scale to assess the relative performances achieved by other principles. Phase 1 calculation is shown below:

Step I: Find Mean *(μ)* & Standard deviation *(σ)*

$$\mu = \frac{\sum Xi}{n} \tag{1}$$

$$\sigma = \sqrt{\frac{Xi - \mu}{n - 1}} \tag{2}$$

where *Xi* is each value of observed parameter and *n* is the number of values.

Step II: Calculate z-score *(Z)*

$$Zi = \frac{Xi - \mu}{\sigma} \tag{3}$$

Step III: Find the z-score from the standard normal table.

A standard normal table or Z-table is a mathematical table for the values of Φ, which are the values of the cumulative distribution function of the normal distribution. We map Z-value in the Z-table to get a numeric value.

Step IV: Convert Z-score into a percentage (Z^{per})

$$Z^{per} = Z * 100 \tag{4}$$

Step V: Calculate requests' split percentage *(Si)*

$$P = \sum Z^{per} \tag{5}$$

Let R be the total requests for a service A:

$$Si = \sum_{i=1}^{n} \frac{R * Zi^{per}}{P} \tag{6}$$

Step VI: Compute the performance by consolidating the handling capacity factor and average response time as follows:

$$Min\ (A) = \sum_{i=1}^{n} (Ei - Si) \tag{7}$$

$$Min\ (B) = \sum_{i=1}^{n} (r_{i(start\ time)} - r_{i(arrival\ time)}) \tag{8}$$

where *A* is the handling capacity factor, *B* is the average response time for the total requests, *E* is the number of requests expected to be served by a compute node that is computed by phase 1, *n* is the number of computing nodes, *R* is the total number of requests, and *S* is the number of

requests actually served by a compute node.

Thus, we calculated the resource utilization factor and average response time. The goal is to identify the principle that maximizes the resource utilization and minimizes the response time of the requests.

Performance comparison: Since handling capacity factor and response time are inversely proportional to the system's performance, we conclude that the principle that yields the minimum value for the above-said parameters are the most efficient ones. The user has to make a trade-off between the objective functions to choose the suitable load balancing principle.

3.7 Features of RAS

Dedicated scheduler tool: Even though there are many contributions on load balancing principles in the cloud, only a few works are focused on scheduler simulators. From the literature review, the authors are confident that RAS is one among the pioneer works in RAS tools.

Graphical User Interface: The front end of RAS is designed in a user-friendly manner facilitating the user to execute experiments in a comfortable manner. After setting up the initial configurations in the dashboard, the RAS wizard assists the user to execute the experiment in a step-by-step method making the process transparent and clear.

Easy configurability: Setting up the configurations for an experiment is made easy because the tool is essentially interactive. Default value setup and metrics for the parameters gives better clarity for the user to input. Before execution of the scheduling strategies, the user is presented with a consolidation of configuration setup. Therefore, a user can concentrate on the system's performance instead of worrying on how to make it perform. **Fig. 3** presents a configuration panel of RAS.

Random generation of inputs: Automatic generation of user base and cloudlet emulates how the requests are originated in the cloud. The user also has the provision to set the value range for the parameters.

Repeatability of experiments: Each experiment is saved with a unique identification, which enables the user to open a particular experiment at any time. The configuration setup can also be changed, and the experiment can be executed once again. This feature allows the user to evaluate the impact of the changes that he made with the system. **Fig. 4** presents the strategy assembly of RAS for an experiment.

Support of different formats of input and output: Cloudlets can be created either by random generation or by importing the input data file in .txt, .xls, and .arff formats. Similarly, the output of the experiment is exportable in .txt, .doc, and .xls formats.

Consolidation of results: If the experiment is repeated for different cloudlets but with the same configuration setup, RAS cumulates the results in a single sleeve. When the experiment is executed repeatedly with varying request sizes, the performance evaluator computes various performance metrics to identify the order of favoring principles.

Graphical output: Wait time of the request, average wait time, response time, average response, turnaround time, average turnaround time, total value earned by a node, average value earned for a unit time, and percentage of resource utilization of the nodes are some of the metrics calculated by the Performance Evaluator. The comparison between these parameters is plotted graphically and presented in **Fig. 5**.

RAS wizard plots the chart as the final step of the experiment which helps the user to export data and plot it on a spreadsheet. RAS supports 53 different types of charts. Users can also export these charts.

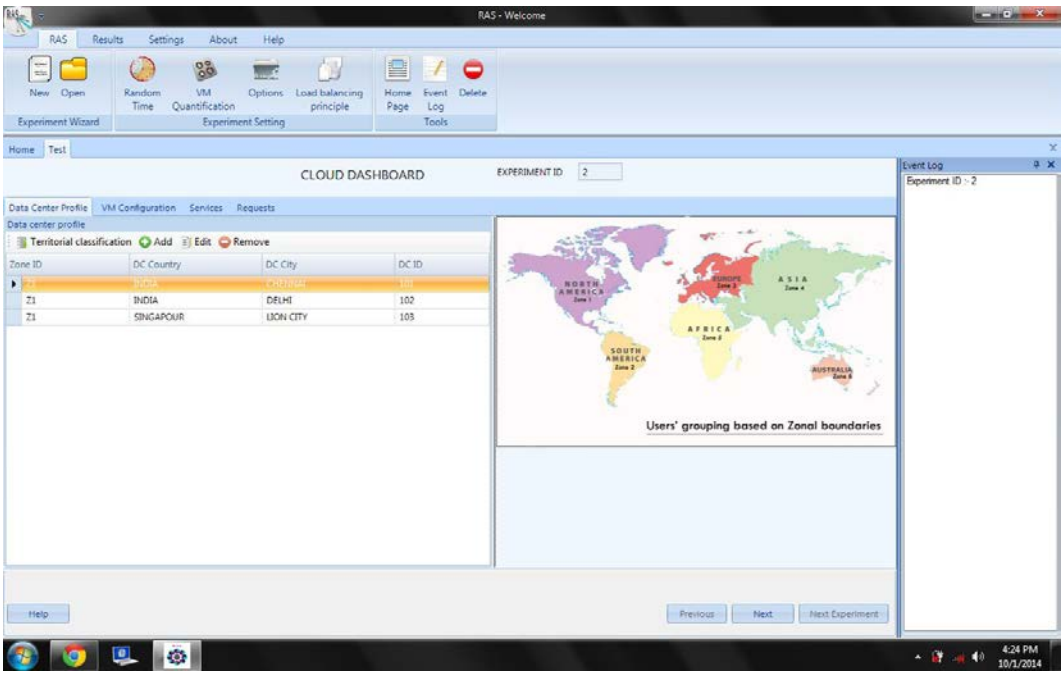

Fig. 3. Setting up the configuration in RAS

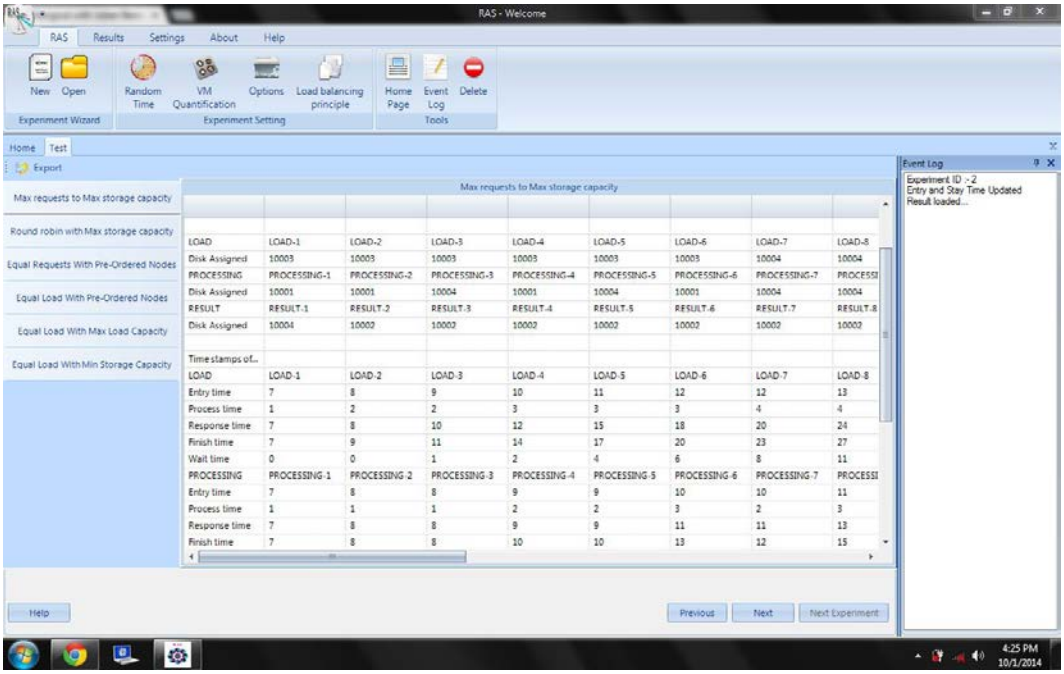

Fig. 4. Requests assignment with nodes

4. Conclusion and Future Directions

Although there has been significant progress in developing simulation tools in recent years, which demonstrates how cloud can be modeled and tested in its different thrust areas, RAS is the forerunner in the direction of dedicated simulation tools for the request assignment process, which evaluates the performance changes with respect to load balancing principles. To hypothetically study the nature of load balancing principles in a large-scale distributed computing environment, we modeled a simulator with an objective policy, impacting parameters and its principal constraints. We observed that the change in the load balancing principle and its influencing parameters has a significant impact on the system's performance. Experimental results identify the most fitting load balancing principle for the problem scenario, and the performances of other load balancing strategies are also compared. Therefore, the authors are confident that this tool will be a test bed for the academia and cloud designers who are focused on service broker policies.

The model presented in this paper focused on the server's perspective of achieving a higher rate of utilization with available resources. Achieving minimum response time or reducing the computing costs can also be the objectives. Identification and inclusion of new objective policies, parameters, and constraints will give more space and dimension to extend the existing model in future.

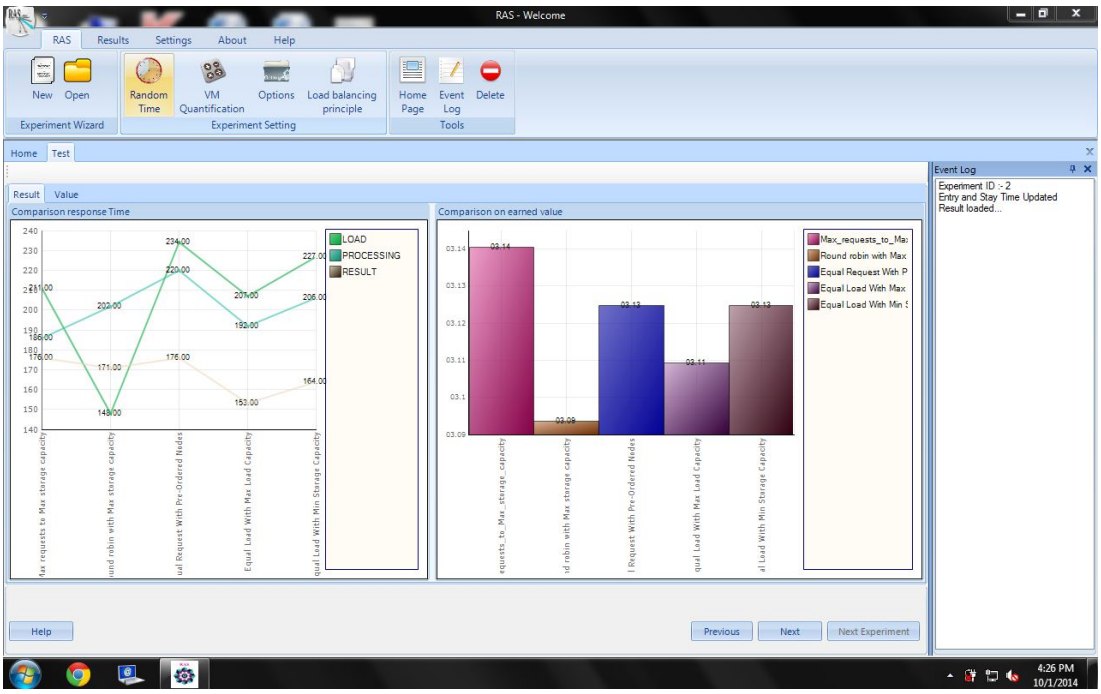

Fig. 5. Performance comparison of different principles

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," *Communications of the ACM Magazine*, vol. 53, issue 4, pp. 50-58, 2010. Article (CrossRef Link)

[2] A. Velte, T. Velte, R. Elsenpeter, *Cloud Computing, A Practical Approach*, McGraw-Hill Education, 2009.

[3] Jinesh Varia, "Amazon Web Services - Architecting for The Cloud: Best Practices," *http://media.amazonwebservices.com/ AWS_Cloud_Best_Practices.pdf*, 2011.

[4] K. Purdy, "How Google Apps works when actual people use it," *http://www.techrepublic.com/blog/google-in-the-enterprise/how-google-apps-works-when-actual-people-use-it/*, 2011

[5] Rackspace Open Cloud Reference Architecture, *http://www.rackspace.com/knowledge_center/article/rackspace-open-cloud-reference-architecture*, 2013.

[6] R. Buyya, J. Broberg, A. Goscinski, *Cloud Computing: Principles and Paradigms*, Wiley Press, New York, USA, 2011.

[7] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, J. Al-Jaroodi, "A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms," *Second Symposium on Network Cloud Computing and Applications (NCCA)*, pp. 137-142, 2012. Article (CrossRef Link)

[8] R. Buyya, M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, Concurrency and Computation: Practice and Experience (CCPE)," vol. 14, no. 13-15, pp. 1175-1220, USA, 2002.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no.1, pp. 23-50, Wiley Press, 2011. Article (CrossRef Link)

[10] G. Ramos, Martins, "ClusterSim: a Java-based parallel discrete-event simulation tool for cluster computing," in *Proc. of IEEE International Conference on Cluster Computing*, pp. 401-410, 2004. Article (CrossRef Link)

[11] A. Núñez, L. Jose, Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero and Ignacio M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator," *Springer, Journal of Grid Computing*, vol. 10, issue 1 , pp. 185-209, 2012. Article (CrossRef Link)

[12] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, "GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds," in *Proc. of Euro-Par 2010 Parallel Processing Workshops*, pp, 305-313, 2010. Article (CrossRef Link)

[13] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, M. Alsaleh, "TeachCloud: A Cloud Computing Educational Toolkit," *International Journal of Cloud Computing (IJCC)*, vol. 2, no. 2-3, pp. 237-257, 2013. Article (CrossRef Link)

[14] B. Wickremasinghe, R. N. Calheiros, R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *Proc. of 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 446-452, 2010. Article (CrossRef Link)

[15] Brucker, Peter, Scheduling Algorithms, *Operations Research & Decision Theory*, Springer, 5th Edition, 2007.

[16] K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, J. Al-Jaroodi, "A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms," in *Proc. of Second Symposium on Network Cloud Computing and Applications (NCCA)*, pp. 137 - 142, 2012. Article (CrossRef Link)

[17] Reverso web services, *http://www.reverso.net/text_translation.aspx?lang=EN*.

[18] Amazon EC2 services, *docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/concepts.html.*

[19] V. Vineetha, "Performance Monitoring in Cloud: Building Tomorrow's Enterprise," *http://www.infosys.com/engineering-services/features-opinions/Documents/cloud-performance-monitoring.pdf*, 2012.

[20] F. Lin, X. Zhou, D. Huang, W. Song, D. Han, "Service Scheduling in Cloud Computing based on Queuing Game Model," *KSII Transactions on Internet and Information Systems*, vol. 8, no. 5, pp. 1554-1566, 2014. Article (CrossRef Link)
[21] R. Arokia Paul Rajan, F. Sagayaraj Francis, "Dynamic Scheduling of Requests Based on Impacting Parameters in Cloud Based Architectures," in *Proc. of the 48th Annual Convention of Computer Society of India*, Advances in Intelligent Systems and Computing, Springer International Publishing, vol. I, series 248, pp. 513-521, 2014. Article (CrossRef Link)
[22] A. Gulati, G. Shanmuganathan, A. Holler and A. Irfan, "Cloud scale resource management: Challenges and techniques," in *Proc. of 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 2011)*, 2011.
[23] S. C. Gupta and V. K. Kapoor, *Fundamentals of Mathematical Statistics*, 14th Edition, Sultan Chand & Sons, India, 2014.

F. Sagayaraj Francis is working as Professor in the Department of Computer Science & Engineering at Pondicherry Engineering College, Pondicherry, India. He holds Ph.D in Data Management from Pondicherry University, India. He published research papers in 15 journals and 13 conference proceedings at international level. His areas of interest includes Database Management Systems, Data Mining and Knowledge Discovery, Data Structures and Algorithms, Knowledge and Intelligent Systems and Automata Theory and Applications.

R. Arokia Paul Rajan is a research scholar in the Department of Computer Science and Engineering at Pondicherry Engineering College, Pondicherry, India. His research area is data management in distributed systems. He published 9 research papers in international journals and conference proceedings.