QoS-based and Scalable Scheduling of OpenSees Tasks in a Virtual Cloud Computing Environment

Dr. Hadi Salimi[1], Seyed Emad Fereshteh Nezhad[2], Dr. Mohsen Sharifi[3]

[1.] Department of Computer Engineering, University of Science and Technology, Iran
[2.] Msc of Computer Engineering, University of Science and Technology, Iran
[3.] Department of Computer Engineering, University of Science and Technology, Iran
emad_fereshtehnejad@yahoo.com

Abstract: In recent years, the amount and complexity of computations required for solving our everyday problems have increased considerably. These problems require higher computing power hence higher number of computing components. Different methods are used for solving such problems such as parallel machines, computer clusters, grid systems and cloud systems. An important issue in all these approaches is scheduling the tasks taking into account Quality of Service (QoS) considerations. This article offers QoS-based algorithm in a cloud computing context. QoS parameters considered in this study include cost and reduction of the average response time. The basic procedure for this algorithm is that every user requests for QoS parameters is valuated using index method. In addition, having received user request, computations are performed and every request is given a priority with the aim of minimizing the cost and according to resource utilization. The result of this valuation of user requests is first used for scheduling activities in real environment. These indexes are then used in the system's execution queue to determine the priority of user request. The proposed algorithm is implemented in a real environment using the OpenSees software. Results from the computations indicate that using the proposed algorithm improves the average execution time by 15% compared to the EDF (Earliest Deadline First) method while increasing the number of OpenSees execution requests. It also reduces resource utilization cost proportionately by 5%. This study reveals that using the proposed method increases the number of requests and at the same time fewer requests are missed.
[Hadi Salimi, Seyed Emad Fereshteh Nezhad, Mohsen Sharifi. **QoS-based and Scalable Scheduling of OpenSees Tasks in a Virtual Cloud Computing Environment.** *Researcher* 2013;5(12):60-68]. (ISSN: 1553-9865). http://www.sciencepub.net/researcher. 7

Keywords: Scheduling, Tasks Graph, OpenSees, Cloud Computing

1. Introduction

At the beginning of the 90s when the number of local workstations and cluster computers were not sufficient to perform scientific work loads, scientific committees started to construct some groups through which they could exclusively share their existing clusters. Thus, the idea of grid computing was formed. The common factor between cluster computing and grid computing models was that through both of them empty and unused computing capacity of workstations was shared.

This sharing was typologically different from one problem to another and did not always guarantee higher efficiency. The grid computing model has had the highest influence in developing scientific works. It has been claimed that the users of a grid environment have easy access to a high number of computing nodes and by adding more computing resources enhance the speed of producing output [2].

Cloud computing is a model based on the internet by which computing resources are shared through a cloud environment and the internet interface. Softwares, information, cloud environment and all other devices are available to users as public tools. Cloud computing is a new supplementary, practical, and commercial model for internet-based IT services. Cloud computing is claimed to be dynamically scalable and most of its resources are found on the internet in form of services.

In the last few years, Amazon cloud computing system, EC2, and other commercial cloud systems have offered computing services with defined service quality. For specific time intervals (usually in hours) until the request time ends, cloud computing resources can only be used for the same interval of time. The flexibility of cloud environment allows users to customize computing resource for several hours or a number of resources for one hour.

One of the challenges of scheduling resources in a cloud computing environment is selecting suitable computing resources in order to execute one user request. In fact, it is impossible to predict which computing resource group has higher efficiency for the user's request. Therefore, the main issue in this phase is allocating certain number of computing resources to an activity in a particular interval of time.

In this article, it has been attempted to offer a strategy that could make selecting and combining QoS parameters simpler by indexing them. QoS parameters considered in this study include the

cost of executing the program on virtual machines and the reduction of average execution time. The general procedure for this approach is that once program execution request is received from the user in the form of a workflow of activities, the QoS parameters chosen by the user (service-level agreements) are valued using indexing method. In addition, once user request is received and assuming that execution is done with minimum cost, computations are simultaneously done according to the amount of resource utilization and prioritized. Result obtained from the valuation of user request is first matched with the result from resource computing to check the feasibility of execution. The result of matching user request valuation and computing resources are then used to determine user request priority in execution queue. Using this strategy, ideal conditions could be matched to real conditions. Results from this evaluation show that using this method causes an average reduction of 15 per cent in resource utilization. It also reduces the average execution time by 15 per cent. The rest of this article is organized as follows: section 2 deals with related works about workflow scheduling in cloud computing. Section 3 explains the hypotheses and models introduced in this paper. In section 4, the architecture designed for scheduler system is described. Section 5 explains how different parts of the main scheduler work. Section 6 describes execution queue holder and the algorithm monitoring execution queue. In section 7, the proposed method is presented in a real environment. Finally in section 8, conclusion is made and future works are explained.

2. Related Work

In [6], workflow scheduling in grid computing is studied. This is study is a two-condition scheduling model. In this study, DCA strategy is used to optimize problem with two independent conditions of cost and execution time. Selected algorithm first chooses the main condition and the user sets the variability percentage of the second parameter. This strategy does not include service quality requirements. Neither does it distinguish between resources and QoS. Finally, it does not use task categorization to reduce bandwidth usage.

In [8], a scheduling algorithm is proposed which is based workflow cost for real-time applications. The purpose of this algorithm is to develop a scheduler that could minimize cost, but is limited by time restrictions applied (forced) by the user. The incoming workflow is divided into subtasks in order to form a direct flow. Jobs\tasks that cannot form a flow are separated and each of them is executed in an independent subcategory.

The method proposed in [7] first provides different users, each of whom needs certain QoS parameters, with different services. Then, in order to present workflow from occurring at the same time, one strategy for scheduling several workflows with different QoS limitations has been proposed. In addition, it introduces a scheduler to check the requirements of optimizing QoS parameters, minimizing execution space and cost by optimizing where softwares are placed.

In [5], a model is proposed based on QoS which allows performance degrader elements to be identified and problem be detected through a fault tolerance strategy.

In [9], a heuristic, greedy method is introduced and in [10] homogeneous earliest finish time algorithm is defined. In both articles, QoS parameters are disregarded. Also, in both methods it is assumed that there is a workflow graph and the strategy is not suitable for real conditions.

In [12], a strategy with throughput maximization for scheduling workflows with heavy transactions is proposed. However, this method is not suitable for scheduling several workflows because by workflows with heavy transactions we mean executing several examples of the same workflow. Therefore, this strategy is primarily useful for one workflow and the notion of several workflows requires that every workflow be completely different with the next one in terms of structure and number of transactions.

In [13], a guided-planner strategy is proposed for scheduling several workflows. This strategy first values all jobs and then decides which one should be done first. In this procedure, executing the tasks with lower value is constantly postponed. In addition, in this strategy, QoS requirement parameters such as cost are not taken into account.

In the strategies introduced, conditions of the execution environment (the bandwidth used, exhaustion of resources) are not taken into account. Also, prioritization or variability ofQoS parameters and matching them to the user's request are not taken into account. In this study, it has been attempted to introduce a strategy in which conditions of the real execution environment are met as well as some QoS parameters such as execution time.

3. Hypotheses and Models

In this paper, two models are considered for presenting the procedures. These models include infrastructure model and software model characteristics of which are explained below.

3.1. Infrastructure Model

In the infrastructure model, a series of physical machines are considered. A virtual monitoring machine VMware ESXi4.1 is installed on

every physical machine. At least four virtual machines are installedon each physical machine by this monitoring machine. This series along with their subsystems are interconnected through a local network. All physical machine are independent of each other and can be on or off independently. This is also true for virtual machines i.e. each installed virtual machine can be on, off or on suspend independent of the other machines. The defined specifications of all physical and virtual machines are identical. Each physical machine is equipped with an Intel Core™2 Duo 2.2 GHz and 4GB of main memory. The operating system installed on each virtual machine is a Windows XP along with supplement V.3. Therefore, all systems are of the same type and identical.

3.2 Software Model

As previously mentioned, OpenSees and TCL Editor softwares have to be installed on each virtual machine. The fourth version of .Net Framework has to be also included in each system. Finally, a copy of the software must be installed on the systems. This software is implemented in order to connect the systems, receive different parts of the program, transfer the results obtained from execution and to report how the OpenSees software is operating.

4. System Architecture

In the architecture of the proposed design as showed in Fig 1 is considered. The user first gives the system the request to execute simulation program along with service level agreement. Every request is first delivered to the super scheduler. The super scheduler consists of three subsections: 1 the section related to receiving user requests; 2. the section responsible for calculating the average time of completing the request in different states and 3. The section setting request priority and producing a list of requests has being executed. After receiving the request, the amount of resources required to execute user request is estimated. In addition, in this computation the degree of a request's priority and the number of resources with which it should be executed are also determined. Finally, the result of these computations is matched with the user request and the system selects the appropriate execution. Then, the request along with the properties of the estimated resources is delivered to the monitoring section to be placed in the execution queue (Fig. 1. Part B.). In this section, the request is placed in the execution queue according to the proposed algorithm. The requests are delivered to the core of the computing cloud (Fig. 1. Part C) in certain time.

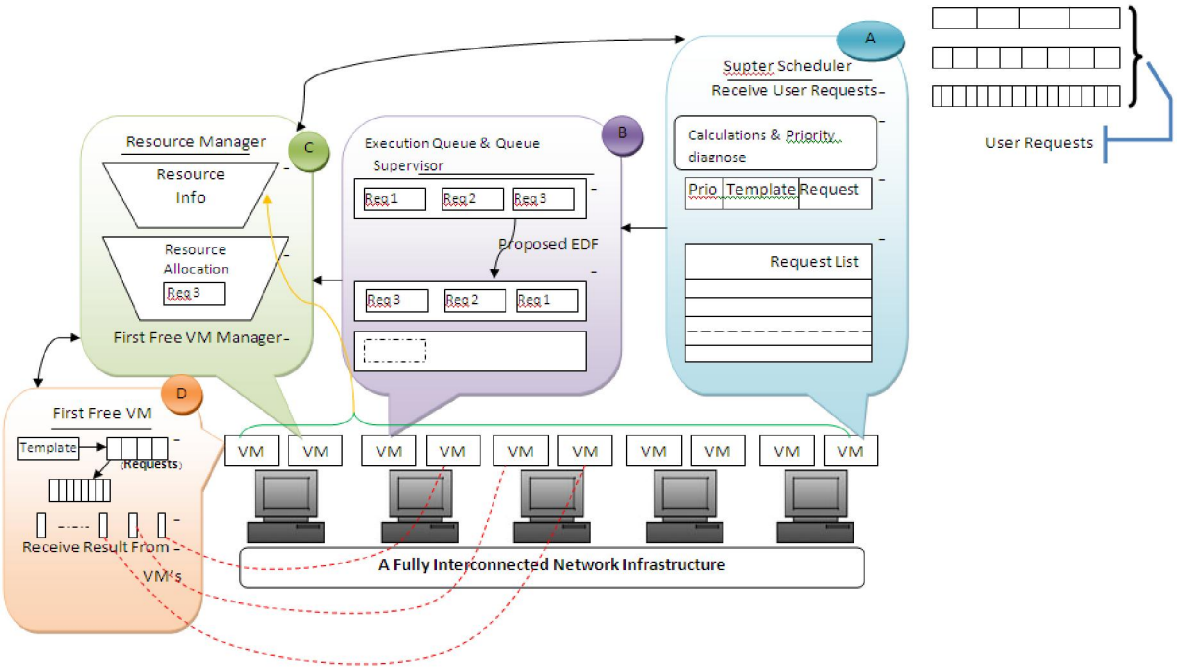

Figure 1: proposed System Architecture

In the core of the proposed strategy, every request is given to the first free virtual machine (Fig. 1. Part D). From that moment on, that virtual machine would act as the initiator and controller of execution process of the task graph. Having received user request, the initiating machine will demand resources from the resources management system according to the properties determined in previous sections and executes the request.

5. Procedure of the Super Scheduler Stages

As shown in Fig. 1, user's request is given to the system first. This request is checked by the first subsection. In this section, the user's request is checked with the assumption of "executing request using one hundred percentage of the virtual machines' computing capacity." This assumption has two advantages: firstly, if the system does not have sufficient resources to execute the request in the shortest time possible, it could be executed with the lowest cost. Secondly, it is a benchmark to identify the user request's degree of promptness. Using this checking, a flag is set which registers the promptness of the request. If in this state the user's request could be executed with one hundred percent of the computing capacity in the identified time, we could easily place it in the system's execution queue taking into account the parameters and prioritization of users' requests. Nevertheless, if using one hundred percent of the computing capacity did not yield an execution in that given time, the request will be on a critical path. It means that in order not to miss the request more resources should be used for the execution to be done in the deadline specified. Therefore, the special flag of this issue is set to the critical state. In this section, completion time and the deadline defined for it are interrelated in four states:

1. Jobs with short deadline and long completion time;
2. Jobs with short deadline and short completion time;
3. Jobs with long deadline and long completion time; and
4. Jobs with long deadline and short completion time.

In subsection two, estimation of the user's request execution resources is computed assuming that different computing capacities are used. It means how much completion time will it have if user's request is executed with 25% of the virtual machines' computing capacity? Where in the execution queue will it be? How much will execution cost be in this state?

This computation is performed for utilizing 50%, 75% and 100% of the capacity. Choosing this categorization and defining coefficients for computing power usage results in simplification in two respects: firstly, a program can be easily divided into several sections and secondly, computing capacity is standardized to create cost-benefit conditions. If user requests are executed using 25% of the virtual machines' computing capacity, there will be short completion time and high execution cost and if they are executed with 100% of the computing capacity, there will be long completion time and low cost.

These computations are delivered to the third subsection of the super scheduler. This subsection is responsible for distributing user requests among virtual machine for execution and scheduling. Its other responsibility is to manage users' requests and create a list of the jobs that are being executed. In the next stage, user's request is delivered to the scheduler along with the four computations. This scheduler compares these computations with the agreement given by the user. This selection determines how user's request is executed. The agreement submitted by the user can be in one of three states. In order of priority value, these states are: the state based on priority, the state based on deadline, the state based on the lowest cost.

After matching and selecting the manner of execution, user's request along with this supplementary information are sent to the queue holder section and monitoring section illustrated in Fig. 1.B. This section plays an important role. It has to determine where in the queue the user request should be placed. In this section, in addition to the usual execution queue, a second queue is formed in which missed jobs are placed. Jobs placed in this queue enjoy a special priority because they must be executed first using customized resources and then the jobs already present in the first queue. Finally, after the user request is placed in the related queue, it will be sent to the resource management section mentioned in Fig. 1.B.

1. Receive users' requests along with service level agreements.
2. Given that one hundred percent of computing resources are utilized, estimate execution time.
3. Receive information related to computing resources from resource management subsystem.
4. If the resources required for the incoming workflow are more than the system's free resources, flag the incoming job as a critical one.
5. Estimate the required time for executing incoming job given that 75%, 50% and 25% of the computing resources are utilized.
6. Compare four computed information groups with the existing information in service level agreement.
7. Select the lowest possible percentage of resource utilization required to meet service level agreements.
8. For the incoming workflow, construct a structure to show priority, the requesting user and the request's priority and add it to the list.
9. Send the structure constructed in step 8 to the queue manager.

Fig. 2. Super Scheduler job processing procedure pseudo code

6. Execution queue monitor and holder

The execution monitoring algorithm is the main and the most important part of the queue holder and queue monitor (Fig 1.B). This algorithm uses a method described below to place requests in execution queue. The logic of this method is based on EDF algorithm ([3], [4]). In the computing section, in addition to computing the four execution states, maximum start time (the maximum time for request execution) is also computed. The advantage of this computation is in the fact that if the job execution start time passes the computed interval during rescheduling of jobs, the work will definitely be missed. This computation is according to the flag set in step 4 in the super scheduler algorithm to show the priority of the request.

In the monitor algorithm, the placing of the request in the queue is determined according to this information. The procedure is that in order to put the request in queue the monitor starts searching until it reach a time shorter than the request deadline. From here on, it can be placed in the interval that has the same length as the request completion time. In this state, in order to create more flexibility for the scheduler, it is better to compute the completion time as T+t. The time difference added to the completion time is according to the system fluctuation proportionate to the missed jobs. For instance, if 10% of the requests are missed in the system, it is better to add 10% of the same length of time to the completion time of each request.

The monitor constructs two queues for execution. The first queue, the common execution queue, is with A, B, C and D priorities. In the prioritization process, there is a highest priority and those requests are placed in the second queue. Among the requests placed in the second queue are the missed jobs. In order to execute jobs placed in the second queue, some of the resources are allocated exclusively. The number of these resources increases proportionate to need and concentration of the second queue. The algorithm for increasing and decreasing resources for the second queue is completely dynamic. First, we allocate 20% of the resources to the second queue. In response to the second queue's demand we increase exclusive resources. At this time, if there is a request in the first queue that would require this amount of resources, the second queue is the priority. Also, if there is a request in the second queue and the number of exclusive resources is insufficient for execution, resources should be taken from the requests of the first queue which are already being processed and only 10% of their execution is completed and the second queue's request should be responded to. If such a request was not found, second queue's request has to wait until enough resources are freed. After the second queue is emptied or the first queue no longer needs resources, they return to their initial amount.

With every request incoming to the first queue, once the monitor finds its place in the queue, reorganizes the set of requests after it. This reorganization is for checking and finding requests which may be lost with this new modification. It is essential that the resources allocated to the second queue are never less than 20% of the entire resources of the cloud environment even if there is resource shortage crisis for execution of requests in the first queue. That is because once all resources in the second queue are freed, it is difficult to get them back in the middle of the execution of the requests in first queue and it results in more requests being missed.

If the user selects the request execution model with the lowest cost, the monitor has to inform the resource management system to choose the virtual machines on the same physical machine. This also leads to the reduction of intra-network transfers and bandwidth usage. Thus, the cost of bandwidth usage is minimized.

7. Efficiency Evaluation

In order to evaluate the efficiency of the proposed design, a prototype system with the following properties was implemented. This software is constructed with the .Net 4 technology and C# programming language in Visual Studio (V. 2010) environment. This software is designed for Windows. Of course, with some modification, it also has the capability to be executed in Linux Ubuntu (Version 10). However, with respect to the infrastructure consideredfor easier execution of OpenSees programs, Windows XP environment was chosen.

The software selected for case study is OpenSees [14] software. Programs written by OpenSees are used in simulating the behavior of a structure subjected to earthquake. For this simulation, using heuristic method and numerical computation methods such as Newton, OpenSees applies pre-defined forces to the user-designed structure and saves the computation results for later analysis or next stages of processing. The structure of programs written by OpenSees can be seen in two perspectives.

First perspective: every program written by OpenSees includes three sections including preprocessing, processing and post-processing. In the preprocessing section, variables are defined, required files are created, initial values are given to the variables or files are filled with initial values. In other words, this section is responsible for defining and creating the structural model. This section is most of the time involved with parts like ROM (Hard Drive). In the processing section, certain number of

processings is done based on the definition of the variables and defied forces. Compared to the preprocessing section, the processing section requires longer time for computation. Most of the programs written by OpenSees include only these two stages. The last section or the post-processing section analyses the output data registered by the processing stage. This stage is takes a lot of time and requires high computational capacity. The results obtained in this stage are also saved in files. It should be noted that if a structure is large, the processing stage takes longer time than the post-processing stage. In this article, however, the post-processing stage in the real example is four times longer than the processing stage.

The second perspective is derived from the first perspective. As mentioned in the first perspective, every program written by OpenSees first defines some files and variables, processes them and finally analyses the results. These stages are reiterated for a certain number (n) of times. Thus, this certain number could be considered as a combination of ones. It means that the three said stages are run n times for different and independent data. Therefore, every execution could be executed separately on an independent system. In the implementation process, this perspective is applied.

The implemented software is designed with the aim of running written programs by OpenSees with reduced average execution time and controlling the cost of executing programs. In this software, the time required for executing OpenSees programs from the beginning of the execution until outputs are collected is measured. In addition, the amount of transactions created in the network for each OpenSees program executed is also measured and saved in record files.

In this software, the proposed algorithm is implemented along with the selected architecture. The result obtained from executions is compared with the usual execution time of each OpenSees program. In order to implement the algorithm and architecture we have chosen, assumptions and parameters are first defined the result of which is described below.

According to the analysis and assumptions considered, service level agreement for the user could be given to the system in three states (patterns). If the completion time is important for the user when requesting execution of simulator program, pattern 2 (Deadline-Priority-Cost) which is based on deadline will be selected. If the user requests several executions of the program and prioritizes them or in other words determines the important and the more important, pattern 1 (Priority-Deadline-Cost) which is based on priority will be selected. Finally, if the user only needs to execute the program with the lowest cost possible, pattern 3 (Cost-Priority-Deadline) which is based on cost is recommended.

Here, the cost parameter consists of two parts. Part one is the resource consumption of the virtual machines per execution of user requests and part two is the bandwidth used for transferring the outputs. According to the analyses, if a parameter affects the computation result in logarithmic terms, parameters have to be multiplied to obtain a formula for each parameter's degree of effect [15]. Therefore, it was attempted to find a suitable coefficient for each parameter showing its effect. Coefficients such as the ones in table 1 could be attributed to each parameter in each pattern.

Table 1: Computed coefficients for QoS parameters in each pattern

Pattern Name	Cost Coefficient	Deadline Coefficient	Priority Coefficient
Based on	1.5	5	13
Based on	1.5	7	7
Based on	2	3	7

The common factors in simulator programs are reiteration and performing a set of computations with different initial values and assumptions. Therefore, a written simulator program could be divided into n parts. These parts are independent of each other. By analyzing OpenSees software, we found out that a program written in OpenSees environment can ideally produce about 2 Mb of output per execution of one part (of n parts). Therefore, if a simulator program written by OpenSees could be divided into n parts, the expected output will be about n*2 Mg in the worst scenario.

Furthermore, according to the measurements, executing one time from n times in worst circumstances would take 4 minutes. Therefore, executing n times would take approximately n*4 times. We may conclude that n parts could be executed on one virtual machine sequentially which would take about n*4 minutes or execute n times simultaneously parallel to each other on n virtual machines in which case execution time would be about 4 minutes. Table t can be drawn according to these assumptions.

Table 2: Computing different states of time and resource consumption per request

Computation Number	Computational capacity utilized	Task per VM	Number of VM	Output and bandwidth usage (Mb)	Completion time (mins)
1	100%	M	K	K*M*2	4*M
2	75%	3M/4	4K/3	(4K/3)*(3M/4)*2	(3M/4)*4
3	50%	M/2	2K	2k*(M/2)*2	(M/2)*4
4	25%	M/4	4K	4K*(M/4)*2	(M/4)*4

It should be noted that if output values are computed in four states, a constant value is obtained whereas it is not so in reality. The value is constant in terms of the overall value of traffic in the network, but the four states are different in terms of network's bandwidth usage. If this constant output value is injected to the network from a virtual machine and wants to be transferred, it will never allow this amount of transaction to exceed the allowed limit (usually 30%) due to control policies on TCP/IP protocol. That is because if the number of transacting virtual machines are high, bandwidth usage will increase proportionately and at a point in time the entire bandwidth will be occupied.

Based on this information, the monitor is responsible for matching the service level agreement to the real conditions of the execution environment. For the pattern based on cost, computations 1 and 2, for the pattern based on deadline, computation 3 and for the pattern based on priority computation 4 are selected. For instance, if a request is estimated to utilize 100% of the computational capacity of priority C or D and user's service level agreement is execution based on cost, computation 1 or 2 is selected. Computation number 3 or 4 can be selected depending on the user selecting the pattern based on priority or the pattern based on deadline and also depending on computation priority obtained in the first stage of the flow graph.

The reason for choosing these assumptions for the pattern based on cost is due to the insignificance of the request's deadline or its not being the priority. Therefore, if its deadline passes in the execution queue, a flag is used which represents this pattern and simply prevents this request from being transferred to the second queue while not falling behind from the current time.

Experiments are performed on a real model in real environment. The program selected for the experiments is a program simulating the behavior of a two-story building in the city of Tehran with high seismic activity [1]. In order to perform the experiments, the requests are selected with two methods and applied to the algorithm. The first method is selecting requests with identical priority and volume. By volume we mean the number of the program's execution loops. First, in order to perform the experiment, three physical machines are used with 10 virtual machines installed on each of them. The scheduling evaluation result is shown in Fig. 3.

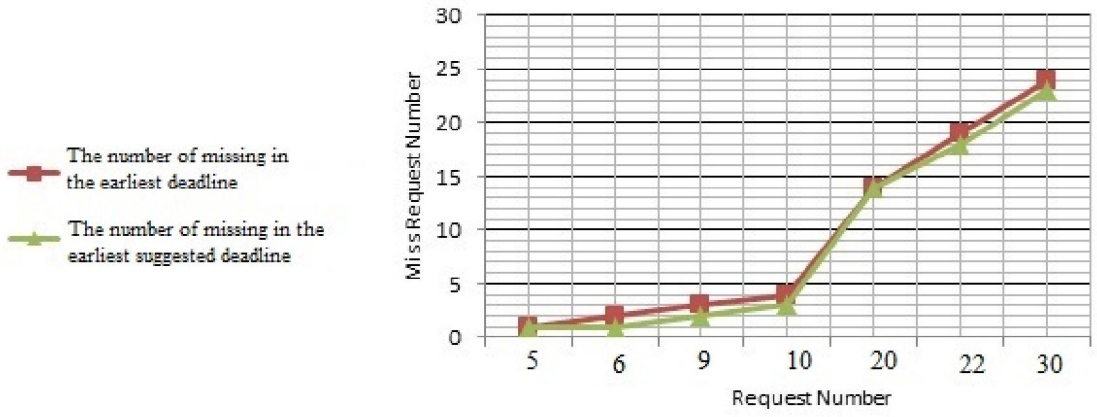

Fig. 3. Evaluation result of two scheduling method with three physical machines (requests' volume are the same and priorities different)

In this test, due to the low number of computation resources, the proposed EDF method performs the scheduling similar to EDF with negligible difference. In this test, by capturing a number of computation resources for executing requests, the proposed EDF method will have no

considerable difference with EDF method because the number of resources is low and limited in proportion to the volume of requests. In the second phase of the experiment, two scheduling methods are implemented on 8 physical machines with 10 virtual machines installed on each of them. In this phase, the volume of requests and their priorities are selected equally. As shown in Fig. 4, the proposed EDF method responds to the requests better than EDF method and fewer requests are missed.

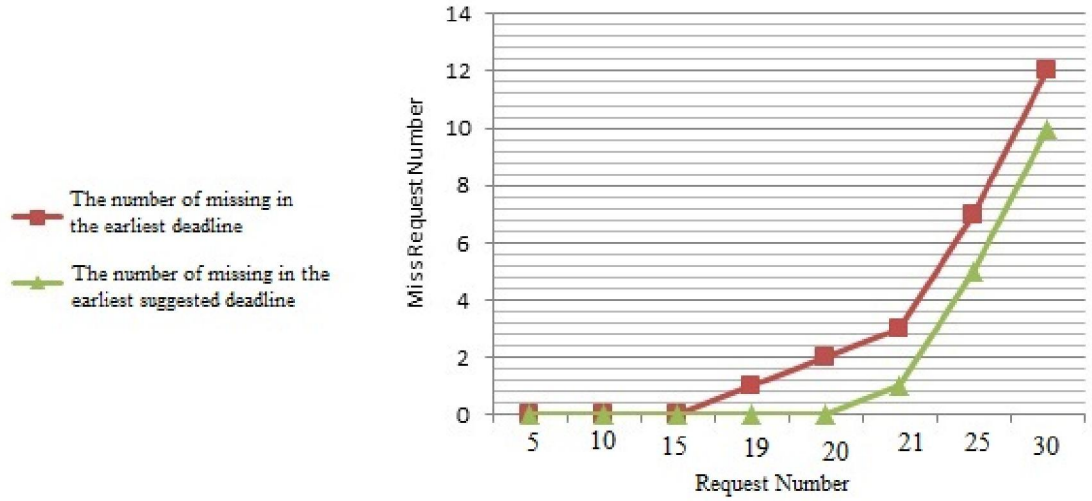

Fig. 4. Evaluation result of two scheduling methods with 8 physical machines (number of requests and priorities are the same)

In the third phase of the experiment, two scheduling methods are implemented on 8 physical machines with 10 virtual machines installed on each of them. But in this phase, the number of requests is the same and their priorities are selected different. The result from evaluation shown in Fig. 5 proves that the performance of the proposed scheduling method in terms of executing requests fares better than EDF.

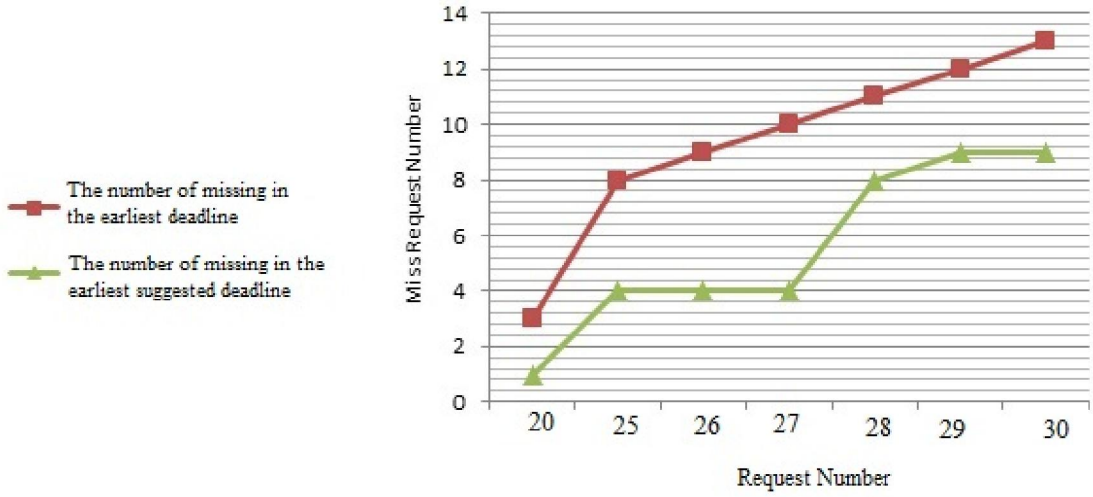

Fig. 5. Evaluation result of two scheduling methods with 8 physical machines (volume of requests are the same and the priorities different)

Evaluations indicate that the scheduler algorithm introduced here better manages the resources when the number of requests increases. This results in fewer requests being missed. Results obtained from the tests and comparing them leads us to the conclusion that the more the number of computation resources, the more optimal the proposed EDF algorithm acts in scheduling the

requests because the proposed method is able to better manage pending requests for execution using indexing method, and by constructing a second queue and capturing a number of resources.

8. Conclusion and Future Work

In this paper, a strategy was introduced by which heavy activities of the OpenSees software can be performed with better management in a virtual cloud computing environment. In addition, with respect to the selected method (indexing method) for receiving QoS parameters from the user, user requests can be valued and matched with existing resources. The proposed strategy was implemented in small scale and in a real context. Results obtained from different experiments reveal that the algorithm proposed in this paper acts on average 15 percent more optimum than EDF. This optimum behavior includes the reduction of average request execution time, reduction of resource utilization and reduction of missed requests. In addition, we may come to the conclusion that the performance of the proposed EDF performs scheduling better than other methods when the number of available resources is higher and with the increase in the number of user requests fewer requests are missed. The proposed method for receiving parameters and evaluating them has paved the way for evaluating scheduler response reliability and the effects of unpredictable accidents with the help of fuzzy logic. Future work will be extending the indexing method to different parameters of QoS and studying indexing on quality parameters such as accuracy or availability.

References:

[1] M. Banazadeh, E. Fereshtehnejad, "System reliability assessment of steel moment frames with different failure mechanisms using Bayesian Probability Network". 8th International Conference on Structural Dynamics, EURODYN 2011, Leuven, Belgium, PP 2985-2992, 4-6 July 2011.

[2] M. Baker, R. Buyya, D. Laforenza, “Grids and Grid technologies for wide-area distributed computing”. Software: Practice and Experience, PP: 1437–1466. 2002

[3] I. Foster, C. Kesselman, The Grid2, Morgan Kauffmann Publishers, 2003.

[4] C. Liu, J. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM 20 (1) (1973) 46-61.

[5] L. Ramakrishnan and D. A. Reed. “Performability modeling for scheduling and fault tolerance strategies for scientific workows”. 17th international symposium on High performance distributed computing ACM, pages 23-34, New York, NY, USA, 2008.

[6] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer. “Bi-criteria scheduling of scientific workows for the grid”, 8th IEEE International Symposium on Cluster Computing and the Grid IEEE Computer Society, pages 9-16, Washington, DC, USA, 2008.

[7] M. Xu, L. Cui, H. Wang, and Y. Bi. “A multiple qos constrained scheduling strategy of multiple workows for cloud computing”, Parallel and Distributed Processing with Applications, International Symposium on, 0:629-634, 2009.

[8] J. Yu, R. Buyya, and C. K. Tham. “Cost-based scheduling of scientific workflow application on utility grids”, 1st International Conference on e-Science and Grid Computing IEEE Computer Society, pages 140-147, Washington, DC, USA, 2005.

[9] R. Sakellariou and H. Zhao, “A hybrid heuristic for DAG scheduling on hetero-geneous systems”, 13th Heterogeneous Computing Workshop (HCW 2004), Santa Fe, New Mexico, USA, April 26, 2004.

[10] H. Topcuouglu, S. Hariri and M. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing”, IEEE Transactions on Parallel and Distribution Systems, vol. 13, no. 3, pp. 260–274, 2002.

[12] K. Liu, J. Chen, Y. Yang and H. Jin, “A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G”, Concurrency and Computation: Practice and Experience, Wiley, 20(15):1807-1820, Oct. 2008.

[13] Z. Yu and W. Shi, "A planner-guided scheduling strategy for multiple workflow applications," International Conference on Parallel Processing Workshops IEEE Computer Society, 2008.

[14] OpenSeez software specification, available at: http://opensees.berkeley.edu/OpenSees/workshops/parallel/IntroductionOpenSees.pdf

[15] J. A. Rice, "Mathematical Statistics and Data analysis", 3rd Edition, Thomson Learning Publishers, 2006.

[16] G. Galen, “What cloud computing really means?", available at: http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031, Visited: 2010-06-02.

[17] Parsian, A.; Fundamentals of probability and statistics for engineering and science students., Center, Isfahan University Press, first edition, 2005

[18] E. Walker. "Benchmarking Amazon EC2 for high-performance scientific computing". The USENIX Magazine, 33(5), 2008.

10/12/2013