Application Scheduling in CloudSim

Pradeeban Kathiravelu
KTH Royal Institute of Technology
Stockholm, Sweden
kpr@kth.se

Abstract

Cloud computing involves a hierarchy of distributed shared resources, such as processor, memory, and bandwidth and applications submitted by users. Applications have specific resource requirements and may require a considerable amount of resources to be allocated exclusively for them for a successful operation. Application scheduling algorithms are implemented to fairly schedule the jobs of the users to the available resources. This paper looks into the three families of application scheduling algorithms — strict matchmaking-based, utility-driven, and QoS-driven algorithms. Prior to test the scheduling algorithms in a real cloud environment, they are often tested on a simulation environment. This research attempts to measure the performance and scheduling efficiency of the algorithms by incorporating them in CloudSim cloud simulation tool and evaluating them against the evaluation criteria.

Categories and Subject Descriptors
D.2.8 **[Software Engineering]**: Operating Systems — scheduling
D.2.8 **[Software Engineering]**: Metrics — performance measures
General Terms
Application Scheduling, Simulation
Keywords — Application Scheduling, Strict Matchmaking-based algorithms, Utility-driven algorithms, QoS-driven algorithms, Cloud Simulation, Virtualization.

1. Introduction

Application scheduling plays a major role in a system that has resources distributed and shared among the users. Fair scheduling of the resources is mandatory for a successful execution of the system. With the ever increasing complexity of cloud environments such as the heterogeneity and geographic dispersion of the cloud resources, as well as the users submitting applications from different geographic locations, scheduling the applications has become a harder task to accomplish, making it a promising research field. Scheduling algorithms are developed to find a resource for any given job that fulfils the job's requirements. Three families of algorithms are developed to address this, naming, strict matchmaking-based algorithms, utility-driven algorithms, and QoS-driven algorithms.

Strict matchmaking-based scheduling algorithms perform well for certain tasks as they focus on total fulfilment of the specified requirement of the job which is specified as an objective function. Utility is a measure of user's satisfaction, which could be expressed as a composite of objective functions that the scheduler aims to optimize. Users have different priorities for each of their requirements and an effective scheduling algorithm that focusses to optimize the user utility should differentiate the weight or the priority assigned to each of the requirements. While strict matchmaking-based algorithms focus to optimize a specific objective function, they do not consider partial requirements satisfaction, where requirements from the users are satisfied partially based on the utility value given to each of the user requirements[16]. Hence, these algorithms do not satisfy the varying scheduling needs of the users in a real world cloud environment.

Utility-driven and QoS-driven algorithms focus to mitigate the shortcomings of strict matchmaking-based algorithms. Utility-driven algorithms consider partial requirement satisfaction, where application requirements and objective functions are considered and satisfied partially by the algorithm based on importance of the requirement to the user. QoS-driven algorithms ensure quality of service measures, in addition to existing objective functions.

This research targets to evaluate the user satisfaction of different application scheduling algorithms. Due to the complicated nature of cloud environments, Cloud Simulation tools are used in early phases of research, development, and testing of cloud applications, opposed to implementing and testing on the real cloud environments. During this research, the scheduling algorithms will be incorporated and evaluated on CloudSim[4], an open source cloud simulation tool implemented using Java.

In the upcoming sections, we will further analyse the application scheduling algorithms and how they behave in cloud environments, by studying their behaviour using CloudSim. We will continue to discuss the preliminary background information on application scheduling algorithms, in section II. Section III discusses design and implementation of the application scheduling algorithms, and how CloudSim is customized and extended to incorporate the algorithms. Section IV consists of evaluation which is a detailed discussion on experimental studies of the scheduling algorithms, efficiencies of the scheduling algorithms on CloudSim, and the results produced by the studies. Section V will drive us to the conclusion of this research, and finally section VI will discuss the possible future work foreseen by this research.

2. Preliminaries

2.1. Strict Matchmaking-based Algorithms

Strict matchmaking-based algorithms focus on a specific objective function and aims at optimizing it while fulfilling the requirements of the applications. First-come first-served (FCFS), round-robin (RR), matchmaking algorithm, minimum execution time (MET), minimum completion time (MCT), min-min, min-max, and max-min[2] are notable examples of strict matchmaking-based algorithms.

First-come first-served schedules tasks according to the order they arrived[10]. Hence a task that arrived first will be considered before the one that arrived later. Round-robin is on the other hand allocates a time slot for each of the tasks. A task will be assigned when its slot is reached, and executed as long as its slot lasts. Once the slot finishes, the slot will be reassigned to the next task in the round and the algorithm moves on serving everyone in a time-shared manner[14]. In a first-come first-served system, if a bigger task comes first, the smaller tasks that arrive later will have to wait for a long time, or even starve to death. In a round-robin system, a bigger task will have to wait for a long time to complete as time is shared among all the tasks by allocating them a time slot regardless of the submission time or the order of submission. This may lead the bigger tasks to time out.

Matchmaking algorithm finds a resource that matches the specification of an application. The specification could be requirements specified for the application such as completion time, operating system or computer architecture. In a heterogeneous cloud environment, overly constrained applications may fail to find the resource with specified requirements in a given time limit. Job scheduling success ratio could be used as a measure to evaluate matchmaking algorithm.

2.2. Utility-driven Algorithms

In a market-oriented cloud environment, objective function is generally a composite of variable requirements such as execution time as well as cost and user priorities. Users have different priorities over these different criteria. A few of these criteria may have a lower priority such that they could be considered lightly or ignored in favour of a parameter that is of a higher priority to a user. This is defined as Partial requirement satisfaction. Importance of each of these parameters is indicated by a value provided by the user for each of these parameters, which is known as the utility value. Utility value of the objective functions could be in the range from 0 to 1, where 1 indicates the highest priority whilst negligible priorities approach 0. Utility-driven algorithms consider the user- or system- defined utility functions and the user determines utility-values for these functions for his applications. Hence, these algorithms could be designed to fit specific business scenarios.

2.3. QoS-driven Algorithms

Some tasks are time constrained and their time-to-deliver is crucial for successful execution of the system. Such system or user critical tasks should be given higher priority. QoS priority-based scheduling algorithms categorize the tasks according to their priority as high and low. QoS Guided Weighted Mean Time-min (QGWMT) and QoS Guided Weighted Mean Time Min-Min Max-Min Selective (QGWMTMMS) are two such QoS-driven algorithms[7].

2.4. Evaluation Criteria

Performance and scheduling efficiency of these algorithms are measured by criteria such as job scheduling success ratio, average user utility, mean user submission time, mean execution time, mean completion time, average resource utilization, and Sufferage. Evaluation criteria for scheduling algorithms are developed based on objective functions, as they measure effectiveness of the algorithms in a straight-forward manner.

As strict matchmaking-based algorithms often focus on a single objective function, they perform excellently for those specific objective functions. Minimum Execution Time (MET) and Minimum Completion Time (MCT) stand as examples for a direct correlation between strict matchmaking-based algorithms and objective functions. Mean execution time is the average of the time that each task spent executing. Mean submission time is the average of the time taken to submit the tasks for scheduling. Completion time is the total of the submission time and the execution time, as execution follows submission. Minimum execution time algorithm targets minimizing the execution time for the tasks.

Since this only focusses on the minimal execution time for a job to be executed by a resource, the resource scheduled to execute the job may not be available or ready to execute upon the job scheduling[11]. This leads the job to wait till the resource becomes available and hence a higher completion time. Minimum completion time algorithm targets minimizing the completion time for the tasks. This does not consider the execution time, and the resource chosen by minimum completion time algorithm may have a high execution time for the job[11].

Job scheduling success ratio is a measure on how many of the submitted tasks were successfully scheduled in the considered time frame without being timed out. Sufferage is the difference between the best and the second best completion time for the given task[13]. The task that suffers the most if a resource is not assigned, is the task that has the highest sufferage. Sufferage algorithm assigns a resource to the task that has the highest sufferage for the resource, minimizing the total sufferage.

3. Design and Implementation

As access to the environments is limited, researches are implemented on simulation environments that try to mimic the real environments. Simulations empower the researchers with an effective and quicker way to test the prototype developments of their research. As cloud computing environments consist of data centers and applications distributed on a planetary-scale, cloud simulations are used in evaluating algorithms and strategies that are under research and development. CloudSim, EmuSim[3], and DCSim[15] are some of the mostly used cloud simulation environments. Simgrid[6] is a toolkit for simulation of application scheduling. OverSim[1] and PeerSim[12] are simulation toolkits for overlay and peer-to-peer networks respectively. Among these simulation environments, CloudSim is frequently used by researchers, because of its extensibility and portability.

3.1. CloudSim

Originally developed as GridSim, a Grid Simulation tool, CloudSim was later extended as a Cloud Simulation environment having GridSim as a major building block[5]. Due to its modular architecture which facilitates customizations, it is extended into different simulation tools such as CloudAnalyst[17], GreenCloud[9], and NetworkCloudSim[8]. Developed in Java, CloudSim is portable. It could easily be incorporated with scheduling algorithms with different parameters since its source code is open. For these obvious advantages, CloudSim was picked as the platform to evaluate the scheduling algorithms, and built from the source code using Apache Maven, incorporating the changes.

3.2. Architecture

Components of a basic cloud environment is depicted using the class hierarchy of CloudSim. Parameters and variables of the objects of these classes depict the characteristics of the components. CPU unit is defined by Pe (Processing Element) in terms of millions of instructions per second (MIPS). Multicore processors are created by adding multiple Pe objects to the list of Pe:s. All processing elements of the same machine have same processing power (MIPS). Similarly, hosts and virtual machines are represented by respective classes and objects. Status of a processing element could be FREE (1), BUSY/Allocated (2), or FAILED (3) indicating its availability. Cloudlet represents the entity that is responsible to represent applications. In the CloudSim terminology (hence in this report) the terms, "cloudlet", "task", and "application" are used interchangeably. Figure 1

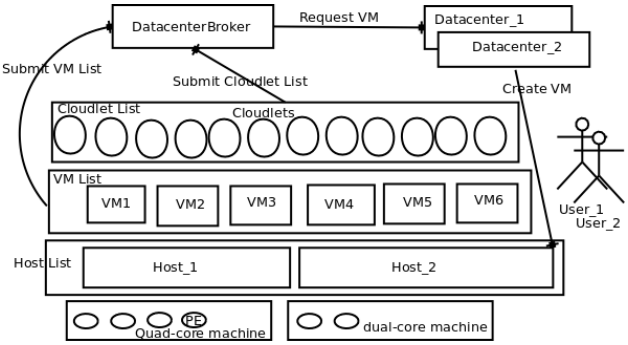

Figure 1. CloudSim scheduling operations

depicts the overview of resource scheduling and the architecture. A list of cloudlets and a list of virtual machines are created and broker decides which cloudlet to be scheduled to execute next. Hosts are defined and each of the VMs is assigned to a host. Each cloudlet is assigned to a VM, and the processing elements are shared among the VMs in a host and among the executing cloudlets in the VMs. Processing elements are the shared resources and cloudlets represent the applications that share these resources among them. Complicated real-world cloud scenarios could be simulated by appropriately extending the available classes.

3.3. Design

Scheduling algorithms are incorporated into CloudSim and evaluated using the constructs of CloudSim. The method, $init()$ calls $initCommonVariable()$, which itself calls $initialize()$ to initialize CloudSim for simulation.

```
CloudSim.init(
    num_user, calendar, trace_flag);
```

DataCenter is the resource provider which simulates infrastructure as a service. *DatacenterCharacteristics* defines the static properties of a resource. *DataCenter* is initialized by,

```
DataCenter datacenter0 =
    createDatacenter("Datacenter0");
```

Then a broker is created by calling,

```
DataCenterBroker broker = createBroker();
```

Virtual machines are created and added to a list of virtual machines, while cloudlets are created and added to a list of cloudlets. These lists are submitted to the broker. Finally simulation is started and broker handles the allocation of VMs to the hosts and cloudlets to the VMs, while deciding which of the available cloudlets to be executed next. *CloudSimShutdown* waits for termination of all CloudSim user entities to determine the end of simulation. An object of the *CloudSimShutdown* class is created by CloudSim upon initialisation of the simulation.

```
CloudSim.init() ->
    CloudSim.initCommonVariable()
```

This avoids the need to manually terminate the user entities upon completion by calling *CloudSimShutdown*(). This object signals the end of simulation to *CloudInformationService*(*CIS*) entity.

```
public CloudSimShutdown(String name,
    int numUser) throws Exception { .. }
```

Total number of cloud user entity plays an important role to determine whether the list of hosts (hostList) should be shut down or not. The hostList will be shutdown, unless one or more users are still not finished.

3.4. Implementation

Scheduling of resources is modelled at host and VM levels in CloudSim. At the host level, fractions of the total available processing element is shared across the VMs running in the host. This is handled by *VmScheduler* classes. Similarly, the resources are shared among the cloudlets running in a single VM at VM level, by *CloudletScheduler* classes. Classes {*X*|*Y*}*SchedulerSpaceShared* implement the space shared system using first-come first-served algorithm, where {*X*|*Y*}*SchedulerTimeShared* implement the time shared system using round-robin algorithm. Here, X refers to Cloudlet and Y refers to Vm, which handle resource allocation at VM and host levels accordingly. *VmAllocationPolicy* chooses a host and assign the given VM to it. The default behaviour is defined by *VmAllocationPolicySimple* which picks the host with less processing elements (Pe) in use as the host for the VM. The application level scheduling - which of the cloudlets from the broker is executed next, is defined at the broker level. Figure 2 depicts the classes involved in the application and resource scheduling.

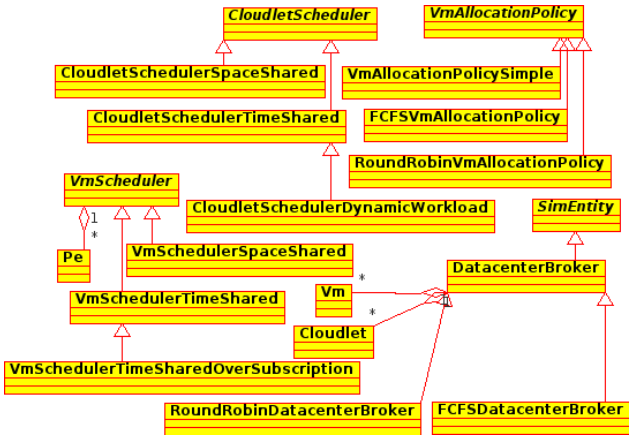

Figure 2. Class Diagram depicting the core classes involved

The examples and default scenarios often use TimeShared systems, which use round-robin algorithm. However, these policies could be mixed and the algorithm implementations for application scheduling were tested against these different combinations of scheduling algorithms at VM and host level.

3.5. Scheduling

In a space shared system with first-come first-served algorithm, a VM is dedicated completely to a cloudlet till it finishes its execution. Hence the allocation of the VM for the respective cloudlet plays a major role in deciding the start and the finish time. A time shared system with round-robin effectively shares the time window against the cloudlets. Default application scheduling is defined in *DatacenterBroker* which picks a random cloudlet from the list of readily available cloudlets waiting to be executed next. *DatacenterBroker* is extended and used to depict the desired application scheduling behaviour.

4. Evaluation

CloudSim is bundled with examples that could be extended to simulate more complicated scenarios. Existing examples and user simulations could be executed from the

folder $cloudsim-3.1-SNAPSHOT/jars$, using a command similar to the one below.

```
java -classpath
    cloudsim-3.1-SNAPSHOT.jar:
    cloudsim-examples-3.1-SNAPSHOT.jar
    org.cloudbus.cloudsim.examples.
    CloudSimExample6
```

4.1. Environment

Development of experiments and evaluation were performed on a platform of Ubuntu 12.04 LTS (precise) - 64 bit, Kernel Linux 3.2.0-56-generic and GNOME 3.4.2. The environment had 1.9 GiB memory and Intel®Core™2 Duo CPU T6600 @ 2.20GHz * 2 processor available. Java(TM) SE Runtime Environment was of the build 1.6.0_31-b04.

4.2. Configurations

CloudSim was configured with different configurations of jobs and resources and the experiments were conducted to study the behaviour of the scheduling algorithms. Extreme conditions for the scheduling algorithms where the program started to hang up were marked as the upper limits and the experiments were ensured not to surpass these.

Testing was performed with 5 virtual machines and up to 4000 cloudlets having different cloudlet lengths randomly generated within the given range. Most of the initial experiments were carried ahead with the VMs having processing elements of 200, 400, 600, 800, and 1000 MIPS, 1 CPU in each virtual machines, and with 200 users.

4.3. Resource Allocation Algorithms

Before evaluating the application scheduling algorithms, CloudSim was first tested with the implementations of resource allocation algorithms with the default policy of application scheduling and VM allocation to the hosts. Space shared schedulers with first-come first-served algorithm and time shared schedulers with round-robin algorithms were tested for both the VM scheduling which is done at host level and the cloudlet scheduling which is done at VM level.

Start times and finish times of the algorithms for the same workload is plotted as figure 3 and figure 4. These plots indicate the submission time and the completion time of the cloudlets. Figure 3 indicates that all the cloudlets were started almost immediately when VM level scheduling was run with round-robin. But first-come first-served started the cloudlets in the order they were submitted.

As shown by figure 4, by switching the available time frame among the cloudlets, all the cloudlets tend to finish at the same time regardless of their starting order when scheduled by round-robin. However in first-come first-served the earlier the cloudlet starts, the earlier it finishes. This behaviour ensured a minimal minimum execution time (MET) for first-come first-served. However, the cloudlets that arrived later had to wait much longer even to have the processing element allocated to them. This increased the starting time or the submission time of the tasks.

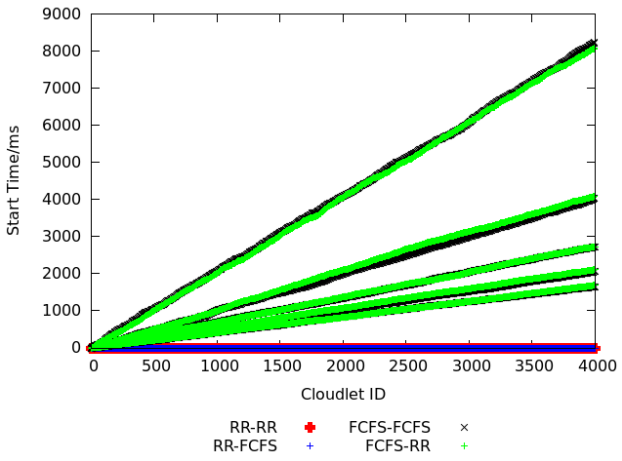

Figure 3. VM and Host level Scheduling Vs Start Time

As virtual machines with 5 different configurations (millions of instructions per second) were used, the initial allocation of the cloudlet to the VM decides the start and finish times of the cloudlet in first-come first-served as a cloudlet is executed in the same VM, once it is allocated. This makes the completion time proportional to $CloudletID/Processor\_speed\_of\_the\_VM$, creating 5 different obvious straight lines of completion time. In first-come first-served, the time waiting for submission dominates the finishing time, as the execution time is very low given that the resources are solely allocated to the cloudlet being executed. Hence, the plots depicting the submission time and the completion time tend to appear as a single line with a marginal difference between them.

The separation based on the initial allocation of cloudlets to a VM is blurred in time shared VM level allocation as VMs are shared across the cloudlets, giving the cloudlets time slots to execute in a round-robin fashion. However, the shorter tasks tend to finish faster and probably during their first slot itself at the VM they were initially allocated. Hence, the first allocation of VM still plays a major role even in a round-robin allocation, as the completion times of the cloudlets still have a clustered behaviour as 5 horizontal but rather distorted lines of finishing times, along with many cloudlets finishing at different times in between those 5 lines. Observations indicate that the VM level scheduling is dominating in submission and execution time than the

host level scheduling, as the effects of the VM allocation on the application scheduling is relatively minimal.

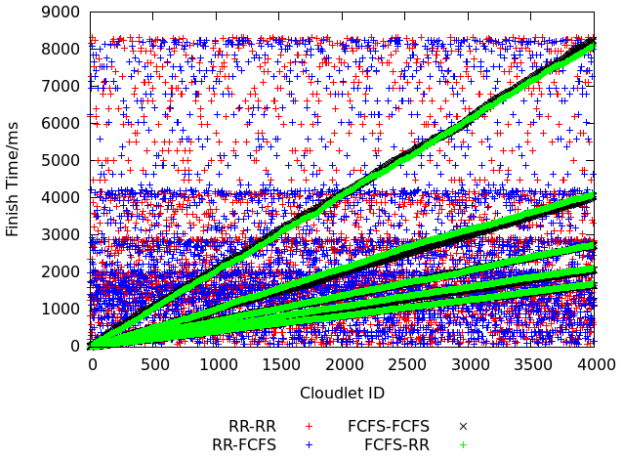

Figure 4. VM and Host level Scheduling Vs Finishing Time

4.4. Application Scheduling Algorithms

Mean execution time, mean submission time, mean completion time, and job scheduling success ratio of the scheduling algorithms were evaluated. Initially cloudlet scheduling with round-robin as well as first-come first-served were measured with over subscription of virtual machines to the hosts. Since not all the cloudlets would run concurrently, having virtual machines with the total CPU higher than the total available processing elements of the host is possible and the over subscription algorithms facilitate that scenario. First-come first-served and round-robin algorithms too were evaluated at host, VM, and broker levels without over subscribing the resources. Maximum resource utility algorithm has an objective function that maximizes the time frame that each of the resource is being utilized, focussing a fair resource usage avoiding over-utilization or under-utilization of a few resources. This is achieved by choosing the hosts with less number of processing elements utilized for the VM. Scheduling with Maximum resource utility was also evaluated. Dynamic allocation considering partial requirements satisfaction was evaluated as an algorithm from utility-driven algorithms family. Architecture, operating systems, and capacity of the VM (in million instructions per second) are given as the requirements to be partially satisfied by the allocation algorithm, with varying values of parameters between 0 and 1, for each of the cloudlets.

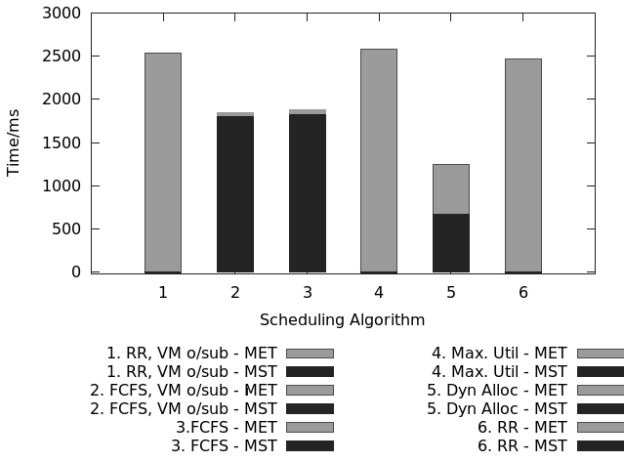

Figure 5. Mean Submission Time and Mean Execution Time of the algorithms

4.5. Observation

Mean submission time and mean execution time of these algorithms are depicted in figure 5. First-come first-served algorithms (both with and without the over-subscription of the VMs) perform better than round-robin (both with and without the over-subscription of the VMs) and maximum resource utility algorithms, when the mean execution time is concerned. Round-robin performs better on mean submission time, as the first slot is assigned to each of the application sooner, where the cloudlets or the tasks that are submitted later have to wait longer in first-come first-served, making the mean submission time higher. First-come first-served still performs better on mean completion time, when all the cloudlets are scheduled simultaneously.

Dynamic allocation algorithm outperformed all the algorithms considerably in the completion time. Mean submission time of the round-robin algorithms and the maximum resource utility algorithms was almost negligible compared to their respective execution time. Mean execution time of the first-come first-served algorithms was negligible compared to their mean submission time. While not outperforming the round-robin on mean submission time or first-come first-served on the mean execution time, dynamic allocation algorithm still was able to reach the lowest mean completion time by performing reasonably well on both submission and execution. Partial user utility is fine tuned in this algorithm according to the tasks that are under consideration, which makes the utility-driven algorithms highly efficient for their intended target applications. Since partial requirement satisfaction lets the user to indicate which of the requirements could be taken mildly and which ones to be taken seriously, waiting for a rare resource which is not a

hard-requirement for the application is reduced. This shows that Utility-driven algorithms perform much better for the tasks that are known to the user, as the utility values could be defined accordingly such that user utility will be maximized.

Job scheduling success ratio remained at 100% for all the scheduling algorithms considered. This is because the experimentation time was sufficient for all the algorithms to complete the scheduling, and there was no task that does not have a resource that it needs to execute. A separate experiment involving a very limited number of resources that are of high demand by the applications, showed that matchmaking algorithm scores poor in job scheduling success ratio where utility-driven algorithm was still able to schedule successfully, given that the requirements could be partially satisfied.

5. Conclusion

Strict matchmaking-based algorithms focus to optimize a specific objective function. Users' satisfaction depends on how much of their requirements are satisfied by the scheduling algorithms. User utility was proven to be reasonably high for all the criteria considered for the utility based algorithm developed considering partial requirements satisfaction. Hence it is mandatory to develop efficient utility based scheduling algorithms to satisfy users with heterogeneous evaluation criteria.

While criteria such as minimum execution time and minimum completion time could be a good start for evaluating scheduling algorithms, many other parameters take a higher precedence in a real cloud environment. Evaluation criteria could be developed based on market requirements such as cost of the cloud resources. QoS-based algorithms focus on such priorities. Developing effective evaluation criteria is essential for measuring user satisfaction with a considerable accuracy. Effectiveness of the simulation environments to depict real cloud scenarios is still a question to be addressed. While it is sufficient to test the prototypes and the algorithms against the simulation environment during the early phases of development, it is essential to test the production-ready algorithms against real cloud deployments.

6. Future Work

Performance of the simulation tools are far from ideal, as they try to portray a geo-distributed decentralized environment using the network and topology simulation code that is serial and manipulating the large global state that is considered consistent. Current multi-core machines and computing clusters could be exploited by the cloud simulation tools avoiding the sequential and centralized execution of the simulations. A decentralized scheduler architecture, such as a hierarchical or a mesh-like architecture, would facilitate enhanced scalability. Nevertheless, the scalability comes with a price in efficiency. The tradeoff was estimated in a conceptual manner during some further experiments. Further investigation would show whether a distributed scheduler architecture would be beneficial overshadowing its shortcomings in efficiency.

7. Acknowledgements

Prof. Luiz Veiga provided the project idea and motivation. Timely suggestions from Prof. Johan Montelius directed me in the right direction throughout the project. Dr. Susanna Lyne helped shaping up the project report and its presentation to its current state. My special thanks goes to these professors who helped me complete the project. As a project that is heavily used by the researchers in their research activities, CloudSim mailing list is always active with a vibrant community. I am grateful to the original developers and the community of CloudSim.

References

[1] Baumgart, I., Heep, B., and Krause, S. 2007. OverSim: A flexible overlay network simulation framework. In *IEEE Global Internet Symposium, 2007* (pp. 79-84). IEEE May 2007.

[2] Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., and Freund, R. F. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing, 61* (6), 810-837.

[3] Calheiros, R. N., Netto, M. A., De Rose, C. A., and Buyya, R. 2012. EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience, 00-00.*

[4] Calheiros, R.N., Ranjan, R., De Rose, C. A. F. and Buyya, R. 2009. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services, *Technical Report, GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory,* The University of Melbourne, Australia.

[5] Calheiros, R.N., Ranjan, R., De Rose, C. A. F. and Buyya, R. 2010. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract.*

Exper. 2011; 41:23-50. Published online 24 August 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.995

[6] Casanova, H. 2001. Simgrid: A toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on* (pp. 430-437). IEEE 2001.

[7] Chauhan,S.S. and Joshi, R.C. 2010. QoS Guided Heuristic Algorithms for Grid Task Scheduling. *International Journal of Computer Applications (0975 - 8887), Volume 2* - No.9, June 2010.

[8] Garg, S. K., and Buyya, R. 2011. NetworkCloudSim: modelling parallel applications in cloud simulations. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on* (pp. 105-113). IEEE.

[9] Kliazovich, D., Bouvry, P., and Khan, S. U. 2012. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing, 62*(3), 1263-1283.

[10] Lee, Y. H., Leu, S., and Chang, R. S. 2011. Improving job scheduling algorithms in a grid environment. *Future generation computer systems, 27*(8), 991-998.

[11] Maheswaran, M., S. Ali, H. Siegal, D. Hensgen, and R. Freund (1999). Dynamic matching and

[12] Montresor, A., and Jelasity, M. 2009. PeerSim: A scalable P2P simulator. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on* (pp. 99-100). IEEE September 2009.

[13] Santos-Neto, E., Cirne, W., Brasileiro, F., and Lima, A. 2005. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *Job Scheduling Strategies for Parallel Processing* (pp. 210-232). Springer Berlin Heidelberg. January, 2005.

[14] Tanenbaum, A. S. 2007. Modern Operating Systems (3rd ed.). Upper Saddle River, NJ, USA:

[15] Tighe, M., Keller, G., Bauer, M., and Lutfiyya, H. 2012. DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management. In *Network and Service Management (CNSM), 2012 8th International Conference on* (pp. 385-392). IEEE.

[16] Vasques, J. and Veiga, L. 2013. A Decentralized Utility-based Grid Scheduling Algorithm. In *28th Symposium On Applied Computing — SAC-2013*.

[17] Wickremasinghe, B., Calheiros, R. N., and Buyya, R. 2010. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on* (pp. 446-452). IEEE 2010.