National College of Ireland

An Efficient Framework To Minimize The Response Time In Resource Scheduling Using A Modified CERS Algorithm

MSc Research Project
Cloud Computing

Adarsh Puttaswamy Gowda
Student ID: 18146236

School of Computing
National College of Ireland

Supervisor: Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Adarsh Puttaswamy Gowda
Student ID:	18146236
Programme:	Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	An Efficient Framework To Minimize The Response Time In Resource Scheduling Using A Modified CERS Algorithm
Word Count:	6062
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	12th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Abstract

Resource scheduling in cloud has always been a critical research issue for any cloud designer. In order to successfully run a cloud deployment, the designers need to first analyze the load requirement on the cloud, followed by the kind of infrastructure available, and other specifications. Based on these specifications the resource scheduling algorithm is designed. In this work, we have taken into consideration the task length, the number of sub-tasks for a given task, the processing capabilities of available virtual machines, and developed a modified version of the existing cost-effective resource scheduling (CERS) algorithm. This modified CERS algorithm is based on minimization of response time. In order to achieve this task, we have proposed a simplistic task sorting mechanism which works along with CERS. Due to the task sorting mechanism, there is a reduction in the system's response time, as shorter tasks are executed faster as compared to longer tasks. This helps in improving the overall throughput of CERS. In this study, we describe the various steps and protocols which were followed in order to develop, test and optimize the performance of the CERS algorithm in terms of response time minimization and increase in the overall throughput. The results are compared with greedy algorithm, and insertion sort mechanism which shows that performance of the CERS with quick sorting is gradually better.

Contents

1 Introduction

Scheduling of resources in the cloud leads to a better performing cloud deployment. These resources can be in the form of physical machines, virtual machines, platforms or the entire infrastructure. Generally, virtual machines are the most used resource in a cloud computing scenario, therefore proper scheduling of the virtual machines is of utmost importance. Having an optimized resource scheduling algorithm benefits the cloud deployment in the following ways,

- Optimization in cost of task execution, due to proper scheduling the tasks are effectively assigned to the most favourable virtual machines, and therefore the cost of task execution is optimized.
- Reduction in running costs, because if the tasks are executed faster, then the overall computation units will get free faster. Thereby provisioning more tasks to be executed, which will reduce the running costs.
- Ability to add more security features, as the tasks are being executed with minimum delay, there is a possibility to add more security to the cloud architecture without compromising much on the delay of the system.

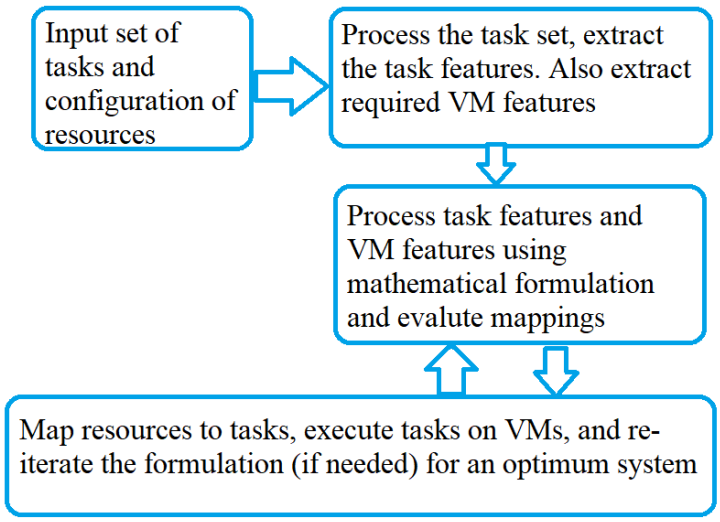

Figure 1: General resource scheduling procedure

From the figure 1, it is clear that the main block in task scheduling is where processing of task features and mathematical formulation is performed. In this report, we are focussing on mathematically implementing the cost-effective resource scheduling (CERS) algorithm, and analyzing its performance under a series of test case scenarios. We also plan to improve the performance of the CERS algorithm by adding the concept of response time minimization into it. The CERS algorithm works using the principle of reducing the cost per service rate, and can be summarized using the following steps,

- Input the task time and the utilization of the VM, and evaluate the value of cost per service rate (q).

- Using this value of q for each task to VM mapping, the array Q is formed, which contains all the 'q' values.
- Sort the array Q in ascending order.
- Schedule the VMs based on the ordering of Q, as lower Q means that the VM should be given higher priority in terms of scheduling.
- Evaluate the number of requests left, and schedule them according to the given time slot.
- Find the number of clock cycles needed for all the tasks to be executed.
- Perform VM scheduling based on the minimum number of clock cycles needed.

These basic steps are explained in more details in the following sections. The next section deals with different algorithms that are available for resource scheduling in cloud, most of which optimize the cost of scheduling. It is followed by our approach, and its implementation details. Finally, we conclude this report with an in-depth analysis of the algorithm, and observations via some test-case scenarios. This report also concludes with some future work that can be performed in order to further improve the quality of this research.

1.1 Research Question

How a modified Cost Effective Resource Scheduling(CERS) algorithm will help in optimizing the response time while scheduling the resources and to what extent could the response time be minimized?

2 Related Work

2.1 State-of-the-art techniques

In this section different algorithms for resource scheduling are studied, and the selection process of our base algorithm is explained. Resource scheduling requires careful planning from the cloud provider side and from the cloud infrastructure side as well. This planning is accomplished with the help of different task-type-aware algorithms, that take into consideration various task parameters in order to execute the tasks on the resources (or virtual machines). The algorithm defined by Xing Jia Wei, Wang Bei and Li Jun in Wei et al. (2017) uses a modified version of genetic algorithm to schedule resources. They have used simulated annealing with a multi-population genetic algorithm in order to improve task scheduling efficiency. Due to simulated annealing the algorithm focusses on finding global optimum rather than local optimum. Therefore, the overall response time reduces, the completion cost reduces, the convergence speed improves and the degree of load balancing reduces. As per their observations, the completion time reduces by 40 percent, the load imbalance reduces by 20 percent, the completion cost reduces by 40 percent and the load imbalance reduces by 20 percent when compared with simple genetic algorithm. Due to these advantages, the simulated annealing algorithm can be used for real-time cloud deployments.

A different approach towards resource scheduling is proposed in Tian et al. (2011), wherein the CPU, memory and bandwidth are used together for all the virtual and physical machines in order to allocate the tasks to the machines. A migration policy is used in order to migrate the tasks which do not satisfy a given allocation constraint, like maximum execution delay, minimum run-time, etc. The proposed algorithm is compared with ZHZJ, ZHCJ and random allocation. The comparison results indicate that the algorithm in Tian et al. (2011) reduces the imbalance level of the tasks, and also has a lower running time when compared with ZHZJ and ZHCJ algorithms. Random algorithm does not perform any complex mathematical operations while mapping, thus it is bound to require less delay. In our recommendation, we would suggest that the algorithm in Tian et al. (2011) must be compared with more algorithms and larger datasets in order to evaluate its real-time performance.

Another bio-inspired algorithm like Wei et al. (2017) is mentioned in Zhang et al. (2018), which uses improved differential evolution algorithm for resource scheduling. The algorithm in this paper proposes the scheduling in a multi-user and multi-provider environment (MUMP). This MUMP environment makes sure that the allocation is done in real-time, because for any use case scenario there are always multiple providers and multiple users of the cloud deployment. The algorithm is compared with round robin, min-min and differential evolution techniques. The comparisons indicate that the proposed MUMP based algorithm reduces the completion time by 30 percent and improves the user-to-provider satisfaction ratio by 20 percent. All this is done while keeping the load ratio of the virtual machines almost constant. For any practical use case, this algorithm can be applied without any modification.

Bio-inspired algorithms can be applied together in order to optimize one-particular area of task scheduling. In the work proposed in Guddeti et al. (2017), a hybrid algorithm which combines cat swarm optimization for load balancing and particle swarm optimization for virtual machine configuration management is defined. Via their algorithm individual optimizations are combined to form a bigger system-level optimization. Their results when compared with simple PSO, round robin, simple cat swarm optimization, ant-colony optimization and exact optimization showcase a 10 percent reduction in processing delay, 20 percent improvement in resource utilization and 10 percent reduction in algorithm complexity. This work is a perfect example of layered-based modular processing, which tends to improve the overall efficiency of resource allocation.

PSO is a very widely accepted optimization algorithm which has been in use for quite a long time. PSO allows for multiple level modifications, right from its fitness function, to its velocity equations. One such modification is done in Ashwin et al. (2014), where the fitness and velocity update equations are optimized in order to minimize the response time of the algorithm. The modified PSO reduces the response time of the algorithm by more than 10 percent, and can be used for fast-scheduling of resources in the cloud. Heuristics based scheduling is proposed using the max-min algorithm in Devipriya and Ramesh (2013), wherein researchers have applied maximum sized tasks to minimum capacity virtual machines. By doing this, the minimum sized tasks execute on higher capacity virtual machines, thereby the processing speed of the system improves drastically. The proposed max-min algorithm changes the concept a bit further, by executing average-sized tasks on slower virtual machines, which further reduces the response time by 10 percent when compared to the original max-min algorithm. A hybrid approach which combines max-min and min-max algorithms is defined in Mittal and Katal (2016), wherein the elapsed time for both max-min and min-max is evaluated, and then based on the comparison a

particular algorithm is selected for scheduling. The results indicate that the proposed combinatorial approach reduces the make-span or response time by 10 percent, when compared with max-min, min-max, RASA, improved-max-min and enhanced version of max-min. These results showcase the superiority of using a pre-calculated metric, rather than fixating on a given policy.

Value of service (VoS) is a novel metric defined in Tunc et al. (2016), wherein the value of the services given by a cloud provider are decided by the virtual machine's previous ability to complete a task within a given deadline and the energy needed by the VM to execute the given task. Using these parameters, a VoS based scheduling algorithm is defined, which uses soft and hard thresholds for scheduling tasks. The VoS based algorithm is compared with a non-VoS based system, and it is observed that the VoS based system is 50 percent more responsive, and 40 percent more energy efficient. Generally, the VoS system is deployed on each of the load-execution units, which increases the overheads. In order to reduce these overheads, a central load balancer is defined in Soni and Kalra (2014). This balancer uses the concept of data-aggregation in order to execute tasks on virtual machines. The information from all the computing nodes is aggregated on a single node, and calculations are done based on these readings. These calculations provide a fair idea about the current load scenarios of the system, and thereby allows for proper resource allocation. The central load balancer reduces the response time by more than 60 percent when compared with round robin, active response time and throttled response time-based balancers.

Clustering is a way to group similar kind data together in order to perform a similar set of operation on it. A clustering-based approach, which uses task grouping based on the cloud's capability to execute a given set of tasks on a given set of machines is proposed in Selvarani and Sadhasivam (2010). Based on this clustering, the algorithm is able to reduce the processing delay by 15 percent, and the processing cost by 70 percent when compared with a non-clustering approach of resource scheduling. Clustering approach is again used to balance network traffic in Das et al. (2003), wherein the algorithm is not a cloud-based algorithm, but can be linearly used for cloud-based systems. Using the proposed algorithm, the mapping between tasks and execution units can be performed in the same manner as the mapping between nodes and base-stations is done. The results indicate that the proposed algorithm reduces delay by 10 percent when compared to first-come-first-serve (FCFS) based scheduling systems. The evaluation of this algorithm must be done on cloud networks to evaluate its real-time performance before actual usage. Like FCFS; round robin and least connection methods are also equally good when it comes to task scheduling. The work in Swarnakar et al. (2018) combines weighted round robin with weighted least connection algorithms in order to improve the performance of each of the individual algorithms. The proposed algorithm reduces the average waiting time by 20 percent, and increases the average resource utilization by almost 15 percent when compared with the individual algorithms. This study can further be extended for more algorithms in order to evaluate its actual performance.

2.2 Cost-aware techniques for scheduling

Including cost awareness to a scheduling system is of primary importance due to 2 major reasons, which are,

- A cost aware scheduling system will never overload the virtual machines.

- The system will always take into consideration Quality of Service while scheduling tasks.

Due to these inherent advantages, the algorithm in Kapur (2015) was studied and selected as our baseline algorithm for this research. Using this study, researchers have claimed that cost effective resource scheduling algorithm takes into consideration the task cost, and allocates the best capacity VMs to the tasks which require higher processing capabilities along with reduced response time. Due to these advantages the resource cost of this CERS algorithm is lower than the existing methods, while the throughput is very high. An in-depth analysis of the CERS system is done in the later sections of this text. In order to evaluate CERS with other algorithms, we have selected the research done in Keivani et al. (2018). This research clearly indicates that simulated annealing and bio-inspired algorithms are the best-in-class, and must be used for any level of resource scheduling.

A bandwidth aware scheduling algorithm is presented in Razaque et al. (2016), wherein researchers have selected the bandwidth of virtual machine in order to allocate tasks. Higher bandwidth machines are assigned with larger tasks, and lower bandwidth machines are assigned with smaller tasks. Due to this bandwidth-based allocation the delay in execution of tasks is reduced by more than 40 percent. While these results look too-good-to-be-true, we too recommend in-depth analysis of such algorithms before actual implementation on cloud deployments. Another soft-computing algorithm that modifies PSO and adds cuckoo search into it is presented in Naresh et al. (n.d.). Using the cuckoo-based PSO, the cost of task execution reduces by 10 percent, while the energy efficiency improves by 15 percent, when compared with simple PSO and improved PSO algorithms. But this algorithm is generally suited for smaller sized tasks, because the complexity of evaluation increases as the task size increases.

A Median Deviation based Task Scheduling (MDTS), which uses Median Absolute Deviation (MAD) of the Expected Time to Compute (ETC) is proposed in Akbar et al. (2016). In this approach the average of the computation time is evaluated, and then minimized by using task time variation optimization. Due to this approach, the overall response time of the algorithm is reduced by almost 25 percent when compared to CPOP, HEFT and MDTS algorithms as mentioned in the same paper. Clustering based methods are supposed to reduce the overall response time of any scheduling system. This has been proved by the work in Al-Rahayfeh et al. (2019), where task size and virtual machine capacity-based clustering is performed in order to map the most matching clusters. Clusters with minimum task time are mapped with VM clusters of minimum capacity in order to improve the overall quality of scheduling. This results in a reduction of makespan by 10 percent, an increase in resource utilization by more than 30 percent and an improvement in trust-level by around 10 percent when compared with the TTSA algorithm.

As seen previously, the bio-inspired algorithms outperform any other statistical approaches. But in contrast the hybrid heuristics method proposed in Wang and Li (2019) beats the PSO, ACO and round robin-based approaches in terms of average delay of execution, energy consumption and reliability. Due to the heuristic-based approach, there is no-uncertainty of the obtained solution, thereby there is an obvious improvement in algorithm reliability as compared to other stochastic algorithms, which do not guarantee optimization. Another CERS based algorithm is presented in Nasr et al. (2019), wherein chemical reaction optimization and ant colony optimization (CRO and ACO) are used in tandem to improve the effectiveness of task scheduling. Using these two methods, and then adding a resource aware deadline constraint to it, guarantees a reduction in

response time, and an improvement in the overall resource utilization by the system. The proposed system is more than 15 percent effective than non-resource aware counterparts. Thus, resource aware algorithms like CERS must be explored in depth for a better resource allocation algorithm development. Similar cost-effective approaches are mentioned in Wang et al. (2017),Arabnejad and Bubendorfer (2015), Sahni and Vidyarthi (2015) wherein the cost of the task is mapped effectively with the capacity of the virtual machine for task execution.The results from these researches have further strengthened our conviction towards selection of CERS as a base-line algorithm for our research.

3 Methodology

In order to design and evaluate the response time minimization CERS algorithm, we followed the given steps,

Selection of simulation platform – Cloudsim

After reviewing various cloud simulation platforms like cloudsim, greencloud, iCanCloud, cloud analyst and open nebula, we observed that the cloudsim provides the ability to add customized algorithms for cloud memory storage, cloud broker selection, virtual machine configurations and task scheduling. Also, cloudsim is being used for most of the research projects currently under study.

Selection of programming language – Java

Cloudsim supports Java, and thus it is our default programming language for developing the algorithms.

Selection of dataset - NASA Ames iPSC/860 log The NASA Ames iPSC/860 log consists of around 7 task sets, and each task set has more than 50k tasks. Each set has tasks which contain the following information,

- Number of tasks in a given set.
- Execution time of each task in the set.

For our research, we need exactly these parameters in order to evaluate the system, and thus, this set was selected for the purpose of evaluation.

Selection of evaluation parameters

System evaluation has to be done in such a way that the precise performance of the algorithms can be evaluated. For this research, we selected the following parameters to evaluate all the 3 systems,

- Number of execution cycles (NEC), this is the number of times all or some of the virtual machines of the cloud are executed to run all the given tasks. This parameter defines the future remaining capacity of the cloud deployment once all the tasks have completed execution.
- Time needed for execution (Te), is the total time needed by our algorithm to allocate resources and execute tasks. This delay should be minimum, so that the overall system response time is good.
- Task mean waiting time (Tmwt), is the average of the waiting time for all the tasks. A low value of Tmwt indicates that the tasks are being executed with best possible effort from the cloud system.

- We have formulated a new parameter named inverse Quality of service(iQoS), which is a combination of 3 minimization parameters which are already defined in our work. These parameters when combined will provide a fused minimization parameter, that decides the overall QoS of the system. Generally QoS needs to be maximized, but this parameter is a combination of 3 parameters which need minimization, thus in order to get a good QoS, this parameter needs to be minimized, thus we have named this parameter as iQoS.

Development of the algorithms Eclipse IDE was used in order to develop these algorithms. For the purpose of evaluation, we manually changed the number of virtual machines in the simulation. For varying the number of tasks, each dataset was read and sets of 1000, 2000, 5k,10k, 20k, 50k and 100k were made in order to evaluate the performance. These tasks were given to the system for evaluation.

The in-depth analysis and algorithm development steps are described in the specifications and implementation sections.

4 Design Specification

The main issue in resource scheduling is the ability of the algorithm to effectively allocate resources to tasks, in such a way that the overall system should have a high QoS. QoS may include the total cloud execution cycles, the algorithmic delay, the complexity of the algorithm, etc. In order to model an effective QoS based task scheduler, we first implemented the CERS algorithm. The flow diagram of the existing CERS algorithm is shown in figure 2. As per the flow diagram the working of the system can be elaborated using the following,

- The input task set is given to the task scheduling system, and the VM parameters are setup. The task parameters include task length, number of tasks, bandwidth needed by the task and RAM needed by the task. While the VM parameters include Mips, RAM, Bandwidth, etc. More details about this are mentioned in the next section.

- Based on the VM and task parameters, the value of 'q', which is the cost per service rate is evaluated.

- The cost per service rate is a ratio of the task cost to the execution rate of the virtual machine. We need a lower value of the 'q' factor, because we need to assign tasks to machines that have faster execution rate.

$$q = (T_{\mathrm{l}} * N_{\mathrm{t}})/(VM_{\mathrm{MIPS}} * VM_{\mathrm{RAM}}) \tag{1}$$

where,
T_{l} is the length of the task
N_{t} is the number of sub-tasks in that particular task
VM_{MIPS} is the MIPS rating of the VM
$VM_{\mathrm{RAM}}$is the RAM memory of the VM

- Based on these 'q' values, a 'Q' array is created. The 'Q' array is created on a per VM per task basis, and must be sorted in ascending order to get the best values of task to VM mapping.

$$Q = sorted(q_{i,j}) \tag{2}$$

where,
$q_{i,j}$ is the q value for the task when executed on VM

- Once this mapping is done, and new tasks are incoming, then update the 'Q' vector and re-sort the array.
- If there are no more tasks, then stop the scheduler process and go-to first step, else check if the updated 'Q' is empty.
- If the updated 'Q' is empty, then schedule the VMs as per the previously sorted 'Q' values.
- If the updated 'Q' is not empty, then mark the arrival rates of the tasks, and schedule faster arriving tasks to faster. operating VMs, and slower arriving tasks to slower operating VMs.
- Evaluate the inverse cost ratio of the VMs for the remaining tasks, and schedule them to the VMs on-demand, as per their 'q' values.

$$I_{cr} = \frac{C_{vm}}{T_l} \tag{3}$$

where,
I_{cr} is the inverse cost ratio of the VM to the task
C_{vm} is the capacity of the VM
T_l is the task length, or the task cost

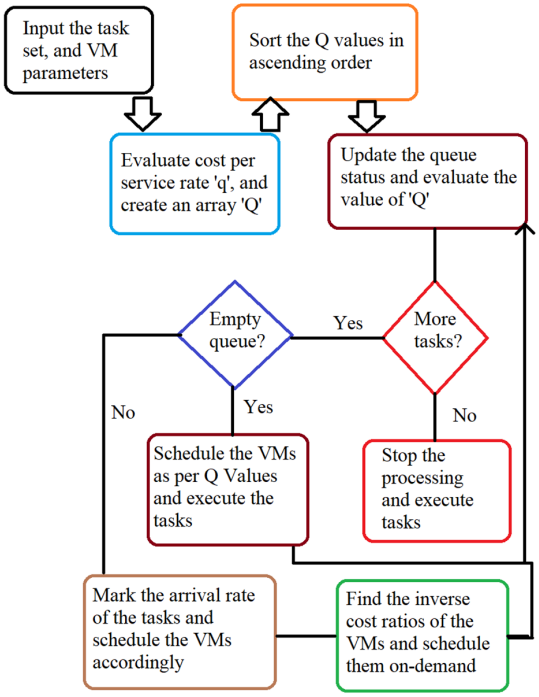

Figure 2: The flow diagram of the CERS algorithm

Based on these steps shown in Fig 3, the tasks are executed on the VMs. But the given algorithm doesn't take into consideration the response time of the tasks. The response time is basically the difference between the incoming time and the outgoing time for the task. The response time should be as minimum as possible, which showcases that the scheduling system is efficient enough to execute tasks which are time-bound. Therefore, we introduced a simple sorting step, which is added just before the task execution begins. This sorting step will sort all the tasks w.r.t. their task length and execution delays. Due to the sorting process, the tasks with minimum delay will be executed first, thereby virtual machines having higher capacity will be able to execute these tasks quickly. This process will reduce the mean task waiting time, thereby improving the overall response time of the system. The flow diagram for the modified system can be seen in figure 3.

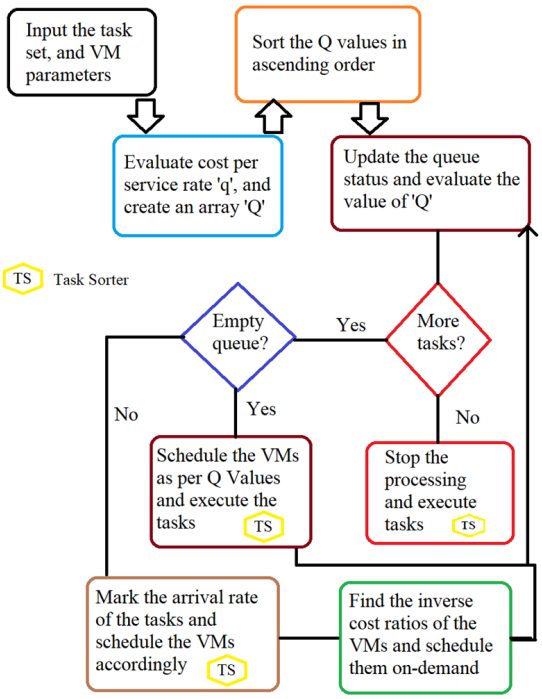

Figure 3: Flow diagram of the modified system

In the figure 3, the TS block is responsible for sorting the tasks in ascending order of execution delay. We have used quick sort and insertion sort methods in order to evaluate the execution delay of the algorithm, details about these evaluations can be found in the evaluation section of this report. The algorithmic implementation of the system is done using Eclipse IDE and cloudsim framework. The details of this implementation are mentioned in the next section.

5 Implementation

In order to implement the original and proposed CERS algorithm the following tools were used,

1. **Eclipse IDE**, used for Java based codebase development.

2. **Cloudsim**, used for simulation of cloud-like environment on local machine.

Eclipse is used for general purpose Java development. Our Eclipse project is named as 'CERS' and consists of the following classes,

- **Task** class, which has the following members,

1. taskId – It is the ID of the task as read from the dataset.
2. numTasks – Represents the number of sub-tasks present in this task.
3. taskLength – It is the length of each sub-task.
4. taskBW – Gives the bandwidth needed to execute the task on a VM.
5. taskRAM – Represents the RAM memory needed to execute the task.
6. compareTo() method – This method is used to sort the tasks using quick sort technique.

- **MyTaskList** class, which contains the list of tasks for insertion sorting. It has the following members,

1. insertionSort() method – Used to perform insertion sort on the list of tasks passed through the class.

- **CERS2** class, this class has the main routines of the project for the existing CERS system. These main routines include,

1. Reading the tasks from the dataset, and putting them into lists.
2. Initializing the Virtual machines with the following parameters,
   - MIPS rating, is the capacity of the machine in terms of millions of instructions executed per second.
   - CPUs, indicates the number of processing units the VM possesses.
   - RAM, amount of memory available with the Virtual Machine.
   - BW, amount of bandwidth present with the Virtual Machine.
   - Size, represents hard disk size of the Virtual Machine.
3. Executing the CERS algorithm as per the flow diagram given in figure 2.
4. Evaluation of results, and putting them into an output file for future checking.

- **CERS_ResponseTimeMinimization** class, implements the CERS algorithm with task sorting. It implements the CERS algorithm as given in the figure 3 of this text. It consists of the following elements,

1. Initializes the tasks and assigns them into lists.

2. Initializing the Virtual machines with the following parameters,

   - MIPS rating,is the capacity of the machine in terms of millions of instructions executed per second.

   - CPUs, indicates the number of processing units the VM possesses.

   - RAM, amount of memory available in the Virtual Machine.

   - BW, amount of bandwidth present in the Virtual Machine.

   - Size, represents the hard disk size of the Virtual Machine.

3. Sorts the tasks using quick sort (internal Java method) before execution.

4. Executes the tasks using the CERS algorithm as mentioned in figure 3.

5. Evaluation of results, and putting them into an output file for future checking.

- **CERS_ResponseTimeMinimizationInsert** class, implements the CERS algorithm with insertion sorting based task sorting. It implements the CERS algorithm as given in the figure 3 of this report. It consists of the following elements,

1. Initializes the tasks and assigns them into lists.

2. Initializing the Virtual machines with the following parameters,

   - MIPS rating, which is the capacity of the machine in terms of millions of instructions executed per second.

   - CPUs,indicates the number of processing units the VM possesses.

   - RAM, amount of memory available in the Virtual Machine.

   - BW, amount of bandwidth present in the Virtual Machine.

   - Size, represents the hard disk size of the Virtual Machine.

3. Sorts the tasks using the MyTaskList class, before execution.

4. Executes the tasks using the CERS algorithm as mentioned in figure 3.

5. Evaluation of results, and putting them into an output file for future checking.

Cloudsim is a general-purpose cloud simulator which is compatible with Java. Being open source, it gives researchers the ability to extend and modify its capabilities as per requirement Calheiros et al. (2011). In our work, we have used the following classes from the cloudsim framework,

- **Cloudlet**, this class initializes a cloudlet in the cloudsim framework.
- **CloudletSchedulerTimeShared**, scheduling of the cloudlets is done with the help of this class so that they can work in tandem to execute tasks.
- **Datacenter**, is used to create and manage cloud Data Centres.
- **DatacenterBroker**, is used to broker between the cloudlets and the data centre.
- **DatacenterCharacteristics**, it defines the properties and rules for the data centre.

- **Host**, this class defines a Host machine.
- **Log**, used to perform logging of events in the simulator.
- **Pe**, is the basic processing element class in cloudsim.
- **Storage**, this class provides different kind of storage configurations for the virtual machine which we are using.
- **UtilizationModel**, defines a sample cloud utilization model for the simulated cloud environment.
- **UtilizationModelFull**, this is an extension to the UtilizationModel class and helps in defining rules for the utilization model for the cloud.
- **Vm**, the basic class for defining and using Virtual machines.
- **VmAllocationPolicySimple**, used to allocate VMs to the cloudlets.
- **VmSchedulerTimeShared**, used to time schedule the VMs as per the defined scheduling policy.
- **CloudSim**, basic cloudsim initialization class.
- **BwProvisionerSimple**, used to provision the bandwidth between the virtual machines.
- **PeProvisionerSimple**, used to provision the processing elements between the virtual machines.
- **RamProvisionerSimple**, this class helps to provision the RAM memory between the VMs.

6 Evaluation

All these classes are required for initialization and working of the cloudsim environment with our proposed model. Once these classes are defined, and the algorithm is performing task allocations properly, then the following metrics are used in order to evaluate the algorithm's efficiency,

- Number of execution cycles (NEC), this is the number of times all or some of the virtual machines of the cloud are executed to run all the given tasks.
- Time needed for execution (Te), is the total time needed by our algorithm to allocate resources and execute tasks.
- Task mean waiting time (Tmwt), is the average of the waiting time for all the tasks.
- Inverse Quality of Service (iQoS), combination of 3 minimization parameters which helps in deciding the overall Qos of the system.

Based on these metrics, we varied the number of virtual machines, the number of tasks and the configuration of the virtual machines in order to evaluate the performance of greedy algorithm, CERS algorithm and CERS algorithm with sorting. The following use business case scenarios were considered while evaluation.

6.1 Case Study 1 : Optimum cloud running cycles for a given virtual machine and task set

In this scenario we are observing the effect of varying the number of VMs and the number of tasks on the cloud running cycles. This will help businesses to evaluate the best configuration of VMs for a given task set.

The results table(Table 1) for this evaluation is shown in the appendix. And from those results we can observe that the overall execution cycles for CERS are reduced by more than half as compared to the greedy algorithm due to the cost effectiveness of the system. These values can be visualized using the following graph(Fig 4) for 10 VMs.

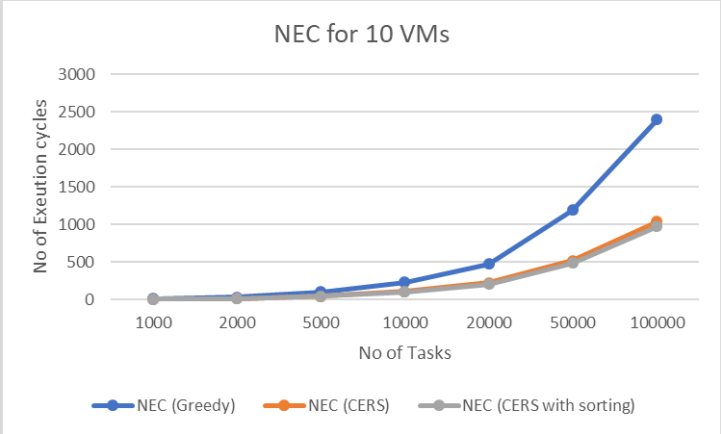

Figure 4: NEC for 10 VMs

Similar observations are done as VMs are increased. The CEC cycles follow a default linear trend as the number of VMs are increased. The trend of these values can be observed from the mean bar chart(Fig 5), using the following illustration,

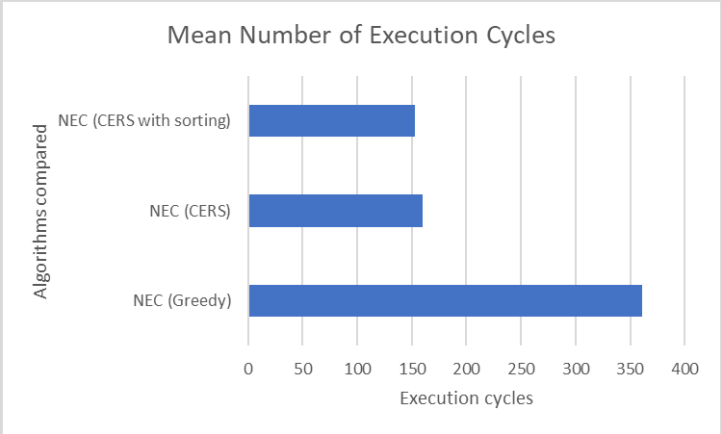

Figure 5: Mean execution cycles

As we can observe from Fig 5, the mean execution cycles are reduced by more than 50 percent when compared to the greedy algorithm.

6.2 Case Study 2 : Effect of task execution delay by changing virtual machines and task set

In this scenario we are observing the effect of varying the number of VMs and the number of tasks on the total execution delay. This will help businesses to evaluate the fastest configuration of VMs for a given task set. But instead of the Greedy algorithm, we have chose to compare CERS with quick sort and insertion sort to check which sorting technique is more efficient in this study.

The results table(Table 2)for this experiment is shown in the appendix. The results calculated in milliseconds(ms) show that the overall execution delay is reduced with the CERS algorithm, but due to insertion sorting the overall delay increases by more than 90 percent, and thus quick sorting is preferred than insertion sorting. The same results can be observed using the following graph(Fig 6),

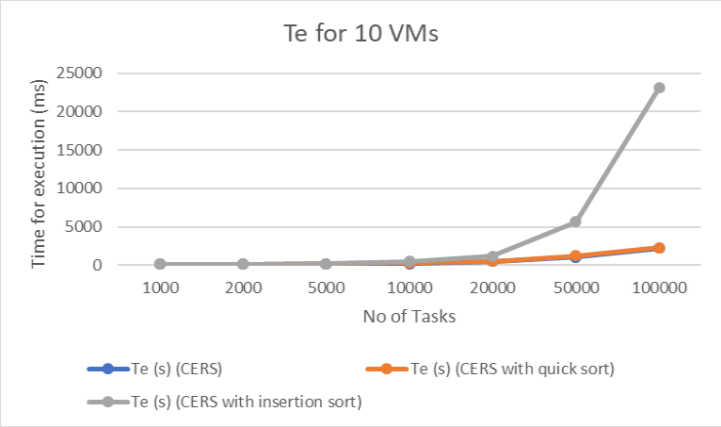

Figure 6: Comparison of delay values

Similar trend is followed as the number of VMs are increased, therefore we plotted a bar chart as shown in Fig 7 for the mean Te value across all VMs, thereby showcasing that the performance of CERS with quick sort is the most optimum out of all the developed algorithms.

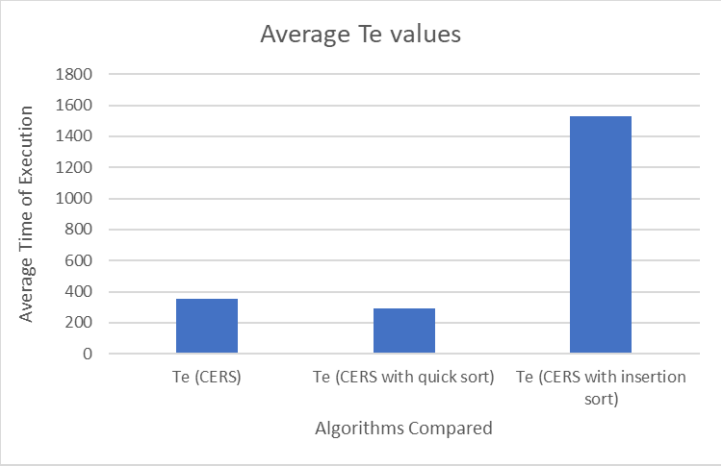

Figure 7: Comparison of average Te values

6.3 Case Study 3: : Effect of task mean waiting delay by changing virtual machines and task set

In this case study, we are observing the effect of varying the number of VMs and the number of tasks on the mean waiting delay of tasks. This will help businesses to evaluate the most responsive configuration of VMs for a given task set.

The results table (Table 3) is attached in the Appendix. From that table, we can observe that the Tmwt values are highly optimized for the case of CERS with sorting because the tasks with shorter duration are executed faster, thereby improving the overall response time of the algorithm. The following graph (Fig 8), can be used to indicate the results,

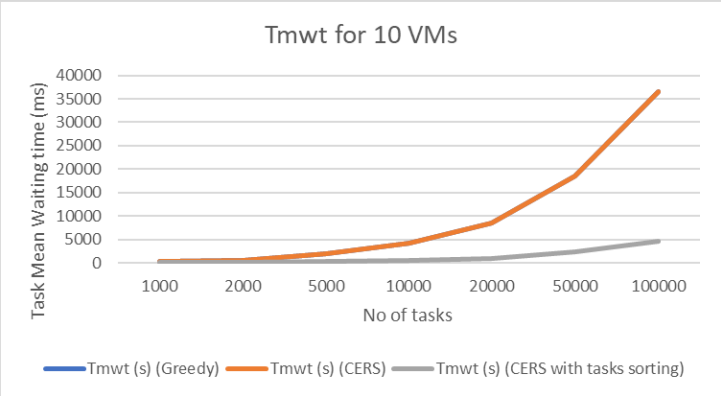

Figure 8: Comparison of Tmwt values

Similar comparison is made for the average Tmwt values, and the following results were evaluated,

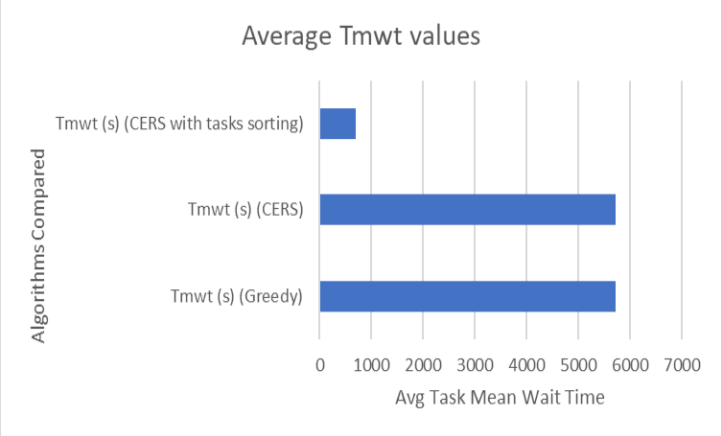

Figure 9: Comparison of average Tmwt values

From the graph as shown in Fig 9, we can observe that the Tmwt values have been improved by more than 30 percent when compared with CERS and Greedy algorithm.

6.4 Case Study 4 : Effect of changing virtual machines and task set on overall user side inverse QoS

We evaluated an inverse QoS parameter, by combining the values of NEC, Te and Tmwt. The formula for this inverse QoS is defined as under,

$$QoS = \frac{NEC}{Max(NEC)} + \frac{Te}{Max(Te)} + \frac{Tmwt}{Max(Tmwt)} \tag{4}$$

The inverse QoS should be as minimum as possible, which will make sure that the given algorithm is evaluated for best NEC, best Te and best Tmwt values. For obtaining QoS value we can take the inverse of the inverse QoS. That value will need to be maximized. For the sake of simplicity, we are taking inverse QoS as the parameter for evaluation of this work. Using the given formula, we obtained the relevant values for the inverse-QoS, whose table (Table 4) is shown in the Appendix.

We visualized these values for 10 VMs, and observed that the system followed the same trend as the number of VMs are increased. A graph is plotted to visualise the same.

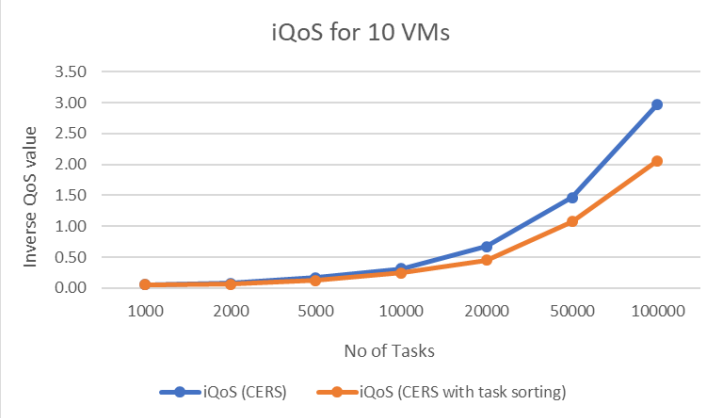

Figure 10: iQoS comparison (lower the better)

Fig 10 shows the graph where we can observe that the iQoS for CERS with sorting is around 10 percent better than the one without sorting. These results help us in identifying that the CERS algorithm can provide good overall QoS with task sorting. The QoS difference increases as the number of tasks are increased, thereby concluding that as the number of tasks increase, the performance of the proposed CERS algorithm with quick sort increases linearly w.r.t. the QoS performance of the CERS without sorting.

6.5 Discussion

The reason for selecting the given parameters for evaluation is cited as follows,

- Number of execution cycles decides the cost required by the virtual machine configuration to execute the given set of tasks on it. This value requires a minimization optimization, and thus should be as low as possible. From our observations, the number of execution cycles are reduced in CERS when compared with greedy algorithm. Moreover, response time aware CERS further reduces these cycles by executing smaller tasks faster.

- Time needed for execution is basically the delay needed for running the algorithm, and for task mapping. This time must be as low as possible. CERS and response time aware CERS have similar delays, while greedy algorithm is quicker than the two. This metric can define the performance of the algorithm when it is clubbed with some other parameter. Because, it is ok to compromise a little on the delay, but the overall system performance must be improved at the same time.

- Task mean waiting time is a user-experience parameter. We have tried to minimize this value via task size sorting, and observed that the mean waiting time reduces drastically. This waiting time decides the speed of the cloud deployment, and thereby the user retention capability of the cloud.

- As described iQoS in the methodology, we have defined a new parameter named iQoS. This parameter must be minimized in order to get better quality of scheduling. For the purpose of evaluation, we have first evaluated the maximum values of all the 3 components in iQoS, and then used the instantaneous values of these components individually to obtain the iQoS value. Our proposed CERS with quick sort is able to achieve around 20 percent improvement in terms of iQoS, which indicates a 20 percent improvement in the overall scheduling quality.

7 Conclusion and Future Work

Our research was mainly based on checking the performance of resource scheduling via modifications in the existing CERS algorithm. In order to perform that task, we had integrated CERS with a response time minimization technique. Using the proposed technique of quick sort on the tasks, the response time of the system reduces, thereby improving the overall responsiveness of the algorithm. Based on the results, we can observe that the traditional CERS algorithm has longer waiting delays as compared to the sorting based CERS, the delay is reduced by 10 percent. The Te of the insertion sort is improved by 20 percent when compared to quick sorting technique. Also, the Te values are found to be lower for CERS than for the greedy algorithm. Moreover, the mean task waiting time is also reduced by 15 percent when compared to the original CERS algorithm without sorting.

As a future work, we can incorporate machine learning and artificial intelligence into the resource allocation process. These algorithms are self-learning and are able to improve the QoS of any system under test. Researchers can explore the complexity of these algorithms before implementing them for the resource schedulers.

References

Akbar, M. F., Munir, E. U., Rafique, M. M., Malik, Z., Khan, S. U. and Yang, L. T. (2016). List-based task scheduling for cloud computing, *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, pp. 652–659.

Al-Rahayfeh, A., Atiewi, S., Abuhussein, A. and Almiani, M. (2019). Novel approach to task scheduling and load balancing using the dominant sequence clustering and mean shift clustering algorithms, *Future Internet* **11**(5): 109.

Arabnejad, V. and Bubendorfer, K. (2015). Cost effective and deadline constrained scientific workflow scheduling for commercial clouds, *2015 IEEE 14th International Symposium on Network Computing and Applications*, IEEE, pp. 106–113.

Ashwin, T., Domanal, S. G. and Guddeti, R. M. R. (2014). A novel bio-inspired load balancing of virtualmachines in cloud environment, *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, IEEE, pp. 1–4.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, *Software: Practice and experience* **41**(1): 23–50.

Das, S., Viswanathan, H. and Rittenhouse, G. (2003). Dynamic load balancing through coordinated scheduling in packet data systems, *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, Vol. 1, IEEE, pp. 786–796.

Devipriya, S. and Ramesh, C. (2013). Improved max-min heuristic model for task scheduling in cloud, *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, IEEE, pp. 883–888.

Guddeti, R. M., Buyya, R. et al. (2017). A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment, *IEEE Transactions on Services Computing* .

Kapur, R. (2015). A cost effective approach for resource scheduling in cloud computing, *2015 International Conference on Computer, Communication and Control (IC4)*, IEEE, pp. 1–6.

Keivani, A., Ghayoor, F. and Tapamo, J.-R. (2018). A review of recent methods of task scheduling in cloud computing, *2018 19th IEEE Mediterranean Electrotechnical Conference (MELECON)*, IEEE, pp. 104–109.

Mittal, S. and Katal, A. (2016). An optimized task scheduling algorithm in cloud computing, *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, IEEE, pp. 197–202.

Naresh, T., Lakshmi, A. J. and Reddy, V. K. (n.d.). Resource optimization using cloud scheduling.

Nasr, A. A., El-Bahnasawy, N. A., Attiya, G. and El-Sayed, A. (2019). Cost-effective algorithm for workflow scheduling in cloud computing under deadline constraint, *Arabian Journal for Science and Engineering* **44**(4): 3765–3780.

Razaque, A., Vennapusa, N. R., Soni, N., Janapati, G. S. et al. (2016). Task scheduling in cloud computing, *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, IEEE, pp. 1–5.

Sahni, J. and Vidyarthi, D. P. (2015). A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment, *IEEE Transactions on Cloud Computing* **6**(1): 2–18.

Selvarani, S. and Sadhasivam, G. S. (2010). Improved cost-based algorithm for task scheduling in cloud computing, *2010 IEEE International Conference on Computational Intelligence and Computing Research*, IEEE, pp. 1–5.

Soni, G. and Kalra, M. (2014). A novel approach for load balancing in cloud data center, *2014 IEEE international advance computing conference (IACC)*, IEEE, pp. 807–812.

Swarnakar, S., Raza, Z., Bhattacharya, S. and Banerjee, C. (2018). A novel improved hybrid model for load balancing in cloud environment, *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, IEEE, pp. 18–22.

Tian, W., Zhao, Y., Zhong, Y., Xu, M. and Jing, C. (2011). A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters, *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, IEEE, pp. 311–315.

Tunc, C., Kumbhare, N., Akoglu, A., Hariri, S., Machovec, D. and Siegel, H. J. (2016). Value of service based task scheduling for cloud computing systems, *2016 International Conference on Cloud and Autonomic Computing (ICCAC)*, IEEE, pp. 1–11.

Wang, B., Li, J. and Wang, C. (2017). Cost-effective scheduling precedence constrained tasks in cloud computing, *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, IEEE, pp. 230–235.

Wang, J. and Li, D. (2019). Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing, *Sensors* **19**(5): 1023.

Wei, X. J., Bei, W. and Jun, L. (2017). Sampga task scheduling algorithm in cloud computing, *2017 36th Chinese Control Conference (CCC)*, IEEE, pp. 5633–5637.

Zhang, P., Wang, X. and Zhou, M. (2018). Multi-user multi-provider resource allocation in cloud computing, *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, pp. 1428–1433.

A Appendix

No of VMs	No of Tasks	NEC (Greedy)	NEC (CERS)	NEC (CERS with sorting)
10	1000	16	8	8
10	2000	34	16	15
10	5000	107	47	45
10	10000	234	111	103
10	20000	478	226	210
10	50000	1196	524	489
10	100000	2400	1044	974
20	1000	8	4	4
20	2000	17	8	8
20	5000	52	23	23
20	10000	117	55	54
20	20000	239	113	111
20	50000	598	262	258
20	100000	1200	522	514
50	1000	3	1	1
50	2000	6	3	3
50	5000	21	9	9
50	10000	46	22	22
50	20000	95	45	45
50	50000	239	104	106
50	100000	480	208	212

Table 1: Comparison of NEC values

No of VMs	No of Tasks	Te(CERS) ms	Te(CERS with q.sort)ms	Te(CERS with i.sort)ms
10	1000	96	110	94
10	2000	109	98	109
10	5000	150	173	188
10	10000	204	310	501
10	20000	498	515	1156
10	50000	1034	1230	5674
10	100000	2186	2250	23056
20	1000	121	84	83
20	2000	95	99	113
20	5000	134	165	208
20	10000	200	266	376
20	20000	449	510	1219
20	50000	1176	1206	5645
20	100000	2142	2134	20827
50	1000	98	90	78
50	2000	97	104	125
50	5000	138	148	218
50	10000	322	260	495
50	20000	445	492	1246
50	50000	1090	1037	5695
50	100000	1961	2175	21108

Table 2: Comparison of Te values in ms (Sorting Algorithms are compared)

No of VMs	No of Tasks	Tmwt(Greedy)ms	Tmwt(CERS)ms	Tmwt(CERS with sorting)ms
10	1000	336	336	44
10	2000	585	585	91
10	5000	2014	2014	225
10	10000	4250	4250	454
10	20000	8505	8505	915
10	50000	18422	18422	2309
10	100000	36570	36570	4651
20	1000	168	168	22
20	2000	292	292	45
20	5000	1007	1007	112
20	10000	2125	2125	227
20	20000	4252	4252	457
20	50000	9211	9211	1154
20	100000	18285	18285	2325
50	1000	67	67	8
50	2000	117	117	128
50	5000	402	402	45
50	10000	850	850	90
50	20000	1701	1701	183
50	50000	3684	3684	461
50	100000	7314	7314	930

Table 3: Comparison of Tmwt values in ms

Number of VMs	Number of Tasks	iQoS (CERS)	iQoS(CERS with task sorting)
10	1000	0.06	0.06
10	2000	0.08	0.06
10	5000	0.17	0.13
10	10000	0.31	0.25
10	20000	0.67	0.46
10	50000	1.47	1.08
10	100000	2.97	2.06
20	1000	0.06	0.04
20	2000	0.06	0.05
20	5000	0.11	0.10
20	10000	0.20	0.18
10	20000	0.42	0.35
20	50000	1.03	0.81
20	100000	1.95	1.50
50	1000	0.05	0.04
50	2000	0.05	0.05
50	5000	0.08	0.08
50	10000	0.19	0.14
50	20000	0.29	0.27
50	50000	0.68	0.58
50	100000	1.27	1.20

Table 4: iQoS comparison for CERS with and without task-sorting

National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Adarsh Puttaswamy Gowda
Student ID: x18146236

School of Computing
National College of Ireland

Supervisor: Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Adarsh Puttaswamy Gowda
Student ID:	x18146236
Programme:	Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	683
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	12th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Adarsh Puttaswamy Gowda
x18146236

1 Introduction

This configuration manual consists of the process of running the CERS code, and the general setup required in order to install the necessary tools needed for the project. The structure of the manual is as follows,

Section Number	Section Name	Purpose
2	System configuration	The basic information about software and hardware needed for the CERS project
3	Project Development	The procedure of data-reading, data-preprocessing and execution of the algorithm on the dataset
4	Description of the codes	There are 4 main classes in our project, we describe each of them in this section

2 System Configuration

2.1 Hardware

Processor: Any CPU from Intel(R) Core(TM) i3 to Core i9 is ok.
GPU: Any (not required for this work).
RAM: More than 4 GB (higher for larger datasets).
Storage: More than 100 GB (including Windows/Linux).
Operating system: Windows / Linux / Mac (Our development is in Java, thus any OS is fine).

2.2 Software

Eclipse: Used to code the implementation in Java.
JDK 1.7: Java Development Kit for compiling and running the codes.
Cloudsim: It is a JAR file, which consists of the cloudsim framework.
Excel: To visualize and analyze the results.

3 Project Development

Our project development is done using the following steps,

Step Number	Title	Description
1	Data collection	The dataset is collected from the Huji parallel workloads website. It is available in raw format, which is then pre-processed
2	Data collection and pre-processing	The raw format data is converted into CSV format by manually opening the file, removing headers, and adding commas between each of the values For this purpose, a simple Notepad or Wordpad with fast replace functionality is needed, because the number of tasks are more than 100k. We used Programmers Notepad due to its fast processing capabilities
3	Task reading	Tasks are read using a custom-made Task reader code, which is described in the later sections
4	Application of Algorithms	Separate codes for Greedy Algorithm, CERS algorithm, and CERS algorithm with different sorting mechanisms are developed
5	Result evaluation	Code snippets are added for evaluation of all the 3 primary parameters for the system
6	Result analysis	Excel tool with formula builder was used in order to perform result analysis and graphing
7	Calculation of iQoS	Excel workbooks were integrated in order to calculate iQoS for the system

Figure 1: Steps for Project Development

3.1 Data collection and Pre-processing

Data is collected from the following Huji Parallel workloads website,

http://www.cs.huji.ac.il/labs/parallel/workload/l_nasa_ipsc/index.html

The dataset consists of NASAs tasks loads for the Numerical Aerodynamic Simulation (NAS) Systems Division. It has a SWF format, which looks as follows,

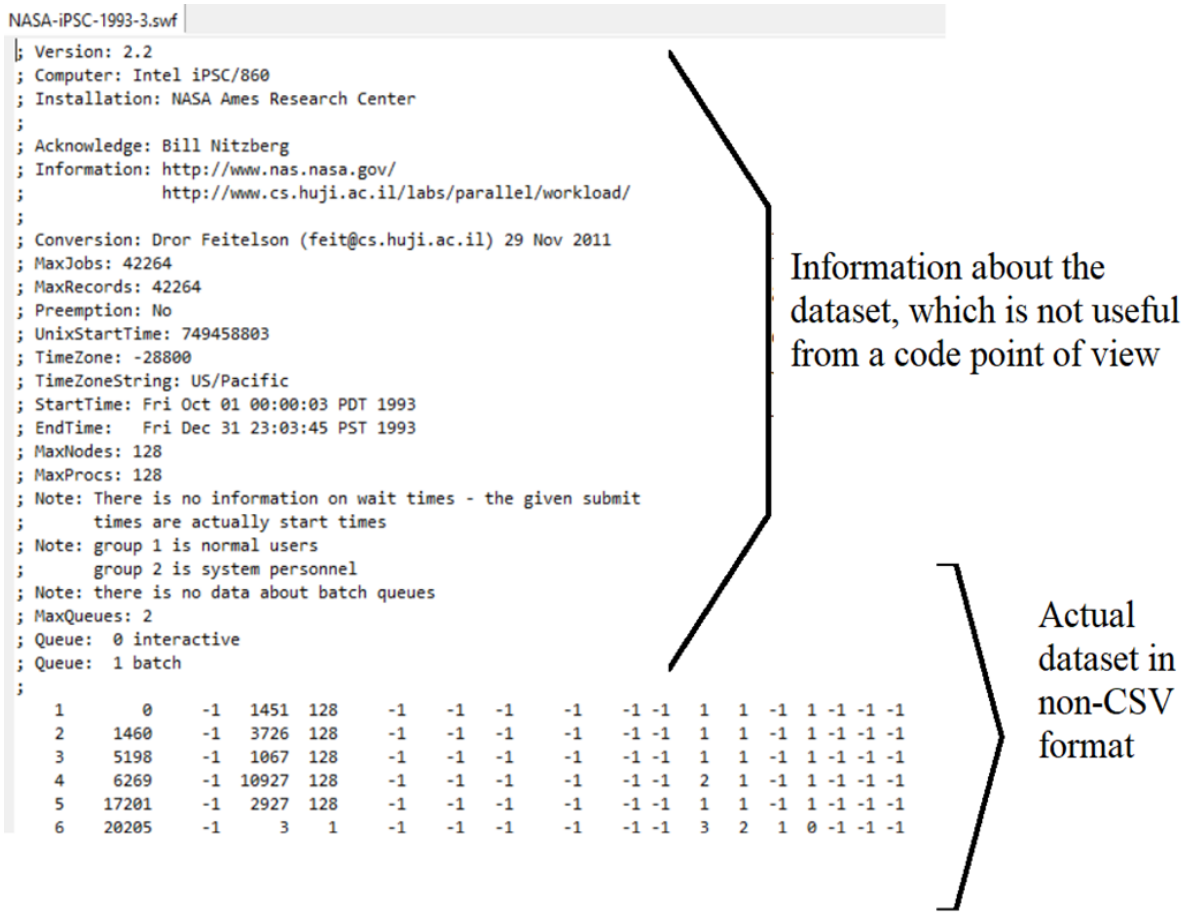

Figure 2: Dataset raw format

We removed the dataset information, and replaced the spaces in the actual data with commas. Once this is done, then we stored the dataset in nasa_set.csv file inside our Eclipse project. The final dataset looks as follows,

nasa_set.csv

```
1,0,-1,1451,128,-1,-1,-1,-1,-1,-1,1,1,-1,1,-1,-1,-1
2,1460,-1,3726,128,-1,-1,-1,-1,-1,-1,1,1,-1,1,-1,-1,-1
3,5198,-1,1067,128,-1,-1,-1,-1,-1,-1,1,1,-1,1,-1,-1,-1
4,6269,-1,10927,128,-1,-1,-1,-1,-1,-1,2,1,-1,1,-1,-1,-1
5,17201,-1,2927,128,-1,-1,-1,-1,-1,-1,1,1,-1,1,-1,-1,-1
6,20205,-1,3,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
7,20582,-1,3,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
8,20654,-1,8,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
9,20996,-1,17,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
10,21014,-1,2,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
11,21043,-1,19,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
12,21097,-1,20,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
13,21142,-1,14,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
14,21206,-1,2,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
15,21360,-1,14,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
16,21405,-1,15,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
17,21449,-1,16,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
18,21496,-1,15,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
19,21568,-1,2,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
20,21655,-1,5,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
21,22008,-1,3,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
22,22083,-1,2,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
23,22418,-1,16,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
24,22463,-1,3,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
25,22468,-1,14,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
26,22519,-1,14,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
27,22565,-1,14,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
28,22628,-1,2,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
29,22802,-1,13,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
30,22846,-1,17,1,-1,-1,-1,-1,-1,-1,3,2,1,0,-1,-1,-1
```

Figure 3: Final dataset for processing

This dataset as shown in Fig 3, is then used by our codes for reading and processing the tasks.

3.2 Task Reading

We have developed a custom task reader code to read the task length and task times. These values are stored inside an array which consists of customized Task objects. The code for the task reader is given as follows,

CERS_ResponseTimeMinimization.java

```
try {
    ArrayList<Task> arrTasks = new ArrayList<Task>();
    BufferedReader br = new BufferedReader(new FileReader(DATASET_FILE_NAME));
    String data;

    int id = 0;
    long totalTaskSize = 0;
    //Read each line
    while((data = br.readLine()) != null) {

        //Get the task details
        String[] data_vals = data.split(",");

        //Get the tasks and their lengths
        //System.out.println(data);
        try {
            int numTasks = Integer.parseInt(data_vals[2]);
            int taskLength = Integer.parseInt(data_vals[3]);

            //Add the task to the list
            Task t = new Task();
            t.taskId = id;
            t.taskLength = taskLength;
            t.numTasks = numTasks;

            //Get the total task size
            totalTaskSize = totalTaskSize + (taskLength * numTasks);

            arrTasks.add(t);
            id++;
        } catch (Exception ex) {
        }
    }
    System.out.println(id + " tasks fetched, total task size:" + totalTaskSize);
```

Figure 4: Task reading code

From the code shown in Fig 4, we can observe that the 2nd and 3rd indices are used for reading the tasks, and these values are stored inside the arrTasks array.

The Task class can be seen from the following figure,

```
public class Task implements Comparable<Task>
{
    int taskId;
    int numTasks;
    int taskLength;
    int taskBW;
    int taskRAM;

    public int totalTaskSize() {
        return numTasks * taskLength;
    }

    @Override
    public int compareTo(Task t) {
        // Sort by lowest task size to highest
        return (this.numTasks > t.numTasks ? -1 :
            (this.numTasks == t.numTasks ? 0 : 1));
    }
}
```

Figure 5: The Task class

From the class(Fig 5) we can observe that the Task class can store various task parameters as needed by our algorithm. Moreover, the compareTo() function performs all the work of sorting the tasks.

4 Description of Codes

There are 5 classes required to run the codes, they are,

- Task – Already defined in section 3, it contains the information about tasks.
- GreedyAlgorithm, CERS2, CERS_ResponseTimeMinimization, CERS_ResponseTimeMinimization – Classes that define each of the algorithms.

The Greedy algorithm is defined in the GreedyAlgorithm.java file, it has the following code(Fig 6),

```
Vm[] availableCloudVMs = new Vm[NUMBER_OF_VMs];
boolean[] processed = new boolean[NUMBER_OF_VMs];
for(int count=0;count<NUMBER_OF_VMs;count++) {
    int mips = 250;
    int cpus = 2;
    int RAM = 512;
    int bw = 1000;
    int size = 1000;

    Vm vm = new Vm(count, count, mips, cpus, RAM, bw, size, "VM:" + count, new CloudletSchedulerTimeShared());
    availableCloudVMs[count] = vm;
    processed[count] = false;
}

double currentMax = 0;
int currentMaxIndex = 0;
vmList = new ArrayList<Vm>();
Date d1 = new Date();

for(int count1=0;count1<availableCloudVMs.length;count1++) {
    for(int count2=0;count2<availableCloudVMs.length;count2++) {
        if(processed[count2] == true)
            continue;

        double usageFactor = availableCloudVMs[count2].getMips() * availableCloudVMs[count2].getPesNumber();

        if(usageFactor >= currentMax) {
            currentMax = usageFactor;
            currentMaxIndex = count2;
        }
    }

    vmList.add(availableCloudVMs[currentMaxIndex]);
    System.out.println("VM " + currentMaxIndex + " added");
    currentMax = 0;
    processed[currentMaxIndex] = true;
```

Figure 6: Code for the Greedy Algorithm

Similarly, the original CERS code is implemented in the CERS2.java file, which has the following code syntax(Fig 7),

```
//Apply the CERS algorithm
double currentMax = 0;
int currentMaxIndex = 0;
vmList = new ArrayList<Vm>();
Date d1 = new Date();

for(int count1=0;count1<availableCloudVMs.length;count1++) {
    for(int count2=0;count2<availableCloudVMs.length;count2++) {
        if(processed[count2] == true)
            continue;

        double qFactor = availableCloudVMs[count2].getMips() / LAMBDA;

        if(qFactor >= currentMax) {
            currentMax = qFactor;
            currentMaxIndex = count2;
        }
    }

    vmList.add(availableCloudVMs[currentMaxIndex]);

    //Add VMs in Descending Q order, and later make it ascending by allocating VMs from the last VM to the first VM
    System.out.println("VM " + currentMaxIndex + " added");
    currentMax = 0;
    processed[currentMaxIndex] = true;

}
 //We need to process in ascending order, so we use the min Q value first
 int vm_number = vmList.size()-1;
 float vm_capacity_left = (float) (vmList.get(vm_number).getMips() * vmList.get(vm_number).getPesNumber());
 long tmwt = 0;
 for(int count=0;count<arrTasks.size();count++) {
     Task t = arrTasks.get(count);
     long tot_task = t.numTasks * t.taskLength;
     float task_length_left = 0;
     if(tot_task <= vm_capacity_left) {
         System.out.println("Task " + t.taskId + ", executing on Vm:" + vm_number);
         vm_capacity_left = vm_capacity_left - tot_task;
     } else {
         task_length_left = tot_task - vm_capacity_left;
         if(vm_capacity_left > 0) {
             System.out.println("Task " + t.taskId + ", executing on Vm:" + vm_number + " for " + vm_capacity_left + " cycles");
         }
         vm_capacity_left = 0;
     }
     if(count > (arrTasks.size()/2))
         tmwt = tmwt + t.numTasks;
     if(vm_capacity_left == 0) {
         //Change the vm
         vm_number--;
         if(vm_number == -1)
         {
             //Check the number of tasks which are remaining, and try to allocate them based on their resource cost
             int best_vm = 0;
             float best_vm_cost_ratio = (float) (vmList.get(best_vm).getMips() * vmList.get(best_vm).getPesNumber()) / task_length_left;
             for(int count2=0;count2<vmList.size();count2++) {
                 float curr_vm_cost_ratio = (float) (vmList.get(count2).getMips() * vmList.get(count2).getPesNumber()) / task_length_left;

                 if(curr_vm_cost_ratio > best_vm_cost_ratio) {
                     best_vm_cost_ratio = curr_vm_cost_ratio;
                     best_vm = count2;
                 }
             }
             System.out.println("Executing remaining task on VM:" + best_vm);
             vm_number = vmList.size()-1;
         }

         team_effort++;
         vm_capacity_left = (long) (vmList.get(vm_number).getMips() * vmList.get(vm_number).getPesNumber());
```

Figure 7: Code for the CERS Algorithm

As we can observe the code works in 2 parts, it first evaluates the 'q' factor and then using VM cost ratio to allocate the tasks. The code for VM initialization can be seen from the following snippet (Fig 8),

```
Vm[] availableCloudVMs = new Vm[NUMBER_OF_VMs];
boolean[] processed = new boolean[NUMBER_OF_VMs];
for(int count=0;count<NUMBER_OF_VMs;count++) {
    int mips = 250;
    int cpus = 2;
    int RAM = 512;
    int bw = 1000;
    int size = 1000;

    Vm vm = new Vm(count, count, mips, cpus, RAM, bw, size, "VM:" + count, new CloudletSchedulerTimeShared());
    availableCloudVMs[count] = vm;
    processed[count] = false;
}
```

Figure 8: Code for VM initialization

Once the algorithm is executed, then we use cloudsim to execute these tasks, the code for cloudsim initialization and task execution can be described as follows(Fig 9),

```
// First step: Initialize the CloudSim package. It should be called
// before creating any entities.
int num_user = 1; // number of cloud users
Calendar calendar = Calendar.getInstance();
boolean trace_flag = false; // mean trace events
int team_effort = 0;
// Initialize the CloudSim library
CloudSim.init(num_user, calendar, trace_flag);
// Second step: Create Datacenters
// Datacenters are the resource providers in CloudSim. We need at
// list one of them to run a CloudSim simulation
@SuppressWarnings("unused")
Datacenter datacenter0 = createDatacenter("Datacenter_0");
// Third step: Create Broker
DatacenterBroker broker = createBroker();
int brokerId = broker.getId();
// Fourth step: Create one virtual machine
// submit vm list to the broker
broker.submitVmList(vmList);
// Fifth step: Create one Cloudlet
cloudletList = new ArrayList<Cloudlet>();
// Cloudlet properties
int id1 = 0;
long length = 1000000;
long fileSize = 300;
long outputSize = 300;
UtilizationModel utilizationModel = new UtilizationModelFull();
try {
    Cloudlet cloudlet = new Cloudlet(id1, length, 1, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);
    cloudlet.setUserId(brokerId);
    cloudlet.setVmId(0);
    cloudletList.add(cloudlet);
    broker.submitCloudletList(cloudletList);
    CloudSim.startSimulation();
    System.in.read();
} catch(Exception ex) {

}
```

Figure 9: The cloudsim execution codes

For CERS with task sorting, we need a sorter class that can sort the tasks in a particular order.

The following snippets show the codes for the sorter classes.

Fig 10 depicts the code for the insertion sort technique.

```
public class MyTaskList extends ArrayList<Task> {
    private static final long serialVersionUID = 1L;

    public void insertionSort()
    {
        int i;
        int pos;
        Task keyelement;
        for (i = 1; i< size();i++)
        {
            keyelement = get(i);
            pos = i;  //position of the key element

            while (pos > 0 && ((Comparable<Task>)get(pos-1)).compareTo((Task)keyelement) >0)
            {
                Task elemPosMinusOne = get(pos-1);
                set(pos, elemPosMinusOne);
                pos = pos -1;
            } //end while loop

            set(pos,keyelement); //insert the key element in the correct position
        }
  }
}
```

Figure 10: Code for Insertion sort

Java has a collection classes which give in-built sorting methods and we have selected the quick sort method. T following code (Fig 11) is used for quick sort,

```
public class MyTaskList extends ArrayList<Task> {
    private static final long serialVersionUID = 1L;

    public void insertionSort()
    {
        int i;
        int pos;
        Task keyelement;
        for (i = 1; i< size();i++)
        {
            keyelement = get(i);
            pos = i;  //position of the key element

            while (pos > 0 && ((Comparable<Task>)get(pos-1)).compareTo((Task)keyelement) >0)
            {
                Task elemPosMinusOne = get(pos-1);
                set(pos, elemPosMinusOne);
                pos = pos -1;
            } //end while loop

            set(pos,keyelement); //insert the key element in the correct position
        }
  }
}
```

Figure 11: Code for task sorting using quick sort

In the given code, the arrTasks array is defined inside the CERS code, while the compareTo() function is defined inside the task class. Collections.sort will be sorting the tasks based on the compareTo() function of the code.