CHAPTER 3
CHAPTER 3
MULTI QUEUE JOB SCHEDULING

3.1 INTRODUCTION

There are various scheduling algorithms that have been discussed in the previous chapter. This chapter presents a Multi Queue job Scheduling (MQS) for cloud computing, the jobs are classified into small, medium and long queue as per the burst time in cloud atmosphere. The proposed research scheduling strategy helps to overcome the problem posed by the existing algorithm. Proposed grouping method helps to improve the Combinational Backfill Algorithm using MQS technique. The main objective of the proposed research is to achieve better utilization of resources and obtain performance enhancements for an application capable of offering various cloud based services. The projected approach improves the scheduler clustering with different kinds of burst time based on jobs in the queue. QoS is a never ending process. The conformance limit is infinity. The service provider frequently analyzes the expectation of the client.

First Come First Serve is the most simple, basic and commonly used method of job scheduling in clusters. In FCFS, jobs are executed as per the arrival of job in the cloud environment. FCFS does need not any help from the queue manager. It is appropriate merely for homogenous type of workload, predominantly for large jobs. In FCFS, parallel job are scheduled at the time order of jobs put forwarded by consumer. It always chooses the first job in the queue to map the suitable computing resources and complete it. This scheduling method system resource exploitation is low and the lowest point case is the large processing job in the head. The waiting job queue may take the computing resources for a long time, while other computing resources are at rest not to be used by extra short jobs as a result, split the computing resources.

Time slicing backfill algorithm made reservation by means of the jobs enter the wall clock register made entry. Time slicing executes the backfill first, estimate the completion time of the entire jobs and then simply implements the program. It
does not furnish any assurance on run time forecast in the vibrant situation. The arrival rates of the jobs are dispatched and low system utilization. A smaller job is waited in the queue as long as it does wait for the previously queued job which completed their execution i.e. small job is allowed to leap forward as long as it does not delay the job at the head of the queue. This denotes that it suffers from fragmentation. It skips over the enhanced condition to consume resources that carry out by scheduling in workload. In open design, it is completed with the help of online manner. It is diverse to determine the resource utilize, make span and throughput. The effective scheduler completely decreases the starvation, however it suffers additional response time and throughput. At last it generates deprived response time. One more drawback in this method is distinguishing the jobs based on any time arrival. Small jobs are moved ahead in the schedule that can fill the resources gap which is generated by FCFS.

Round Robin scheduling is the method in which a FCFS queue is controlled with a monotonous time quantum for every job. EASY backfilling picks and chooses a small job to backfill if it does not impediment the start time of the first job in the queue. The resource utilization is enhanced. Requisite the client estimated run-time of job is lesser. The small jobs are capable to grant more opportunities for backfilling. It is more flexible to backfill. The large job may not be wanted as long as the arrival of others jobs in the queue.

In Combinational Backfill Algorithm, small jobs are getting high priority and avoid large jobs which lead to starvation. The multiple small jobs are grouped together and backfill which maximize the resource utilization but longer jobs are failed to backfill and gets starved. Backfilling allows smaller jobs to move forward in the schedule as long as such movement does not allow the other scheduled jobs in the queue. Space sharing algorithm results in poor usage of jobs despite it takes more waiting and response time that are relatively high. Parallel job scheduling strategies has been widely studied in the past. Existing backfilling scheduling algorithms are only available for one queued job backfilled to schedule and create waste of resource gaps.
Backfilling is an optimization in the framework of variable partitioning. In variable partitioning, users define the number of processors required for each job and this number does not change during the execution. Thus, jobs can be described as requiring a processor/time space that always draws time on the horizontal axis and processors on the vertical axis. The jobs then run on dedicated partitions of the requested size. The name variable partitioning reflects the fact that the partitions are created in different sizes as needed. In backfilling, users also provide an estimate of the runtime. This enables the scheduler to predict the time of jobs termination and, thus, when the next queued jobs are able to run. In particular, it is possible to identify the holes in the schedule and small jobs that can fit into these holes. This is the concept of backfilling algorithm. It is desirable that a scheduler with backfilling supports two conflicting goals: to move as many short jobs forward as possible in order to improve utilization and responsiveness and to avoid starvation for large jobs and, in particular, to be able to predict when each job runs. Different versions of backfilling balance these goals in different ways.

An alternative strategy for conservative backfilling is EASY scheduler which is aggressive backfilling the small jobs that are moved ahead to fill in holes in the schedule, provided that did not delay the first job in the queue. In conservative approach small jobs move ahead only if not delayed on any job in the queue that produces essentially the same benefit in terms of utilization.

3.2 AN EFFICIENT MQS FOR CLOUD COMPUTING

The user submitted jobs are classified based on ascending order of their completion time and it grants equal weightage to all jobs. It is used to improve the customer satisfaction because the customer needs are changing based on the current trends and technology. It decreases the starvation by using the dynamic resource allocation of jobs to pick the best suitable jobs among the available and does not decrease the performance of the computer nodes. The queue manager gives the resources for the basic network. The queue manager is a part of the scheduler which manages the utilization of all the resources in the cluster. It keeps the track of the systems, which are all currently running the jobs by balancing the load among the metascheduler and its disposal. It indicates the scheduler to schedule the output of the
job which is collected back by the queue manager. The scheduling process and resource allocation is based on real time scheduling with unpredictable events.

Three different queues are formed as small, medium and long based on ascending order that measured in terms of burst time of the jobs.

(i) Small queue stores first 40% of jobs.
(ii) Medium queue stores next 40% of jobs.
(iii) Long queue stores remaining 20% of jobs.

The proposed method gives importance to all the jobs because many clients requested in cloud computing and every client expect the fast, reliable and prompt service. The methodology grants equal importance for all in dynamic selection. The best allocation reduces the time and availability of space in an effective manner without compensating the QoS and customer needs.

Drawbacks of Traditional Algorithm:

In the conventional scheduling policy like FCFS, SJF, EASY and CBA clustering, the jobs are not based on their burst time. The fundamental ideas behind the scheduling algorithms are:

(i) In the FCFS job scheduling, the arrival time of jobs wait in queue line as per the arrival.

(ii) SJF scheduling algorithm gives only importance to shortest jobs whereas it ignores the medium and long jobs.

(iii) RR Scheduling is also known as time-slicing scheduling based on quantum time. The interrupt is generated at periodic intervals usually 10-100 ms (milli seconds). The process does not finish within the scheduled time and hence the subsequent job is waiting in the queue.
(iv) Fragmentation happens at several stages that shows waste of energy and extends the price tag of client on pay per consumption.

![Architecture of MQS for Cloud Computing](image)

**Figure 3.1 Architecture of MQS for Cloud Computing**

In the above Figure 3.1 shows the Architecture of MQS using dynamic selection of jobs based on the sorted burst time. The client submitted jobs that are entered into the service provider. The resource manager allocates needed processor based on the client job size called as Cloudlet. The resource manager carries out two
vital roles in queuing of jobs and scheduling. Scheduled jobs are stored in the queued based on execution time. The various categories are long job queue, medium job queue and small job queue. In cloud computing dynamic quantum time implies that the client requests are not predefined one. On the basis of their job completion time the execution set changes their priority for the newly arrived jobs. It grants importance to all jobs.

![Figure 3.2 Queue Forming Technique](image)

The above Figure 3.2 shows the queue forming technique. The newly arrived jobs are updated from the already scheduled job in the queue. The main function of queue manager is handling the scheduled jobs and dispatched them. It sends each job identity to their corresponding queue.

Then the stored jobs enter into the quantum time which denotes environment. In cloud environment, the aim of Metascheduler is to handle submission of a service and manages the resources belonging to the Cloud. Furthermore, Metascheduler also
aims to build the best pool of available resources for the implementation of the requests issued by a user. It is complex to identify the number of steps and time of execution of each job arrived, so that the Metascheduler is dynamic. To achieve Quality System in cloud environment, the proposed MQS algorithm provides to backfill the available jobs. The metascheduler becomes dynamic in nature because of the newly arrived jobs that are entered into scheduling strategies which estimate how long the job runs using that predictions concept. The CloudSim resource simulator uses internal events to simulate the execution and allocation of PEs’ share to Cloudlet jobs. When jobs arrive, the MQS starts their process and executes them. Whenever a new Cloudlet job arrives, scheduler updates the processing time of existing Cloudlets and then adds this newly arrived job to the execution set.

CloudSim schedule is an internal event to be delivered at the earliest completion time of the smallest job in the execution set. The tag number of the newly arrived job is matched with the recently scheduled event, arrived job completes its execution. Depending on the number of Cloudlets in execution and the number of PEs in a resource, CloudSim allocates the appropriate PE share to all Cloudlets for the event duration. It should be noted that Cloudlets sharing the same PE would get an equal amount of PE share. The completed Cloudlet is sent back to its originator (broker or user) and removed from the execution set. CloudSim schedules a new internal event to be delivered at the forecasted time of the remaining Cloudlets.

Algorithm to find the anchor point to check the availability of the processor:

(i) Search the anchor point:

(a) Scanning the summary and find the first point where adequate processors are available to execute this job. This is called the anchor point.

(b) Starting from this point, continue scanning the entire summary to find out the remaining available processors until the jobs expected termination.
(c) If not available, return to (a) and continue the scanning to find the next possible anchor point.

(ii) Update the entire summary to reflect the allocation of processors to this job, starting from its anchor point.

(iii) If the job's anchor is the current time, start it execution immediately.

**Combinational Backfill Algorithm**

CBA scheduling is the state of the free resources, searching suitable jobs from the waiting jobs queue, it selects a group of small jobs to backfill the resources gap to maximize the utilization of resources in clusters. However, existing backfilling scheduling algorithms manage the available jobs that are queued and backfilled to schedule. The resources gap can't be fully utilized. It can select multiple jobs combined from the waiting job to backfill and maximize the use of idle resources.

The major drawback in CBA was that a pair of backfilling is optimum for scheduling the jobs. There are various hybrids of algorithms like CBA with FCFS, CBA with SJF, CBA with EASY and so on. The complexity arose to choose the pair of backfilling that is suitable for effective use of computer nodes and processor needed.

The following assumptions made in CBA are:

In order to facilitate the description of the process of combinational backfilling algorithm for parallel job scheduling, the assumptions are given as follows.

(i) All the jobs are independent to each other. There is no data communication between the jobs during their time execution.

(ii) Jobs in the queue to be scheduled and adjusted dynamically and randomly as needed. There is no precedence constraints relationship among the jobs.
(iii) Jobs submitted to clusters system by the clients must have the estimated execution time. The system estimated execution time of jobs is used based on the average history.

(iv) A job can be a parallel job, and it can also be a serial job which requires only a single computing node to execute.

(v) A job may request ‘n’ computing nodes to execute. Then the computing nodes have been assigned the job and started simultaneously, when the system allocates resources to the job.

(vi) The network bandwidth for data transfer between computing nodes never impacts on the execution time of jobs. In global scheduler time of data transfer of a job are stored and considered. The execution time of each job is not restricted by the network bandwidth.

(vii) The priority of all the jobs is the same. The CBA does not take into account the priority of jobs. CBA processes the parallel job scheduling at three states namely a new job arrives, the running job is finished, the job fails to run or the user terminated the job.

GROUPING ALGORITHM

Grouping the jobs in scheduling is that according to the state of the free resources, searching suitable jobs from the waiting jobs queue, it selects a group of jobs using adaptive and parameterized job grouping method to backfill the resources gap to maximize the utilization of resources in clusters.
Grouping consists of various phases like:

**Phase 1: Initialization**

(i) Get Cloud application and QoS requirements from the user. Determine the total number of job and the size of each job in the file.

(ii) Divide the jobs into three categories (small, medium and large) based on the frequency distribution of the file size, facilitating the grouping of jobs of different file sizes. The three categories of classification mentioned in the queue forming technique is called class interval. If class interval $\leq$ minimum variation, then assume that all the files are of similar size (one category). Compute the mean for each category.

(iii) Obtain the policy details of each resource: number of CPUs, maximum allowed wall-clock-time, CPU time for executing a job and processing cost per second.

**Phase 2: Benchmark**

(i) For each resource, select ‘p’ jobs randomly from the job list and send to the resource.

(ii) Receive all the jobs.

(iii) Select the latest arrived input files, within the last ‘m’ minutes (or time units). Perform average analysis to get the following parameters for each file category-resource combination:

- job CPU time, job wall-clock-time, job processing cost, transmission time for job and output, available network bandwidth, output file size.

(iv) Using the above parameters obtained in Phase 1 and Phase 2 to perform the following:
(a) Check if all the remaining jobs can be processed successfully within user’s QoS requirements.

(b) If (a) is achievable, then for each resource, determine the total number of jobs that can be allocated from each file category. If (a) is not achievable, then advise the user on suitable QoS requirements and exit.

(v) Send and temporarily store the job as per category list.

**Phase 3: Job Scheduling and Deployment**

1. The number of waiting jobs and running stages are collected from the Cloud Monitoring Service: number of jobs in waiting and running stages.

2. Determine the number of jobs ‘x’ to be grouped. Based on the categories the jobs are grouped let it be ‘x’.

   Check the following conditions:

   (i) Job_Group_CPUtime <= Resource maximum allowed CPU Time

   (ii) Job_Group_Wall-Clock-Time <= Resource maximum allowed wall-clock-time

   (iii) Job_Group_Transmission_Time<=max Job_Batch transmission_time

   (iv) Output_Group_Transmission_Time<=max Output_Batch transmission-time

   (v) x <= Total jobs allocated for the resource

3. Select and merge ‘x’ jobs from the unprocessed job list using job merge _program. Send the job group to the resource and job_split_program will be called for further job executions.

4. Repeat steps 1-3 for the remaining resources.
5. Repeat step 4 for ‘n’ iterations, completing one cycle.

6. Repeat steps 3 and 4 in Phase 2, followed by Phase 3 until all the jobs are successfully processed.

Table 3.1 Input Data

<table>
<thead>
<tr>
<th>Job id</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>P₇</th>
<th>P₈</th>
<th>P₉</th>
<th>P₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst Time (ms)</td>
<td>15</td>
<td>90</td>
<td>85</td>
<td>40</td>
<td>35</td>
<td>20</td>
<td>95</td>
<td>70</td>
<td>65</td>
<td>45</td>
</tr>
</tbody>
</table>

For simplicity, the job P₁, P₂,…,P₁₀ with burst time as per the table 3.1 taken as input data to implement the existing various types of scheduling algorithms. Based on the input, the concept of scheduling job in FCFS, SJF and CBA are listed in the table 3.2, 3.3 and 3.4 respectively.

Table 3.2 Output in FCFS

<table>
<thead>
<tr>
<th>Job id</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>P₇</th>
<th>P₈</th>
<th>P₉</th>
<th>P₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst Time (ms)</td>
<td>15</td>
<td>90</td>
<td>85</td>
<td>40</td>
<td>35</td>
<td>20</td>
<td>95</td>
<td>70</td>
<td>65</td>
<td>45</td>
</tr>
</tbody>
</table>
In the above Table 3.2 shows the output of FCFS that have 10 different job id with various burst time. At the time of execution, the jobs are dispatched on the basis of arrival time. There is no preference in scheduling.

**Table 3.3 Output in SJF**

<table>
<thead>
<tr>
<th>Job id</th>
<th>P₁</th>
<th>P₆</th>
<th>P₅</th>
<th>P₄</th>
<th>P₁₀</th>
<th>P₉</th>
<th>P₈</th>
<th>P₃</th>
<th>P₂</th>
<th>P₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst Time (ms)</td>
<td>15</td>
<td>20</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>65</td>
<td>70</td>
<td>85</td>
<td>90</td>
<td>95</td>
</tr>
</tbody>
</table>

The above Table 3.3 depicts the output of SJF with various types of job id that are scheduled in the order of job id P₁, P₆, P₅, P₄, P₁₀, P₉, P₈, P₃, P₂ and P₇ based on their burst time of input given.

**Table 3.4 Output in CBA**

<table>
<thead>
<tr>
<th>Job id</th>
<th>P₁</th>
<th>P₄</th>
<th>P₅</th>
<th>P₂</th>
<th>P₃</th>
<th>P₆</th>
<th>P₈</th>
<th>P₇</th>
<th>P₉</th>
<th>P₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst Time (ms)</td>
<td>15</td>
<td>40</td>
<td>35</td>
<td>90</td>
<td>85</td>
<td>20</td>
<td>70</td>
<td>95</td>
<td>65</td>
<td>45</td>
</tr>
</tbody>
</table>

The above Table 3.4 shows the output of CBA with different jobs and their execution time. The scheduling process takes place in the order of P₁, P₄, P₅, P₂, P₃, P₆, P₈, P₇, P₉ and P₁₀.
The following Table 3.5 shows the proposed MQS algorithm grants opportunity for all the jobs.

**Table 3.5 Output in MQS**

<table>
<thead>
<tr>
<th>Job id</th>
<th>P_1</th>
<th>P_{10}</th>
<th>P_2</th>
<th>P_6</th>
<th>P_9</th>
<th>P_7</th>
<th>P_5</th>
<th>P_4</th>
<th>P_8</th>
<th>P_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (ms)</td>
<td>15</td>
<td>20</td>
<td>45</td>
<td>65</td>
<td>90</td>
<td>35</td>
<td>40</td>
<td>70</td>
<td>85</td>
<td>90</td>
</tr>
</tbody>
</table>

The above different table shows the comparison of same input data with various algorithms. The order of jobs after scheduling are P_1, P_{10}, P_2, P_6, P_9, P_7, P_5, P_4, P_8 and P_3.

(i) In small queue job id P_1, P_6, P_5, P_4 are queued.

(ii) In medium queue job id P_{10}, P_9, P_8, P_3 are queued.

(iii) In long queue job id P_2 and P_7 are queued.

The scheduler takes job id P_1 and P_6 from small queue and job id P_{10} and P_9 from medium queue and job id P_2 from long queue correspondingly and repeats for all jobs.

The sequence in 1^{st} Iteration is P_1 \rightarrow P_6 \rightarrow P_{10} \rightarrow P_9 \rightarrow P_2

The sequence in 2^{nd} Iteration is P_2 \rightarrow P_4 \rightarrow P_8 \rightarrow P_3 \rightarrow P_7

**3.3 RESULTS AND DISCUSSION**

The CloudSim resource simulator uses internal events to simulate the execution and allocation of PEs that share to Cloudlet jobs. When jobs enter into the
queue manager, MQS starts their execution jobs. When new job arrives, scheduler updates the burst time in the existed cloudlets execution set. The CloudSim assigns the tag number for each job. Tag number and scheduled event are same and also it denotes the completion of the job. The CloudSim decides the number of PE that is based on the number of jobs in the system. Once job is finished, it is removed from the execution set. CloudSim schedules and delivered the event based on the predicted burst time of the left over jobs. The simulation of the MQS algorithm and the Cloudlets’ execution in small queue $P_1$, $P_6$, $P_5$ and $P_4$ are queued, in medium queue $P_{10}$, $P_9$, $P_8$ and $P_3$ are queued and in long queue $P_2$ and $P_7$ are queued. The scheduler take job id $P_1$ and $P_6$ from small queue, job id $P_{10}$ and $P_9$ from medium queue and job id $P_2$ from long queue correspondingly. This is repeated for all jobs and here the sequence of job becomes

$$P_1 \rightarrow P_6 \rightarrow P_{10} \rightarrow P_9 \rightarrow P_2 \rightarrow P_5 \rightarrow P_4 \rightarrow P_8 \rightarrow P_3 \rightarrow P_7$$

3.4 COMPARISON OF MQS WITH TRADITIONAL BACKFILL ALGORITHM

![Job Completion Ratio](image.png)

Figure 3.3 Performance of MQS with Traditional algorithm
The above Figure 3.3 shows the performance of MQS with Traditional algorithm. The X-axis shows the number of cloudlet and Y-axis represents the processing time of the job. Traditional algorithm is backfilled with the help of FCFS. The number of cloudlet is processed in the CloudSim that is low when compared to MQS. The cloudlet size is below 2500 FCFS provides positive result. The size of cloudlet above 2500 the MQS provides better results.

3.5 COMPARISON OF MQS WITH CBA

The following Figure 3.4 shows the performance of MQS with CBA. The cloudlets are represented in X-axis and processing time is represented in the Y-axis for scheduled jobs completion ratio.
The number of cloudlets between 1300 - 2200 the CBA provides better results, if the cloudlet size is above 2200 and above the MQS provides efficient result.

3.6 COMPARISON OF MQS WITH EXISTING ALGORITHMS

The following Figure 3.5 shows the performance of MQS with Traditional Backfill and CBA algorithms. The result of MQS gives better and satisfactory one when compared to the traditional backfill algorithm.

![Job Completion Ratio](image)

**Figure 3.5 Performance of MQS with Existing Algorithms**

Based on the equal priority, number of processor needed and burst time aspect the MQS algorithm returns better results. The proposed scheduling algorithm attains the optimum usage of resources for cloud computing and attains high resource utilization and provides QoS in cloud environment.
Comparative study of GA, ACO and PSO algorithm

The comparative study of various algorithms like Genetic Algorithm, Ant Colony Optimization and Particle Swarm Optimization for allocation, it distributes the jobs. GA is widely used for various application oriented techniques like searching to find the nearest value to optimization solution for the problem. It requires more computing power and large space for storage. In GA values are represented in chromosome, followed by semantics is defined by the encoding method. When encoding is over the initial population, it generates the initial search. In every generation it performs the fitness function, selection, crossover, mutation and termination.

Initial population:

In the initial population the chromosome are generated randomly. It creates random values.

Fitness function:

Every chromosome is evaluated by fitness function. The main function of fitness function is to measure the performance and whether the complete the process within the expected finishing time.

Selection:

The main function of selection is to identify the best fit chromosome and select the chromosome from the current population to next.

Crossover:

It is used to combine or mix all the pairs of chromosomes, to generate the next generation. Crossover selects the chromosome from individuals and mixes their genes and produces a new pair of chromosome.

Mutation:

Mutation process is just like crossover, the difference is interchanging the value within the particular pair. Mutation made small changes in each chromosome. It helps to identify the new pair.
Termination of the algorithm:

It helps to identify the suitable values from the given set of chromosome. If the condition is satisfied the give the optimum solution for the problem.

**GA, ACO and PSO Algorithms**

**GA Pseudocode [74]**

(i) Initialization of the population generated randomly  
(ii) Calculate the fitness function of each node  
(iii) While the scheduling process did not meet the termination condition 
   a. Implement the Genetic Operator such as Selection, Crossover, Mutation and Evaluate of individuals  
   b. Create a new population  
(iv) End while

**ACO**

ACO is a kind of scheduling algorithm based on the activities of ants in the forest. It helps to resolve very large complex scheduling problems and gives better results. It is effective for local search when compared to global or meta search. The time taken to finish all the jobs is good and reduce the load imbalance. It takes more response time and cost. The main advantage of ACO changes its work according to their present environment, because cloud computing is dynamic based. It has so many kinds of ACO algorithm they are available like ant colony system, fast ant system, rank based and max-min based algorithm.

**ACO Pseudocode [75]**

Declaration  
While (until condition met)  
{  
For (every ant)  
{  
Selection;
Building;
} 
While ()
{
Choose
Update Pheromones
}
};

PSO

The below Figure 3.6 shows the flow chart of PSO. The main advantage of PSO is speedy process when compared to GA and ACO. It also provides good result for large processing of jobs. There are various advantages of PSO are scalability, flexible and opt for dynamic changing environment. The peculiar feature of PSO is to find solution without any previous information. It has two vectors like position and velocity.

![Flowchart of PSO](image)
Each particle is assumed to have a local memory that keeps its previous position called pbest. The pbest is interchanged by the current position of the each particle and its position vector. The neighbour local gbest are also adjusted based on velocity and dimension of the particle. The adjusted velocity changes the dimension of the position. Every time the particle updates the velocity and position of the particle.

**PSO Pseudocode [80]**
Generate the required variable
Assign the value for the variable
Find gbest
While()
{
    Generate value()
    Calculate value()
    Update ()
}
End

**Simulation of Comparative Results**

![Job Completion Ratio](image)

**Figure 3.7 Overall Performances of GA, ACO and PSO**
The above Figure 3.7 shows that comparative study of various jobs scheduling algorithm produce the satisfactory results. The allocation of jobs gives the enhanced results and makespan rate is good. Number of cloudlets between 250 - 500 makespan GA gives speedy result. The cloudlets between 1000 - 2500 the makespan of ACO gives better result and in the PSO makespan between 3500 and above it gives better result. If the cloudlet size is low, GA is suitable, for moderate stable ACO gives good performance and for more number of cloudlet PSO provides good results. The job completion measured in terms of number of cloudlets and processing time (MS). It helps to achieve the effective resource utilization and gives QoS for the clients.

3.7 SUMMARY

This chapter describes the MQS approach effectively and identifies the drawbacks in the traditional system. It can identify the fragmentation that occurs during the time of scheduling, with the help of MQS that provide results and shows it performance well against the traditional and CBA. Furthermore, it overcomes the drawbacks on improved backfill. Similar scheduling algorithms have been tried by many researchers in the past. These experiments however were done based on MQS. Not only has it finds the unused space in the processor, it also finds a better resource balance.