# A Novel Dynamic Priority Based Job Scheduling Approach for Cloud Environment

**Er. Shakeel Ahmad [1], Er. Imtiyaj Ahmad[2], Er. Sourav Mirdha[3]**

*[1,2]M.Tech. Student, Computer Science & Engineering, International Institute of Engineering & Technology, Samani, Kurukshetra, Haryana, India*
*[3]Assistant Professor, Computer Science & Engineering, International Institute of Engineering & Technology, Samani, Kurukshetra, Haryana, India*

---------------------------------------------------------------------***---------------------------------------------------------------------

**Abstract -** *Scheduling of jobs is a foremost and difficult issue in Cloud Computing. Utilizing cloud computing resources efficiently is one of the Cloud computing service provider's ultimate goals. Today Cloud computing is on demand as it offers dynamic flexible resource allocation for trustworthy and definite services in pay-as-you-use manner, to Cloud service users. So there must be a provision that all resources should be made available to demanding users in proficient manner to satisfy their needs. In this dissertation author has proposed a new dynamic priority based job scheduling algorithm in cloud computing to optimize the problem of starvation. The priority in proposed algorithm is based on multiple criteria such as CPU Resource Requirement, IO Resource Requirement and JOB criticality. The proposed model aims to reduce the waiting time, turnaround time of jobs and to increase the throughput and CPU utilization of complete system. A comparison with SJF algorithm in terms of waiting time, turnaround time and total finish time is performed. Simulation of work has been done on CLOUDSIM.*

***Key Words***: **Cloud Computing, Task Scheduling, Cloudsim, Shortest Job First**

## 1. INTRODUCTION

Cloud Computing is a term used to illustrate both a platform and type of application. As a platform it supplies, configures and reconfigures servers, while the servers can be physical machines or virtual machines. On the other hand, Cloud Computing describes applications that are extended to be accessible through the internet and for this purpose large data centers and powerful servers are used to host the web applications and web services [1].
NIST is a well accepted institution all over the world for their work in the field of Information Technology. NIST defines the Cloud Computing architecture by describing five essential characteristics, three cloud services models and four cloud deployment models is shown in figure 1 where layered architecture is shown [2]
On demand self service, broad network access, resource pooling, rapid elasticity and measured services are 5 essential characteristics of Cloud computing which explains there relation and difference from the traditional computing system.

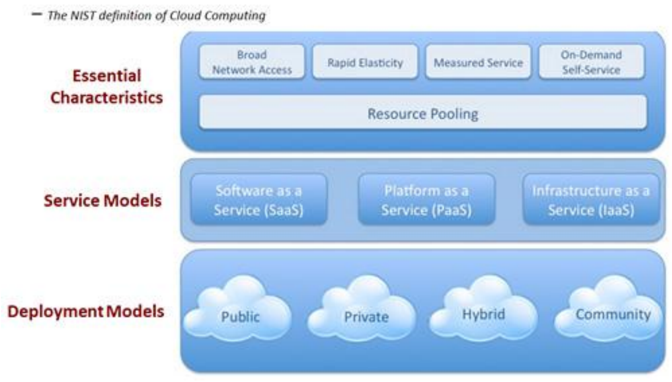

The NIST definition of Cloud Computing
Essential Characteristics
Broad Network Access
Rapid Elasticity
Measured Service
On-Demand Self-Service
Resource Pooling
Service Models
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)
Deployment Models
Public
Private
Hybrid
Community

**Fig-1:** Cloud computing model given by NIST [2]

## 2. JOB SCHEDULING

Scheduling is a process of finding the capable resources that can execute the cloud requests (tasks) at specific times that satisfy specific performance quality measure such as execution time minimization, as specified by cloud users. The main goal of job scheduling is to achieve a high performance computing and the best system throughput [3].

Schedulers employ a function that takes into account the essential objectives to optimize a specific outcome. The commonly used scheduling reason in a cloud computing environment is related to the tasks completion time and resource utilization. The scheduler uses a particular policy for mapping the tasks to suitable Grid/Cloud resources in order to satisfy user requirements. However, the bulk of these scheduling strategies are static in nature. They produce a good plan given the current state of Cloud resources and do not take into account changes in resource accessibility. On the other hand, dynamic scheduling considers the current state of the system. It is adaptive in nature and able to fabricate efficient schedules, which ultimately reduces the completion time of tasks as well as improves the overall performance of the system [4].

### 2.1 Starvation

**Starvation** is a resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

Starvation generally occurs in a Priority based scheduling System where high priority requests get processed first. Thus a request with least priority may never be processed.

**Aging** is a technique to reduce starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the requests priority as time passes and must ensure that a request will eventually be the highest priority request [5].

## 3. CLOUDSIM SIMULATOR

Cloudsim is a new generalized and extensible simulation framework that enables flawless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and management services. Cloudsim has the following novel features:

1. Support for modeling and instantiation of large scale Cloud computing infrastructure, including data centers on a single physical computing node and java virtual machine
2. Independent platform for modeling data centers, service brokers, scheduling, and allocations policies
3. Accessibility of virtualization engine, which assist in creation and management of multiple, independent, and co-hosted virtualized services on a data center node
4. Flexibility to switch between space-shared and time-shared allocation of processing cores to virtualized services.

Cloudsim is implemented in JAVA language which is based on the object oriented programming concepts. Class defines as abstract unit in OOP concepts [6].

## 4. RELATED WORK

D. Dutta et al. in [7] suggested a genetic algorithm approach to cost based multi QoS job scheduling. A model for cloud computing environment has been also proposed and some popular genetic cross over operators, like PMX, OX, CX and mutation operators, swap and insertion mutation are used to produce a better schedule. The algorithm assures the best solution in finite time.
P. Kumar et al. in [7] have discussed various forms of mapping cluster topology requirements into Cloud environments to achieve higher reliability and scalability of application carry out within Cloud resources and enabling the scheduler to make best use of CPU utilization while remaining within the constraints imposed by the need to optimize user Quality of Service (QOS). The focus of the paper is to provide a dynamic scheduler that aims to maximize user satisfaction. Thus the job details submitted by the user will include job prioritization criteria i.e. the allocated budget and the deadline required by the user, enabling the scheduler to maximize CPU utilization while remaining within the constraints imposed by the need to optimize user Quality of Service (QOS).
M. Paul et al. in [9] have proposed scheduling mechanism which follows the Lexi – search approach to assign the tasks to the available resources. The scheduled task will be preserved by a load balancing algorithm that allocate the pool of task into small partition and then distribute into local middleware. Cost matrix was generated from a probabilistic factor based on some most vital condition of efficient task scheduling such as task arrival, task waiting time and the most important task processing time in a resource. The recommended method considered the scheduling problem as the assignment problem in mathematics here the cost matrix gives the cost of a task to be assigned into a resource. Cost had been considered as credit or the probabilistic measurement thus only the processing time of a job is not been given importance but the other issues are considered such as the probability of a resource to be free soon after executing a task so that it will be available for other waiting job. Job which has the highest probability to get a resource as well as the resource which fits better for a job is assigned in a manner that one resource get one job at a time. The load balancing mechanism in the central middleware decreases the overhead of scheduling on a single middleware by partitioning the job queue thus scalability issues is well maintained and making the duplication of the partitioned job queue ensures the fault tolerant in the cloud since if any of the client fail then that job could be reassigned into another client by another local middleware as the local middleware interact each other for every job updates. The proposed methodology does not need any complex network architecture than other job scheduling network architecture in the cloud.
C.S. Pawar et al. in [10] had put forwarded an algorithm which considered preemptive task execution and multiple SLA parameters such as memory, network bandwidth, and required CPU time. Proposed algorithm dynamically reacts to fluctuating work load by preempting the current executing task having low priority with high priority task and if preemption is not possible due same priority then by creating the new VM form globally accessible resources. An achieved experimental results show that in a situation where resource contention is severe proposed algorithm (PBSA ) perform better than CMMS in resource contention situation and affords better utilization of resources.
A. Tumanov et al. in [11] discussed the need for and an approach for accommodating diverse tenant needs, based on having resource requests indicate any soft (i.e., when certain resource types would be better, but are not mandatory) and hard constraints in the form of composable utility functions. They proposed scheduler that acknowledges such requests that can then maximize overall utility, perhaps weighted by priorities, taking into account application specifics. Done Experiments with a prototype scheduler, called alsched, reveal that support for soft constraints is important for efficiency in multi-purpose clouds and that composable utility functions can provide it.
A. Jain et.al. in [12] has critically evaluated the performances of different scheduling algorithms found in literature. The

request time for the three policies applied (Round Robin, Equally spread current execution load, Throttled Load balancing) are same which means there is no effect on data centers request time after changing the algorithms. The cost analysis illustrated for each algorithm is calculated in the experimental work. The cost calculated for virtual machine usage per hour is same for two algorithms Round Robin, Equally spread current execution load but Throttled Load balancing algorithm lessen the cost of usage, so Throttled Load balancing algorithm works more efficiently in terms of cost for load balancing on cloud data centers.

## 5. PROPOSED WORK

Modular representation of proposed approach is shown below in figure 2.

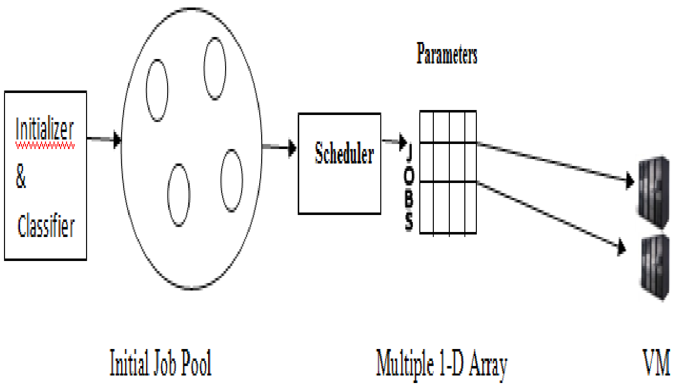


**Fig -2:** Modular representation of proposed scheduling approach

Initialize and scheduler are the major modules. Their functionality is as follows:

- **Functionality of Initialize & Classifier**

It is the module that has been generated in Cloudsim which allocates the five chosen characteristics value to each job and assigns the initial priority. Initial job pool is created in this module. The functionality of this module is stated below:

1. It creates the JOBS randomly through CLOUD SIM
2. With each incoming job, some parameters as associated to all the incoming JOBS
3. Each 1-D array represents the attribute associated with Job or Process to be executed.
4. IO resource requirement, CPU requirement, Arrival time, job execution time and job criticality are static parameters.
5. Priority, Wait time, turnaround time and finish time are calculated dynamically.

- **Functionality of Scheduler**

Scheduler is a module which is responsible for allocation of jobs to virtual machines on the basis of some priority value.

1. JOBS in job pool will be initially arranged in ascending order of their arrival time.
2. Priority for each job is calculated based upon the values of CPU Requirement, Resource Requirement and Job Criticality.
3. Jobs are arranged in descending order of priority.
4. Allocations of JOBs to VM are at runtime depending upon the availability of VM.
5. Searching of VM based upon least execution time is also done to allocate the unassigned jobs.
6. Priority of unassigned jobs is again calculated and incremented by 1 if wait time of job exceeds wait threshold value.
7. Non-Preemptive dynamic Scheduling is performed

"**Starvation Optimizer Scheduler**" is dynamic algorithm based upon the priority assigned to each task. The algorithm starts its operation by first creating the job pool where in jobs are created and five characteristics (Arrival time, CPU execution time, CPU requirement, IO resource requirement and job criticality) are associated with each job. These characteristics form the basis to calculate or assign the initial priority for each job. Once the priority of jobs is calculated, jobs are sorted in descending order of priority. Higher priority jobs are assigned to virtual machines. For the remaining unassigned jobs search for VM having least execution time is done before allocation of jobs. The assignment of job to VM depends upon their priority. If the waiting time of job exceeds the wait threshold value, priority of job is incremented by one. Finally wait time, turnaround time and finish time for each job is calculated. Following formulas are used for calculation:

**Wait Time: =** Start Time – Arrival Time
**Turn Around Time:** Finish Time –Start Time
**Throughput:** (CPU Clocks used in process execution)/ (Total Clocks)*100

Algorithmic form of starvation optimizer scheduler is as follows:

```
1  Enter the number of jobs to be executed.
2  While (J!=NULL) // J is Job Pool
3  For each Job ($j_i$) ϵ J
       Initialize arrival_time, execution_time, cpu_requirement, IO_requirement & Job_criticality.
   End For
4  Arrange all Jobs $j_i$ in the ascending order of arrival time.
5  For each Job ($j_i$) ϵ J
   Calculate the Job_Priority ($P_i$) based upon cpu_requirement,IO_requirement & Job_criticality.
   End For
6  Arrange jobs in descending order of priority.
7  Allocate high priority jobs to VM for execution.
8  For each Job ($j_i$) ϵ J
     a) Compare job_wait_time ($WT_i$) with wait_threshold
```

```
        b)  If job_wait_time (WT$_i$) > wait_threshold THEN
            Increment the priority for job (j$_i$) by 1
        End for
9       Search VM ( V$_k$) having least execution time.
10      For each Job (j$_i$) ϵ J
        Allocate high priority jobs to VM( V$_k$) and calculate
        wait time and turnaround time by using following
        formulae:
          a)  Wait Time = Start Time (j$_i$) – Arrival Time( j$_i$)
          b)  Turn Around Time= Finish Time(j$_i$) –Start
              Time(j$_i$)
          c)  Update the status of Job to complete.
        END While
```

## 6. RESULTS & ANALYSIS

### 6.1 Simulation configuration

A simulation program is implemented in JAVA language with the help of Cloudsim tool kit to optimize the starvation problem in cloud environment. Simulation is implemented under following set of assumptions:

- Type of Scheduling: Non pre-emptive and Dynamic.
- For same priority jobs FCFS scheduling policy will be used.
- Highest priority value is 3 and lowest is 1.
- Waiting threshold value is 40 second.

Random job pool of six jobs is created. For each job - Arrival Time, CPU Clock, CPU Requirement, Resource Requirement and Job Criticality is provided as input characteristic.

Table 1 shows values for the above characteristics associated with each job
.

**Table -1**: Input Data Set for SOS Algorithm

| **Process** | **Arrival Time** | **CPU Execution** | **CPU Requirement** | **Resource Requirement** | **JOB Criticality** |
|---|---|---|---|---|---|
| P0 | 2 | 10 | 3 | 3 | 3 |
| P1 | 2 | 20 | 1 | 1 | 1 |
| P2 | 2 | 30 | 2 | 2 | 2 |
| P3 | 3 | 25 | 1 | 2 | 3 |
| P4 | 3 | 20 | 3 | 2 | 3 |
| P5 | 3 | 10 | 3 | 5 | 2 |

### 6.2 Results

Scheduler will assign the jobs to VM and for each job finish time, wait time and turnaround time is calculated as output. The obtained output characteristics values are shown in table 2.

**Table -2**: Output Data Set for SOS Algorithm

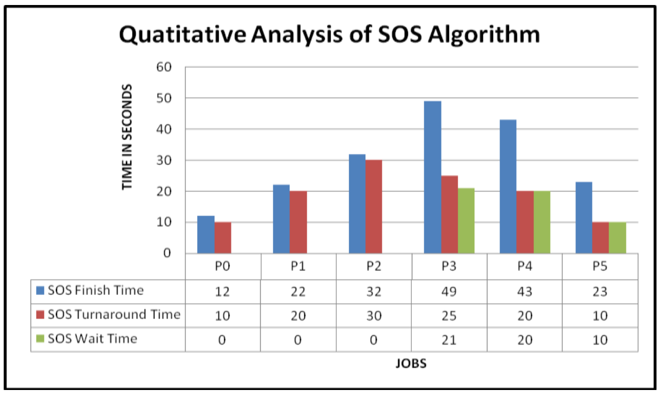
| **Process** | **Finish Time** | **Turnaround Time** | **Wait Time** | **Priority** |
|---|---|---|---|---|
| P0 | 12 | 10 | 0 | 2 |
| P1 | 22 | 20 | 0 | 1 |
| P2 | 32 | 30 | 0 | 1 |
| P3 | 49 | 25 | 21 | 3 |
| P4 | 43 | 20 | 20 | 3 |
| P5 | 23 | 10 | 10 | 2 |

Evaluation summary for all the jobs comprising of total finish time, CPU Utilization, throughput, average turnaround time and average waiting time is presented in table 3.

Table 3: Evaluation Summary for SOS

| **Parameters** | **Values** |
|---|---|
| Total Finish Time | 49 |
| CPU Utilization | 0.87 |
| Throughput | 40 |
| Average Turnaround Time | 19.16 |
| Average Waiting Time | 8.5 |

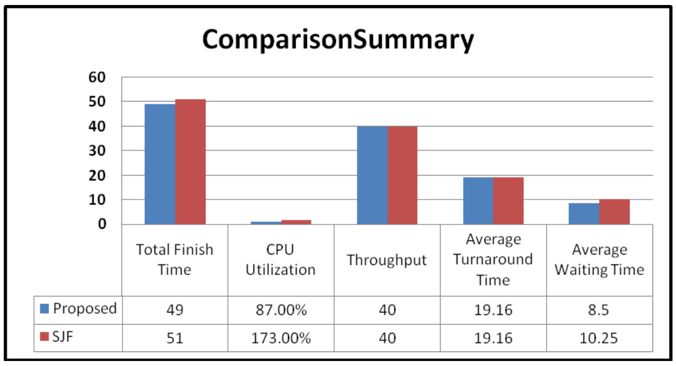
Quantitative analysis of SOS algorithm is presented in figure 3. For each job- finish time, turnaround time and wait time is depicted with different colors. Finish time is shown in blue color, turnaround time in red color and wait time in green color. As for the first three jobs, the wait time is zero, so the weight time bar is not figured.



| | P0 | P1 | P2 | P3 | P4 | P5 |
|---|---|---|---|---|---|---|
| 🟦 SOS Finish Time | 12 | 22 | 32 | 49 | 43 | 23 |
| 🟥 SOS Turnaround Time | 10 | 20 | 30 | 25 | 20 | 10 |
| 🟩 SOS Wait Time | 0 | 0 | 0 | 21 | 20 | 10 |

**Fig -3:** Quantitative Analysis of SOS Algorithm

Quantifying results in figure 4 shows that by dynamically increasing the priority of jobs, average waiting time and total finish time of the complete system is reduced approximately to 23% and 4% respectively. Hence problem of starvation is optimized in "Starvation Optimizing Scheduler" algorithm.



| | Total Finish Time | CPU Utilization | Throughput | Average Turnaround Time | Average Waiting Time |
|---|---|---|---|---|---|
| 🟦 Proposed | 49 | 87.00% | 40 | 19.16 | 8.5 |
| 🟥 SJF | 51 | 173.00% | 40 | 19.16 | 10.25 |

**Fig -4:** Quantitative Analysis of SOS Algorithm

## 7. CONCLUSIONS

Job scheduling problem is important and challenging issue in Cloud Computing. Utilizing cloud computing resources proficiently and gaining the highest profits with job scheduling system is one of the Cloud computing service providers' ultimate goals.
Research done earlier in this area was focused on mapping of tasks to machines efficiently but still problem of starvation persists. So to resolve this issue major focus of this paper has been put on optimizing the starvation. An algorithm "SJF" mainly suffers from this problem. New algorithm is generated "**Starvation Optimizing Scheduler**" which aims to reduce the starvation. Following objectives have been met satisfactorily which are stated below:

- Jobs are allocated to VM's dynamically at run time.
- The average waiting time, average turnaround time and total finish time of jobs are reduced.
- Starvation problem is optimized.

Also this work can be extended in future in the following way:

1 In this work, author has input the jobs only once under different arrival time specification, but no work is defined for the job input during the job execution. In future, work can improved by including the anytime participation of user in terms of job input.

2 In this work, jobs are defined in non-preemptive way, but in future thee technique of preemption can be used for allocating resources to jobs.

## REFERENCES

[1] A. Jain and R. Kumar, "A Taxonomy of Cloud Computing," International Journal of Scientific and Research Publications. vol. 4(7), Jul. 2014, pp. 1-5.

[2] G. Brunette & R. Mogull, "Security guidance for critical areas of focus in cloud computing v2. 1," Cloud Security Alliance, 2009, pp. 1-76.

[3] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg & I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility". Future Generation computer systems, vol. 25(6), 2009, pp. 599-616.

[4] H. Topcuoglu, S. Hariri & M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," Parallel and Distributed Systems, vol. 13(3), 2002, pp.260-274.

[5] M. Rahman, S. Venugopal & R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," In e-Science and Grid Computing, IEEE International Conference, 2009, pp. 35-42.

[6] R. Buyya, R. Ranjan & R.N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," High Performance Computing & Simulation, HPCS'09. International Conference, 2009, pp. 1-11.

[7] D. Dutta & R.C. Joshi, "A genetic: algorithm approach to cost-based multi-QoS job scheduling in cloud computing environment". In Proceedings of the International Conference & Workshop on Emerging Trends in Technology, 2011, pp. 422-427.

[8] P. Kumar, N. Nitin, V. Sehgal, D. S. Chauhan & M. Diwakar, "Clouds: Concept to optimize the Quality of Service (QOS) for clusters," In Information and Communication Technologies (WICT), 2011, pp. 816-821.

[9] M. Paul, D. Samanta, & G. Sanyal, "Dynamic job Scheduling in Cloud Computing based on horizontal load balancing," International Journal of Computer Technology and Applications (IJCTA), vol. 2(5), 2011, pp. 1552-1556.

[10] C.S. Pawar & R.B. Wagh, "Priority Based Dynamic resource allocation in Cloud Computing," In Cloud and Services Computing (ISCOS), 2012, pp. 1-6.

[11] A. Tumanov, J. Cipar, G.R. Ganger & M.A. Kozuch,"Algebraic scheduling of mixed workloads in heterogeneous clouds," InProceedings of the Third ACM Symposium on Cloud Computing, 2012, pp. 25-30

[12] A. Jain and R. Kumar, "A Comparative Analysis of Task Scheduling Approaches for Cloud Environment," International Conference On Computing for Sustainable Global Development, 2016, pp. 2602-2607.