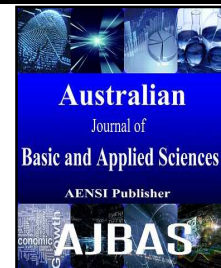




ISSN:1991-8178

Australian Journal of Basic and Applied Sciences

Journal home page: www.ajbasweb.com



Scheduling Jobs Using Oppositional-GSO Algorithm in Cloud Computing Environment

¹S. Parthasarathy and ²C. Jothi Venkateswaran

¹Assistant Professor Department of MCA Valliammai Engineering College, Anna University

²Associate Professor Department of MCA, Presidency College, Chennai – 5

ARTICLE INFO

Article history:

Received 3 October 2015

Accepted 31 October 2015

Key words:

Cloud Computing, execution time, oppositional operation Scheduling, Producer operation, ranger operation, scrounger operation

ABSTRACT

The basic idea of our research is to scheduling jobs using oppositional-GSO algorithm in cloud computing environment. Cloud computing is an emerging domain that requires more algorithm and techniques for various process. The scheduling process in cloud computing platform needs a good algorithm to schedule the jobs of different users. In this paper, initially we generate a population that contains a group of members and the members consist of the number of users and their respective jobs. The motto of our research is to schedule the user given jobs efficiently. We separate the members from the population based on the fitness to perform different operations to generate new members. We calculate the fitness for the new members and iterate the process until we get a stable best member for repeated iteration. We then schedule the jobs of the users based on the best member obtained.

© 2015 AENSI Publisher All rights reserved.

To Cite This Article: S. Parthasarathy and C. Jothi Venkateswaran., Scheduling Jobs Using Oppositional-GSO Algorithm in Cloud Computing Environment. *Aust. J. Basic & Appl. Sci.*, 9(33): 145-155, 2015

INTRODUCTION

Computing is being converted to a model comprising of services that are commoditized and distributed in a manner alike to traditional services such as water, electricity, gas, and telephony. In such a model, users access services derived from their needs without respect to where the services are hosted or how they are distributed. A number of computing paradigms have promised to distribute this utility computing vision and these comprise cluster computing, Grid computing, and more recently Cloud computing Buyyaa, R., Yeo, C.S., *et al.* (2009) Cloud computing is an budding information technology that shifts the way IT architectural solutions are put forward through moving towards the theme of virtualization: of data storage, of local networks (infrastructure) as well as software Leavitt, N., *et al.*, (2009) Weinhardt C., *et al.* (2009) Cloud computing offers full scalability, reliability, high performance and comparatively low cost feasible solution contrast to committed infrastructures. It is the application offered in the form of service over the internet and system hardware in the data centers that gives these services. This technology has the potential to access a common collection of resources on request. It offers tremendously striking to cash-strapped IT departments that are required to deliver superior services under pressure. If the cloud can be used by

general customers on pay per use basis, then it is known as public cloud. If the customer builds their own applications and run their own internal infrastructure, then it is known as private cloud. The integration of both public and private clouds is known as hybrid cloud.

Cloud computing is a development based on internet and use of computer technology Chen, S., *et al.*, (2011) Sun, A., *et al.* (2011) The cloud is a figure of speech for the Internet and is an abstraction for the multifaceted infrastructure it conceals. Cloud computing is a method of enabling ubiquitous, convenient, on-demand network admission to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be hurriedly provisioned and unconfined with least management effort or service provider interaction. It defines in three models Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) Tayal, S., *et al.* In cloud computing, each application of users will run on a virtual operation system, the cloud systems distributed resources among these virtual operation systems. Every application is entirely different and is independent and has no link amid each other whatsoever, for example, some necessitate more CPU time to compute complex task, and some others may need more memory to store data, etc. Resources are sacrificed on activities processed on each individual unit of service. To assess direct costs of

Corresponding Author: S. Parthasarathy, Assistant Professor Department of MCA Valliammai Engineering College, Anna University
E-mail: sarathy30@gmail.com

applications, every individual's use of resources such as CPU cost, memory cost, I/O cost, etc. must be measured. When the direct data of each individual resources cost has been measured, more accurate cost and profit analysis Brimson, J.A., *et al.*

Many cloud applications necessitate workflow processing in which tasks are processed derived from their control or data dependencies. As workflow scheduling is an eminent NP-complete issue Yu, J., and Buyya, R., *et al.* (2008) many heuristic and meta-heuristics methods have been proposed for distributed systems like grids Liu, K., *et al.* (2009) In cloud computing, pricing is based on the level of Quality of Service (QoS) offered. Usually, the service providers charge more prices for high QoS. Therefore, users may not always need to complete the workflows previous to they require. As an alternative, the users prefer cheaper services with lower QoS that are enough to meet their requirements Le, K., *et al.* (2009) The Cloud workflow management systems with scheduling algorithms have been proposed by some projects. However, scheduling workflows based on users' QoS necessities (e.g. deadline and budget) has been given very little awareness in these existing Cloud workflow management systems Zhangjun, W., *et al.*, Ke, L., *et al.* (2011)

In this paper we propose a job scheduling algorithm using oppositional-GSO in cloud computing environment. Scheduling is used to decided that which job has to assign how much of resources, computing components, which job to be allocated to which computing component. The scheduling of jobs from different users is an important step to execute the jobs effectively. To schedule each job from different users, we use Oppositional-GSO algorithm as scheduling algorithm to execute the jobs effectively. In first, we generate a population that contains number of members which consist of number of users with their request of jobs to be executed. We apply the Oppositional-GSO algorithm to identify the best schedule to process the jobs of the respective users. To identify the best schedule, we first calculate the fitness for each member in the population and we separate the members based on best fitness, worst fitness, members having fitness greater than the threshold and remaining members. After applying those

operations we would get newly formed members and we iterate the process until we get a stable producer for the repeated iterations. We then schedule the job based on the fitness value.

II. Motivation of our work:

Nowadays cloud computing is an emerging field, requiring more algorithm and techniques for various process of cloud computing. The scheduling process in cloud computing platform needs a good algorithm to schedule the process or jobs requesting from various users of cloud computing environment. The request can be from any platform so scheduling is indispensable one when more number of users needs the particular jobs. Let us assume there has n number of resources $R = \{R_1, R_2, R_3, \dots, R_n\}$ in cloud and N number of users $U = \{U_1, U_2, U_3, \dots, U_N\}$ and the users may request m number of jobs $U_N = \{J_1, J_2, J_3, \dots, J_m\}$. If two or more users request same sequence of jobs that access same resource, then it would hamper the process and it needs a scheduling process to schedule the jobs efficiently. To schedule the users requested jobs efficiently, we intend to develop a heuristic search-based scheduling algorithm in cloud computing environment. Accordingly, multiple criteria will be taken for scheduling various jobs located in various servers. Then, the scheduling will be done in the brokers using any one of the heuristic search based optimization algorithm. Additionally, different jobs with different constraints will be considered and the cloud computing environment is simulated with the help of Cloudsim tool.

III. Proposed Scheduling Algorithm In Cloud Computing Environment:

This section delineates our proposed scheduling algorithm using heuristic search method in cloud computing environment. The Fig.1 shows the process of a user performing a task. There would be N number of users and each user would give m number of jobs. A resource can be used for only one execution of job at a time i.e. we can't use a resource for two jobs simultaneously. The scheduling broker is used to schedule each job from the users to the respective resources to execute the job.

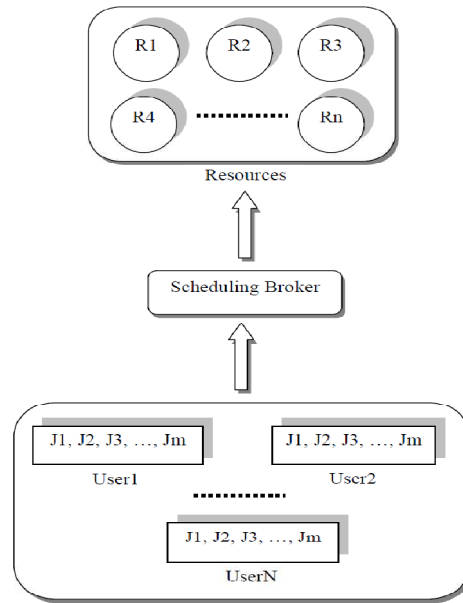


Fig. 1: Process of a user performing a task

The scheduling of jobs from different users is an important step to execute the jobs effectively. To schedule each job from different users, we use Oppositional-GSO algorithm as scheduling algorithm to execute the jobs effectively. The population of the GSO algorithm is known as group and each individual in the population is called member. The GSO Maguluri S.T., and Srikant, R., *et al.* (2012) is based on the concept of animal scanning mechanism for resource searching. The population of the GSO algorithm consists of three kinds of members which

are producers, scroungers and rangers. A group contains only one member as producer and remaining members are scroungers and rangers. Each kind of members would do different operations to generate a new member. We iterate the process of generating new members based on those operations. In our Oppositional-GSO algorithm, we include an opposition operation to generate a new member with the producer operation, scrounger operation and ranger operation. The Fig.2 shows the process of our proposed technique

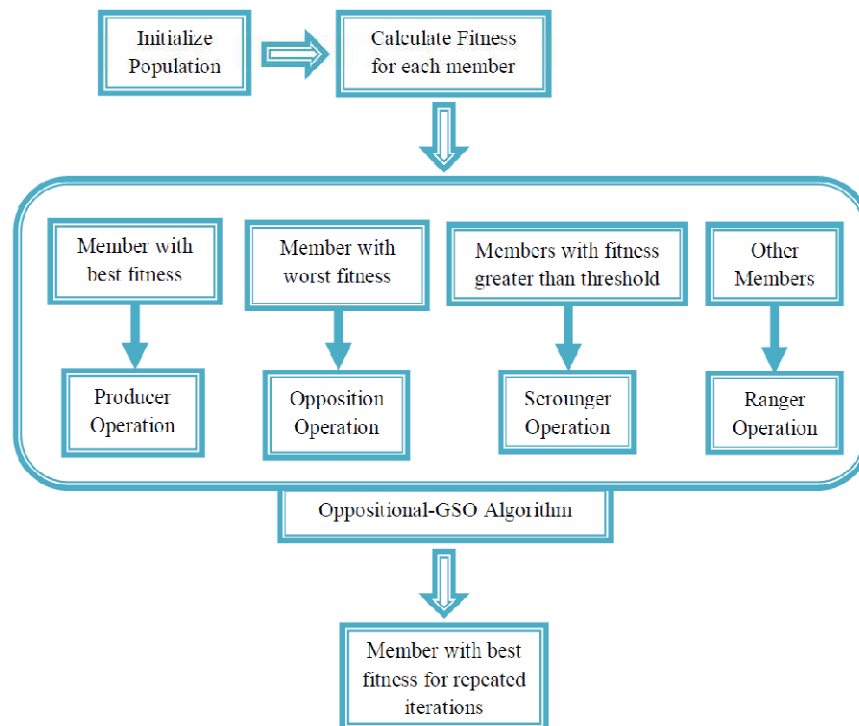


Fig. 2: Process of our proposed technique

The Fig.2 explains as follows: initially we generate a group of members which is called population. The members contain the number of users and the jobs that the users give to execute. We calculate the fitness values for each member and separate the members as four categories based on the fitness values. The first and fourth categories contain only one member and the remaining categories would have more than one member. The member in the first category is based on best fitness value and the member in the fourth category is based on worst fitness value and the members in the second category is based on fitness values greater than the threshold and the third category contains remaining members. We do the producer operation for the first category and we do the scrounger operation for the second category and we do the ranger operation in the third category and we do the opposition operation in the

fourth category. The opposition operation is an additional operation we include in the GSO process that contains the producer operation, scrounger operation and ranger operation.

3.1. Job Allocation To Resource:

To initialize our process, we have to generate a population that contains a group of members by setting the number of users and the required jobs for corresponding users. Each member in the population is generated based on the number of users and their jobs i.e. each member would have the number of users we set with their respective jobs. For example we set the number of users as three and the number of jobs requested by the users as three and the number of members in the population as five, the sample generated initial population is shown in Fig.3

Users	M1	M2	M3	M4	M5
U1	1	2	1	1	3
	3	3	2	3	1
	2	1	3	2	2
U2	2	3	1	1	1
	1	2	3	2	3
	3	1	2	3	2
U3	2	1	3	2	3
	3	2	1	3	1
	1	3	2	1	2

Fig. 3: Sample initial population generated

In the Fig.3, U1, U2 and U3 represents the number of users; and M1, M2, M3, M4 and M5 represents the number of members in the population and 1,2,3 represents the jobs requested by the users. In this Fig.3, consider there has three resources and the job execute by the first resource is represented by numerical value one and the job execute by the second resource is represented by numerical value two and the job execute by the third resource is represented as numerical value three. We give certain

execution time interval for each jobs i.e. for the job execute in the first resource, we set the execution time interval as three; and for second resource, the execution time interval as two; and for the third resource, the execution time interval as one. For example we take the first member from the population to execute our job allocation process. The Fig.4 shows the first member of the population we generated and the Fig.5 shows each user with their respective jobs separately

U1	1	J1
	3	J2
	2	J3
U2	2	J1
	1	J2
	3	J3
U3	2	J1
	3	J2
	1	J3

Fig. 4: First member of the population generated

	J1	J2	J3
U1	1	3	2
U2	2	1	3
U3	2	3	1

Fig. 5: each user with their respective jobs separately

Consider we have three resources to do three different jobs. While a resource is on process, it will not take another job to process in the specified time interval and different jobs from same user will not

get executed in same time interval. The Fig.6 shows the job allocation to the resources for the users in first member of the population.

R1	R2	R3
U1	U2	-
U1	U2	-
U1	U3	-
U2	U3	U1
U2	U1	U3
U2	U1	-
U3		U2
U3		
U3		

Fig. 6: Sample job allocation to resources for users

The Fig.6 is based on the users and their required jobs of the first member of the population we generated and it explains as follows: the first job of the first user requests service from the first resource, so the first three time intervals of the first resource is allocated to the first user because the first resource requires three time intervals to execute the job. While the first user uses the first resource in the first three time interval, the scheduling broker will not allocate any job of the first user to any resource in the first three time interval and also the scheduling broker will not allocate any user's job to the first resource while the first resource is scheduled for some other user. After allocating first job of the first user to the respective resource, the scheduling broker will schedule the first job of the second user. The first job of the second user requests service from the second resource and the scheduling broker allocated first two time interval of the second resource to the second user because the execution time interval of the service from second user is two. Thereafter the scheduling broker checks the first job of the third user. The first job of the third user requests service from the second resource and the scheduling broker allocate the third and fourth time interval of the second resource to the third user. The scheduling broker then checks the second job of the first user. The second job of the first user requests service from the third resource. The scheduling broker allocates the fourth time interval of the third resource because the first user uses the first three time intervals to execute the jobs in the first resource and the

execution time interval of the third resource is one. Similarly, the scheduling broker allocates the jobs to the respective resources for every user.

3.2. Fitness calculation:

This section explains the fitness calculation of the members in the population generated for our technique. The fitness is calculated based on the time intervals taken by each resource. After allocating all the jobs requested by each user to the respective resources, we calculate the total time interval taken by each resource. We also include the idle time interval which is before the last execution of a job in a particular resource while calculating the total time interval. It is denoted by an equation below:

$$TR_n = \text{total time interval used by } R_n$$

In the above equation, TR_n is the total time interval taken by n^{th} resource. Where n varies from one to three because used three resource for our example. In our example shown in Fig.6, the total time interval taken by the first resource is nine and the total time interval taken by the second resource is six and the total time interval taken by the third resource is seven. The fitness value is then based on the maximum time interval. It is shown by an equation below:

$$fit(M_i) = \max(TR_n)$$

Where, $fit(M_i)$ denotes the fitness value of i^{th} member of the population we generated and $\max(TR_n)$ denotes the maximum time interval

obtained among all the resources for the i^{th} member. Similarly, we calculate the fitness for all the members we generated and the best fitness is decided by minimum value i.e. a member that has $fit(M)$ value as small compared to other members is chosen as best fitness. The worst fitness is based on the maximum value i.e. a member that has $fit(M)$ value as high compared to the other members is considered as worst fitness. Thereafter, we classify the members for different operations based on the fitness. We take the member with best fitness i.e. $fit(M)$ value as small for producer operation and we take the member with worst fitness i.e. $fit(M)$ value as large for opposition operation and we take the members that has the fitness greater than a threshold we set for scrounger operation and we take the remaining members for ranger operation.

3.3. Producer operation:

The member that has best fitness value is chosen as producer and the operation performed by the producer is explained as follows: animals scan the environment to search for food. Scanning is an essential part of search orientation. It is a set of mechanism by which animals move sensory receptors and sometimes their bodies or appendages to capture information from the environment. The major scanning mechanism used by various animal species is employed by producer. In s -dimensional search space, the i^{th} member at the z^{th} searching bout (iteration) has current position as $y_i^z \in R^s$, head angle as $\lambda_i^z = (\lambda_{i1}^z, \dots, \lambda_{i(s-1)}^z) \in R^{s-1}$ and head direction as $F_i^z(\lambda_i^z) = (f_{i1}^z, \dots, f_{is}^z) \in R^{s-1}$ which can be evaluated from λ_i^z through polar to Cartesian coordinates transformation:

$$f_{i1}^z = \prod_{p=1}^{s-1} \cos(\lambda_{ip}^z)$$

$$f_{ij}^z = \sin(\lambda_{i(j-1)}^z) * f_{i1}^z$$

$$f_{is}^z = \sin(\lambda_{i(s-1)}^z)$$

The scanning field of the vision is generalized to a s -dimensional space which is characterized by maximum pursuit angle $\omega_{\max} \in R^{s-1}$ and maximum pursuit distance $d_{\max} \in R^1$. The attitude of producer y_p at z^{th} iteration is as follows: the producer first scans at zero degree and then scans at right hand side hypercube and then scans at left hand side hypercube. It is shown by equations below:

$$y_{zero} = y_p^z + \eta_1 d_{\max} F_p^z(\lambda^z)$$

$$y_{right} = y_p^z + \eta_1 d_{\max} F_p^z\left(\lambda^z + r_2 \frac{\omega_{\max}}{2}\right)$$

$$y_{left} = y_p^z + \eta_1 d_{\max} F_p^z\left(\lambda^z - r_2 \frac{\omega_{\max}}{2}\right)$$

In the above equations y_{zero} represents zero degree scan, y_{right} represents right hand side hypercube scan, y_{left} represents left hand side hypercube scan, $r_1 \in R^1$ is a normally distributed random number with zero mean and standard deviation as one and $r_2 \in R^{s-1}$ is a random sequence in the range (0,1). Thereafter, the producer will find the best point with best resource i.e. with best fitness. If the best point has a better resource than its current position, then it will move to this point or else it will stay in its current position and turn its head to a new angle:

$$\lambda^{z+1} = \lambda^z + r_2 \gamma_{\max}$$

The γ_{\max} in the above equation denotes the maximum turning angle. If the producer cannot discover a better area after a iterations, it will twist its head back to zero degree:

$$\lambda^{z+a} = \lambda^z$$

3.4. Scrounger operation:

In each search bout, a number of group members are selected as scroungers. The scrounger will explore for the moment to join the resource found by the producer. The process involved in scrounger operation is area copying, following and snatching. Area copying is the process of searching immediate area around the producer and following is the process of pursuing another animal without any searching behavior and snatching is the process of taking resource directly from the producer. In GSO algorithm the common process of area copying of the scrounger behavior is adopted. The area copying attitude of the i^{th} scrounger at z^{th} iteration is shown by an equation below:

$$y_i^{z+1} = y_i^z + r_3 (y_p^z - y_i^z)$$

The scrounger can be modeled as a random walk towards the producer. The $r_3 \in R^s$ in the above equation denotes the uniform random sequence in the range (0,1).

3.5. Ranger operation:

The group members which are less efficient foragers than the dominant will be dispersed from the group and it is called rangers. The rangers operation includes random walk and systematic search

strategies to discover resources efficiently. In GSO algorithm, the random walk which is the most efficient searching behavior for randomly distributed resources is employed by rangers. If the i^{th} member from the group is chosen as ranger at z^{th} iteration, it generate a random head angle λ_i which is shown by an equation below:

$$\lambda_i^{z+1} = \lambda_i^z + r_2 \gamma_{\max}$$

In the above equation γ_{\max} is a maximum turning angle and it chooses a random distance. The random distance is shown by the equation below:

$$d_i = a \cdot r_1 d_{\max}$$

After it chooses a random distance, it will move to a new point which is given below:

$$y_i^{z+1} = y_i^z + d_i F_i^z (\lambda^{z+1})$$

3.6. Opposition operation:

The opposition operation of our proposed technique is as follows: we convert the sequence of

jobs given by the users to opposite order. For example, consider there has five resources to execute five different jobs and a user gives the sequence of jobs to execute as 1, 3, 2, 5, 4, then based on opposition operation it would be converted as 5, 3, 4, 1, 2. The formula to convert the sequence is given below:

$$O = \text{total no. of resource} - (\text{current job} - 1)$$

In the above equation, O is opposition operation and the total number of resources is five based on the aforementioned example and if we consider the current job as 2, the calculation is as follows: $5 - (2 - 1)$ which is equal to 4. Similarly, we convert for all the users in the member which we obtained as worst fitness value. After performing all the operations, we iterate the process i.e. we will repeat the process until we get a stable member as producer for repeated iterations. The scheduling broker then schedule the jobs for each user based on the producer member we eventually obtained. The Fig.7 shows the algorithm of our Oppositional-GSO algorithm.

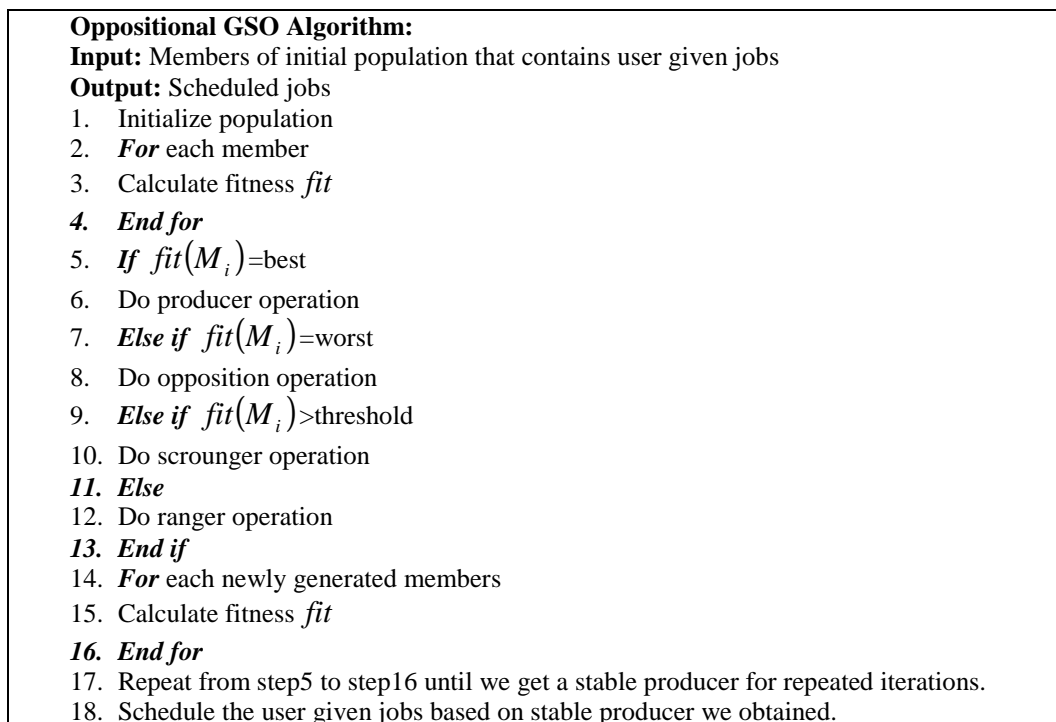


Fig. 7: Oppositional-GSO Algorithm

RESULT AND DISCUSSION

This section delineates the results we obtained for our proposed technique with comparison to the existing GSO algorithm i.e. instead of our Oppositional-GSO algorithm we applied existing GSO algorithm [11] and genetic algorithm (GA). We used three setups to compare the performance: the first one is all the users' requests same sequence of jobs and the second setup is half users requests same

sequence of jobs and another half users request another same sequence of jobs and the third setup is all the users requests different sequence of jobs.

4.1. Experimental Setup:

Our experiment is done on a system that has the system configuration as i5 processor with 4GB RAM and our technique is implemented in java (jdk 1.6). We initially generated a population that has members which contains the number of users and their

requested jobs i.e. each member in a population would have same number of users with their requested jobs in different schedules. We used three different setups to compare the performance of our system with the existing one. The users in each member of the population requested same sequence of jobs for the first setup. For example, there have five resources and all the users requests same sequence of five jobs from the resources, then all the members generated in the population would be in same sequence initially. The usage time we set for the resources are three for first resource, two for second resource, one for third resource, five for

fourth resource and four for fifth resource. Although, the users give same jobs to execute in same sequence, the scheduling algorithm schedules the job for each user to the corresponding resources. In second setup, half of the users request same sequence of jobs and half of the users request another same sequence of jobs i.e. if we use ten users, five users would request same sequence of jobs and the other five users request another same sequence. In third setup, all the users request jobs in different sequence. The Fig.8 shows a sample initial population generated based on third setup.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
U0	[5, 3, 1, 2, 4]	[5, 4, 3, 1, 2]	[5, 2, 4, 1, 3]	[2, 5, 3, 1, 4]	[5, 3, 4, 1, 2]	[1, 5, 3, 2, 4]	[1, 4, 5, 2, 3]	[5, 2, 4, 1, 3]	[4, 5, 3, 1, 2]	[3, 5, 4, 1, 2]
U1	[1, 5, 4, 3, 2]	[4, 1, 2, 3, 5]	[2, 3, 4, 1, 5]	[4, 3, 1, 2, 5]	[2, 3, 1, 4, 5]	[2, 1, 3, 4, 5]	[1, 4, 2, 3, 5]	[4, 3, 1, 2, 5]	[4, 3, 5, 1, 2]	[4, 2, 5, 3, 1]
U2	[5, 2, 4, 1, 3]	[1, 3, 4, 5, 2]	[3, 4, 1, 5, 2]	[4, 1, 2, 5, 3]	[5, 3, 2, 1, 4]	[2, 1, 4, 3, 5]	[4, 2, 5, 3, 1]	[4, 5, 1, 2, 3]	[3, 2, 4, 5, 1]	[4, 1, 5, 3, 2]
U3	[5, 4, 2, 1, 3]	[2, 1, 4, 3, 5]	[1, 4, 3, 5, 2]	[5, 4, 3, 2, 1]	[2, 5, 3, 4, 1]	[1, 2, 5, 3, 4]	[2, 5, 1, 4, 3]	[1, 4, 3, 5, 2, 3]	[1, 5, 3, 1, 5, 3]	[5, 4, 1, 3, 2]
U4	[4, 1, 5, 3, 2]	[5, 4, 2, 3, 1]	[2, 3, 5, 1, 4]	[5, 3, 1, 4, 2]	[1, 2, 5, 4, 3]	[4, 5, 3, 1, 2]	[1, 5, 3, 4, 2]	[3, 5, 1, 4, 2]	[5, 3, 1, 4, 2]	[1, 4, 5, 3, 2]
U5	[3, 2, 5, 4, 1]	[4, 5, 3, 2, 1]	[1, 3, 2, 4, 5]	[1, 2, 3, 4, 5]	[5, 3, 2, 4, 1]	[4, 5, 3, 2, 1]	[3, 5, 1, 2, 4]	[5, 4, 3, 1, 2]	[4, 5, 1, 2, 3]	[3, 2, 4, 5, 1]
U6	[5, 3, 1, 2, 4]	[4, 2, 5, 3, 1]	[1, 3, 4, 5, 2]	[1, 3, 4, 5, 2]	[4, 1, 5, 2, 3]	[4, 1, 5, 3, 2]	[1, 5, 3, 2, 4]	[5, 2, 1, 3, 4]	[4, 5, 1, 3, 2]	[5, 4, 2, 1, 3]
U7	[1, 2, 4, 5, 3]	[1, 2, 5, 4, 3]	[2, 1, 4, 5, 3]	[4, 3, 5, 2, 1]	[2, 4, 1, 3, 5]	[5, 2, 1, 3, 4]	[1, 2, 3, 5, 4]	[1, 3, 2, 5, 4]	[4, 3, 1, 2, 5]	[2, 1, 5, 3, 4]
U8	[2, 3, 5, 4, 1]	[2, 5, 1, 3, 4]	[1, 3, 2, 5, 4]	[2, 5, 4, 3, 1]	[4, 5, 1, 2, 3]	[4, 3, 5, 1, 2]	[1, 4, 2, 3, 5]	[3, 1, 2, 4, 5]	[5, 3, 2, 1, 4]	[5, 3, 2, 1, 4]
U9	[4, 3, 5, 1, 2]	[1, 5, 4, 2, 3]	[4, 1, 2, 5, 3]	[1, 2, 4, 5, 3]	[5, 1, 4, 2, 3]	[2, 3, 1, 5, 4]	[4, 1, 2, 5, 3]	[3, 2, 1, 4, 5]	[3, 1, 5, 2, 4]	[2, 3, 1, 4, 5]

Fig. 8: Sample population generated based on third setup

The Fig.8 shows the sample population generated with ten members contains users with different sequence of jobs i.e. based on third setup

and the best schedule obtained after applying our technique is shown in Fig.9.

Users	Best Schedule
U0	[5, 3, 1, 2, 4]
U1	[1, 4, 3, 2, 5]
U2	[3, 5, 1, 2, 4]
U3	[5, 3, 2, 1, 4]
U4	[3, 5, 1, 2, 4]
U5	[4, 3, 2, 1, 5]
U6	[2, 1, 3, 4, 5]
U7	[1, 5, 2, 3, 4]
U8	[2, 1, 5, 3, 4]
U9	[2, 1, 5, 3, 4]

Fig. 9: Best scheduled jobs

4.2.Performance Comparison:

We compared the performance of our technique with the existing GSO algorithm and GA algorithm. The performance is compared based on two cases. In first case, the performance is evaluated for time taken to execute the jobs using three different setups; and

in the second case, the performance is evaluated for the total time taken to execute using the proposed and the existing algorithms.

4.3.Performance Based On Execution Of Jobs:

In this section, the performance is compared based on the time taken to execute the scheduled jobs using three different setups as follows

4.4. Performance Based On First Setup:

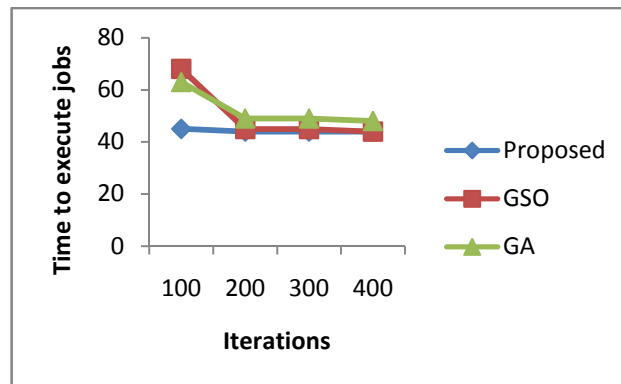


Fig. 10: Performance comparison for first setup

The Fig.10 shows the performance comparison of our technique with the existing algorithms for the first setup. Here when we set the iteration as hundred, the time taken to execute the scheduled jobs for our technique is forty five seconds and for the GSO technique it is sixty eight seconds and for GA technique it is sixty three seconds. This shows that our technique scheduled the jobs better which obtained less execution time than the existing technique. When we set the iteration as two hundred, our technique executed the jobs in forty four seconds and the GSO technique executed the jobs in forty five seconds and the GA technique executed the jobs in forty nine seconds. When we set the iteration as three hundred, the execution time of scheduled jobs

This section shows the performance comparison of our technique and the existing techniques based on the first setup which is all the users request same sequence of jobs. We evaluated the execution time of scheduled jobs for different number of iterations.

is forty four seconds for our proposed technique and it is forty five seconds for the GSO technique and it is forty nine seconds for the GA technique. When the iteration is four hundred, the proposed and GSO techniques executed the scheduled jobs in forty four seconds and the GA algorithm executed the jobs in forty eight seconds.

4.5. Performance Based On Second Setup:

This section shows the performance of our technique compared to the existing techniques based on the second setup which is half of the users request same sequence of jobs and another half request another same sequence of jobs. We evaluated the performance for different iterations

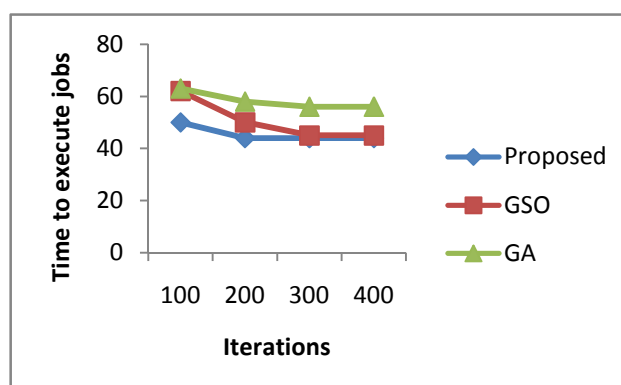


Fig. 11: Performance comparison for second setup

The Fig.11 shows the performance of our technique compared to the existing GSO and GA techniques based on second setup. Here when the iteration is hundred, the execution time for the scheduled jobs using our technique is fifty seconds and it is sixty two seconds using GSO technique and it is sixty three seconds using GA technique. When

the iteration is two hundred, the execution time of scheduled jobs using our proposed technique is forty four seconds and the execution time using the GSO technique is fifty seconds and it is fifty eight seconds using GA technique. When we set the iteration as three hundred, our technique executed the jobs in forty four seconds and the GSO technique executed

the jobs in forty five seconds and the GA technique executed the jobs in fifty six seconds. When the iteration is four hundred, the execution time is forty four seconds using our proposed technique and it is forty five seconds using the GSO technique and it is fifty six seconds using the GA technique.

4.6. Performance Based On Third Setup:

This section shows the performance of our technique compared to the existing GSO and GA techniques based on the third setup which is all the users request jobs in different sequence. The performance of both the techniques is evaluated for different iterations.

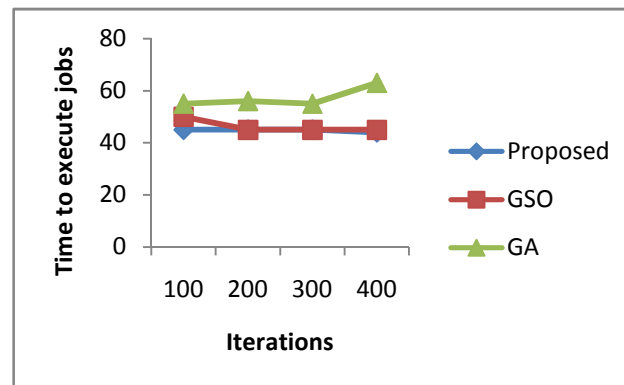


Fig. 12: Performance comparison for third setup

The Fig.12 shows the comparison of our technique and the existing GSO and GA techniques based on third setup for different iterations. Here when we set the iteration as hundred, the execution time for the scheduled jobs using our technique is forty five seconds and the execution time for the scheduled jobs using GSO technique is fifty seconds and it is fifty five seconds using GA technique. The execution time of the scheduled jobs using the proposed and the GSO algorithms is forty five seconds and it is fifty six seconds using GA algorithm when the iteration is two hundred. The time taken to execute the scheduled jobs using the

proposed and the GSO technique is forty five seconds and it is fifty five seconds using GA algorithm. When we set the iteration as four hundred, the execution time is forty four seconds using our technique and it is forty five seconds using GSO technique and it is sixty three seconds using GA technique.

4.7. Performance based on total time for execution:

In this section, the performance based on the total execution time of each algorithm is compared by varying the resources used. The Fig.13 shows the total execution time comparison of algorithms.

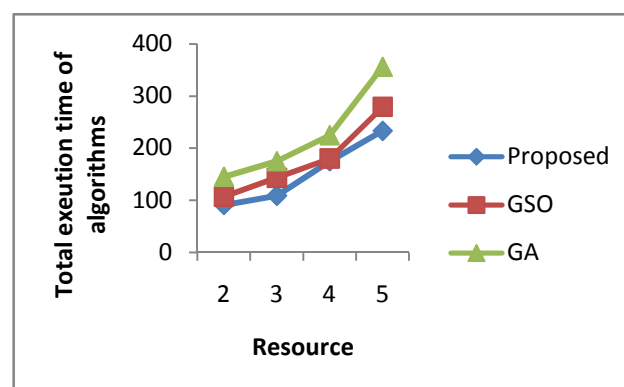


Fig. 13: Total time taken to execute the algorithms

Here when we used two resources, the total execution time of algorithm based on the proposed technique is 91 seconds and it is 107 seconds based on GSO algorithm and it is 145 seconds based on GA algorithm. When the resource used is three, the total

execution time based on the proposed technique is 109 seconds and it is 143 seconds based on GSO algorithm and it is 175 seconds based on GA algorithm. When the resource used is four, the total execution time based on the proposed technique is

175 seconds and it is 180 seconds based on GSO algorithm and it is 225 seconds based on GA algorithm. When the resource used is five, the total execution time based on the proposed technique is 233 seconds and it is 279 seconds based on GSO algorithm and it is 356 seconds based on GA algorithm.

Conclusion:

In this paper we have proposed a scheduling algorithm which is Oppositional-GSO algorithm using heuristic search methods in cloud computing environment. Here, initially we generated a population that contains a group of members with different jobs and their respective jobs. We calculated the fitness for each member and based on the fitness we applied different operations such as producer operation, scrounger operation, ranger operation and oppositional operation to generate new schedule. We iterated the process and we chose a best member to schedule the user requested jobs and to execute it. We compared our proposed algorithm with the existing GSO algorithm and GA algorithm in terms of time taken to execute the scheduled jobs and total execution time of the algorithms. We set three different setups to evaluate the performance of our technique in terms of time taken to execute the scheduled jobs. The performance comparison showed that in most cases our algorithm obtained less execution time than the existing algorithms in terms of time taken to execute the scheduled jobs and total time taken to process the algorithms. This implies that our technique scheduled the jobs of the users better than the existing algorithms and our technique took less time to finish the whole process compared to the existing techniques.

REFERENCES

- Armstrong, P., 2010. "Cloud Scheduler: a resource manager for distributed compute clouds", Distributed, Parallel, and Cluster Computing.
- Bitam, S., 2012. "Bees Life Algorithm for Job Scheduling in Cloud Computing", The Second International Conference on Communications and Information Technology.
- Brimson, J.A., 1991. Activity Accounting: An Activity-based Costing Approach, John Wiley & Sons.
- Buyyaa, R., C.S. Yeo, 2009. Srikumar Venugopal, James Broberg, Ivona Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Published in Journal Future Generation Computer Systems, 25: 6.
- Chen, S., T. He, H.Y.S. Wong, K.L.L. Tong, 2011. "Secondary Job Scheduling in the Cloud with Deadlines", IPDPS Workshops.
- He, S., Q.H. Wu and J.R. Saunders, 2009. "Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior", IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 13(5).
- Ke, L., J. Hai, C. Jinjun, L. Xiao, Y. Dong and Y. Yun, 2010. "A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on Cloud Computing Platform", International Journal of High Performance Computing Applications, pp: 1-16.
- Le, K., J. Chen, H. Jin and Y. Yang, 2009. "A Min-Min Average Algorithm for Scheduling Transaction Incentive Grid Workflows," 7th Australasian symposium on Grid Computing and e-Research (AusGrid), Australia, pp: 41-48.
- Leavitt, N., 2009. "Is Cloud Computing Really Ready for Prime Time?," Computer, 42: 15-20.
- Liu, K., 2009. "Scheduling Algorithms for Instance Intensive Cloud Workflows", Ph.D Thesis, Swinburne University of Technology, Australia.
- Maguluri, S.T., and R. Srikant and L. Ying, 2012. "Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters", INFOCOM.
- Sun, A., T. Ji, Q. Yue, F. Xiong, 2011. "IaaS Public Cloud Computing Platform Scheduling Model and Optimization Analysis", IJCNS, 4: 12.
- Tayal, S., 2011. "Tasks scheduling optimization for the cloud computing system", International Journal of Advanced Engineering Sciences And Technologies, 5(2): 111-115.
- Weinhardt, C., A. Anandasivam, B. Blau and J. Stosser, 2009. 'Business Models in the Service World', IT Professional, 11: 28-33.
- Yu, J., and R. Buyya, 2008. "Workflow Scheduling Algorithms for Grid Computing," In: Xhafa F, Abraham A (eds) Metaheuristics for scheduling in distributed computing environments. ISBN: 978-3-540-69260-7. Springer, Berlin.
- Zhangjun, W., L. Xiao, N. Zhiwei, Y. Dong and Y. Yun, 2011. "A Market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems", Journal of Supercomputing.