Cloud Task Scheduling Using Nature Inspired Meta-Heuristic Algorithm

Syed Hasan Adil[1], Kamran Raza[2], Usman Ahmed[3], Syed Saad Azhar Ali[4], Manzoor Hashmani[5]
Department of Computer Science[1,2,5], College of Computer Science[3], Department of Electrical Engineering[4]
Iqra University[1,2,5], King Khalid University[3], Universiti Teknologi Petronas[4]
Karachi, Pakistan[1,2,5], Abha – Aseer, Saudi Arabia[3], Bandar Seri Iskandar, Malaysia[4]
hasan.adil@iqra.edu.pk[1], kraza@iqra.edu.pk[2], lusman@kku.edu.sa[3], saad.azhar@petronas.com.my[4], mhashmani@iqra.edu.pk[5]

Abstract— **In this paper we investigate the application of Meta-Heuristic for cloud task scheduling on Hadoop. Hadoop is an open source implementation of MapReduce framework which extensively used for processing computational intensive jobs on huge amount of data over multi-node cluster. In order to achieve an efficient execution schedule, the scheduling algorithm requires to determining the order and the node on which tasks will be executed. A scheduling algorithm uses execution time, order of task arrival and location of data (i.e., assign task to the node which contains the required data) to determine the best execution schedule. We use Particle Swarm Optimization (PSO) to determine the tasks execution schedule and compare with tasks schedules obtained from other techniques like Genetic Algorithm (GA), Brute Force (BF) algorithm, First In First Out (FIFO) algorithm and Delay Scheduling Policy (DSP) algorithm. The results of this study prove the significance of PSO algorithm for cloud task scheduling over other algorithms.**

Keywords-Task Scheduling; Cloud Computing; Meta-Heuristic; Particle Swarm Optimization; Brute Force Approach; Hadoop; MapReduce.

I. INTRODUCTION

Cloud Task Scheduling Problem (CTSP) [1-4] is a highly important area of research in the modern cloud computing based IT infrastructure. Due to the emergence of large number of public cloud service providers (i.e., public cloud is the one in which an independent service provider provides computing services on their own infrastructure and thousands of organizations avail this to host their services) and private cloud infrastructure (i.e., private cloud is the one which is deployed by a single organization for its own use usually in their own data center). Public cloud service providers offer Service Level Agreement (SLA) [5] to those organizations which want to utilize the cloud computing services. These SLA include service availability, disaster recovery, throughput, and response time etc. So, in order to meet these SLAs, service provider needs to ensure that their existing infrastructure supports their commitments. One of the important factors which need to be fulfilled is the execution time. In traditional computing environment, execution time of a task is fixed once the CPU and all the required resources are allocated to it. However, in cloud computing environment the execution time of a task is dependent on the node on which the computation is performed. The variation in execution time is due to the locality of the data to computing node.

Virtualization [6, 7] is the primary technology used by cloud infrastructure. Therefore, the computing node offered by cloud infrastructure is a virtual machine instead of physical one. These virtual machines are deployed over many physicals machines using some hypervisor. The cloud infrastructure consists of multiple server racks which may be distributed over multiple geographical locations. Each rack contains multiple physical servers, and each physical server contains multiple virtual machines. Servers on the same rack are directly connected to the same switch but those on different racks are either connected through any switch or may be through routers depending on the actual location of the servers. The connectivity used to establish communication includes copper, Fiber or SAS. The data required by a computing node (i.e., virtual machine) will be either available through Direct Attached Storage (DAS) connected to the physical server, Network Attached Storage (NAS) accessible through some Fiber/Ethernet switch, or Storage Area Network (SAN) accessible through some fiber/Ethernet switch. The distance between the computing and data nodes plays a vital role in the actual execution time required to execute any tasks (i.e., execution reduces with the decrease in the distance between computing and data node and also depends on the underlying communication and storage technology).

Hadoop [8-10] is the most popular framework used to utilize the highly distributed and massive computing power provided by cloud infrastructure. Hadoop Distributed File System (HDFS) is a distributed file system used by Hadoop framework. HDFS distributes a single file into multiple data nodes; it first split the file into 64 MB size chunk and distributes it over different data nodes, multiple copies of same data is placed on multiple (i.e., depends on the configuration) data nodes to handle data loss due to any failure.

Hadoop MapReduce is a framework for writing distributed applications that reliably process very large amount of data stored on HDFS. The concept of map-reduce framework is to move the computation to the node (i.e., closer to data node)

978-1-4799-7812-0/15/$31.00 ©2015 IEEE

which contains the data required to be processed by a map/ reduce task. In traditional distributed computing models, the computing nodes fetch data from some data storage over the network which suffers bottleneck like network bandwidth. The task scheduler used by the MapReduce task needs to ensure that every task should schedule to the node which contains the required data. When the data required by the map/reduce task is not available on the local data than it needs to request from remote node. As the distance between the data node and computing node increases (i.e., from same rack to different rack to different geographical location) than the computational time of the task also increases accordingly.

Assuming the computation power of the each virtual machine is same and there is no waiting time for the task than the computation time of the task will be dependent on the locality of the data. Hence, the performance gradually decreases with respect to distance of the locality. The technology used to host the data and the connectivity also plays a vital role in task completion and that is the dependency MapReduce tried to eliminate.

The traditional scheduling algorithms [11-13] like First Come First Serve (FCFS), Shortest Job First (SJF) and Round Robin (RR) algorithm do not well suits for MapReduce task scheduling. Initially, Hadoop also used FCFS algorithm for tasks scheduling but due to this, interactive tasks were impossible to execute on Hadoop based cloud computing environment. Therefore, in order to improve the scheduling process to facilitate the execution of interactive applications, Hadoop introduced Fair Scheduling (FS) algorithm. The FS algorithm has two core steps: (I) it divides resources using max-min fair scheduling, and (II) data locality (i.e., moving the computation near to data on which it has to execute the required task).

In this research, we investigate the application of meta-heuristics for scheduling [14-19] of tasks on Hadoop based MapReduce computing framework. The objective of meta-heuristic is to minimize the task completion time to make applications interactive (i.e., it includes waiting time and execution time. The execution time is itself dependent on the data locality).

There are *'n'* MapReduce jobs available for execution at a single instance of time. Each MapReduce job contains 'm' tasks which need to be executed on *'k'* computing nodes. Each of *'k'* computing nodes has HDFS and therefore holds a chunk of dataset. Each chunk of dataset is replicated on multiple nodes (i.e., each node contains multiple chunks of dataset and same chunk is available on multiple nodes to provide fault tolerance). This makes MapReduce scheduling problem as a type of combinatorial problem. Each of the *'m'* tasks of *'n'* jobs can be executed on any of the *'k'* computing nodes. However, our objective is to schedule as many jobs as possible on the node in which the dataset required by the task is available. Therefore, our objective function is based on the minimization of task completion time.

The rest of the paper is organized as follows: Section II outlines the optimization problem for task scheduling. Section III includes the discussion about scheduling problem used to find the solution. Section IV describes the BF algorithm for scheduling. Section V describes the PSO algorithm for task scheduling. Section VI defines the basic architecture of the scheduling system. In Section VII, we compare: (I) the result of PSO with BF, GA, DSP and FIFO on benchmark problem and (II) the result of PSO with BF for randomly generated problems. We present our discussion and conclusion in Section VIII.

II. Optimization Problem For Task Scheduling

Our optimization objective is to minimize the cumulative completion time of all available tasks also known as Makespan.

First of all we need describe the notations used in the model:

N	:	Available machines, *n*={1,2, …,N}
J	:	Available jobs, j={1,2, …,J}
T	:	Available tasks in each job, t={1,2, …,T}
MR(n)	:	Contains the time after which a machine *'n'* becomes free and starts executing assigned tasks.
ET(j,t)	:	Contains the estimated execution time for task *'t'* of the job *'j'*
A(j,t)	:	It contains the machine on which the task *'t'* of job *'j'* is executed (i.e., A(j,t) = n)

$$\mathrm{MET(n)} = \mathrm{MR(n)} + \sum_{\substack{j \in J \\ t \in T \\ A(j,t)=n}} ET(j,t) \qquad (1)$$

The total completion time for execution of all the tasks assigned to a machine *'n'* is obtained using equation (1),

Objective function,

Our objective function which minimizes the Cumulative Tasks Completion Time (CTCT) among all alternative schedules is given in equation (2),

$$\textbf{Minimize}\ \ \mathrm{CTCT} = \max\big(\mathrm{MET(n)}\big),\ 1 \le n \le N \qquad (2)$$

III. Problems Used For Scheduling

We used benchmark problem [1] as shows in Table I, II and III to compare our PSO [17] scheduling algorithm with GA, FIFO, BF and DSPA. We also generate some random problems to compare the schedule generated using PSO and BF, the detailed mechanism we used to generate our random problems are shown in Table IV.

The benchmark problem has three jobs and each job consists of two independent tasks (i.e., order of execution does not matter for tasks within a job). Table I shows the estimated execution time taken by each task. The sample setup has three

nodes Hadoop cluster on which all tasks will be scheduled. The existing load on each of the three nodes is shown in Table II. Table III shows the data node local for each task (i.e., node which contains the data required to process by a task). If a task is run on a node which does not contains the required data than the task execution time will be increased by certain factor (i.e., here a non-local task will take **1.2** times longer than a local task). For example, the data required by Task 1 is available on node 2 while data required by Task 2 is available on node 3. If these tasks will be scheduled on some other node than we have to multiply its corresponding local execution time with **1.2**.

TABLE I. LOCAL TASK EXECUTION TIME [1]

	Job 1		**Job 2**		**Job 3**	
	Task1	***Task2***	***Task3***	***Task4***	***Task5***	***Task6***
Time Units	8	7	2	4	1	7

TABLE II. CURRENT LOAD OF SLOTS [1]

	Job 1	**Job 2**	**Job 3**
MR(n)	0	1	1

TABLE III. DATA LOCATIONS [1]

Data Location	**Task#**
Node 1	Task 3, Task4
Node 2	Task 1, Task5
Node 3	Task 2, Task6

We have also generated random data for further evaluation of the proposed PSO algorithm for scheduling. The range of values used for each parameter to generate the random problems are shown in Table IV.

TABLE IV. PARAMETER USED TO GENERATE RANDOM [1]

Parameter	**Value Range**
Tasks Cost	Between 1 and 10
Current Loads	Between 0 and 2
Total Jobs	Between 3 and 4 (Two tasks in per job)
Total Machines	Between 3 and 4
Data Locations	Location of data required by tasks are equally and randomly distributed over the available machines

IV. BRUTE FORCE ALGORITHM FOR TASK SCHEDULING

Brute Force algorithm is an exhaustive search technique which consists of two phases, (I) in the first phase it generates all possible candidate solutions and (II) in the second phase it evaluate all candidate solutions to find the one which gives the optimum solution of a particular problem. BF algorithm guarantees to find optimum solution of a combinatorial problem. However, it is either difficult or impossible to solve large problem using BF algorithm due to its very long execution time. Therefore, we can only solve small problems with BF algorithm and compare with other algorithms (i.e., meta-heuristic algorithms) to assess the quality of the solution. If the results of the meta-heuristic algorithm are comparable with the results of BF algorithm than meta-heuristic algorithm is a good choice for solving large problems.

To compare the quality of results, we have first found the solution of task scheduling problems from BF algorithm and then compared it with the solution found by the PSO algorithm. For example, a scheduling problem having three jobs and each job contains two tasks which we need to schedule on three nodes cluster. In the first step we have to generate all possible schedules by using three nested loops with each loop iterates for three times, it means there are 3 ^ 6 = 729 total iterations of BF algorithm. In the second step we will calculate the completion time for each of the seven hundred twenty nine possible solutions and keep the one with the minimum completion time.

V. PSO ALGORITHM FOR TASK SCHEDULING

The important aspect of any meta-heuristic algorithm to converge to an optimal solution is the use of seed solutions. Seed solutions are initial feasible solutions for the problem which are used by an optimization algorithm to further the process of finding an optimal solution. These solutions are one of the important factors for rapid convergence of any optimization algorithm. Many techniques have been employed by researchers to generate seed solutions, depending on the nature of the problem. These include selecting a random solution, analytical solution based on some restricted model of problem, and/or a heuristic based solution. Each of these techniques has its own positive and negative aspects. However, in absence of a proper heuristic, random seed solution is frequently used to generate a seed solution.

Since PSO is a population based algorithm, seed solutions need to be generated for each particle. In our implementation random solution technique is used to generate the seed solutions. Each particle is equally probable and a complete feasible (i.e., not necessarily optimum) solution based on its position in the $J * T$ dimensional space (i.e., No. of jobs * No. of Tasks in each job). For example, if we have 3 jobs and each job consists of 2 tasks than we have a six dimension solution space where each dimension has a value range between *1* and $\underline{N}$ (i.e., Index of machine on which a job task will be run).

Seed solution generation

- For each particle in the swarm, Initialize each of the $J * T$ dimensional solution space with a value between *1* and $\underline{N}$ (i.e., the node on which the task will execute).
- Each randomly initialized particle represents a possible feasible schedule for execution of all available tasks.

- For each initialized member of the population, calculate the *CTCT*.
- Assign best position (solution) p_j of each particle in the swarm to its own value (i.e., initially each particle current position and best solution should be same which will later update by the PSO algorithm).
- Assign best position p_g of the whole swarm using best solution of all the particles in the swarm.

Finding optimal cloud schedule using PSO algorithm

Repeat the following steps until the maximum number of iteration is reached or until the convergence criterion is met.

- The objective function given in equation (3) is minimized for any number of machines *N* for *J* * *T* tasks.
- Apply PSO algorithm using equations of velocity and position [17] to generate the new positions of *P* particles (i.e., P is the population size) in $J \times T$ dimensional search space. Here, the position of each particle in each of the $J \times T$ dimensional search space represents index of machine on which the task will execute and the whole particle represents one complete possible solution to cloud task scheduling problem.
- As PSO works in the real domain (i.e., position of particle in each of $J \times T$ dimensional space has any real value) but the allowed value for each dimension is between *1* and *N* integer values only. Therefore, we need to convert this complete range of real number between *1* and *N*. To deal with this issue, we used the round function to convert real values into integer values between *1* and *N*.
- For each particle's updated solution (i.e., particle position) generated by the previous step, we apply the same heuristic (i.e., as applied to create seed solution) to generate more solutions and the one which give best result is updated as the particle's new solution.
- Update best position (solution) p_j of each particle in the swarm based on the old and new position of the particle with the one having minimum value of the objective function.
- Update best position p_g of the whole swarm using best solution of all the particles in the swarm.

VI. Architecture of Job Scheduling System

The general architecture of job scheduling system is shown in *Fig. 1*. Multiple users submit jobs through cloud interface for execution. Each job contains multiple tasks which it needs to schedule on the available virtual machines. First, a process estimates execution time of each individual task in a job and adds the job along with the estimated execution time to scheduling job queue. In the next step, scheduling process gets all available tasks for scheduling and generates the optimal schedule for execution from PSO algorithm. Later, this schedule is used to execute tasks on available virtual machines.

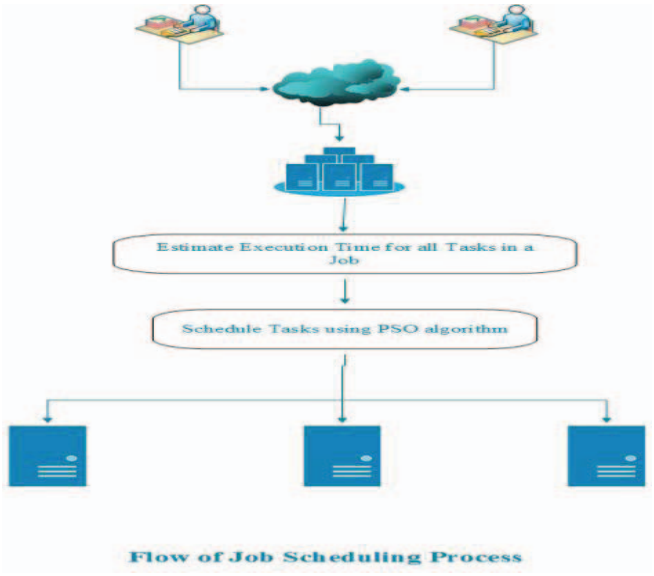


Fig. 1. General architecture of job scheduling system

VII. Results

We have performed the evaluation of our algorithm on (I) benchmark scheduling problem [1] and (II) randomly generated problems (i.e., using parameters defined in Table IV). Results for the benchmark problem is shown in Table V, VI and VII. *Fig. 2* represents the detailed analysis of the results for benchmark scheduling problem. *Fig. 3*, *Fig. 4* and *Fig. 5* show the detailed analysis of the results obtained for randomly generated scheduling problem.

The load on individual nodes for schedule generated through BF and PSO algorithms are shown in Table V and VI respectively. Table VII shows the detail comparison of schedule duration and algorithm scheduling time for BF, PSO, GA, FIFO and DSP algorithms.

The best schedule for any scheduling problem can be achieved using BF algorithm. So, if we can find the schedule using BF algorithm than this can be used for comparison with the schedule obtained from other algorithms. However, due to the combinatorial nature of the scheduling problem, it is very difficult and even impossible to find schedule using BF algorithm for large scheduling problems (i.e., problems with many jobs and tasks).

The benchmark scheduling problem is a small scheduling problem with three jobs and each of the jobs contains three tasks on a three machine cluster. Therefore, we solved this problem using BF algorithm and compared it with PSO and GA [1] algorithms. The best completion time obtained through BF algorithm for this problem is **12.4** and our PSO based algorithm manages to find the same results. However, earlier best result for the same problem using GA was **12.8**. The execution time taken by BF algorithm is **8.6** times slower than proposed PSO algorithm.

The second evaluation is performed on randomly generated scheduling problem as defined in Table IV. For this comparison, we have generated hundred problems having three jobs and each job contains three tasks (i.e., total nine

tasks) on three node cluster. These scheduling problems can be easily solved using BF because of its small size. So, we first find optimum solution of hundred random scheduling problems from BF algorithm and then compare it with solution of hundred random scheduling problems found using the proposed PSO algorithm.

Fig. 2 shows the completion time of hundred random problems from BF and PSO algorithm. It clearly shows that the results obtained from both algorithms are either same or very close to each other. However, *Fig. 3* shows the number of iterations and *Fig. 4* shows the time taken to schedule each of hundred random problems from BF and PSO algorithm. The number of iterations used to solve by BF algorithm was fixed for a particular size of problem (i.e., almost 20,000 iterations used by BF algorithm) but PSO algorithm is executed for any number of iterations (i.e., 100 iterations used by BF algorithm and in almost all the cases best result was obtained with 25 iterations). Similarly, *Fig. 4* shows that the BF algorithm used between 5000 and 6000 ms and PSO algorithm used less than 10 ms to find the schedule of each problem.

We have also included an analysis of hundred randomly generated bigger scheduling problem having five jobs and each job contains ten tasks (i.e., total fifty tasks) on five node cluster. These scheduling problems are difficult to solve using BF algorithm because of its large size (i.e., 5^{50} iterations). So, we just solved all hundred random scheduling problems from PSO algorithm. *Fig. 5* shows the completion time, number of iterations and time taken to schedule each problem. We have run PSO algorithm for hundred iterations but the algorithm successfully able to find best solution in less than 25 iterations (i.e., solutions did not change after 25 iterations).

VIII. CONCLUSION

In this research paper, we have performed a detailed analysis of the application of meta-heuristic to solve real time task scheduling problem. We have solved one benchmark problem and hundred random generated small size problems with both BF and PSO algorithm and hundred random generated large problems using PSO algorithm. For the benchmark scheduling problem, the PSO finds the optimum result (i.e., BF and PSO results are same) which could not be found using GA, FIFO and DSP. Similarly, for small randomly generated problems, PSO found the results very close to the results found by the BF. For large random generated scheduling problems, PSO converge to its optimum value in less than 25 iterations. Therefore, it is evident to say that PSO shows its potential to solve real time scheduling problem and can be used to schedule tasks on Hadoop cluster.

TABLE V. BRUTEFORCE SCHEDULE FOR BENCHMARK PROBLEM

Node	Tasks	Completion Time
Node 1	Task 2, Task 4	12.4
Node 2	Task 1, Task 3, Task 5	12.4
Node 3	Task 6	8

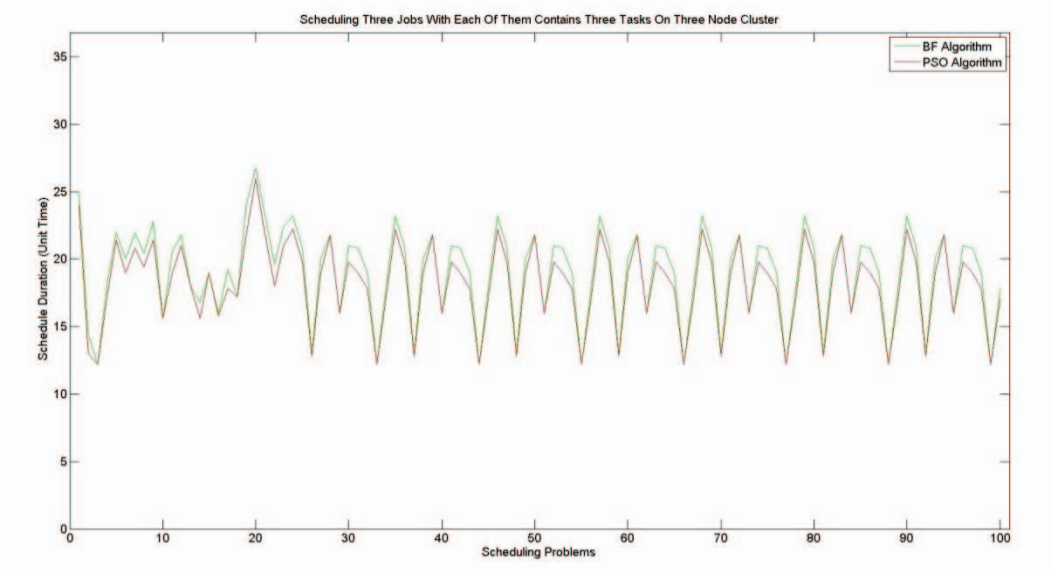
TABLE VI. PSO SCHEDULE FOR BENCHMARK SCHEDULING PROBLEM

Node	Tasks	Completion Time
Node 1	Task 2, Task 4	12.4
Node 2	Task 1, Task 3, Task 5	12.4
Node 3	Task 6	8

TABLE VII. COMPARISON OF ALGORITHMS RESULT FOR BENCHMARK SCHEDULING PROBLEM [1]

Algorithm	Best Schedule Time	Algorithm Time for Scheduling
BF	12.4	67.239 ms
PSO	12.4	7.785 ms
GA	12.8	N/A
FIFO	18	N/A
DSP	15	N/A

Fig. 2. Schedule durations of hundred nine jobs random problems



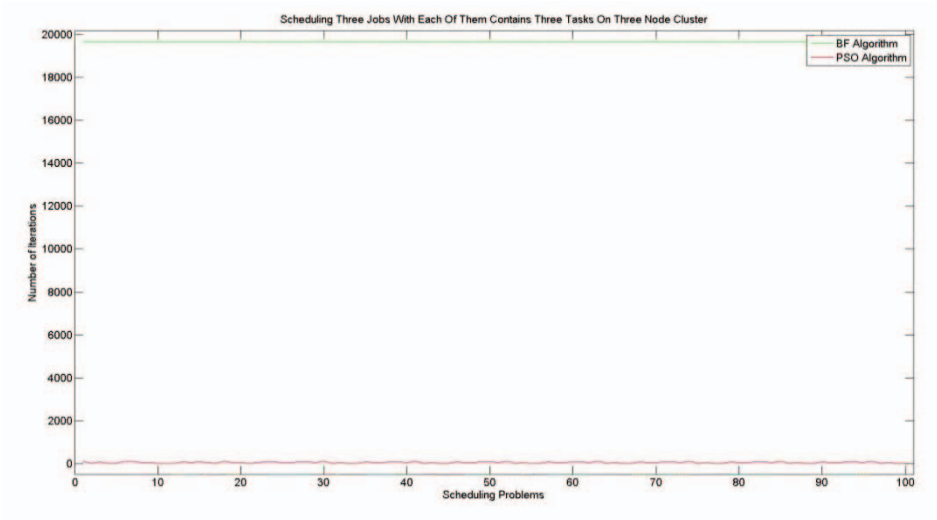


Fig. 3. Iterations used for hundred nine jobs random problems

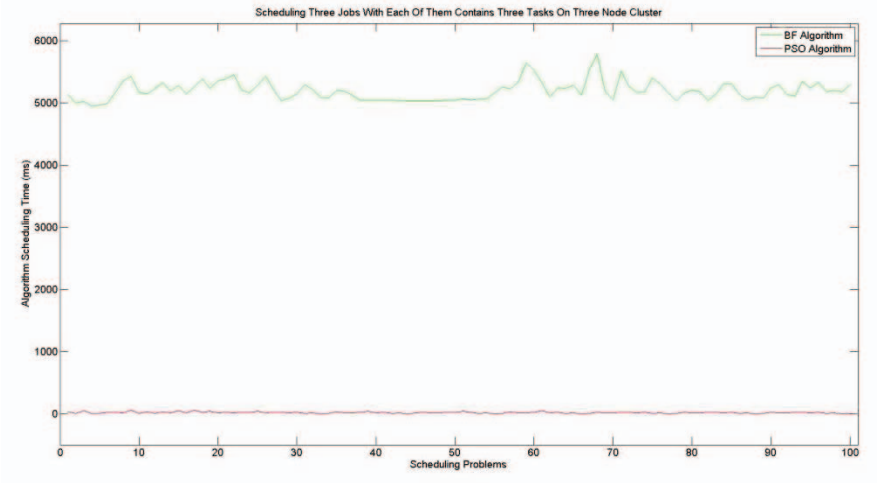


Fig. 4. Algorithm scheduling time for hundred nine jobs random problems

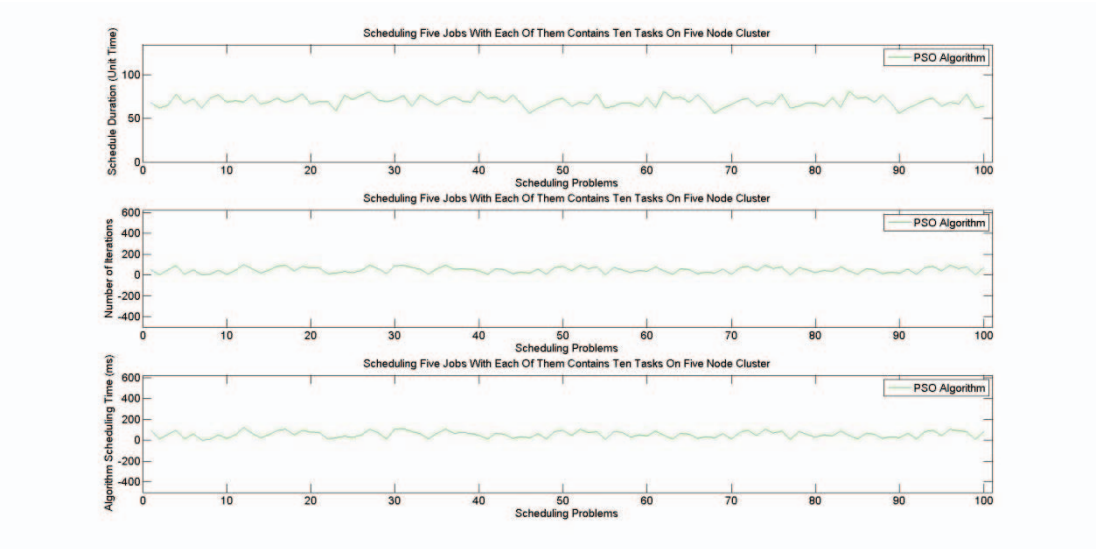


Fig. 5. Schedule durations, Algorithm scheduling time and Iterations used for hundred fifty jobs random problems

REFERENCES

[1] Y. Ge, & G. Wei, "Ga-based task scheduler for the cloud computing systems," International Conference on Web Information Systems and Mining (WISM), vol. 2, pp. 181-186, 2010.

[2] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, & I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," ACM Proceedings of the 5th European conference on Computer systems, pp. 265-278, 2010.

[3] B. T. Rao, & L. S. S. Reddy, "Survey on improved scheduling in Hadoop MapReduce in cloud environments," International Journal of Computer Applications, vol. 34, no. 9, 2011.

[4] R.P. Goldberg, "Survey of Virtual Machine Research," Computer, pp. 34-45, 1974.

[5] P. Patel, A. H. Ranabahu, & A. P. Sheth, "Service level agreement in cloud computing," http://knoesis.wright.edu/library/download/OOPSLA_cloud_wsla_v3.pdf, 2009.

[6] J. Fortes, R. Figueiredo, & P. A. Dinda, "Introduction: Resource Virtualization Renaissance," *Computer*, vol. *38*, no. 5, pp. 28-31, 2005.

[7] M. Pastorelli, A. Barbuzzi, D. Carra, M. Dell'Amico, & P. Michiardi, "HFSP: size-based scheduling for Hadoop," International Conference on Big Data, pp. 51-59, 2013.

[8] T. White, "Hadoop: The definitive guide," O'Reilly Media, Inc, 2012.

[9] D. Borthakur, "The hadoop distributed file system: Architecture and design," The Apache Software Foundation, 2007.

[10] J. Dean, & S. Ghemawat, "MapReduce: a flexible data processing tool," Communications of the ACM, Vol. 53, no. 1, pp. 72-77, 2010.

[11] A. Silberschatz, P. B. Galvin, & G. Gagne, "Operating system concepts," Addison-Wesley, 2013.

[12] J. Kay, & P. Lauder, "A fair share scheduler," Communications of the ACM, vol. 31, issue 1, pp. 44-55, 1988.

[13] R. L. Graham, "Bounds on multiprocessing timing anomalies," SIAM journal on Applied Mathematics, vol. 17, no. 2, pp. 416-429, 1969.

[14] J. Kennedy and R. C. Eberhard, "Swarm intelligence," Morgan Kaufmann Publishers, 2001.

[15] S. H. Adil, S.S.A. Ali, A. Hussaan, K. Raza, "Hybridization of Multiple Intelligent Schemes to Solve Economic Lot Scheduling Problem Using Basic Period Approach," Life Sci J, vol. 10, no. 2, 2013.

[16] S. Qamar, & S. H. Adil, "Comparative analysis of data mining techniques for financial data using parallel processing," In Proceedings of the 7th International Conference on Frontiers of Information Technology, 2009.

[17] J. Kennedy, & R. Eberhart, "Particle swarm optimization," International Conference on Neural Networks, pp. 1942–1948, 1995.

[18] M. Szmajduch, & J. Kołodziej, "Data-aware Scheduling in Massive Heterogeneous Systems," In Proc. 29th Eur. Conf. Modell. Simul. ECMS, pp. 608-614, 2015.

[19] D. Grzonka, M. Szczygieł, A. Bernasiewicz, A. Wilczyński, & M. Liszka, "Short analysis of implementation and resource utilization for the OpenStack cloud computing platform," In Proc. 29th Eur. Conf. Modell. Simul. ECMS, pp. 608-614, 2015