The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

An Efficient Method for Improving Backfill Job Scheduling Algorithm in Cluster Computing Systems

Zeynab Moradpour Hafshejani[1]
Islamic Azad University, Shabestar, Iran

Seyedeh Leili Mirtaheri[2]
Iran University of Science and Technology, School of Computer Engineering, Tehran, Iran

Ehsan Mousavi Khaneghah[3]
Iran University of Science and Technology, School of Computer Engineering, Tehran, Iran

Mohsen Sharifi[4]
Iran University of Science and Technology, School of Computer Engineering, Tehran, Iran

***Abstract*— One of the most important issues in cluster computing systems is the efficient use of resources to increase the performance of systems and hence decrease their response times. These objectives can best be pursued by job schedulers in cluster computing systems. However, existing schedulers in cluster computing systems do not use resources efficiently. This paper proposes a new method for efficient allocation of submitted jobs to resources. Jobs consist of threads that are arranged in a two-dimensional matrix. Using the horizontal scanning of this matrix, threads of different jobs are allocated to different processors, preventing resources becoming idle. In previous method of scheduling, the resources are allocated to total threads of a job synchronization but in proposed method, the resources are allocated to threads of various jobs. In new method if there aren't available resources enough for a job, threads of different jobs can run thus waste of resources are minimum. Simulation results of our proposed scheduling method show quicker cluster system response time than FCFS and Backfilling scheduling methods.**

Keywords- cluster computing system; scheduling; computing resource; thread; response time.

I. INTRODUCTION

A great number of large scale applications in science, engineering and economy are performed on parallel systems in order to achieve high performance computing. Some clusters of computer have been placed as effective and appropriate substitutes for parallel computers owing to rapid advancements in producing powerful microprocessors and high speed networks and standard software tools. Therefore, parallel computing extensively migrated from expensive supercomputers to cheaper clusters. Clusters are formed of several stand-alone computers which are connected together via a network and manage the resource sharing of workstations and the distribution of computational capability effectively [9, 11].

The scheduling topic in distributed systems and clusters is one of the significant topics in more utilization of available resources and reducing the response time. Job scheduling softwares are amongst important softwares in clusters. Aim of job scheduling concerns appropriate allocation of processors and subsequently maximizing the use of system resources and reducing the average response time. Parallel job scheduling is one of the subjects on which considerable researches have been carried out. A parallel job consists of several processors performing simultaneously all which carry out a certain computation [10].

In distributed systems and clusters the scheduling is conducted in three methods according to the sharing policy of processors as follows: 1. Space-sharing 2. Time- sharing 3.Hybrid (a composition of space-sharing and time-sharing).

In space sharing policy the number of processors P is divided into N partition every which is used exclusively for performing a job. In time-sharing policy the possibility of using a collection of processors is not given exclusively to a job and instead the possibility of share using of processors is given to severed jobs by giving a time slice from each processor to each job, e.g., round robin.

In performing parallel jobs the space-sharing policy is mostly used. This policy is implemented in two ways of static and dynamic. The static mode in which the size of partition is fixed is implemented in three models of Fix, Variable and Adaptive [1, 2, 4]. In Fix partitioning model, the size of partition which is determined according to the properties of workload in the start time of the system remains fixed for a long time and it changes only with restarting the system.

The advantage of this method is easy implementation and its disadvantage involves non-adaptively between the size of partition and the job request to processor and as a result internal fragmentation occurs.

In Variable partitioning model, the size of partition is determined based on user's request when proposing the job. The problems of this model relates to the size of partition, which is not affected by system load changes and the tendency job for monopolizing the system resources.

In Adaptive partitioning model, the size of partition depends on both user's request and system load.

Although the two recent models are better than the Fix model, there are still problems of not releasing resources, fragmentation and waste of resources because assigning the partition is conducted for the life time of a job.

The idea behind Dynamic partitioning policy is to take away idle processors from a job and allocate them to another job in order to be able to increase the performance of processors. The problems of this method are extra overhead caused by repartitioning and the need of coordination between the application and operating system to take away processors from the application. To lessen the

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

overhead it is necessary to wait until computations reach a desirable point and as a result of this the system overhead decreases (end of a computational phase) [1, 2, 3].

Most of the algorithms in space-sharing give rise to the formation of sets of unused resources in system. To resolve the fragmentation problem in FCFS[1], the Backfilling is proposed that eventually brings about a major improvement in the performance of resource. In this method, when there are no enough processors in the queue for the next job, the algorithm goes forward in the queue until it reaches a job whose required processor is less or equal to the number of free processors and runs that.

Despite the improvement that the backfilling achieves in using resource, if there is no job that requires fewer processors than free processors, the free processors will remain idle. Another defect of this method is that small jobs requiring fewer resources may queue for long and wait until the running of a large job finishes and afterwards resources are released. Estimating the running time of jobs is amongst other important topics in the backfilling method. Estimation methods are always affected by error.

For decreasing and improving these problems a new method is proposed. This method use threads for resource allocating instead of allocating resource to total process of a job.

A process is an activity within the system that is started with a command, shell script, or another process[16] and a thread is a kind of dispatchable process with few overhead that need few resources to run.

In this method, a matrix for keeping incoming jobs to system in the node having been introduced as a node receiving incoming jobs in cluster is defined. The number of columns of this matrix is equal to the number of incoming jobs to system and the number of rows of this matrix is equal to the number of threads of the biggest incoming job to the node and it is changeable.

With matrix horizontal scanning, resource allocate to first threads of some jobs instead of allocating resource to total processors of a job simultaneously. So, contrary to last algorithms if there aren't enough resources for a job, available resources don't remain idle and threads of some jobs use idle resources.

We used Simgrid tool for simulating method. Evaluations show because resources don't allocate to total processes of a job simultaneously, it is possible that response time be more than backfilling algorithm for proposed jobs in head of queue but because don't remain resource idle and use of resource optimally, overall system's response time decreases and system's utilization increases.

The rest of this paper is organized as follow. Section 2 discusses the related work. Section 3 introduces the architecture of our scheduling method in cluster system. In section 4, we simulate and evaluate this strategy with Simgrid in part 5. Finally, we end this paper with conclusions.

II. RELATED WORKS

As it was considered, for resolving the fragmentation problem in FCFS, the backfilling is offered that brings about a major improvement in the performance of resources. In this method, when there are no enough processors in the queue for the next job, it goes forward in the queue until it reaches a job whose required processor is less or equal to the number of free processors and runs that. There is also a more efficient algorithm called preemptive backfilling and if there are no jobs with high priority in this method, resources will be reserved for those and then resources will be given to the job with low priority. If job with higher priority is proposed, running job is finished in one of these cases such as: suspend-resume, checkpoint-restart, kill-restart and job with higher priority get resources. The Maui scheduler uses *preemptive backfilling* in addition to the *backfilling* and *advanced reservation* [1].

With attention to reserved resources parameter in backfilling, the classifications of this technique are as follows:

- ESAY Backfilling
- Conservative backfilling
- Flexible Backfilling
- Multiple-queue backfilling

If reservation takes place only for the first available job, it is called easy backfilling which ultimately might delay other jobs and hence the conservative backfilling technique was designed. In this method, reservation is carried out for all jobs before the backfilled job to prevent any delays. These two methods are not dynamic and the waiting time of the job in the queue is not considered and as a result the Flexible backfilling method was designed. In this algorithm, each job waiting in the queue is given a slack factor for the computation of which the waiting time of job in the queue is considered (for example if the slack factor is equal to three this means that jobs may wait three times the average waiting time). Important jobs have smaller slacks than other jobs. "Lawson and Amirin" presented another method called multiple-queue backfilling in which each job is given to a queue in accordance with its expected running time and each queue is given to a partition of the parallel system and only the jobs of this queue can be run. Conducted examinations show that the advantage of this method to the single-queue method involves the possibility that small jobs in the queue delay behind long jobs is less [5, 6, 7, 8].

Despite the improvement that the backfilling achieves in using resources and even the usage of dynamic method in space-sharing and release of processors upon finishing the running of their related processes and the possibility of repartitioning and more use of free resources, if there is no job that requires less processors than free processors, the free processors will remain idle.

[1] First come first service

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

Another defect of this method is that small jobs requiting fewer resources may queue for long and wait until the running of a large job finishes and afterwards resources are released. Estimating the running time of jobs is amongst other important topics in the backfilling methods. Estimating the running time is determined either by user when submitting a job or by system with using the data and information which have been achieved from previous running history either estimation methods are always affected by errors [12].

Raised problems of these inaccurate estimations concern illogical prediction of the running time of the relevant job via user. For example, in LSF[2] user is given a command to determine the running time of the job and now if the running time of the job exceeds this time the job will be killed. Another problem involves over estimation of running time. Therefore if the job finishes sooner, these resources cannot be used for running a job with higher priority because of being used by the backfilled job. Of course, the LSF has resolved this problem by using checkpoint. Another problem is also resulted when there is no appropriate job for backfilling due to inaccurate estimation of long running time that causes wasting resources [7].

In order to reduce the above-mentioned problems, a new method is introduced, which improves the problems increasingly.

One of approach for improving use of resources in the past was to add a time-sharing dimension to space-sharing using a technique called gang scheduling or co-scheduling(All tasks of a parallel job are always co-scheduled to run concurrently.). This technique virtualizes the physical machine by slicing the time axis into multiple space-shared virtual machines that use backfilling. Algorithm can be represented by a matrix, in which the rows represent time slices and the columns represent processors. Each row of the matrix defines a virtual parallel machine, which has the same number of processors as the physical machine but runs slower. We use these virtual machines to run multiple parallel jobs. Multiprogramming level (MPL) in general depends on how many jobs can be executed concurrently, but is typically limited by system resources.

This approach opens more opportunities for the execution of parallel jobs, and is thus quite effective in reducing the wait time [14].

One of approach for resolving backfilling problems in the past was Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems. In this approach the system is divided into multiple disjoint partitions by classifying jobs according to their duration (not their requested number of processors). Initially, processors are divided evenly among the partitions. As time evolves, the partitions may exchange processors therefore idle processors in one partition can be used for backfilling in another partition. Therefore, partition boundaries become dynamic, allowing the system to adapt itself to changing workload conditions. Furthermore, the policy does not starve a job that requires the entire machine for execution. Each partition contains its own separate queue of jobs. When a job is submitted, it is classified and assigned to the queue in one of partitions after all previously queued high priority jobs but before the first low priority job in the queue [13]. One of approach's disadvantages is Inaccurate running time estimation by user that is caused job put in unsuitable queue. Therefore, running time of other jobs is affected. Furthermore it have overhead of repartitioning but there isn't need to repartitioning and user running time estimation in new proposed method.

Another proposed approach was Combinational backfilling scheduling (CBA) in 2010. Combinational backfilling scheduling is that according to the state of the free resources, it selects a group of small jobs to backfill the resources gap to maximize the utilization of resources in clusters. There are two special cases for CBA scheduling strategy. Firstly, when there are no combinational jobs in the waiting jobs queue, the degradation of CBA is EASY backfilling. Secondly, when there are no jobs in the waiting jobs queue, the degradation of CBA is FCFS scheduling algorithm. The results of experiments show that the CBA algorithm improves the utilization of resources in clusters compared with EASY backfilling and FCFS algorithm [11].

This approach has inaccurate running time estimation problems, too. Furthermore if there aren't combinational jobs for free resources, the resources remain unused.

Above policies have disadvantages such as need to repartitioning, additional overhead is caused by it, cost of context switching, need to user runtime estimation and problems of inaccurate estimations and to remain idle resource unused but in proposed method, firstly, it doesn't need to user runtime estimation and therefore it don't have inaccurate estimations problems, secondly small jobs after big jobs don't wait very much for running and threads of small jobs are running with threads of big jobs simultaneously, thirdly don't need to go forward in the queue for reaching a job whose required processor is less or equal to the number of free processors. Therefore, resources don't remain idle even if there isn't a smaller job in the queue. In new approach, threads of jobs start to run and free resources don't remain idle and it doesn't have costs of context switching and it's overhead.

III. PROPOSED METHOD

The proposed method carries out the scheduling of jobs for efficient use of the resources of system and is implemented as follow:

In this method, contrary to space-sharing policy and time-sharing policy in both which the complete allocation of job is used, thread definition is used. In this manner that in space-sharing policy all required resources of a job must be available for its start or in time-sharing policy that a collection of resources must be shared between several jobs

[2] load sharing facility

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

and a time quantum of source is allocation to each job in series and this manner continues until the end of execution of all jobs; however, the proposed method acts as follows:

The required resources are given to first process of several jobs instead of being given to all processes of a job. Thus, a matrix for keeping incoming jobs to system in the node having been introduced as a node receiving incoming jobs in cluster is defined. The number of columns of this matrix is equal to the number of incoming jobs to system and the number of rows of this matrix is equal to the number of threads of the biggest incoming job to the node and it is changeable such as the following matrix:

						J_3p_6		
						J_3p_5		J_1p_5
			J_6p_4			J_3p_4		J_1p_4
	J_8p_3		J_6p_3			J_3p_3	J_2p_3	J_1p_3
	J_8p_2	J_7p_2	J_6p_2		J_4p_2	J_3p_2	J_2p_2	J_1p_2
	J_8p_1	J_7p_1	J_6p_1	J_5p_1	J_4p_1	J_3p_1	J_2p_1	J_1p_1
9	8	7	6	5	4	3	2	1

Figure 1. Matrix of incoming jobs in receiving node

To allocate these jobs to available nodes in the cluster, the method initially creates the node of a queue and selects the members of this queue in the following manner:

Instead of allocating the processes of a job completely to the queue, for instance the j1 processes of *p1* to *p5* (*j1p1* to *j1p5*) it moves forward with a horizontal view to the matrix and observes then column of matrix all together and allocates the threads of the first row of this *n* job to resources(processors). Then, this row shifts to up and the threads of the second row insert this *n* job into the queue and in this manner it advances until the last row of the matrix. In this allocation, the quantity of *n* could vary. If *n* is quall to 1, it will become that of FCFS model and as a result all threads of a job will be allocated to processors.

The quantity of *n* is chosen by considering the first column of the matrix and setting the quantity of *n* by the number of threads of the available job in this column. Thus, it selects an *n*n* hypothetical matrix and examines if in this matrix there is any job the number of threads of which is more than *n*. in this case if there is any job with the number of threads more than *n*, it changes the dimensions of the hypothetical matrix to the number of the threads of that job and continues until it reaches a matrix having no job with the number of thread more than the dimensions of the matrix. After this, it starts inserting the members of this matrix horizontally into one-dimensional queue like figure2, 3. To select the dimensions of next matrix again the first job after the matrix is considered to repeat the aforementioned cycle additionally with attention to the number of threads of the job.

In this method, this opportunity is given to *n* jobs, that is, with each thread any job having maximum size in the matrix find the possibility of being run and do not delay greatly. Now upon the availability of resources, the jobs available in the queue find the possibility of being run. For example:

						J_3p_6		
						J_3p_5		J_1p_5
			J_6p_4			J_3p_4		J_1p_4
	J_8p_3		J_6p_3			J_3p_3	J_2p_3	J_1p_3
	J_8p_2	J_7p_2	J_6p_2		J_4p_2	J_3p_2	J_2p_2	J_1p_2
	J_8p_1	J_7p_1	J_6p_1	J_5p_1	J_4p_1	J_3p_1	J_2p_1	J_1p_1
9	8	7	6	5	4	3	2	1

Figure 2. Matrix of incoming jobs in node

In the beginning, with selecting the available job in the first column, which has 5 threads, a 5*5 matrix is considered; however, because the third available job in matrix has 6 threads, the dimensions of the matrix change to 6*6 and consequently by scanning the horizontal rows of the matrix threads are inserted into the queue.

						J_3p_6		
						J_3p_5		J_1p_5
			J_6p_4			J_3p_4		J_1p_4
	J_8p_3		J_6p_3			J_3p_3	J_2p_3	J_1p_3
	J_8p_2	J_7p_2	J_6p_2		J_4p_2	J_3p_2	J_2p_2	J_1p_2
	J_8p_1	J_7p_1	J_6p_1	J_5p_1	J_4p_1	J_3p_1	J_2p_1	J_1p_1
9	8	7	6	5	4	3	2	1

.	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
.	1	6	3	2	1	6	4	3	2	1	6	5	4	3	2	1
.	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p	p
	4	3	3	3	3	2	2	2	2	2	1	1	1	1	1	1

Figure 3. Matrix of incoming jobs in queue

Afterwards, the queue is observed in FCFS and thus it starts allocating these threads to available processors in the cluster. Each thread may release the source in variant times depending on its running time and with the release of the

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

some next thread of the queue is allocated to free resource like figure 4.

This question may be raised that if there is data dependency between threads, how is the need of data solved?

The method performs as follows:

The first thread become dangler and observes the queue. If the relevant thread was allocated to a resource, it receives the result from that source, otherwise in case that the relevant thread had not been allocated yet to a source for running, it would have been given the priority "1" which is inserted into the queue of the CPU of dangled thread and is placed in front of the dangled thread to be run and provide the required data of the dangled thread.

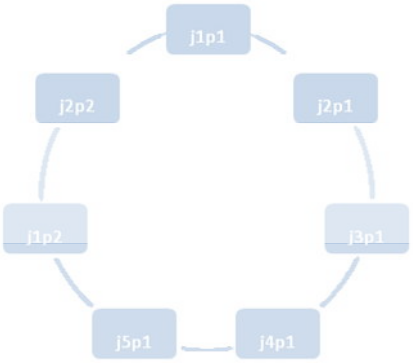

Figure 4. Example of resource allocation in cluster

The above method has several advantages and disadvantages. It seems that if *n* becomes bigger than an amount, it causes the first available job in the queue to have more running time than the common FCFS model; however, it also has some advantages, that is, by choosing an appropriate amount for that this delay in running could be ignored.

Firstly, in this method contrary to the backfilling algorithm there is no need to the running time by user and these matters increasingly because user's inaccurate estimations led to many problems mentioned above. Secondly, small jobs after long jobs do not wait considerably until the running of big job finishes and then they are run and this method can decrease the running time of small job and simultaneously with the running of some of the threads of big job the threads of small job also find running time.

Thirdly, in the event that the number of free resources is less than the number of required resources of available job in the head of the queue, there will be no need to search for finding the job requiring less number of resources and jobs are run in the order of possibility and even if small jobs do not exist in the queue, resources will not stand idle and the execution of the threads of jobs will continue and consequently no source will stand idle.

IV. Evaluation

Scheduling of processes onto processors of a parallel machine has always been an important and challenging area of research. Its importance stems from the impact of the scheduling discipline on the throughput and response times of the system [15]. Therefore with improve these results, utility of system increase. These terms is defined like this: Throughput is number of performed jobs per time unit. Response time is different between times of entering job to system and completing job. Utilization is percent of time that system is running really [1].

Simulation has been done with using Simgrid simulation software for three models like FCFS, backfilling, improved backfilling and results for response time parameter are described below.

a. Analyzing the overall response time of system

In this proposed method an approach like round robin instead of FCFS is used. Therefore, in contrary to backfilling model, the total processes of a job do not receive resources and there is a possibility that the response time of a job is more than other approach, but because it does not remain resources stay unused and the threads of jobs can run. Therefore, the usage of resources is maximized and the overall response time of system will be improved. A simulation shows that the response time in FCFS is more than backfilling and that the proposed method has a response time better than backfilling and FCFS.

In FCFS, jobs are scheduled based on being proposed and entering queue and if there aren't enough resources in the system for next job in the queue, the job can't run and the resources remain idle till enough required resources is prepared. Improvement in backfilling is more than FCFS. When there are no enough processors in the queue for the next job, the method goes forward in the queue until it reaches a job, whose required processor is less or equal to the number of free processors, and runs that. When other used resources are free, the job in the head of the queue can run. If there is no job that requires fewer processors than free processors, the free processors will remain idle. However, simulation methods show that the overall response time decreases considerably. The simulation was carried out in several numbers of jobs.

There are charts of 3 experiments with 1292, 2018 and 3113 proposed jobs in figure 5, 6, 7.

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

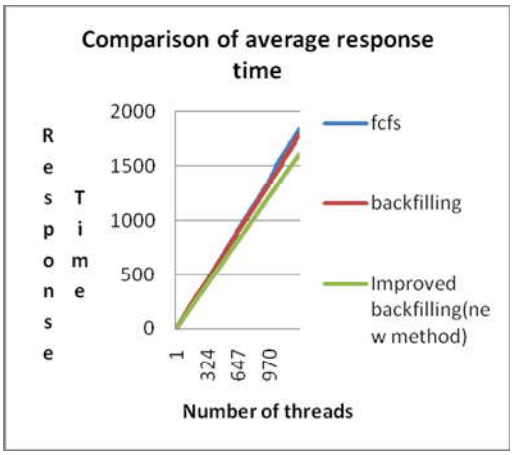

Figure 5. Comparison of overall response time in 3 Scheduling methods in 1292 proposed job

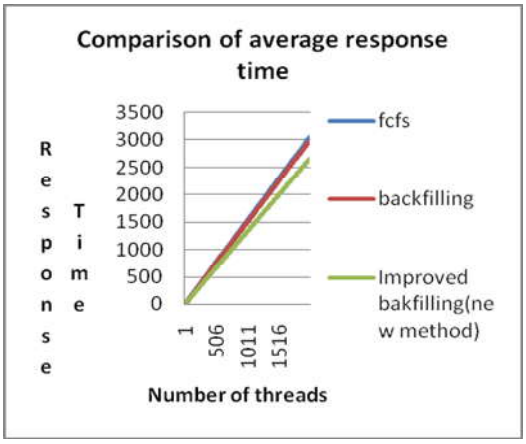

Figure 6. Comparison of overall response time in 3 Scheduling methods in 2018 proposed job

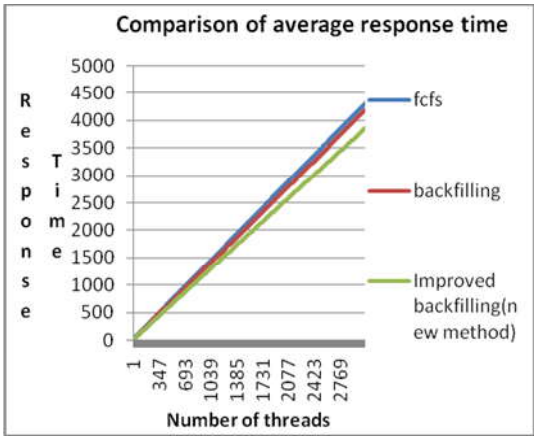

Figure 7. Comparison of overall response time in 3 Scheduling methods in 3113 proposed job

Last chart is obtained with 3113 proposed jobs. As mentioned, this method works based on threads and if there is a minimum of required resources for running a thread, the thread uses the resources and run. In this way, resources aren't wasted and the use of resources is maximized. If there are not sufficient free resources, the relevant thread waits until enough resources become available. Therefore, wasting resources is not considerable.

"Fig. 8" is obtain of last experiment with 3113 proposed jobs and represents that the overall response time of system in new method decreases thus the utility of system increases. For example for running 3113 proposed jobs, overall response time of system is decreased from 4313 and 4197 in FCFS and backfilling to 3861 in new method. The total threads of a job are running simultaneously in normal approaches of scheduling. This means that the total required resources of a job must be prepared so that threads can receive resources and run. If there is adequate free resources, next jobs of the queue will run sequentially. If small jobs are proposed later than big jobs, they must wait to realize their required resources which are used by big jobs. Therefore, the small jobs after big jobs will have high response time.

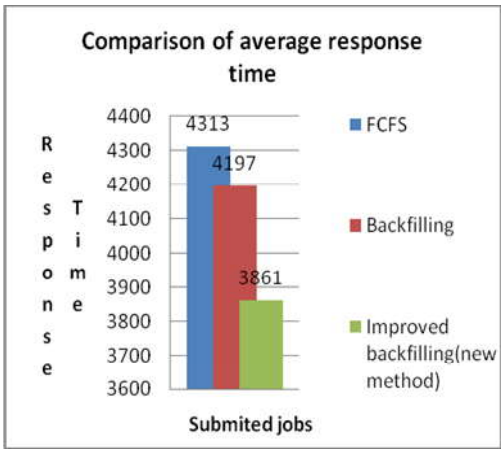

Figure 8. Comparison of overall response time of system

However, the new-proposed method relies on the horizontal scanning of jobs matrix. The first threads of some jobs start to run; then the second threads; after this the third threads and again this rhythm would repeat. Therefore, the distribution of resources between jobs is more reasonable and controlled and if big jobs are proposed before small jobs, it would be possible to run small jobs faster and to decrease their waiting time. Although existing jobs in the head of the queue may have had greater response time than previous approaches, in contrary to FCFS and backfilling algorithm, its advantage is that wasting resources is minimal

The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, Special Issue:
The Proceeding of International Conference on Soft Computing and Software Engineering 2013 [SCSE'13],
San Francisco State University, CA, U.S.A., March 2013
Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

and the use of resources is most. As the simulation charts demonstrate, there is a decrease in response time average in this method.

In previous approach, a job starts to run completely and it is possible that this job may not be able to run entirely due to lack of sufficient resources, whereas in the proposed approach, which is based on threads, even if there are not enough resources for a job, threads of some jobs can still run. This eventually allows jobs to run and wasting resources is minimized. The new method proceeds without a dependency on estimating of the running time of jobs and as a result inaccurate estimation does not exist anymore. Estimation of the jobs running time is conducted by user or system via either using historical data through repeated executions of the job or compile-time analysis. The estimation method is always affected by errors. One of the problems of inaccurate estimation concerns illogic forecast of the jobs runtime by user. For example, if the runtime is underestimated and the real runtime of job is more, the job will be killed. Furthermore, overestimation of the runtime could be caused the inappropriate selection of jobs, while the new-proposed method avoids inaccurate estimations.

V. CONCLUSION

As highlighted in previous cases, in this proposed method the overall response time of system would decrease because process of various jobs can receive resources instead of receiving resources by the total process of a job simultaneously. Therefore, if there are not sufficient resources to run a job, threads of jobs still can run and resources don't remain unused. In addition, the new method has the advantage of allowing the jobs after big jobs to run faster, which in fact prevents these jobs from waiting for running of big jobs and realizing resources. Consequently, the waiting time and response time of next small jobs in queue would decrease. In this approach, if there are not sufficient resources to run a job, threads of jobs still can run. Thus, other free resources would not remain idle and jobs would be able to run and the utilization of system could increase.

Although, this proposed method has advantages but there is some disadvantage such as it is possible that response time of a job is more than before specially for jobs at the first of queue or it is possible that primary-submitted jobs are not respond immediately. Moreover, despite the disadvantages of the new method for primary-submitted jobs and response time of a job, it is preferred to FCFS and backfilling because of its features such as optimum usage of resources, increasing utilization of system, and decreasing the overall response time of system.

In this paper, a new method proposed for improving backfilling scheduling. In the future work, it would be interesting to study scheduling methods in other distributed systems and extend of this new method in these systems and new cluster which aren't central.

REFERENCES

[1] I. Grudenić, "Scheduling Algorithms and Support Tools for parallel systems," 2008. [Online].

[2] J. H. Abawajy and S. P. Dandamudi, "Time/Space Sharing Distributed Job Scheduling Policy in a Workstation Cluster Environment," in Parallel Computing in Electrical Engineering, 2000. PARELEC 2000. Proceedings. International Conference on, 2000 , pp. 116-120.

[3] G. S. K. Amit Chhabra, "Simulated Performance Analysis of Multiprocessor Dynamic Space-Sharing Scheduling policy," IJCSNS International Journal of Computer Science and Network S 326 ecurity, vol. 9 No.2, Feb. 2009.

[4] P. D. Y. Sivarama, "Performance Sensitivity of Space-Sharing Processor Scheduling in Distributed-Memory Multicomputers," in Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998, Ottawa, Ontario K1S 5B6, Canada, 1998, pp. 403-409.

[5] C. Scheduling, "Cluster Scheduling," 159.735.

[6] L. R. Dror G. Feitelson, "Parallel Job Scheduling—A Status Report," Computer Engineering Institute-Universit¨at Dortmund, Dortmund, Germany, A Status Report 44221.

[7] J. Wang and W. Guo, "The Application of Backfilling in Cluster Systems," in Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on , Beijing, 100876, P.R.China, 2009, pp. 55-59.

[8] A. M. G. Adam K.L. Wong, "evaluating the EASY-Backfill Job Scheduling of Static Workloads on Clusters," in Cluster Computing, 2007 IEEE International Conference on , 2007, pp. 64-73.

[9] A. T. S. C. ,. Y. S. Jiannong Cao, "A taxonomy of application scheduling tools for high performance cluster computing," Springer Science, vol. 9, p. 355–371, 2006.

[10] D. G. F. Edi Shmueli, "Backfilling with lookahead to Optimize the Performance of Paralell Job Scheduling".

[11] S. Yi, Z. Wang, S. Ma, Z. Che, and Feng, "Combinational Backfilling for Parallel Job Scheduling," in Education Technology and Computer (ICETC), 2010 2nd International Conference on , china, 2010, pp. V2-112-V2-116.

[12] S. K. Dimitriadou and H. D. Karatza, "Job Scheduling in a Distributed System Using Backfilling with Inaccurate Runtime Computations," in Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on , 2010, pp. 329-336.

[13] E. S. Barry G. Lawson, "Multiple-Queue Backflling Scheduling with Priorities and Reservations for Parallel Systems," springer, vol. 2537, no. Lecture Notes in Computer Science,Job Scheduling Strategies for Parallel Processing, pp. 72-87, 2002.

Doi: 10.7321/jscse.v3.n3.64 e-ISSN: 2251-7545

[14] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," in Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, 2000, pp. 133-142.

[15] Y. Zhang, A. Sivasubramaniam, J. Moreira, and H. Franke, "Impact of Workload and System Parameters on Next Generation Cluster Scheduling Mechanisms," in Parallel and Distributed Systems, IEEE Transactions on , 2001, pp. 967-985.

[16] K. Milberg. (2007, Mar.) Process priority and control on AIX.