A Comparative Study of Batch Scheduling Strategies for Parallel Computing System

Mohammad Shahid, Zahid Raza, and Deo PrakashVidyarthi

***Abstract*—A parallel computing system helps the user to exploit the parallelism inherent in the job to ensure its execution in the minimum time. This is done by distributing the job over multiple processors available in the system. Scheduling a job on such a system gains paramount importance and is always desired from a scheduling policy. Efficient job execution exploits the software parallelism in the job to map it with the available parallel hardware. A batch scheduler schedules the similar jobs by pooling them together and allocating them on the appropriate processor. Batch scheduling is useful as it allows the sharing of resources among many users and programs. Further, it avoids the idling of resources thereby increasing the utilization to a greater extent. A batch scheduler ensures proper amalgamation of software parallelism with hardware parallelism. This paper compares the performance of three batch scheduling policies viz. First Come First Serve (FCFS), Turnaround Based Scheduling Scheme (TBSS) and Batch Scheduling Scheme (BS). Simulation study is performed to analyze the performance of these three strategies under various test conditions that involves varying the hardware and software parallelism to observe the effect on the Turnaround Time (TAT) for the batch of jobs.**

***Index Terms*—Batch scheduling, JPDG, parallel computing, turnaround time (TAT).**

I. Introduction

The computer was introduced as a single processor machine with limited capabilities and the aim of making the computation faster. Computational aspect aims to have faster machines that can complete the job in smallest possible time. Computer systems have also evolved drastically over the period of time. The system witnesses the introduction of many features like hierarchical memory system, cache, spooling, buffering, pipelining, context switching to name a few to induce parallelism in the sequential machine. Newer tools were developed resulting in improved software applications too. A major objective of all these exercise was to minimize the turnaround time of the job execution. Eventually, this thirst for improvements led to multiprocessor and multi computer machines providing the parallel and distributed computing environment to the job. This evolution has resulted in the realization of today's very efficient high end computing environment in the form of cluster, grid and cloud computing.

A job demanding execution may be considered as a group of sub-jobs (modules). A job is said to be highly parallel if the degree of interaction between the various modules is low and vice versa. The modular nature of the jobs helps in representing it as a Job Precedence and Dependence Graph (JPDG) which provides the information of the number of modules in the job, the degree of parallelism and the degree of interaction between the interactive modules.

Job scheduling schemes can be static or dynamic depending on whether the requirement of scheduling is offline (batch scheduling) or online. Online scheduling is often employed when the job requires immediate scheduling and the scheduler have no information about the job's requirements. Batch scheduler clubs together various similar jobs and schedules them for execution with the single objective of the turnaround minimization. The scheduling policy for the batch can be decided easily based on the targets as the requirements are known beforehand. In the case of online scheduling the scheduler has no idea about the incoming jobs and therefore the requirements are unknown till the job actually reaches for execution.

On a uni-processor scheduling the objective is to decide the way in which the CPU time slice will be given to the jobs assigned to the processor. The popular scheduling schemes are First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR) etc. and is entirely the prerogative of the Operating System. In the case of the multiprocessor machine, scheduling helps in assigning the jobs to the appropriate node to ensure the minimum turnaround time for the jobs submitted. The appropriateness of the nodes depends on various factors, considered for allocation,viz. the processing speed of the node, number of jobs already assigned to the node, reliability of the node or even a combination of some objectives.

Scheduling a job on a parallel processing system is the problem of assigning the given job comprising of sub-jobs on the appropriate nodes mostly in order to minimize the job execution time. Thus, it can be defined as the problem of mapping the sub-jobs (modules) as per the JPDG to the processor graph. A sample job scheduling problem is illustrated in Fig. 1[1].

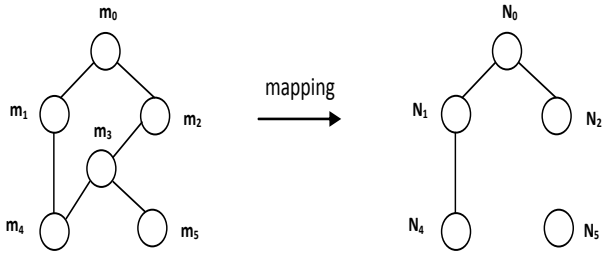

Fig. 1. A sample job scheduling problem.

Manuscript received September 21, 2012; revised November 12, 2012.

The authors are with the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi - 110067, India (e-mail: mdshahid.cs@ gmail.com, zahidraza75@gmail.com, dpv@mail.jnu.ac.in).

A batch of jobs can be considered as a set of many jobs with each job being represented as a Directed Acyclic Graph (DAG) in the form of JPDG. Each job reflects the precedence of the sub-jobs in the job and the degree of interaction between them. Software parallelism in the job(s) is decided by two factors viz. precedence levels between sub-jobs and the number of sub-jobs at each precedence level. A job is said to be highly parallel if the degree of interaction between the sub-jobs is very low. The parallelism is further improved with more number of sub-jobs available for execution at each precedence level. Fig. 2 shows two single jobs comprising of five sub-jobs in the states of low and high parallelism.

For a batch of jobs, parallelism can be viewed at either the job level or at both the job level and the sub-jobs level in accordance to the JPDG.The work has referred to the sub-jobs as modules and has been used interchangeably.

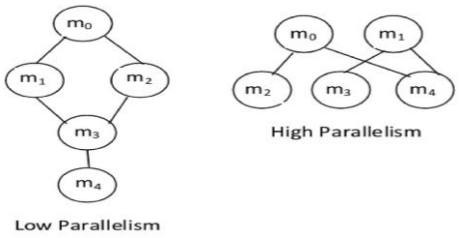

Fig. 2. Jobs with low and high parallelism.

The extent to which the parallelism exhibited by the batch can be exploited depends on the scheduling scheme which is used to allocate these jobs (sub-jobs). If the scheduling scheme is such that it views the parallelism in the batch only at the job level, the requirements are confined only till allocating the individual jobs on the appropriate resources. The best scheduling strategy will be realized when the parallelism is seen at the job level along with the in depth parallelism offered by the sub-jobs of the jobs internally. Thus, in this case, the scheduler should be capable of identifying individual jobs and allocate the sub-jobs of the job under consideration to appropriate nodes so as to exploit the parallelism in the batch to its maximum.

The paper has been divided into five sections. Section II reports some similar models. Section III highlights the FCFS, TBSS and BS scheduling schemes used for comparison in this work. Section IV presents the experimental study and their analysis. The paper ends in Section V detailing the conclusions drawn from the study.

II. Related Work

A lot of work has been done to suggest and evaluate scheduling models for parallel processing systems. Some issues and approaches in the parallel job scheduling are discussed in [2]. Another work presenting some recent developments and challenges of parallel job scheduling spanning various domains like workloads, requirement characterization, resource management etc. has been discussed in [3]. A survey of some approaches that have proven influential with parallel systems has been presented in [4]. A scheduling model for the parallel system by monitoring the job's computation granularity and communication pattern and then scheduling them has been reported in [5]. Duplication of machines in the batch scheduling environment to minimize the make span for the parallel processing system is presented in [6]. Fuzzy systems have found use in the work [7] for scheduling of the parallel jobs based on process grain size. An online batch scheduling model is also presented in [8] and [9]. Various approaches to schedule a mixed batch with interactive loads finds place in the literature in[10].

III. FCFS, TBSS and BS

This paper extends the work presented in [11] to focus on comparing the performance of First Come First Serve (FCFS) and Turnaround Based Scheduling Scheme (TBSS) with the batch scheduling scheme [12]. The various assumptions for the study are listed below.

1) A batch of N jobs J_j (j=1 to N) may be submitted to the system at any time with the batch having jobs of similar nature.
2) Each job J_j is submitted in the preprocessed manner with all the information about itself like the number of modules M_{ij} (i=1 to M) comprising the job, number of instructions (I_i) in each module.
3) The clock frequency (f_k) of each node is known to the system.
4) The previous workload on the nodes (T_k) is periodically updated in the system before scheduling each batch of jobs.
5) The study has considered the sub-jobs of the jobs to be non interactive i.e., there is no communication requirement between them while executing. However, in practice, the communication requirements may also play a decisive role in scheduling the sub-jobs.
6) The jobs forming the batch are assumed to be non interactive.
7) The queuing time for the batch is not considered while calculating the turnaround time for the batch.
8) Comparison of the scheduling policies is based mainly on how effectively they are able to minimize the turnaround time of the batch.

Since, each job in the batch is considered to be in the form of sub-jobs, the turnaround time calculation can be considered at the lowest level i.e. the Turnaround Time (TAT_{ki}) offered by a node P_k to the sub-job (module) M_{ij}. It can be calculated as the sum of the processing time (E_{ijk}) of the module on the node under consideration and the workload (T_k) corresponding to the previous modules allocated and pending on that node. This is represented as

$$TAT_{ki} = \left[\sum_{i=1}^{m} (\mathrm{E}_{ijk} X_{ijk} + T_k) \right] \quad (1)$$

Here, $E_{ijk} X_{ijk}$ represents the processing time of the node P_k under consideration calculated for node P_k for module m_i of size I_i of job J_j as

$$E_{ijk} = I_i (1/f_k) + n\alpha \quad (2)$$

where x_{ijk} is the vector indicating the assignment of module $\mathrm{m_i}$ of job J_j on node P_k. It assumes a binary value. It is 1 if

the module is allocated to the node and is 0 otherwise. T_k is the time to finish execution of the present modules on the node P_k.

FCFS is the batch scheduling scheme which works on the basis of scheduling the jobs in order of their arrival. Here the parallelism is exhibited only till the job level with each job being allocated to the node that has the minimum workload (T_k) at that moment of time. Here, even if the sub-jobs of the jobs can be executed parallel, the scheduler is incapable of exploiting this feature. The FCFS algorithm is shown in the box below.

```
FCFS
{
Submit the batch of jobs Jj(i=1to N) in the desired
   format in order of their arrival // As per Section III
Select the node Pk with minimum value of Tk
For each job (Jj)
  {
     For each module Mij
       {
         Assign the module to the selected node Pk
         Update Tk to reflect the inclusion of new job
       }
  }
Compare Tk of all the nodes on which allocation has
been made
TAT for batch = Highest TK
  // for nodes on which allocation has been made
}
```

TBSS is the scheduling strategy which exhibits parallelism at the sub-job level. Here, the scheduler schedules the job by scheduling each job module independently as suggested by the JPDG of the job. A module is allocated to thenode which offers the minimum turnaround time as per equation (1). The process is repeated for the remaining modules of the job and for all the remaining jobs. In this case, the allocation of modules is done on those nodes which are the fastest with least previous workload. Since the allocation is a result of considering the node attributes the inherent parallelism in the job is exploited resulting in an allocation pattern which may not necessarily be on one node only in contrast to the FCFS policy. The TBSS algorithm is explained as below.

```
TBSS
{
Submit the batch of jobs Ji (i=1to N) in the desired format
   in order of their arrival    // As per Section III
For each job (Jj)
  {
    For each module Mij (j=1to J)
        For each node Pk (k=1toK)
          Select the node with minimum Tk as per eq. (1)
          Assign the module to the selected node Pk
          Update Tk to reflect the inclusion of the new
           module
  }
Compare Tk of all the nodes on which allocation has
been made
 TAT for batch = Highest TK
     // For nodes on which allocation has been made
}
```

The notable difference between FCFS and TBSS policies is that in case of FCFS once a node is selected all the modules (sub-jobs) of the job are allocated on that the node only. Thus, irrespective of the presence of sub-jobs which can be executed parallel, the sub-jobs get executed sequentially on one node. In case of TBSS, although it considers the parallelism at the sub-jobs level, the nature of the scheme does not allow the next job in the batch to be executed till the previous one has finished execution. Thus, the parallelism exhibited by this scheme is only till the sub-jobs level.

BS strategy [12] is the scheduling scheme which considers the parallelism both at the job level as well as the sub-job level making it very suitable for batch mode of execution. Here, the modules of the individual jobs of the batch are partitioned into various levels as per the JPDG of the jobs. Each level corresponds to the job modules from the individual jobs of the batch which can run in parallel both at the job level and at the sub-job level. Fig. 3 illustrates the partitioning of a sample batch into various levels.

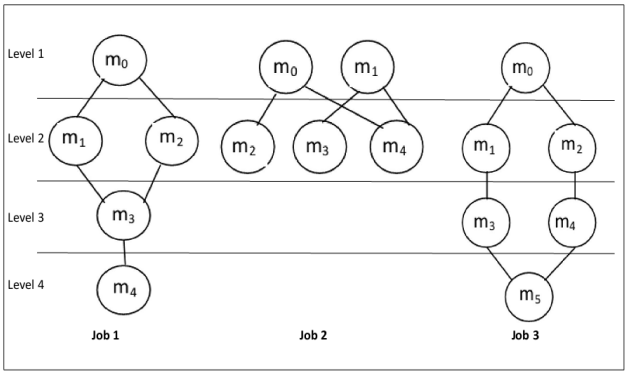

Fig. 3. Partitioning the batch into levels.

The BS scheduler schedules the job modules for the given level by allocating each module at a given level on a node using round robin strategy. The process starts from the first module being allocated on the node with minimum previous workload (T_k)and the following module allocated on the next best node till the last module of that level. Thus, BS enables us to allocate job modules of different jobs in parallel at the job level and as well at the sub job level. Allocating modules to nodes independently for each level ensures the proper load balancing while exploiting the parallelism in the batch to its maximum leading to the most efficient utilization of the computational resources.

IV. Experimental Study

Experiments were carried out to compare the behavior of the FCFS, TBSS and BS scheduling schemes by scheduling the batch of varying size and degree of parallelism using these schemes and analyzing them for the following cases:

Case I: Observing the TAT for the same batch of jobs with more than one job by varying the number of nodes constituting the system.

Case II: Observing the TAT by keeping the number of nodes same but varying the batch size.

Case III: Observing the TAT of the batch by varying the number of nodes and modules. Batch has only one job.

The algorithm for BS is presented in the box as follows.

```
BS
{
Submit the batch
                    // Submit the batch comprising of jobs
Group the batch
   // Group together the job modules as per their levels
For l = 1 to Lmax
  // Lmax being highest precedence level among all jobs
do
{
    For i= 1 to N //for number of jobs
do  {
          For i = 1 to M
// for the modules with precedence level 'l'
do  {
                  For k=1 to K
// for all the available nodes
do  {
                       Compute (CTijkL)
// Calculate CTijkL for the module under consideration /
        CTijkL =  Eijk. +Tk
          }
Select Node & Assign Module
 // Assign the module on the node with minimum CTijkL //
Thus generate the Allocation Vector

Mark the Selected Node
// Mark the selected node as used so that no further
   // allocation will be made on this node till all the
  // other nodes have also got allotted one or the other
 // module} }
Compute NECkL for the selected nodes
// NECkL is the sum of execution cost of assigned
modules and ready time at node Pk of level L

Update Tk = max (NECkL)

// Tkl for the next level is equal to the maximum value
 // of NECkl of the current level
    }
        Compute Turnaround Time (TAT)

         // TAT = max (NECkL) at the highest level
}
```

The various parameters used in the study are listed in Table I.

TABLE I: SYSTEM PARAMETERS

S. No.	Parameter	Notation Used	Range
1	Number of Nodes	P_K	5-30
2	Clock Frequency of Nodes	f_k(MHz)	10-20
3	Time to finish Previous Workload	T_k (μS)	20-100
4	Number of Jobs in the Batch	J_j	3-25
5	Number of Modules in a Job	M_{ij}	10-50
6	Number of Instructions in a Module	I_i	300-500
7	Number of Levels in the Jobs	L_i	1-5

In Case I, a batch of jobs was created along with a parallel processing system. The same batch is then scheduled using FCFS, TBSS and then BS strategy while varying the number of nodes available for execution and observing the TAT. Table II and Fig. 4 represents some of the results obtained. The experiment was done on different data sets resulting in jobs with reasonable parallelism at the sub-job level but the result is along the same lines as shown in Fig. 4.

Observations:

1) It is observed, from Fig. 4, that the TAT of the batch for FCFS, TBSS and BS gradually decreases with the increase in the number of nodes. This is because of the fact that with more nodes the chances of the exploitation of the software parallelism inherent in the job increases.
2) Further, it can be observed that if the number of nodes are very small, the difference in the TAT observed by the three policies is not much owing to the limited hardware parallelism which limits the software parallelism of the scheduling policy
3) Since, BS considers the parallelism in the batch at both the job and the sub-job level, it offers the best turnaround time consistently. FCFS reports the worst performance out of all policies which can be understood by the fact that it considers the parallelism in the job only at the job level. This results in losing the parallelism at the sub-job level. TBSS performs better than FCFS as it tries to allocate the sub-jobs to the best nodes resulting in exploitation of the parallelism at the sub-job level. However, the performance is not comparable to BS as TBSS cannot indulge jobs as well as the sub-jobs at the same time while scheduling the job.
4) It is seen that FCFS saturates fastest followed by TBSS and BS as FCFS exhibits least software parallelism than TBSS or BS. The fall in TAT observed in the case of BS is steepest as more nodes gives it an opportunity to allocate more modules which can be run in parallel at a given level. Thus, more is the match between software and hardware parallelism for BS, better is its performance.

TABLE II: OBSERVATIONS FOR CASE I

	No of Nodes	FCFS	TBSS	BS
Batch size 6 (Job size=15-12)	**3**	754.6875	747.5972	776.5625
	5	590.3333	674.4920	600.0243
	10	522.0909	419.1636	317.5101
	15	481.0833	393.9368	245.5833
	20	463.6429	355.6661	185.3053
	25	452.0000	343.8477	174.4746
	35	463.6429	323.7912	139.9097
	50	438.3571	318.2184	121.1632
	80	425.5000	309.7801	109.4132
	100	481.8571	313.7111	104.9507

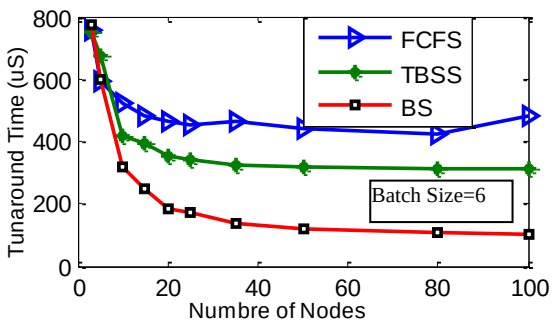

Fig. 4. Turnaround time v/s number of nodes.

Table III and Fig. 5 represents the observations for Case II in which the number of nodes in the system are kept constant and the TAT is observed by varying the batch size submitted for execution.

TABLE III: OBSERVATIONS FOR CASE II

	Batch size	FCFS	TBSS	BS
No. of Nodes=20 No. of Modules in a Job =15-10	**3**	514.9091	200.3005	140.6327
	5	532.6364	402.2040	210.5000
	7	572.7273	434.6892	216.7719
	12	595.0000	649.2888	319.6383
	15	572.4000	729.7893	403.3187
	23	605.9091	819.8765	549.0379
	25	712.3434	890.9876	634.9812

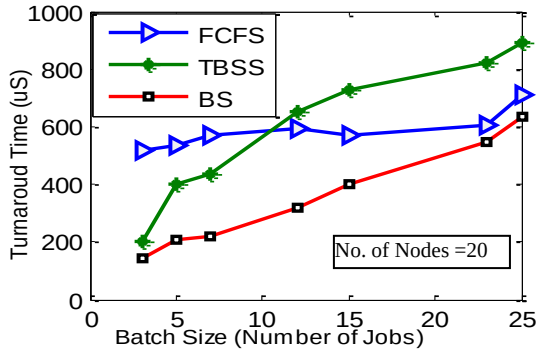

Fig. 5. Turnaround time v/s batch size.

Observations:

1) The TAT keeps on increasing when the batch size is increased while keeping the number of nodes fixed for all the scheduling schemes viz. FCFS, TBSS and BS which is expected in such case.
2) Under any given condition, BS performs the best followed by FCFS andTBSS. This is because of the capability of BS to allocate sub-jobs of independent jobs simultaneously resulting in the minimum TAT offered to the job.
3) TBSS performs better when the batch size is small. But the performance of TBSS gradually reduces as the batch size grows. This is due to the fact that when the number of nodes is limited, TBSS which primarily focuses on the batch parallelism at the sub-job levelcannot exploit the software parallelism at its best.
4) FCFS performs nearly the same as the batch size increases as it schedules the batch while focusing only on parallelism at the job level. Thus till the number of jobs is less or equal to the number of nodes the turnaround does not change substantially.
5) As the number of jobs increases, the difference in the TAT reported by FCFS and BS gradually reduces. This is due to the fact that for given number of nodes with increasing batch size, more and more number of nodes gets loaded with job. This results in effectively reducing the mismatch between hardware and software parallelism which otherwise is there in FCFS. Further, if the job exhibits very low parallelism, FCFS may even report better TAT than BS.

Table IV and Fig. 6 represents the results obtained from Case III. The study of Case III was divided into two parts referred to as case III(a) and Case III (b) respectively. In case III (a), the TAT offered by FCFS, TBSS and BS was observed while varying the number of nodes in the parallel computing system for the same batch size of a single job comprising of some modules. In Case III(b), the TAT is observed while varying the number of modules in the batch having a single job keeping the number of nodes same. The results are summarized as Tables IV-V and represented in Fig. 6-7.

TABLE IV: OBSERVATIONS FOR CASE III (A).

	No of Nodes	FCFS	TBSS	BS
Batch size = 1 (No. of Modules in the Job =20-25)	**5**	685.8000	238.8283	229.8616
	10	550.3500	164.2692	142.2192
	15	530.8421	137.7390	128.7164
	20	531.8421	126.1895	116.1724
	25	530.8421	120.2438	119.5965
	30	525.8235	123.3588	118.2838
	35	524.8235	113.3421	115.5559115.5559
	40	526.3500	111.3421	113.5471
	45	531.8421	116.5789	113.0079
	50	533.8421	117.4691	115.0079

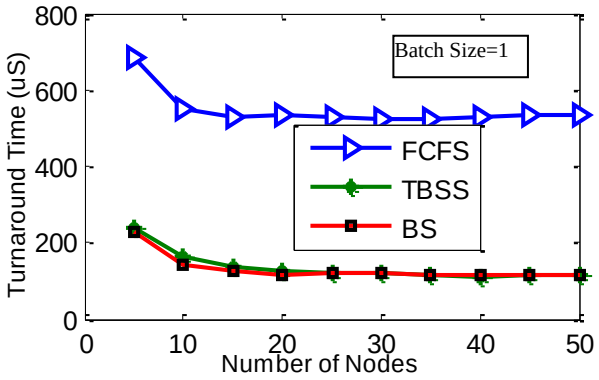

Fig. 6. Turnaround Time v/s Number of Nodes with Batch Size=1.

Observations:

1) The TAT keeps on reducing with the increase in the number of nodes. The fall is sharp for all the policies in the very beginning because of the mismatch between hardware and software parallelism getting reduced. Afterwards, the rate of improvement becomes low.
2) TBSS and BS performs best and almost at par. The reason for this could be attributed to the fact that BS strategy starts resembling TBSS when the batch has only one job thus both of them considering the job parallelism at the sub-job level only.
3) FCFS performs worst as increasing the number of nodes does not affect it much as it focuses on the parallelism at the job level only. Thus, an increase in the number of nodes makes the difference only at the early start to the job with incapability of the scheme in exploiting the parallelism at the sub-job level.

TABLE V: OBSERVATIONS FOR CASE III (B).

Batch size= 1 No of Nodes=15 (No. of Modules in the Job =20-25)	No of Modules	FCFS	TBSS	BS
	5	151.5000	83.3125	81.4853
	10	254.1765	110.0162	92.1353
	15	340.4444	113.8824	99.8140
	20	466.7222	126.2138	126.2138
	25	535	134.1032	139.2848
	30	726.8824	136.8281	129.5573
	35	819.0556	171.2769	168.1573
	45	928.9444	185.0259	183.9424

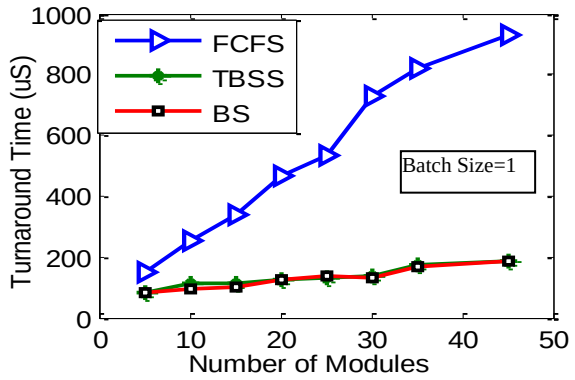

Fig. 7. Turnaround time v/s number of modules with batch size=1.

Observations:

1) The TAT offered by FCFS, TBSS and BS keeps on increasing with the increase in the number of modules for the batch of single job as expected.
2) In this case also BS and TBSS performs at par and emerge as best policies with FCFS reporting the worst results. Again, the reason of BS and TBSS performing better is the same as reported earlier which is that both consider parallelism at the sub-job level unlike FCFS. Therefore, when a single job is there in a batch both BS and TBSS behaves almost identical.
3) FCFS performs worst because an increase in the number of modules adds substantially to the workload as it is incapable of harnessing the parallelism at the sub-job level which could have been run in parallel easily as is the case with BS and TBSS.
4) The performance of BS gets a lead over TBSS with an increase in the degree of parallelism in the job internally as BS can allocate these sub-jobs much faster and in parallel as compared to TBSS.

V. CONCLUSION

Batch scheduling is a useful exercise when we have similar jobs to be executed as it results in an effective utilization of the resources. The work compared the performance of three scheduling schemes viz. First Come First Serve (FCFS), Turnaround Based Scheduling Scheme (TBSS)and Batch Scheduling (BS) for parallel computing system. FCFS considers the scheduling by considering parallelism at only the job level, TBSS considers it at the sub-job level and BS considers it at both the job and the sub-job level. The objective of the work was to observe the effect on the system's performance when there is a mismatch between hardware parallelism and software parallelism and the three scheduling policies mentioned above proves to be an ideal candidate for the study. Simulation study was performed to observe the TAT by varying the number of available nodes (hardware parallelism) while keeping the batch size same and varying the batch size and the degree of parallelism in the jobs in the batch (software parallelism) for the same number of nodes. The experiments were even extended to observe the effect of the scheduling schemes on the TAT when the batch contains only a single job. Simulation study reveals that the BS strategy performs best in almost all conditions as it provides the best match between mapping the software parallelism in the application to the available hardware parallelism.TBSS was observed to be performing well when either the batch size is small or the batch has a single job making it more suitable for smaller applications. FCFS performed worst as it considers parallelism only at the job level resulting in the inability to exploit the parallelism (if any) at the sub-job level.

REFERENCES

[1] D. P. Vidyarthi, B. K. Sarker, A. K. Tripathi, and L. T. Yang, *Scheduling in Distributed Computing Systems*, Springer, ISBN 978-0-387-74480-3, 2009.
[2] D. G. Fietelson and L. Rudolph, "Parallel job scheduling: issues and approaches," *Lecture Notes in Computer Science*, vol. 949/1995, pp. 1-18, 1995.
[3] E. Frachtenberg and U. Schwiegelshohn. New challenges of parallel job scheduling. [Online]. Available: http://www.cs.huji.ac.il/~etcs/pubs/papers/frachte.
[4] J. Weinberg. Job Scheduling on Parallel Systems. [Online]. Available: http://cseweb.ucsd.edu/users/j1weinberg/#weinberg06researchExam.
[5] E. Frachtenberg, D. G. Fietelson, F. Petrini, and Fernandez, "Adaptive parallel Job Scheduling with Flexible Co Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 11, 2005, pp. 1066-1077.
[6] Y. Zhang, C. Bai, and S. Wang, "Duplicating and its applications in batch scheduling, lecture notes in operations research," *The Fifth International Symposium on OR and Its Applications*, pp. 108-117, 2005.
[7] S. V. Sudha and K. Thanushkodi, "Process grain sized based scheduling of parallel jobs using genetic fuzzy systems," *International Journal of Computer Theory and Engineering*, vol. 2, no. 5, pp. 765-772, October 2010.
[8] H. Chen, Y. P. Zhang, and X. R. Yong, "On-line scheduling on a single bounded batch processing machine with restarts," *First International Workshop on Education Technology and Computer Science*, DOI 10.1109/ETCS.2009. vol. 196, pp. 868-871, 2009.
[9] N. Gans and G. V. Ryzin, "Optimal dynamic scheduling of a general class of parallel-processing queuing systems," *Advanced Applied Probability*, pp.1130-1156, 1998.
[10] I. Ashok and J. Zahorjan, "Scheduling a mixed interactive and batch workloadon a parallel, shared memory supercomputer," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 616-625, 1992.
[11] R. Zahid and V. D. Prakash, "A comparative study of FCFS and TBSS scheduling strategies for parallel computing system," *Second International Conference on Information and Multimedia Technology (ICIMT'10)*, Hong Kong, China, December, 2010.
[12] R. Zahid and S. Mohammad, "A batch scheduling strategy for computational grid," *Second International Conference on Meta Computing (ICoMeC'11)*, Goa, December 15-16, 2011, pp. 52-57.

Mohammad Shahid is a Ph. D. scholar at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi. He holds Masters degree in Computer Science and Applications from Aligarh Muslim University and M. Tech. Degree in Computer Science and Technology from JNU.

Zahid Raza is currently an Assistant Professor in the School of Computer and Systems Sciences,Jawaharlal Nehru University, India. He has a Master degree in Electronics, Masters degree in Computer Science and Ph.D. in Computer. Prior to joining JNU, he served as a Lecturer in Banasthali Vidyapith University, Rajasthan, India. His research interest is in the area of Grid Computing He is a member of IEEE.

Deo Prakash Vidyarthi received Master Degree in Computer Application from MMM Engineering College Gorakhpur and PhD in Computer Science from Jabalpur University (work done in Banaras Hindu University, Varanasi). He was associated with the Department of Computer Science of Banaras Hindu University, Varanasi for more than 12 years. Joined JNU in 2004 and currently working as Associate Professor in the School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi. Dr. Vidyarthi has published around 50 research papers in various peer reviewed International Journals and Transactions (including IEEE, Elsevier, Springer, World Scientific, IGI, Inderscience etc.) and around 25 research papers in the proceedings of peer-reviewed International conferences in India and abroad. He has authored a book (Research Monograph) entitled "Scheduling in Distributed Computing Systems: Design, Analysis and Models" published by Springer, USA released in December, 2008. Dr. Vidyarthi is also editor of a book on "Future internet Design" published by IGI-Global released in February, 2012. He has contributed chapters in many edited books. He is in the editorial board of two International Journals and in the reviewer's panel of many International Journals. Dr. Vidyarthi is member of IEEE, International Society of Research in Science and Technology (ISRST), USA and senior member of the International Association of Computer Science and Information Technology (IACSIT), Singapore. Research interest includes Parallel and Distributed System, Grid Computing, Mobile Computing.