The Journal of Systems and Software 108 (2015) 110–133

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Level based batch scheduling strategy with idle slot reduction under DAG constraints for computational grid

Mohammad Shahid*, Zahid Raza[1], Mohammad Sajid

School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi 110067, India

ARTICLE INFO

Article history:
Received 27 June 2014
Revised 3 June 2015
Accepted 5 June 2015
Available online 16 June 2015

Keywords:
Computational grid
Level based batch scheduler
DAG constraints

ABSTRACT

Scheduling in a grid environment optimizing the given objective parameters has been proven to be NP-complete. This work proposes a Level based Batch scheduling Strategy with Idle slot Reduction (LBSIR) while considering the inter module communication within the modules of the jobs represented using Direct Acyclic Graph (DAG) with the objective of optimizing the turnaround time and response time for a computational grid. The model works in two phases, allocation phase and idle slot reduction phase. Allocation phase begins by dividing the batch into a number of partitions as per the precedence level/depth level followed by the assignment of sub-jobs/modules from the partition to the best fit node in terms of the execution time offered for all the partitions. The idle slots generated during the allocation phase in each depth level are then reduced by inserting the best fit modules into these slots in the idle slot reduction phase after allocation of modules from higher depth level. Levelized allocation ensures minimizing the average response time being very useful for user interactive applications. An experimental study of the proposed strategy has been performed by comparing it with other similar methods having common objectives for evaluating its place in the middleware.

1. Introduction

High performance computing approaches as a successor of heterogeneous distributed computing provides the motivation of aggregation of underutilized resources by cooperative inter organization resource sharing resulting in grid computing. A computational grid is hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities. Grid is important in the sense that it can result in the utilization of the underutilized resources extending the benefits to both the consumers and the service providers in a cost effective way. Owing to the participatory nature of the grid there is an expected heterogeneity in its various components be it resources, applications, platforms, demand patterns to even the objectives for which the grid is realized and used. Further, the resources can join or leave the grid at any time making it very dynamic in nature. Due to this reason scheduling in grid becomes a very challenging job which becomes even more complex with the added concerns about the security (Foster and Kesselman, 2006; Zhu and Ni, 2013).

Scheduling in computational grid is very important due to the fact that inappropriate scheduling of tasks can fail to exploit the actual potential of the system and can counterbalance the benefits from parallelization due to excessive communication overhead or under-utilization of resources. Therefore, an effective job scheduling scheme becomes one of the core concerns in order to exploit the capabilities of resources in the grid that are underutilized in an efficient way. Scheduling for grid is different from scheduling on a uniprocessor system in which the major objective is to keep the CPU busy as much as possible. In the case of grid, it becomes the mapping of tasks to a selected group of computational machines from available multiple suitable administrative domains so that the inherent parallelism of the job(s) can be exploited while meeting the scheduling objective(s). The grid could serve the job by optimizing either one or multiple objectives varying from turnaround time, resource utilization, speed up, flow time, matching proximity, fault tolerant, reliability and security (Karatza, 2001; Vidyarthi et al., 2009; Xhafa and Abraham, 2010).

This work proposes a centralized Level based Batch scheduling Strategy with Idle slot Reduction (LBSIR) which has two phases named allocation phase and idle slot reduction phase. The model considers the jobs in the batch being represented as a Direct Acyclic Graph (DAG) divided into levels with the modules at each level capable of being executed in parallel. For a batch, modules of all the

* Corresponding author. Tel.: +91 9717786525.
E-mail addresses: mdshahid.cs@gmail.com (M. Shahid), zahidraza@mail.jnu.ac.in (Z. Raza), sajid.cst@gmail.com (M. Sajid).
[1] Tel.: +91 9818765936.

http://dx.doi.org/10.1016/j.jss.2015.06.016
0164-1212/© 2015 Elsevier Inc. All rights reserved.

jobs at a given level are then allocated to the suitable nodes executing simultaneously from the first level to the last with the objective of minimizing the turnaround time and the response time. The scheduler at first selects the modules of various jobs in the batch lying at the same level with assignment of modules being done level-wise starting from first level and being done one by one from the first job to the last one for each level. Allocation phase begins by dividing the batch into partitions as per the precedence level/depth level followed by sorting the partitions and assigning the sub-jobs/modules from all partitions to the best fit node in such a way that the job execution time gets minimized. After allocation of modules from each depth level is over, the idle slots generated are reduced by inserting the best fitting modules from the subsequent levels in the idle slot reduction phase as discussed in Section 4. Partition and allocation of whole batch into groups as per the depth levels ensures minimizing the average response time too as for the first and the following partitions the modules from all the jobs for a given level are scheduled simultaneously.

Rest of the paper is organized as follows. A brief of the work done in the domain has been presented in Section 2. The problem formulation has been explained in Section 3 including the scheduler as well as grid description, various assumptions related to them and optimization criteria. The proposed batch mode method based on idle slot reduction is presented in Section 4, discussing the allocation and idle slot reduction phases separately with algorithmic template followed by an illustration, design motivation for the proposed method and its time complexity. Section 5 presents the simulation study discussing various observations from the experiments for these strategies along with their evaluation by comparing with the models with similar nature and objective. Finally, the paper ends with the conclusions drawn for the work as Section 6.

2. Related work

Scheduling in grid environment faces a number of challenges due to the heterogeneous and dynamic nature of jobs as well as resources ranging from searching for appropriate resources in collections of geographically distributed heterogeneous computing systems to making scheduling decisions while meeting the optimization parameters and the quality of services (QoS) constraints. In addition, the authentication and authorization, secure and reliable file transfer, distributed storage management and resource scheduling across organizational boundaries are issues which make scheduling even more complex (Karatza, 2001; Vidyarthi et al., 2009; Xhafa and Abraham, 2010). Owing to the heterogeneous and dynamic environment with different scheduling objectives and constraints on tasks and resources, it is well established that the scheduling (mapping) problem for a grid is NP-complete (Garey and Johnson, 1990). Therefore, many heuristics for scheduling have been proposed to gain sub-optimal solutions to the problem. These heuristics can be either static with scheduling decisions being made prior to execution enabling the scheduler to make efficient scheduling decisions due to prior knowledge of the resources and application (Braun et al., 2001; Daoud and Kharma, 2008; Maheswaran et al., 1999b; Papazachos and Karatza, 2010), dynamic where scheduling decisions are made during the execution making it more challenging due to the dynamism involved in the resources as well as jobs (Maheswaran and Siegel, 1998; Maheswaran et al., 1999a; Wang et al., 2005), immediate mode in which the jobs are dispatched as soon as they are receive (Ruyan et al., 2009; Xhafa et al., 2007b) or the batch mode where first a group of jobs is formed followed by scheduling leading to collective scheduling decisions being made for better mapping of the jobs (Cai et al., 2002; Xhafa et al., 2007a; Leah et al., 2006; Ran and Han, 2009; Zhang et al., 2009).

Grid being a distributed system, many issues that are relevant to job scheduling in the distributed system becomes equally concerning. An overview of the heterogeneous distributed computing and related issues has been reported in Maheswaran et al. (1999b) with an insight into the various aspects of scheduling in Distributed Computing Systems with possibility of their use in any system requiring distributed processing being presented in Vidyarthi et al. (2009). A job scheduling scheme for a distributed system with heterogeneous processors with priorities for jobs has been presented in Karatza (2001). Here, the focus is on improving the overall system performance proposing probabilistic, deterministic and adaptive policies with the objective of optimizing the throughput, response time and waiting time of the system along with an analysis of their merits and demerits. Examination of the Gang scheduling strategies of jobs which are bags of independent gangs in a heterogeneous system for a prioritized job scheduling environment using migration has been presented in Papazachos and Karatza (2010). A very useful performance comparison of eleven static scheduling heuristics for independent tasks for heterogeneous distributed systems has been done in Braun et al. (2001) evaluating the appropriateness of each heuristic under different circumstances. A scheduling algorithm for static task scheduling employing prioritizing the tasks followed by the scheduling decision is presented in Daoud and Kharma (2008). A strategy for immediate and batch mode scheduling for independent tasks and observing their performance based on the consistency in the ETC matrix and arrival rate of the tasks is reported in Maheswaran et al. (1999a). A non-preemptive based remapping scheme for minimizing the overall execution time by using the run time values of the subtask completion time and availability time for heterogeneous systems is presented in Maheswaran and Siegel (1998). Owing to the NP complete nature of heterogeneous distributed systems and the large search space, meta heuristics suits well for optimization problems but the results and convergence depends greatly on the initial step. This problem is addressed in Izakian et al. (2009) with a pure heuristic approach.

Batch mode scheduling scheme is also very popular as it can result in an efficient matching of the software parallelism with the available hardware parallelism. The study (Xhafa et al., 2007a) considers the allocations of jobs to resources using batch mode methods for four parameters viz. make span, flow time, utilization and matching proximity using the benchmark model proposed by Braun et al. (2001) for different matrices. The work is also evaluated varying the consistency in computing and the heterogeneity of jobs and resources. Both Braun et al. (2001) and Xhafa et al. (2007a) have been used as the basis of experimentation and evaluation in this work. The work (Ruyan et al., 2009) presents a study on the online scheduling to minimize the make span on an unbounded parallel batch machine. Polynomial Time Approximation Schemes (PTAS) for bounded batch scheduling is presented with the objective to assign jobs to batches and sequence the batches so as to minimize the total completion time while jobs are executed non-preemptively (Cai et al., 2002) and also while considering the precedence constraints (Ran and Han, 2009). A proof that the greedy set cover algorithm gives a 4-approximation for perfect graphs and its improvement for a number of different classes of graphs has been reported in Leah et al. (2006). Workflow computations are a major programming paradigm for scientific applications. A scheduling scheme has been proposed in Zhang et al. (2009) that aggregates a workflow scheme and a batch queue to acquire resources for each sub component with an approach to reduce the wait time. Most of the work reported schedules the batch while considering only a few parameters leaving scope for the problem to be reconsidered using other appropriate parameters (Braun et al., 2001; Cai et al., 2002; Ran and Han, 2009; Ruyan et al., 2009) while in some other places in the literature mapping done to optimize even more than one parameter has also been reported (Xhafa et al., 2007a,b). Some heuristics where whole task represented as DAG is partitioned into groups according to depth level and then scheduling done level by level with a focus to find the optimal schedule at each depth level to eventually achieve an optimized make-span of whole job or batch have also been reported (Iverson et al., 1995; Saovapakhiran et al., 2011; Shahid and Raza, 2014). The strategy (Shahid and Raza, 2014) results in the generation of idle slots in the process which leaves the processor unutilized. On other hand the scheme suggested in Saovapakhiran et al. (2011) does not consider the communication cost between the modules.

A survey on static scheduling represented by a directed task graph on a multiprocessor system to minimize the completion time is shown in Kwok and Ahmad (1999). The intention of this survey is to explain different scheduling algorithms and also to observe their merits in terms of performance as well as time-complexity. The complexity of the problem enhances when task scheduling is to be made in a heterogeneous environment, where the processors are unlike and have different execution time for the similar task. List scheduling heuristics for DAGs are also reported for generating good quality schedules at a reasonable cost. List scheduling algorithms begin with the initialization of list (thus the name) of task and nodes by first assigning some priorities to each node and task. Some important list scheduling algorithms available in the literature include Dynamic Level Scheduling (DLS) (Sih and Lee, 1993), Earliest Time First (ETF) (Hwang et al., 1989), Heterogeneous Earliest Finish Time (HEFT) (Topcuoglu et al., 2002), Critical path on Processor (CPOP) (Topcuoglu et al., 2002) and Levelized Min time (LMT) (Iverson et al., 1995). LMT heuristic is based on depth level assignment of modules in a group on the node having minimum completion time for heterogeneous computing environment (Iverson et al., 1995). HEFT (Hwang et al., 1989) works in three phases. In the task prioritizing phase, it uses rank (upward or downward) to generate a topological sorting for dependent tasks. In the task selection phase, the algorithm selects the task with highest rank calculated from the ready list. In the processor selection phase, it selects the machine that allows the Earliest Finish Time of the selected task. However, the HEFT algorithm uses an insertion policy that tries to insert a task in an earliest idle time between two already scheduled tasks on a processor if the slot is large enough to accommodate the task. The CPOP algorithm is similar to HEFT. In the task prioritizing phase, CPOP uses the sum of upward rank and downward rank to give a priority to each task in the DAG. Here, tasks are categorized into critical path tasks and non-critical path tasks. The CPOP algorithm defines a Critical Path (CP) processor as the processor that minimizes the overall execution time of the critical path assuming all the critical path nodes are mapped onto it. In the task selection phase, the CPOP algorithm selects the task with highest priority from the ready tasks. In the next phase which is the processor selection phase, if the selected task is a CP task, it is mapped to a CP processor otherwise it is mapped to the processor that minimizes its earliest finish time. In DLS, pair with maximizing dynamic level which is the difference of module rank and earliest start time of module on that node for each step is selected. Here, computational cost of a module is assumed to be the median of the computational cost on all nodes.

In another work, a scheduling heuristic for DAG is presented to reduce the different number of possible scheduling variations emerging in a series of independent tasks (Sakellariou and Zhao, 2004). A scheduling algorithm known as Longest Dynamic Critical Path algorithm for heterogeneous distributed computing systems with a bounded number of processors has been proposed in Daoud and Kharma (2008). This algorithm uses different features to select tasks that enables algorithm to efficiently schedule selected tasks in a heterogeneous computing environment. Authors in the work (Yang et al., 2011) proposed a scheduling algorithm based on the Bayesian Optimization Algorithm (BOA) for heterogeneous computing environments. BOA initializes a Bayesian networks and learns all the dependencies between tasks in the network according to the task graph of multiprocessor scheduling problems to choose few best possible combinations of the task schedule. BOA, then executes these selected sequences of tasks on the processor by the heuristic based priority as used in list scheduling approach. The work in Bajaj and Agrawal (2004) presents a task duplication based scheduling algorithm for heterogeneous network systems which provided exceptional solutions for task scheduling applications with complexity $O(V^2)$. In the work (Arabnejad and Barbosa, 2014), a Predict Earliest Finish Time (PEFT) for heterogeneous computing systems is proposed. The algorithm has the same time complexity as other algorithm i.e. $O\ (v^2p)$ for v tasks and p processors. However, addition of look-ahead concept improves the algorithm in terms of the make-span.

A new list based scheduling algorithm has been reported in Shi and Dongarra (2006) for handling the scheduling of workflow applications represented by weighted directed acyclic graphs. This algorithm has two phases, in the first phase, the algorithm evaluates the effect of the percentage of capable processors when the task node weights are being assigned. Later, the algorithm regulates the efficient earliest finish time policy by including the average communication cost between the current scheduling node and its children throughout the processor selection phase. Direct hyper-graph is used in Deveci et al. (2015), for representation of maximum volume and the number of messages sent and received by a processor. Authors introduced a multi objective, multi level hyper-graph partitioner which takes into account a variety of prioritized communication metrics and optimizes all of them together in a one-phase approach. An extremely good quality survey of the Pegasus system is provided in paper (Deelman et al., 2014) demonstrating its capabilities that have been developed during the last decade to fulfill the need of different computing applications. The design, development and evolution of the Pegasus workflow management system was illustrated to map with the workflow descriptions on distributed computing infrastructures. A resource efficient workflow scheduling problem has been addressed via Maximum Effective Reduction (MER) algorithm (Lee et al., 2015). The major contribution of MER is to recognize the optimal trade off position among increasing makespan and to decrease the resource usage. Discovering such optimal point is of great practical importance that leads to improvement in resource utilization, reductions in resource provisioning and saving energy consumption. In the Cloud environment the workflow scheduling is a challenging problem. The work in Abrishami et al. (2013) adapts the Partial Critical Paths algorithm to propose two special polynomial time complexity scheduling algorithms with one-phase algorithm called IaaS Cloud Partial Critical Paths and a two-phase algorithm referred as IaaS Cloud Partial Critical Paths with Deadline Distribution.

Since last decades the research community has been interested in matching and scheduling algorithms to allow multiple DAG structured applications to allocate resources of the network in the large scale heterogeneous environments. The paper (Iverson and Ozguner, 1998) proposed a matching and scheduling structure where several applications raced for computational resources on the network. The motive of Zhao and Sakellariou (2006) is to optimize the makespan as well as to attain fairness due to the delay at each DAG because of the wait incurred between competing resources. The work (Zhu et al., 2007) proposes a Serve On Time (SoT) algorithm that combines all dynamic multi tasks in such a way that all the tasks will obtain the rights to be served as soon as possible. In paper (Ren and Yu, 2012) the authors proposed a multi-priority multi-DAG mixed scheduling algorithm based on minimizing the data transmission time and task completion time to deal with the problem in multiple DAGs workflow having the same priority. Grid scheduling approach with Deadline Driven coordinator is reported in Melnyk (2013) with the idea being to minimize the total completion time while meeting the deadline constraints.

LBSIR considers both the parallelism at the batch level and the sub-job/module level in terms of the availability of the computational resources. The strategy is different from the available strategies in the sense that it considers many attributes of the jobs and the resources for scheduling like node speed, existing workload on the nodes, communication requirements among the interactive modules in the jobs extended even to the batch level giving a broader scope to the scheduling decisions. Many available models in the literature focus on scheduling the entire job to the suitable node (Xhafa et al., 2007a). While saving on the communication cost, these approaches result in a serial execution of the job modules which otherwise could have executed in parallel thus wasting the inherent job parallelism. If the same situation is considered for a batch of jobs it results in a substantial negligence of the module level parallelism as these modules across the jobs in the batch can be run in parallel.

The DAG scheduling models reported in the literature considering data transfer within modules suffers from communication delay in case of workflow having sufficient amount of inter module communications. The proposed work considers the communication cost of various modules in the batch on a node simultaneously after each depth thereby efficiently minimizing the effect of the communication costs at each level. This results in a significant gain for the entire batch. Since the size of the modules of the various jobs at each depth level is different, idle slots are generated in the process. The second phase of the model then tries to minimize them using insertion of modules from the subsequent levels thus further improving the performance gain. The process and motivation have been explained in detail in Section 4.

All the methods proposed in the literature have different strong points, yet, none of them can be rated the best under all situations for all the parameters. This is due to the nature of the problem i.e. if the communication overhead is removed as in Xhafa et al. (2007a), we lose the parallelism present in the jobs. If parallelism within the job is exploited some data transfer within modules takes place as reported earlier. Exploitation of the inherent parallelism in the job cannot be done without any cost i.e. some communication overhead will have to be tolerated. This work makes an attempt to establish a trade-off between communication overhead and exploitation of parallelism within the jobs in the batch as the proposed strategy has an intention of exploitation of parallelism to a greater extent with as possible lower communication cost.

3. Problem formulation

The scheduling strategy being proposed in the work schedules a batch of jobs in the grid environment while meeting the objective of minimizing the job turnaround time and the response time. This section presents an insight into the various components related to the work e.g. the notation used, grid and scheduler description, batch/job description and various data structures used along with the problem statement and the parameter estimation.

3.1. Notations used

To describe the various models and parameters involved in the process, the various notations used in the study are as under

- $\boldsymbol{K}$**:** Number of computational nodes in the grid
- $\boldsymbol{N}$**:** Number of jobs in the batch
- $\boldsymbol{N_k}$**:** k^{th} computational node
- $\boldsymbol{J_i}$: Job identifier
- $\boldsymbol{m_i}$**:** Number of modules in the job i
- $\boldsymbol{m}$**:** Average number of modules in the jobs
- $\boldsymbol{M_l}$**:** Number of modules in the partition ρ^l
- $\boldsymbol{M}$**:** Average number of modules in the partitions
- $\boldsymbol{m_{ij}}$**:** Module j of job i
- $\boldsymbol{\rho^l}$**:** Partition of lth depth level from the batch which is the set of all modules from the batch lying at depth level l i.e. $\rho^l = \{m_{ij}: \forall i \in N, j \in M_i \ \&\ \text{depth\_level}\,(m_{ij}) = l\}$
- $\boldsymbol{m^l{}_{ij}}$**:** Module j of job lying at depth level l or $m_{ij} \in \rho^l$
- $\boldsymbol{l_i}$**:** Highest level of precedence in DAG of job J_i
- $\boldsymbol{L}$**:** Highest level of precedence of batch which is computed as $L = \max\,(l_i, \forall i \in N)$
- $\mathbf{RT}_{\boldsymbol{k}}^{1}$**:** Previous workload already assigned to node N_k at beginning of allocation
- $\boldsymbol{D_{kr}}$**:** Inter node distance between the nodes N_k and N_r
- $\mathbf{IMC_{ipq}}$**:** Inter module communication between the modules m_{ip} and m_{iq}
- $\boldsymbol{E_{ijk}}$**:** Processing time of a module m_{ij} of job J_j on node N_k
- $\mathbf{IS}^{\boldsymbol{l}}$**:** Idle slot list of depth level l. This list contains idle slots on all computational machines of size w after execution of all modules from ρ^l i.e. $\text{IS}^l = \{S_k: w_k, \forall N_k\}$.
- $\boldsymbol{S_k}$: Idle slot on the node N_k
- $\mathbf{AFT}^{\boldsymbol{l}}{}_{\boldsymbol{ij}}$**:** Completion time of module $m_{ij} \in \rho^l$.
- $\boldsymbol{M_{ij-v}^{l-s}}$: Preceding module of $m_{i\,j}$ in the job J_i, at previous depth level $l-s$, where $v = 1, 2, \ldots, j-1 \ \&\ s = 1, 2, \ldots, l-1$.
- $\boldsymbol{PM_{ij}^{l}}$**:** Set of all preceding modules lying at previous levels $(l-s)$ of all modules $m_{i\,j}$ of depth level l, $PM_{ij}^{l} = \{M_{ij-v}^{l-s}: : \forall i \in N, j \in M_i, v = 1, 2, \ldots, j-1 \ \&\ s = 1, 2, \ldots, l-1\}$
- $\mathbf{ST}_{\boldsymbol{ij}}^{\boldsymbol{l}}$**:** Start time of module $m_{ij} \in \rho^l$
- $\mathbf{CC}_{\mathbf{ipqkr}}^{\mathbf{l-s,\,l}}$**:** Communication cost of pth and qth modules m_{ip} and m_{iq} lying at level $l-s$ and l assigned to nodes N_k and r respectively
- $\overline{C}$: Average communication cost of all modules m_{ij} of batch.
- $\text{CC}_{\boldsymbol{k}}^{\boldsymbol{l}}$**:** overall communication overhead on node N_k at depth level l.
- $\mathbf{CC_{LBSIR}}$: Average communication cost share in allocation of batch by LBSIR variants
- $\mathbf{CC_{HEFT}}$: Average communication cost share in allocation of batch by HEFT
- $\mathbf{ET}_{\boldsymbol{k}}^{\boldsymbol{l}}$**:** Execution time of assigned modules of level l at node N_k
- $\mathbf{TCT}_{\boldsymbol{k}}^{\boldsymbol{l}}$**:** The Total Completion Time for node N_k in a given level l.
- $(\mathbf{ET}_{\boldsymbol{k}}^{\boldsymbol{l}})'$**:** Execution time of inserted modules on idle slot $S_k \in \text{IS}^l$ at depth level l at node N_k
- $(\mathbf{TCT}_{\boldsymbol{k}}^{\boldsymbol{l}})'$**:** Updated Total Completion Time for node N_k after insertion in a given level l.
- $\boldsymbol{\mu}$**:** Service Rate of queuing system at global queue
- $\boldsymbol{\lambda}$**:** Arrival rateon queuing system at global queue
- $\boldsymbol{B}$**:** B is batch size
- $\boldsymbol{Q_u}$**:** Q_u is the queue unit that servicing node can handle as a unit.
- **TAT:** Turnaround Time for the batch submitted to grid
- $\boldsymbol{U_s}$**:** Average utilization of the grid system
- **FT:** Flow Time of the batch submitted
- **RT:** Average Response Time of the batch

3.2. The grid and scheduler descriptions

The physical architecture of the considered computational grid is multi domain. The computational resources i.e. machines/nodes contributed from these domains makes the grid a huge collection of heterogeneous resources. A domain can be seen as a collection of computers at various sites and its own administrative and security policy. A site is resembled by a stand-alone supercomputer or could be a cluster of workstations controlled by management software. The resources from every domain are interconnected using interconnection network e.g. a High speed WANs such as the Internet or a dedicated private connection link. The prediction of communication performance is difficult in shared WANs which provide varying performances (Zhu and Ni, 2013). In the grid system, as soon as the jobs are submitted at any site in the domains, these are batched on a single machine known as the dispatch and evaluation node. All scheduling decision of the batch like allocation, keeping track of the available

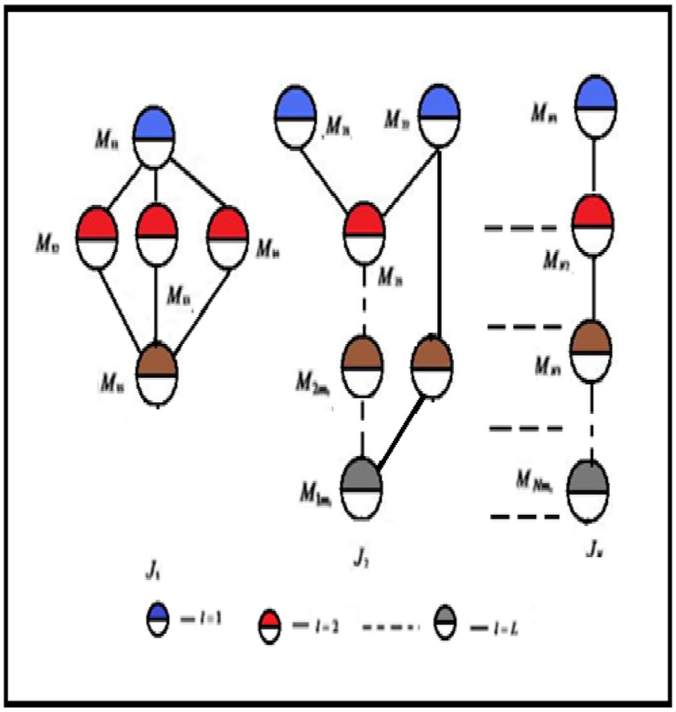
Fig. 1. Typical batch with levelized JPDG.

grid resources, their present state and the status of the job execution from time to time are done by Grid Scheduler (GS) loaded at this node. GS, being level based, used in the centralized batch mode, a global queue is maintained at the dispatching node.

3.3. Job model

A Batch of N independent jobs, $\beta = \{J_i: i = 1, 2, \dots, N\}$, is considered for execution in which each job is represented as Direct Acyclic Graph (DAG). In the batch, each job can be divided into various sub-jobs (modules) having precedence level in accordance with depth level and may require some inter module communication with modules of the same job at earlier depth levels. A typical depiction of such scenario is presented in Fig. 1 where individual modules in the jobs of the batch are divided in a number of depth levels depending on their order of precedence in the DAG and their communication requirements with the other modules in the job. Various jobs in the batch may be of varying depth levels. Modules of the jobs having same depth level are shown in the same color e.g. modules $M_{11}, M_{21}, M_{22}, \dots, M_{N1}$ having precedence level 1 while $M_{12}, M_{13}, M_{14}, M_{23}, \dots, M_{N2}$ have precedence level 2 and are represented by light blue and red colors respectively. Therefore, as can be seen in Fig. 1, M_{15} is seen to be dependent on M_{12}, M_{13}, M_{14} to start its execution whereas the modules/sub-jobs at the same depth level within the batch e.g. $M_{12}, M_{13}, M_{14}, M_{23}, \dots, M_{N2}$ may execute in parallel. It is assumed that the job pre-processing (partitioning) is done prior to scheduling.

The degree of communication depends on two parameters viz. inter module communication (i.e. byte exchanges) and inter node distance between computational machines. The computing method for visualizing the communication cost $\mathrm{CC}_{\mathrm{ipqkr}}^{l-s,l}$ between two modules of the same job is presented in Fig. 2 (Shahid and Raza, 2014). It considers two modules M_{ip} and M_{iq} of job the J_i assigned on node N_k and N_r respectively. M_{iq} is assumed to be dependent on M_{ip} with their inter module communication requirement represented as $\mathrm{IMC}_{\mathrm{ipq}}$.

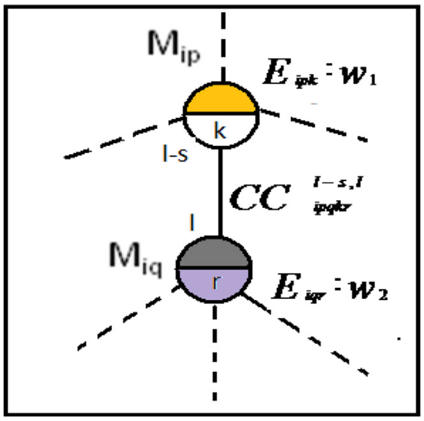

Fig. 2. Sample of communication cost computation.

The communication cost of modules $M_{\mathrm{iq}} \in \rho^l$ and $M_{\mathrm{ip}} \in \rho^{l-s}$ assigned on nodes N_k and N_r respectively is represented by $\mathrm{CC}_{\mathrm{ipqkr}}^{l-s,\,l}$ and can be computed as (Shahid and Raza, 2014)

$$\mathrm{CC}_{\mathrm{ipqkr}}^{l-s,l} = w\left(\mathrm{IMC}_{\mathrm{ipq}} * D_{kr}\right) \tag{1}$$

where D_{kr} is inter node distance between the nodes N_k and N_r, $i = 1, 2, \dots, N$, $s \in Z^{+}$, $s \geq 0$, $1 \leq l \leq L$, $p, q = 1, 2, \dots, m_i$ and w is the proportionality constant as $\mathrm{CC}_{\mathrm{ipqkr}}^{l-s,\,l}$ is directly proportional to the $\mathrm{IMC}_{\mathrm{ipq}}$ and D_{kr}. If the relation between $\mathrm{CC}_{\mathrm{ipqkr}}^{l-s,\,l}$ and $\mathrm{IMC}_{\mathrm{ipq}}$, D_{kr} is liner, the value of w will be taken one.

3.4. Computational machine model

The computational grid under consideration has a set of K computational machines/nodes $R = \{N_k, \forall i \in K)$ with the following characteristics:

- Number of heterogeneous computational machines (K) to participate in the mapping of batch of the independent jobs.
- Machines/nodes are well suited only for compute intensive jobs.
- Computing capacity (C_k: $k = 1, 2, \dots, K$) of each machine (in MIPS).
- Initial Ready time (RT_k^1) which is the amount of time after that machine N_k will have finished the previously assigned modules. This parameter measures the previous workload of the machine.
- Inter node distances (D_{kr}) among the nodes in the grid. D_{kr} actually measures the distance between nodes N_k and N_r by finding the number of hop counts between them.
- The Expected Time to Compute matrix E ($N \times m_i \times K$), in which the component E_{ijk} is the expected execution time of module m_{ij} of job J_i on machine N_k.

3.5. Problem statement and the parameter estimation

The problem is finding the mapping M for batch $\beta = \{J_i$: $\forall i \in N\}$ submitted for execution on the computational node set $R = \{N_k$: $\forall i \in K)$ from the grid system in such a way that schedule (S) generated for the considered batch of the jobs by this M is optimized in terms of the optimizing criteria:

$$M: \ \beta \rightarrow \ R \tag{2}$$

Optimize P (S), where S is the schedule produced by mapping M and P is the optimization criterion as discussed next subject to the conditions listed below:

(i) Each module is assigned on only a single computational node.
(ii) Each computational machine may be assigned one or more modules of the batch.

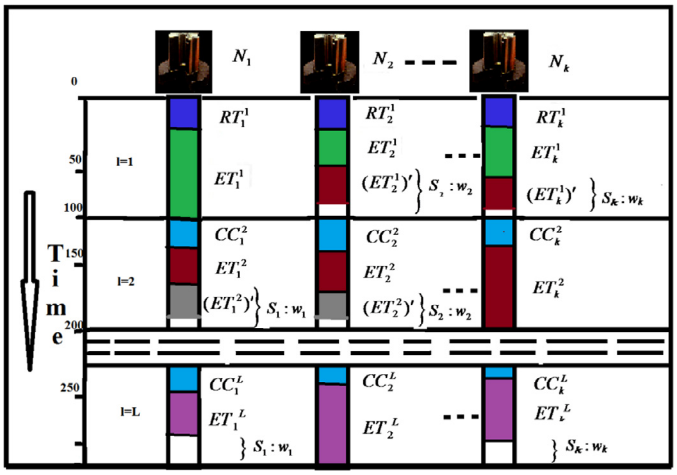

Fig. 3. A sample level-wise execution of jobs with idle slot reduction in the batch. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

(iii) Each module preserves the precedence constraints as appearing in the jobs represented by DAG.

Allocation begins by partitioning all modules of the batch submitted for execution in the partitions (ρ^l: $1 \le l \le L$) in accordance with depth level which can be executed in parallel. This way, modules get assigned level wise from each job of the batch resulting in an efficient exploitation of the software parallelism available in the batch. As well, level wise allocation ensures preservation of the dependence constraints. Afterward, for each level, modules are considered one by one for allocation, on the nodes deemed best for the execution of that module. After allocation of each partition, the idle slots $S_k \in \text{IS}^l$ at that depth level are removed/reduced by using idle slot reduction scheme by using methods described in Section 3. A sample level-wise execution of batch and idle slot reduction in a typical batch has been illustrated in Fig. 3.

As described earlier, the allocation is done level-wise starting from the first level till the last level L observed in the batch. The execution time (ET_k^l) of modules $m_{ij} \in \rho^l$ belonging to the same partitions on the nodes are represented with the same color. For example modules from partitions ρ^1, ρ^2, and ρ^3... are represented by green, dark red, gray and so on respectively. Similarly, communication cost (CC_k^l) of modules $m_{ij} \in \rho^l$ with the modules $m_{in} \in \rho^{l-s}$, ready time (RT_k) and idle slot $S_k \in \text{IS}^l$ on the node N_k are shown by using sky blue, blue and white color respectively. Idle slots S_k are shown by middle brackets indicating their size w_k and some of the idle slots are allocated to the modules from next partitions if they fit well. For example, modules $m_{ij} \in \rho^2$ are assigned on the slot S_2 and S_k of size w_2 and w_k respectively shown by dark red color in the level 1 with the execution time $(\text{ET}_2^1)'$ and $(\text{ET}_k^1)'$respectively. The remaining small portion in white color indicates the residual idle slot reduced in size as major portions of S_2 and S_k are assigned to the modules $m_{ij} \in \rho^2$. These idle slots can be fully removed if we have the modules fully fitting into them from the next partitions which can happen only rarely especially in the most heterogeneous environment like grid. The idle slots in the last level are bigger in the size as they cannot be reduced in absence of any successor.

Execution time for the assigned modules $m_{ij} \in \rho^l$ on node N_k is represented as ET_k^l and can be computed as the sum of the expected time to compute value (E_{ijk}) for these modules. The value of ET_k^l can be written as

$$\text{ET}_k^l = \sum_{N_k \leftarrow \forall\, m_{ij} \in \rho^l} E_{ijk} \tag{3}$$

The net communication cost CC_k^l on node N_k at level l can be estimated as the maximum of the communication cost of modules $m_{\text{iq}} \in \rho^l$ assigned on node N_k with the preceding modules $m_{\text{ip}} \in \rho^{l-s}$ assigned in the previous level on N_r (any node from the resource set) and can be written as

$$\text{CC}_k^l = \max_{m_{\text{iq}} \in \rho^l,\ m_{\text{ip}} \in \rho^{l-s} \&\ N_r \in R} \left(\text{CC}_{\text{ipqkr}}^{l-s,\ l}\right) \tag{4}$$

where $1 \le i \le N$, $1 \le s \le l-1$, $1 \le l \le L$ and p, q & k, r are module and computational node identifiers respectively. For the first level, CC_k^l will be zero due to the fact that no preceding modules exist for them. The total completion time (TCT_k^l) of modules $m_{ij} \in \rho^l$ assigned on N_k is the actual finishing time of the considered modules and can be estimated as

$$\text{TCT}_k^l = \begin{cases} \text{RT}_k^l + \text{ET}_k^l & \text{for } l = 1 \\ \text{CC}_k^l + \text{ET}_k^l & \text{otherwise} \end{cases} \tag{5}$$

RT_k^1 is the ready time of node N_k for the first depth level which is the workload assigned to node N_k prior to the allocation of the current batch. As mentioned earlier, after allocation of modules belonging to each partitions ρ^l some idle slots are left on some of the nodes. The idle slot list, $\text{IS}^l = \{S_k\text{: } w_k, \forall N_k\}$ for depth level l is maintained by computing the size of each idle slot on the respective computational node as

$$\underset{S_k \in \text{IS}_k^l}{S_k} = \max_{\forall\, k} \left(\text{TCT}_k^l\right) - \text{TCT}_k^l \tag{6}$$

Further, the idle slots $S_k \in \text{IS}_k^l$ on nodes are reduced (removed if possible) by inserting the modules $m_{ij} \in \rho^{l+1}$ fitting in them as per the algorithm suggested in Section 3. TCT_k^l on the nodes in which additional modules from next level are assigned are updated with the execution time of the inserted modules. The updated total completion time on node k $(\text{TCT}_k^l)'$ can be written as

$$\left(\text{TCT}_k^l\right)' = \begin{cases} \text{TCT}_k^l + \left(\text{ET}_k^l\right)' & \text{otherwise} \\ \text{TCT}_k^l & l = L\ \&\ \widehat{N}_k \end{cases} \tag{7}$$

where $(\text{ET}_k^l)'$ is the execution time of the inserted modules on idle slots $S_k \in \text{IS}_k^l$, $0 \le (\text{ET}_k^l)' \le S_k$ and $\widehat{N}_k$ is the computational node having non-reducible idle slots which is not of proper size indicating that modules from next partitions cannot be fitted into it or there is no modules in the next level as in case of last level with $l = L$.

It is always desired that the resultant allocation should ensure the optimum values of the desired QoS parameters for any given scheduling scheme; TAT and response time being two such parameters. Turnaround Time (TAT) for the batch is estimated as the time period between submission of the batch on the grid system and its completion. TAT is estimated as the sum of the queuing time and the overall completion time for the batch and can be written as

$$\text{TAT} = \text{QT} + \sum_{l=1}^{L} \max_{\forall\, k} \left(\text{TCT}_k^l\right)' \tag{8}$$

where QT is the Queuing Time which is estimated as the average waiting time of the batch in the global queue as

$$\text{QT} = \begin{cases} \frac{1}{\mu-\lambda} & \text{if } B \le Q_u \\ \frac{1}{\mu-\lambda}\left(\frac{B}{Q_u}\right) & \text{if } B \ge Q_u \end{cases} \tag{9}$$

where B is batch size & Q_u is the queue unit that the servicing node can handle as a unit. The average utilization (U_s) of the grid system and the average flow time (FT) for the batch for a given allocation can be computed as

$$U_s = \frac{\sum_{l=1}^{L}\sum_{k=1}^{K}\left(\mathrm{TCT}_k^l\right)'}{K * \max_{\forall k}\left(\mathrm{TCT}_k^l\right)'} \tag{10}$$

$$\mathrm{FT} = \sum_{l=1}^{L}\sum_{k=1}^{K}\left(\mathrm{TCT}_k^l\right)' \tag{11}$$

Response time is one of the important parameters for the user interactive jobs. It is estimated as the time period between the job submission and appearing of the first response. The average response time for the batch can be taken as the average start time of the jobs in the batch and computed as

$$\mathrm{RT} = \frac{1}{N}\left(\sum_{i=1}^{i=N}\mathrm{ST}_{i1}^1\right) \tag{12}$$

4. Proposed method

This section presents the Level Based Batch Scheduling Strategy based on Idle Slot Reduction (LBSIR) while analyzing its various features. The strategy works on the basis of first allocating the job modules of the batch as per levels and then trying to reduce the idle slots generated in the process to improve the turnaround time in the two phases viz. allocation phase and idle slot reduction phase.

4.1. Allocation phase

Prior to scheduling, the whole batch is partitioned into groups called partitions according to the depth level. In the allocation phase, these modules from each partition are then assigned one by one onto the computational nodes. The best fit node selection for allocation of each module is approximately similar to static level based batch scheduling strategy (SLBBS) (Shahid and Raza, 2014) but the module selection in each step is different. In SLBBS modules in each partition are selected in the order as

- Modules from job J_{N-1} are assigned earlier than that of J_N.
- In the case of modules from the same job, the modules $m_{i\,j-v}^l$ will be assigned before modules m_{ij}^l.

In the case of LBSIR, in each partition the largest or smallest modules available in the partitions in terms of their size are selected and are allocated to the computational machines. Allocation is accomplished with the following considerations:

- Modules from each partition are sorted as per their size i.e. increasing/decreasing.
- Modules are then selected one by one in the order generated by sorting.
- Allotment is done on the best fit node which offers the least aggregate time as suggested in algorithmic template at step 10. Accordingly, for the first level the contribution from the communication cost component is zero as it has no preceding modules.
- The values of ET_k^l at each allocation is updated.
- Allocated module is removed from the pending list after each allocation.
- Finally, values of CC_k^l and TCT_k^l are computed after allocation of each partition.

An approximation for the size of modules from ETC matrices generated randomly by using ETC simulation benchmark is taken as the average of the expected time to compute values on different computational machines. It is expected that a larger value of this average execution time on different machines reflect the module with larger actual size and vice versa even in the highly heterogeneous and inconsistent environment. As soon as the size of modules from each partition ρ^l is computed, the sorting is done in the desired descending or ascending order followed by the allocation.

For the partition sorted in the descending order, each time largest available module in the partition is selected and assigned on the best fit node as per step 8 in the algorithmic template. In this case the strategy is referred to as the Level based Batch scheduling Strategy based on Idle slot Reduction with Largest Module Selection (LBSIR-LMS). The assignment of a larger job module of DAG on the best fit node (available as well as fastest) result in maximizing the utilization of the grid resources for the batch. This scheme is neither pure LJFR nor Max Min as LJFR assigns its job with maximum workload to the resource (idle) having highest computing capacity and Max Min tries to map the entire job with latest completion time on the machine having earliest finish time. LBSIR-LMS begins by assigning the modules level wise starting from the first level, second level and so on. For the first depth level it works as a hybrid of LJFR and Max Min as modules with maximum workloads are mapped to the machine with earliest finish time. For the remaining depth level, allocations are made with minor changes with largest modules being mapped to the machines having minimum aggregate time for the selected module as suggested in the template at step 10. Therefore, allocation is made based on two parameters named least execution time (ETC value) for the selected modules and earliest availability of the machine/node for that module. The allocation process is repeated for every module as well as partition until all modules from the batch get assigned.

In contrast to LBSIR-LMS, partitions sorted in the ascending order will result in the assignment of smaller modules to the best fit node referred as LBSIR–SMS and is expected to minimize the turnaround time, flow time and the average response time for the batch. This scheme is appropriate when most of the job modules in any partition are long ones and very few are small. If modules with lighter workload are more and larger one are only a few in any partition, largest modules should be assigned first and followed by the reaming modules for better performance. This situation motivates toward another approach at each depth level that some larger modules should be assigned first and the remaining modules getting assigned later combining the best features from the both schemes discussed above referred as LBSIR (LMS-SMS). This requires the partitions to be sorted both in descending as well as ascending order and allocation may be made alternatively from these lists in the desired fashion.

4.2. Idle slot reduction by insertion phase

In this phase, a scheme for reducing the idle slots resulting from the level wise allocation in the allocation phase is presented. The idle slots on computational node in any given depth level is reduced by inserting the modules of size best fitting in them as shown in Fig. 3. In the idle slot reduction phase, the following points are considered:

- Find idle slot list $\mathrm{IS}^l = \{S_k: w_k\}$ for the depth level l.
- Find PM_{ij}^l, AFT_{ij}^l and ST_{ij}^{l+1}.
- Idle slots are reduced by assigning the modules of proper size from successive depth level so that these modules may be best fit to those slots on the computational machine.
- Modules are inserted in such a way that they preserve the DAG constraints of the jobs.
- For any given idle slot, the adjacent modules in the DAG to the modules assigned on that machine (i.e. having idle slot) only are considered so that communication overhead due to insertion can be minimized.
- This scheme avoids machines sitting idle as much as possible in turn improving the resource utilization and TAT.
- TCT_k^l is updated with the execution time of the modules that are inserted during this phase.

An algorithmic template for the same is presented in the box below:

```
LBSIR ()
Input: Batch $\beta$ and Set R of K computational machines
Output: A schedule S on $\beta$ onto R
1.  Partition batch
    // Divide the batch modules as per their precedence (depth) levels and find $\rho^l$ set of modules where each set has $M_l$ modules
2.  Sort Partition
    // Sorting partition in accordance with the module size in increasing or decreasing order
3.  Generate ETC matrices
        // $E_{ijk}$ as per ETC simulation benchmark model
4.  Compute Queuing Time // as per Eq. (9)
5.      Allocation ()
6.      {
7.        For all L do
8.        for all $\rho^l$ do//for all modules of lth partition in the sorted order
9.        For all K
10.           Select the Best Fit Node    // Select the node for the current module
                                          //which offers the least aggregate time 't' where
                $t = \mathrm{RT}_k^l + \mathrm{ET}_k^l + E_{ijk}$     for $m_{ij} \in \rho^1$
                $t = \mathrm{ET}_k^l + E_{ijk}$             otherwise
            end for
11.                 Allocate module $m_{ij} \in \rho^l$ on selected machine and update schedule
                            $N_k \leftarrow m_{ij}$ & $S(i, j) \leftarrow k$
12.           Update Execution Time $\mathrm{ET}_k^l$
                        $\mathrm{ET}_k^l = \mathrm{ET}_k^l + E_{ijk}$
13.         end for
14.         Compute $\mathrm{CC}_k^l$ //as per Eq. (4)
15.         Compute $\mathrm{TCT}_k^l$ as per Eq. (5)
16. Idle slot reduction ()
17.   {
18.       Get $\mathrm{ST}_{ij} \in \rho^l$ & $\mathrm{AFT}_{ij} \in \rho^l$
19.       Get $\mathrm{IS}^l$ list of idle slots at depth level l
20.       for all $\rho^{l+1}$ do
            // all modules of depth level l+1
21.       for k = 1 to K do
22.       If (($\mathrm{AFT}_{ij-v} \in \rho^l \le \mathrm{ST}_{ij} \in \rho^{l+1}$) &($E_{ijk} \le S_k \in \mathrm{IS}^l$))
            //assign the module $m_{ij} \in \rho^{l+1}$ to node $N_k$
23.           $N_k \leftarrow m_{ij}$ & $S(i, j) \leftarrow k$
24.         Update idle slot list
                          // $S_k = (S_k - E_{ijk})$
25.                       Update $\mathrm{TCT}_k^l$ as per Eq. (7)
26.         end if
27.             end for
28.         end for
29.         }
30. end for
31. }
32. Compute Turnaround Time (TAT)
        // as per Eqs. (8) and (9)
33. Compute Average Utilization
        // as per Eq. (10)
34. Compute Flow time // as per Eq. (11)
35. Compute Average Response Time (RT) as per Eq. (12)
36. Get schedule S and values of optimizing parameters as output
```

An example is illustrated to explain both the phases of the LBSIR. In allocation phase partitions are sorted in the descending order making it LBSIR-LMS. Consequently, larger modules are selected first and mapped to the best fit node. For the sake of simplicity only three machines at the time of allocation are taken. A batch of three jobs $B = \{J_1, J_2, J_3\}$ is considered for execution represented as DAGs having 5 modules each as shown in Fig. 4. Highest precedence level in the batch is $L = \max(L_1, L_2, L_3, L_4) = 4$. Let the arrival rate of the batches on global queue and service rate of dispatch and evaluation node be 0.03 & 0.05 batches per unit of time respectively and the queue unit Q_u for the system be 10,000 MIs. The information regarding the batch B and computational nodes considered are presented in Tables 1–3.

The batch B is divided into partitions as per the depth level and sorted in decreasing order as $\rho^1 = \{m_{11}, m_{21}, m_{31}\}$, $\rho^2 = \{m_{13}, m_{24}, m_{12}, m_{32}, m_{23}, m_{22}\}$, $\rho^3 = \{m_{14}, m_{33}, m_{34}, m_{25}\}$ and $\rho^4 = \{m_{15}, m_{35}\}$.

Table 1
Job information.

Depth level/ETC information		l	E_{ij1}	E_{ij2}	E_{ij3}
J_1					
	M_{11}	1	70.8	35.4	44.25
	M_{12}	2	74	37	46.25
	M_{13}	2	82	41	51.25
	M_{14}	3	80	40	50
	M_{15}	4	40	20	25
J_2	M_{21}	1	62.8	31.4	39.25
	M_{22}	2	58	29	36.25
	M_{23}	2	73	36.5	45.625
	M_{24}	2	75	37.5	46.875
	M_{25}	3	59	29.5	36.875
J_3	M_{31}	1	56	28	35
	M_{32}	2	74	37	46.25
	M_{33}	3	66	33	41.25
	M_{34}	3	60	30	37.5
	M_{35}	4	20	10	12.5

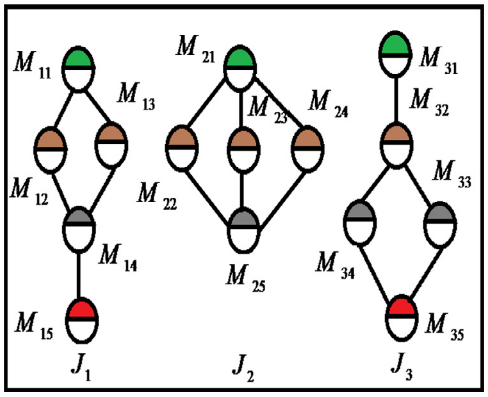

Fig. 4. A sample of batch of three jobs.

Table 2
Inter module communication information for jobs.

		M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
J_1	M_{11}	0	2	3	0	0
	M_{12}	2	0	0	5	0
	M_{13}	3	0	0	4	0
	M_{14}	0	5	4	0	0.5
	M_{15}	0	0	0	0.5	0
J_2		M_{21}	M_{22}	M_{23}	M_{24}	M_{25}
	M_{21}	0	2	3	2	0
	M_{22}	2	0	0	0	5
	M_{23}	3	0	0	0	3
	M_{24}	2	0	0	0	2
J_3		M_{31}	M_{32}	M_{33}	M_{34}	M_{35}
	M_{31}	0	2	0	0	0
	M_{32}	2	0	3	2	0
	M_{33}	0	3	0	0	0.1
	M_{34}	0	2	0	0	1.5

Table 3
Computational grid information.

Ready time information			
Node identifier	N_1	N_2	N_3
RT_k^l (μS)	50	30	90
Inter-node distances information (in hop counts)			
	N_1	N_2	N_3
N_1	0	2	4
N_2	2	0	5
N_3	4	5	0

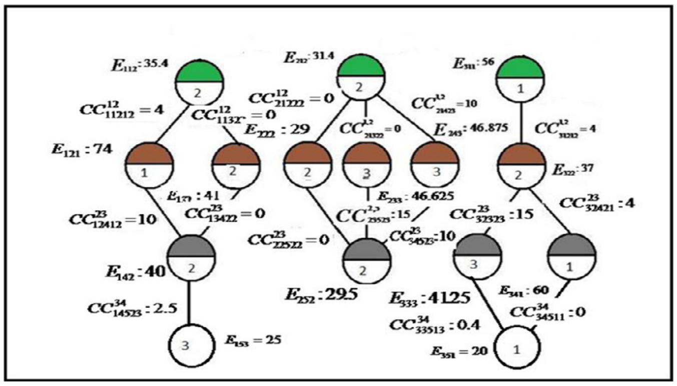

Fig. 5. Allocation and cost representation for the whole batch using LBSIR (LMS).

As soon as the allocation of modules from first partition are completed as shown in Figs. 5 and 6, m_{11}, m_{21}, and m_{31} get assigned on nodes N_2, N_2, and N_1 respectively. The idle slots list IS^1 of first depth level on the computational node becomes $\mathrm{IS}^1 = \{s_2$: 9.2, s_3: 16}. The succeeding modules of these modules $m_{11}, m_{21} \in \rho^1$ in partition ρ^2 are m_{12}, m_{13} and m_{22}, m_{23}, m_{24}. These modules cannot fit into slots s_2 or s_3 as modules execution time (i.e. ETC values for m_{12}, m_{13} are for node P_2 are 37, 41 respectively which is more than 9.2) being more than the slots size. The idle slots at node N_1 and node N_3 are of size 34 and 18.5. Again, after allocation of modules belonging to the second level, idle slot list IS^2 becomes $\mathrm{IS}^2 = \{s_1$: 34, s_3: 18.5} and modules from next partition ρ^3, again cannot fit into the slots generated during execution in depth level 2 by the same reason as mentioned above. Finally, we get idle slot list IS^3 for depth level 3 as $\mathrm{IS}^3 = \{s_1$: 20.5, s_3: 28.5}. Now, the module $m_{15} \in \rho^4$ is not best fit for idle slot s_1: 20.5 $\in \mathrm{IS}^3$ as E_{153} is equal to 40. Module m_{15} is best fitted on idle slot s_3: 28.5 $\in \mathrm{IS}^3$ as $E_{151} + \mathrm{CC}^{34}_{14523} = 25 + 2.5 = 27.5 \leq s_3$. Again, module $m_{35} \in \rho^4$ is best fitted on idle slot s_1: 20.5 $\in \mathrm{IS}^3$ as $E_{351} + \max(\mathrm{CC}^{34}_{34513}, \mathrm{CC}^{34}_{33533}) = 20 + \max(0.4, 0) = 20.4 \leq s_1$.

The communication cost (CC^l_k) on node N_k is considered as the maximum communication requirement among all modules assigned on N_k with previous level modules PM^{l-s}_{ij}. For example, the modules m_{14}, m_{25} assigned on node N_2 in depth level 3 having

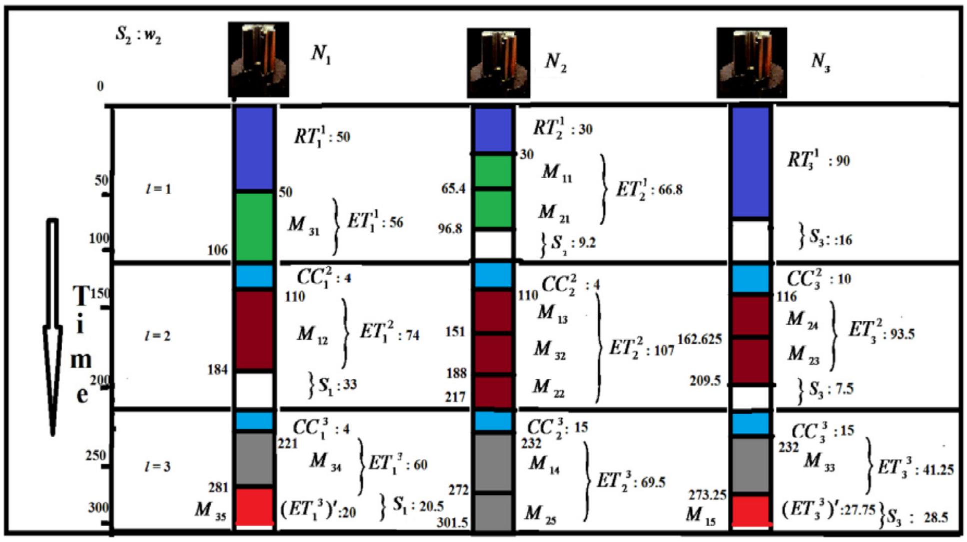

Fig. 6. Level-wise allocation of batch using LBSIR (LMS).

communication with the modules m_{12}, m_{13}, m_{22}, m_{23} and m_{24} for 10, 0, 0, 15, 10 unit time respectively are shown in Fig. 5. The $CC_2^3 = \max(10, 0, 0, 15, 10) = 15$ unit time as shown in Fig. 6. The allocation of whole batch using idle slot reduction scheme with largest module selection on the computational machines is showing complete schedule, communication requirements within modules and their execution times as in Fig. 5. Further, complete processing of batch for all depth levels on computational nodes along with their Ready Time (RT_k^l), Start Time (ST_{ij}^l) and Actual Finish Time (AFT_{ij}^l) of modules, Execution Time (ET_k^l) and net Communication Cost (CC_k^l) at each node has been illustrated in Fig. 6 as shown below.

Finally, turnaround time (TAT) for the batch can be calculated by using Eq. (8) as

$$TAT = 50 + 301.5 = 351.5 \text{ unit of time}$$

The Resources Utilization (U_s) of the grid system can be calculated by using the appropriate values from Fig. 6 and Eq. (10) as

$$\sum_{l=1}^{L} (TCT_1^l)' = 50 + 56 + 4 + 74 + 4 + 60 + 20 = 268$$

$$\sum_{l=1}^{L} (TCT_1^l)' = 30 + 35.4 + 31.4 + 4 + 41 + 29 + 37 + 15 + 40 + 29.5 = 287.3$$

$$\sum_{l=1}^{L} (TCT_3^l)' = 90 + 15 + 46.875 + 46.625 + 15 + 41.25 + 27.75 = 282.50$$

Therefore,

$$U_s = (837.8)/(3^*287.3) \cong 0.9720 = 97.20\%.$$

Further, Speed up (S_s) of the system over the best node in terms of capacity i.e. node 2, Flow Time (FT) and average Response Time (RT) for batch of the jobs submitted from different users can be calculated using Eqs. (11) and (12) as

$$S_s = TAT_s/TAT = 555.3/351.5 = 1.5798$$
$$FT = T_1 + T_2 + T_3 = 837.8 \text{ unit of time}$$
$$RT = (30 + 65.5 + 50)/3 = 48.5$$

The TAT and utilization for the batch by SLBBS (Shahid and Raza, 2014) calculated for the same batch is realized as 380.5 time unit and approx. 87% respectively. Therefore, it can be established that the use of LBSIR substantially improves the scheduler performance.

4.3. Motivations for the proposed method and limitations

The level wise allocation takes inspiration from the fact that the serial execution of the whole DAG on a single machine is unable to exploit the inherent parallelism within the DAG as suggested in the work (Xhafa et al., 2007a). The proposed work faces the communication overhead which was not in Xhafa et al. (2007a) but also exploits inherent parallelism within the jobs of the batch. In the proposed method LBSIR, the communication requirement is managed in such a way that the communication overhead is reduced. The communication within the modules is made parallel each time after execution of the partition owing to the availability of parallel architecture of computational hardware with the ability of each machine to receive/transfer data bytes simultaneously. Therefore, it is expected by the proposed implementation that in presence of sufficient degree of parallelism within DAGs and sufficient number of computational nodes the performance will be better than former due to the exploitation of parallelism within the DAGs. Further, the proposed strategy with both its variants has been designed for the following reasons:

- The batch mode methods result in better mapping of the batch on the computational resources owing to the prior information about the jobs in the batch as well as the resources.
- Most of the batch mode methods available focus only on the parallelism available at the job level implying serial execution of the individual job on a single resource thus wasting the parallelism, if any, at the sub job level.
- Proposed methods exploit parallelism at both job level as well as the sub job/module level.
- Preserves DAG constraints and deals with communication requirement by using level attribute of the DAG being more convenient to implement.
- Slicing all modules of the same depth level from the jobs of the batch and their level wise execution minimizes the response time of the batch as all jobs are executing in the first as well as remaining levels. Accordingly, this method becomes quite suited for the jobs from interactive users.
- Each time smallest or largest available modules get assigned to best fit node which leads to improvement in the turnaround time or utilization.
- Insertion based scheme effectively reduces the idle slots thereby approaching toward optimum value of the turnaround time and utilization.
- Since, hardware cost has reduced significantly over time and the same trend is expected to continue, we expect to have even more computational power in the future at a substantially lower cost. Therefore, in case where high degree of parallelism is available in the batch along with high hardware parallelism coupled with low communication requirements results in boosting the performance of the proposed batch strategies.
- Average communication cost share within turnaround time for LBSIR variants significantly lesser than peers HEFT, CPOP and DLS.

Average communication cost incurred by LBSIR, CC_{LBSIR} can be written as the summation of the net communication cost CC_k^l over nodes and depth levels as seen in Fig. 3 and can be estimated as

$$CC_{LBSIR} = \frac{1}{K} \sum_{l=1}^{L} \sum_{k=1}^{K} (CC_k^l) \tag{13}$$

The value of CC_k^l can be computed using Eq. (4). The average communication cost incurred by allocation of batch of N jobs by DAG scheduling heuristics considered for comparison in this work viz. HEFT, CPOP and DLS with m modules each in average can be written as

$$CC_{HEFT} = \frac{N^*m}{K} (\bar{C}) \tag{14}$$

where $\bar{C}$ is average communication cost between two modules.

If $\bar{C} \cong CC_k^l$ (i.e. range of modules communication cost is small),

$$(CC)_{LBSIR} = \bar{C}^* L$$

Therefore,

$$\frac{CC_{LBSIR}}{CC_{HEFT}} = \frac{Nm}{KL} \tag{15}$$

In real scenario, $N \gg K$ & $m \gg L$. Therefore, $CC_{LBSIR} < CC_{HEFT}$.

Although, proposed batch mode methods have many virtues to use these methods in the real grid environment, however, these are not free from certain limitations. Some of these are listed below:

- The case of low parallelism in the batch and high communication cost along with the availability of very few computational machines becomes the bottleneck in the performance.
- If the arrival rate of the jobs is very low, the average waiting time in the global queue becomes more for batch formation thus affecting the turnaround time of the batch significantly due to centralization.

4.4. Time complexity of the proposed method

Time complexity of an algorithm quantifies the amount of time taken by it to run as a function of size of the input and computed under various phases as:

- Partition the batch as per the depth level: The process of partitioning N jobs of the batch into L partitions has three loops viz. outermost corresponding to the number of partitions, middle one for the number of jobs and inner loop for the number of modules m_i in the jobs J_i. Accordingly, the time complexity can be written as of the order $O(L^*N^*m)$, where m is the average number of modules in the jobs.
- Sorting the partition: This process corresponds to the sorting of each partition in an ascending or descending order in allocation phase. The order of sorting a partition ρ^l of size M_l is $O(M_l * M_l)$. Hence, the time complexity of sorting all L partitions is $O(L*M*M)$ where M is the average number of modules in the partitions.
- Allocation/Idle slot Reduction: The time complexity of allocation of the partitions and reduction of idle slots produced during allocation are of same order. Therefore, time complexity of only allocation phase is computed. As presented in the algorithm, for each iteration of outermost loop for the depth levels (L), the middle loop for average number of modules (M) in the partitions is executed for M times. Same way, for each M the inner most loops is executed for all nodes (K). Therefore, the order of time complexity for allocation will be $O(L*M*K)$.

If a task is completed as three phases, the net complexity is taken as the maximum complexity of either phase. Number of jobs (N) in the batch and number of nodes (K) considered for allocation are smaller than M and m is comparable to M for the batch having more parallelism within the jobs. Therefore, the complexity for the algorithm can be computed as:

$$= \max\left(O(L*N*m),\ O(L*M*M),\ O(L*M*K)\right)$$
$$\cong O(L*M*M).$$

Number of levels in a graph is equal to $O(\log m)$. Therefore, the time complexity of LBSIR can be written as $O(M^2 \log m)$.

5. Simulation study

Simulation experiments were performed to evaluate the proposed strategies by observing the allocation of the batch of jobs on the grid for different input parameters. The experiments were conducted using MATLAB 7.60 on Sun Fire X4470 Server having eight cores with 14 GB RAM. The parameters related to grid are responsible for the simulation of the grid environment being generated for heterogeneous computational machines of a specified number in the specified range of capacities and inter node distance. A random batch generator generates random batch of DAGs in a specified range of job size, module size, inter module byte exchange for communication and the parallelism within the batch (Yang and Gerasoulis, 1994).

The simulation study has been done by using ETC simulation benchmark model for static heuristics. The benchmark of these instances are widely accepted for the mapping problem for grid due to better resemblance with the real highly heterogeneous and non-consistent environment having following characteristics (Braun et al., 2001; Xhafa et al., 2007a):

- It uses a combination of three matrices X, Y and Z with the notation XYZ to define 12 cases. X represents the matrix types i.e. consistent (C), inconsistent (I) or semi consistent (S), Y the task heterogeneity i.e. high (H) or low (L) and Z the node heterogeneity i.e. high (H) or low (L). As an example, SHL interprets a case of semi consistent with high job heterogeneity and low nodes heterogeneity.
- The distribution used in the study is the uniform distribution for generating input parameters.

The proposed LBSIR strategy with its variants has been compared with state of art batch scheduling strategies Min Min, Max Min, Sufferage and LJFR-SJFR which only exploits the parallelism at the job level (Xhafa et al., 2007a). Further, some other DAG scheduling heuristics (Sih and Lee, 1993; Topcuoglu et al., 2002) i.e. HEFT, CPOP and DLS are considered for comparison taking into accounts not only execution time but also the communication cost incurred by the communicating modules. HEFT has been proven to be having quadratic time complexity and it produces the shortest schedule lengths for random graphs when compared with 20 heuristics (Canon et al., 2008). Still, HEFT algorithm is highly competitive as it gives a schedule which has a comparable schedule length over others with a lower time complexity. Batch implementation of HEFT is performed in two ways as:

- HEFT-1: In this implementation, jobs are taken from the batch one by one to generate a schedule for execution one after another by using HEFT algorithm given in Topcuoglu et al. (2002). This implementation exploits only module level parallelism while jobs are processed serially.
- HEFT-2: In this version, two pseudo tasks, one for entry and another for exit are considered. The resultant DAG is generated by connecting all entry tasks of each jobs of the batch to pseudo entry tasks and all exit tasks of the jobs to pseudo exit tasks. In this way, a DAG is created for the whole batch and HEFT algorithm is applied to execute this DAG.
- CPOP is only implemented as HEFT-1 and communication cost is included to compute ranks for modules and computation cost of module is assumed as the mean of the computation cost on all nodes in DLS.

Since, performance of LBSIR (LMS-SMS) is expected to be between LBSIR (LMS) and LBSIR (SMS) it has been excluded from the discussion. The study has been divided in three parts to observe the performance while varying the heterogeneity and consistency of the system, parallelism in the batch and computational nodes in the system as discussed in the coming section. The various input parameters used in the various experiments which are conducted for evaluation of the proposed model are listed in Table 4.

5.1. Varying heterogeneity of the jobs and nodes and consistency of the computing system

In this section, LBSIR variants are evaluated by varying the heterogeneities of the jobs in the batch and computational nodes in the grid system while varying the consistency of the computational nodes. As discussed in Braun et al. (2001) and Xhafa et al. (2007a), the range of Expected Time to Compute (ETC) for M_{ij} on N_k are varied as heterogeneity vary from low to high for both the jobs and the nodes. The range of execution time (E_{ijk}) for four combinations of heterogeneities viz. XHH, XLH, XHL, XLL are 1–3,000,000, 1–100,000, 1–30,000 and 1–1000 respectively. Note that for all the instances, the number of jobs in the batch is 256 and each job has 512 modules with the number of computational nodes being 16. Therefore, the total number of modules becomes 2^{17}. For each instance, turnaround time and resource utilization, flow time and response time values are computed 100 times and then its mean is reported. Accordingly, the analysis presented in the succeeding sections is carried out for a mean of 100 data sets computed for all the four parameters. The remaining input parameters for the experiments are as follows:

$$\text{Ready Time }(\mathrm{RT}_k^l) = 0 - 10,000,\ \text{Number of depth level }(l)$$
$$= 16,\ \text{Number of modules of each depth level in the job}$$
$$= 32,\ \text{Inter Node distance between the nodes } = 1 - 1000$$

Table 4
System parameters.

S. no.	Parameters	Notation used	Range
1	Number of nodes	N_k	8–128
2	Time to finish previous workload	RT_k^l (μS)	0-10000
3	Number of jobs in the batch	J_j	4–256
4	Number of modules in a job	M_{ij}	128–1024
5	Expected time to compute for M_{ij} on N_k	E_{ijk}	1–3,000,000
6	Number of levels in the jobs	L_i	8–128
7	Inter node distance between the nodes N_k and N_r	D_{kr}	1–1000
8	Inter module communication between M_{ip} and M_{iq}	IMC_{ipq} (KBs)	1–3000
9	Service rate	μ	0.05
10	Arrival rate	λ	0.03
11	Queue unit	Q_u (MIs)	200,000

5.1.1. *Batch consisting of non communicating dependent modules*

Figs. 7–10 present the behavior of all four parameters by varying the heterogeneities of jobs and nodes and consistency of the grid system. In this case, modules in the batch are still subjected to dependencies due to the DAG constraints within the jobs while the communication requirement among them is considered negligible (zero). Therefore, inter module communication (IMC_{ipq}) becomes zero for all pairs of modules. Hence, the net communication cost ($CC_{ipqkr}^{l\ l-s}$) for all pair of dependent modules by using Eq. (1) becomes zero. LJFR-SJFR is observed to offer the worst performance among all heuristics for the turnaround time for all 12 instances while HEFT-1 and CPOP offers worst average response time and have been excluded from the results reported in Fig. 7(a)–(c) and Fig. 10(a)–(c) respectively.

Observations:

1. As can be seen in Fig. 7, LBSIR (SMS) outperforms all other strategies with LBSIR (LMS) just below par as compared to LBSIR (SMS) with the objective of minimizing the turnaround time. This is due to the allocation of smallest available module in each partition at each depth level to the best fit node. Between Min Min and Max Min also the same pattern is observed for make span for the same reason although in this case the entire job is allocated to the suitable node. LBSIR (SMS) achieves on an average 25–70% and 20–50% improvement in the turnaround time for all 12 instances over Min Min and HEFT-2 heuristic depending on the heterogeneity in jobs and resources respectively. The reason behind better performance of LBSIR variants is the level wise assignments of modules which exploit the parallelism existing at each depth level with the communication requirement being negligible in this case making it the most suitable candidate for the situation. The turnaround time value is observed to be higher when the heterogeneity is high for both the jobs and the nodes as compared to the case with low heterogeneity for both and in between for the case with alternating heterogeneity as high heterogeneity results in higher ETC range as discussed earlier. HEFT-2 performs better than Min

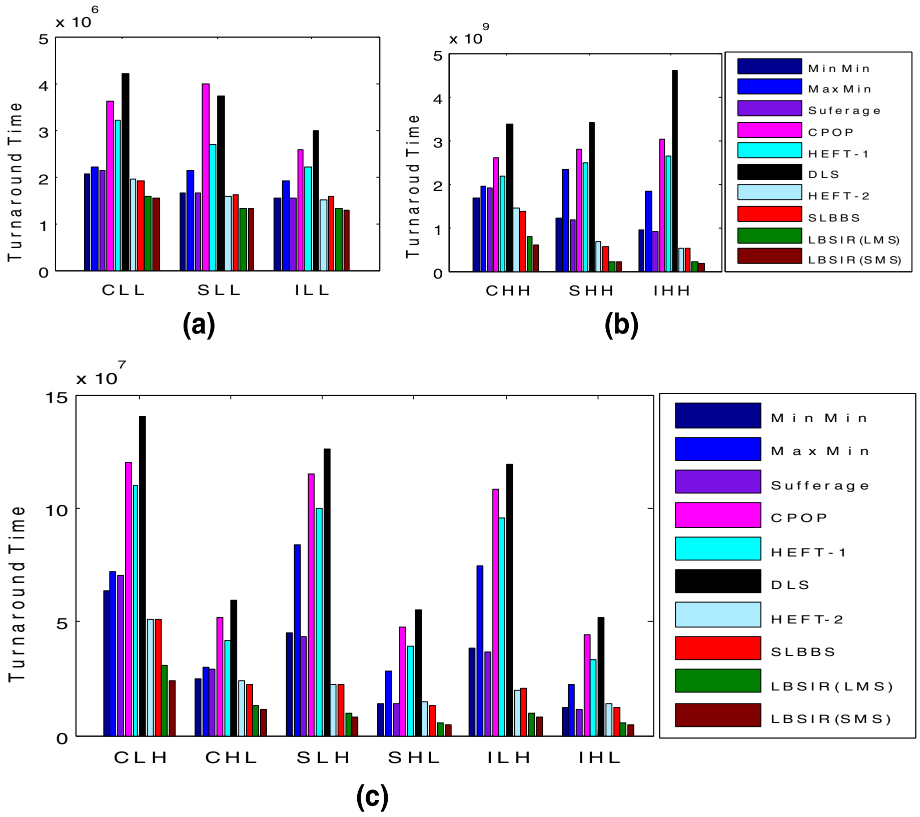

Fig. 7. (a) Turnaround time vs XLL. (b) Turnaround time vs XHH. (c) Turnaround time vs XHL or XLH.

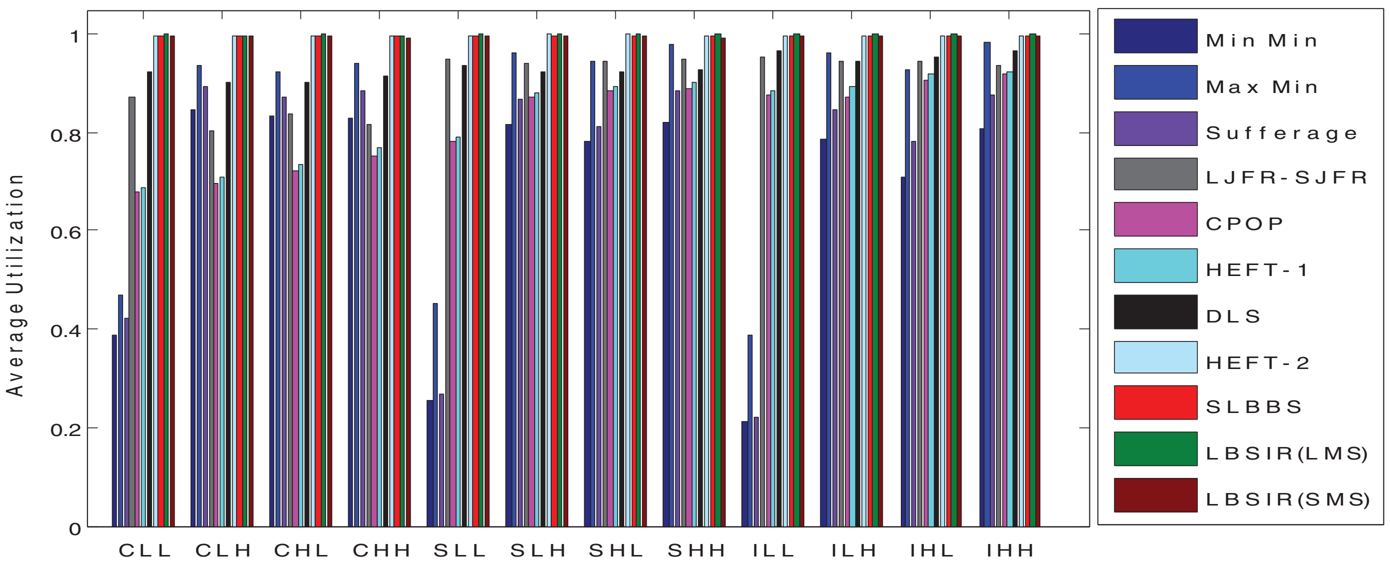

Fig. 8. Average utilization vs instances.

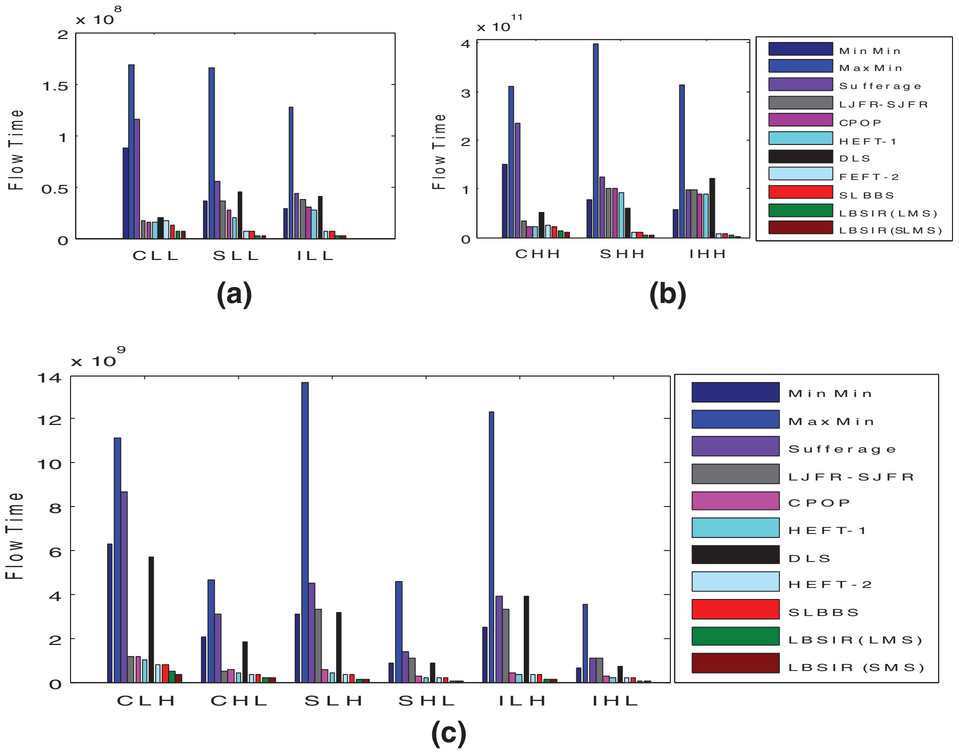

Fig. 9. (a) Flow time vs XLL. (b) Flow time vs XHH. (c) Flow time vs XHL or XLH.

Min due to zero communication cost. Max Min and Sufferage are observed to be offering the turnaround time better than HEFT-1, CPOP and DLS and inferior to Min Min. Sufferage performs better than Max Min for all 12 instances and at par with Min Min for CHH, SHH and IHH is shown in Fig. 7(b). DLS followed by CPOP performs poorer to HEFT-1. Similarly, LBSIR (SMS) and LBSIR (LMS) exhibit approximately 20–60% improvement on SLBBS on account of turnaround time and flow time as per the case. This gain can be attributed to the utilization of the unused idle slots at each depth level through insertion from modules from the succeeding partitions at higher depth levels. All three variants of LBSIR have observed to offer TAT varying within the range of approximately 2% variation.

2. Both LBSIR variants and SLBBS perform better than other heuristics for average resource utilization for all 12 instances as shown in Fig. 8. Min Min performs worst among all, Max Min being better than Sufferage and LJFR-SJFR being very low for the cases with low task and node heterogeneity XLL. Gain in utilization for higher batch size like batch size of 256 or more on LBSIR variants over SLBBS is negligible due to the reason that for a larger batch ($\sim 2^{17}$ modules in this case) all the nodes keeps busy. HEFT-2 too performs close to LBSIR variants and SLBBS.

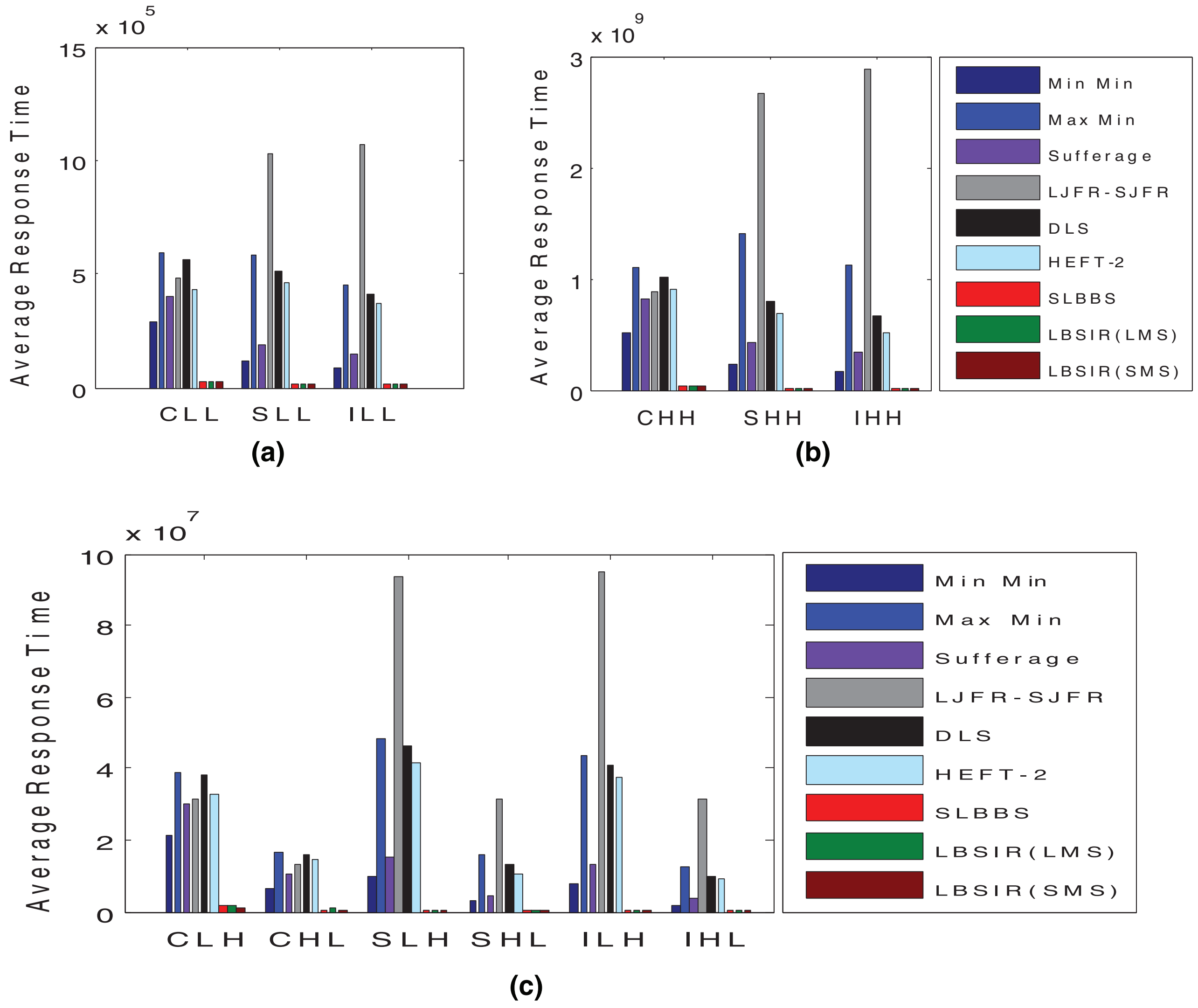

Fig. 10. (a) Average response time vs XLL. (b) Average response time vs XHH. (c) Average response time vs XHL or XLH.

3. LBSIR (SMS), LBSIR (LMS) and SLBBS outperform all strategies with LBSIR (SMS) being much superior to all. HEFT-2, HEFT-1, and CPOP are better than Min Min which exhibits a better performance than Max Min and Sufferage for all 12 instances and lagging with LJFR-SJFR in four instances CLL, CHL, CLH, and CHH as presented in Fig. 9(a)–(c). Max Min is observed to be the worst among all for all 12 instances.
4. In terms of the average response time, LBSIR variants perform the best as shown in Fig. 10(a)–(c). Of the remaining heuristics, Min Min performs better than Max Min, Sufferage, LLJFR-SJFR, HEFT-2 and DLS for all 12 instances. SLBBS performs comparable to LBSIR variants in terms of the average response time as level based batch scheduling strategies depend only on allocation of modules from the first depth level and idle slot reduction has no effect on the response time as it only inserts modules in the idle slots from the higher levels. HEFT-2 and DLS perform superior to Max Min and LJFR-SJFR with HEFT-1 and CPOP being the worst.

Therefore, it can be concluded that for the batch of jobs consisting of dependent modules having no inter module communication, LBSIR (LMS) and LBSIR (SMS) outperform the remaining heuristics. The performance of LBSIR (LMS) and LBSIR (SMS) improves slightly on all parameters for all 12 instances if we consider modules of the jobs in the batch that are not subjected to dependencies.

5.1.2. *Batch consisting of dependent and communicating modules*

Here, the same batch is considered as in the previous section but with a difference that the modules are dependent and may communicate. Communication requirements significantly affect the turnaround time as compared to other parameters. Therefore, this section only presents the results focusing on turnaround time as shown in Fig. 11(a)–(c) while summarizing the effect on other parameters. The inter module communication (IMC_{ipq}) and inter node distance (D_{kr}) is taken to be in the range of 1–3000 (MIs) and 1–1000 (hop counts) for all pairs of modules and nodes respectively resulting the range of net communication cost ($\text{CC}^{ll-s}_{\text{ipqkr}}$) using Eq. (1) to be 1–3,000,000 which is same as the range of ETC values for XHH instances and more than other 9 instances out of 12. Remaining input parameters are same as used in the previous section. HEFT-1, CPOP and DLS performs inferior to LJFR-SJFR on turnaround time for all 12 instances due to higher communication cost share in turnaround time and is excluded from the results shown in Fig. 11(a)–(c) for turnaround time. The observations for the same are as follows:

Observations:

1. As can be seen in Fig. 11(a)–(c), LBSIR (SMS) again outperforms the other heuristics and achieves on an average 40–60% improvement in turnaround time for all instances over Min-Min other than CLL,

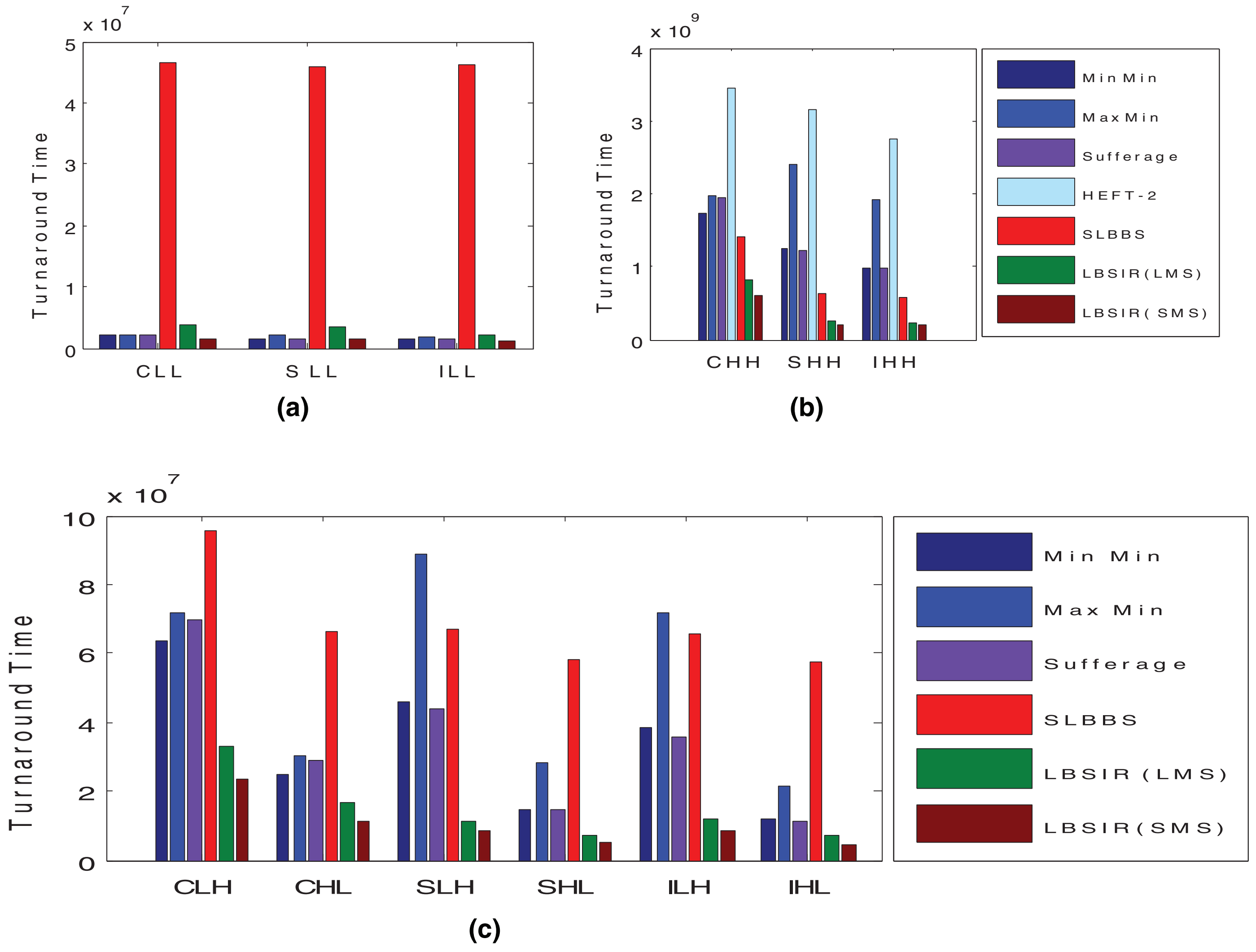

Fig. 11. (a) Turnaround time vs XLL. (b) Turnaround time vs XHH. (c) Turnaround time vs XHL or XLH.

SLL, ILL, LBSIR (SMS) is at par, LBSIR (LMS) is lagging to Min Min and SLBBS is performing poorer, only better than LJFR-SJFR in three instances CLL, SLL and ILL out of the twelve instances as shown in Fig. 11(a). The reason behind this is that in this case, job and node heterogeneities become low for both and the range of ETC values for module is 1–1000 while the communication cost of the modules is still between 1 and 3,000,000 which is many folds higher than the range of execution time. SLBBS performs worst as in this case there are many idle slots generated in the process and are not getting optimized. The same is improved by LBSIR (LMS) and by LBSIR (SMS) being able to accommodate modules from the higher depth levels by assigning the smallest modules to the best fit nodes and exploiting the inherent parallelism in the jobs of the batch to its maximum. This in turn reduces the idle slots as well muting the effect of the communication cost in the process in a better way. While Min Min heuristic assigns the complete job to a single node that eliminates the communication cost during execution among modules, in contrast to the proposed methods, Min Min is not capable of exploiting the inherent parallelism present in the jobs that can be exploited by executing modules in parallel. The tradeoff between communication cost and exploitation of parallelism in the batch is not balanced due to the communication cost ranging much more than the range of ETC values in these three instances. LBSIR (LMS) is slightly poorer than LBSIR (SMS) as smallest module selection at each time in LBSIR (SMS) improves the turnaround time in the same way as Min Min outperforms Max Min on the count of make span. Further, LBSIR (LMS) is performing better than Min Min, Max Min, Sufferage and LJFR-SJFR for other nine remaining instances due to increase in the range of ETC value for the modules by having at least one of the heterogeneity high that reduces the mismatch between the range of the communication cost and the execution time. As heterogeneities become high either for nodes or for jobs, performance boosts up. Finally, performance of proposed LBSIR variants is the best in the case of CHH, SHH and IHH because in these cases the range of both communication cost and execution cost of the modules becomes the same.

2. Min Min, Max Min, Sufferage and LJFR-SJFR is independent of the communication cost among modules because in these methods complete jobs are assigned on nodes by which communication costs incurred within the jobs becomes zero. The performance behavior of these batch mode methods is same as in the previous section.

Both LBSIR variants perform better than all other heuristics on the count of utilization, flow time and average response time in this case as well with the performance order of the heuristics nearly as discussed in the previous section. It is observed that the strategies involving communication cost exhibits a little increased flow time as compared to Min Min and Max Min, Sufferage and LJFR-SJFR. Further, in LBSIR and SLBBS, all the modules at the first depth level responsible for deciding the response time start almost in the same span. Further, due to this levelized nature of the LBSIR algorithm there is no effect of the communication cost until the second level. Other DAG scheduling strategies deteriorate as far as response time is concerned even more than as reported earlier. It is to be noted that Min Min, Max min, Sufferage and LLJFR-SJFR are invariant of the communication cost for all parameters.

In this section, it should be noted that the performance of LBSIR variants depend on the tradeoff between the communications cost and the execution times. If the communication cost between modules is comparable to the execution time of modules (being in the same range), performance of LBSIR variants perform better as seen in the instances CHH, SHH and IHH. When the range of execution time is between 1 and 30,000 or 1 and 100,000 respectively for alternate heterogeneities (XLH, XHL) and the communication cost in the range 1–3,000,000 for all instances being more than the range of the execution cost, LBSIR variants still outperforms. But their performance suffers for turnaround time for the instances (CLL, SLL and ILL) with respect to Min Min where the range of execution time is 1–1000 which is extremely lower than the range of communication cost but their performance is still better than others being one of the significant features. The effect of communication requirement on turnaround time and flow time is much more than any other parameter. On the other hand if communication cost range (1–3,000,000) is decreased up to the range of execution time (1–1000) making both comparable even in the case of the instances like CLL, SLL and ILL, the performance of LBSIR variants again boost up.

5.2. *Varying parallelism in the batch*

This section discusses the effect of variation in the inherent parallelism in the batch i.e. parallelism at depth level for the fixed computational nodes. Degree of parallelism of the batch can be defined as the amount of modules from the batch which can be executed in parallel. Thus the parallelism observed at each depth level in the batch depends on the number of independent modules in the partition. Parallelism within the batch is increased by increasing the number of independent modules in the specified depth levels/partitions and vice versa. Number of independent modules in the partitions can be increased in the following three possible ways:

(1) Reducing the depth levels for fixed number of modules in the jobs
(2) Increasing the number of modules in the fixed number of jobs
(3) Increasing the batch size for fixed depth levels

5.2.1. *Varying the batch size with variable load*

This part of the study discusses the effect on various parameters under study by varying the parallelism in the batch by varying the batch size from small to large. Batch size, in turn, depends on two factors viz. number of jobs in the batch and the number of modules in the considered jobs. In the experimental analysis, the batch size has been varied from 4 jobs to a larger batch size of 128 jobs for the fixed 256 modules in each job. The results are presented in Figs. 12–15. The other common input parameters considered here as follows:

Number of Nodes $(K) = 16$, Number of Modules in a Job $(m) = 256$, Number of depth level in the jobs $(l) = 16$, Number of modules of each depth level in the job $= 16$, Ready Time $(RT_k^l) = 0 - 10,000$, Inter Node distance between the nodes $= 1$–100, Inter Module Communication $(IMC_{ipq}) = 1$–1000, ETC $= 1$–$3,000,000$, Machine Heterogeneity $=$ high, Job Heterogeneity $=$ high, Consistency $=$ Inconsistent.

Observations:

1. As expected, the turnaround time, average utilization, flow time and average response time go on increasing when the batch size is increased while keeping the number of nodes fixed for all the scheduling strategies as shown in Figs. 12–15.
2. For all batch sizes from 4 to 128, LBSIR (SMS) and LBSIR (LMS) perform better than all considered strategies on turnaround time. Between the heuristics other than LBSIR, for a variation of the batch size from 8 to 32 the performance of SLBBS is observed to be better than Min Min, Sufferage and Max Min, HEFT-1, HEFT-2 (Small batch), DLS and CPOP. Still, HEFT-2 performs superior to Min Min, Sufferage, Max Min, HEFT-1, DLS, CPOP and SLBBS (for large batch) as HEFT-2 is implemented as batch mode which results in better mapping than HEFT-1. LBSIR variants achieve more than 50% gain over SLBBS in terms of the turnaround time for all batch sizes. The performance of HEFT-1, DLS, and CPOP including HEFT-2 is inferior to LBSIR variants as these methods have a significant share of communication cost in turnaround time.
3. LBSIR (LMS) performs best for average utilization for all batch sizes as presented in Fig. 13. All heuristics which assigns the entire job on a single suitable machine exhibits extremely poor utilization for small batch with the gap bridging with an increase

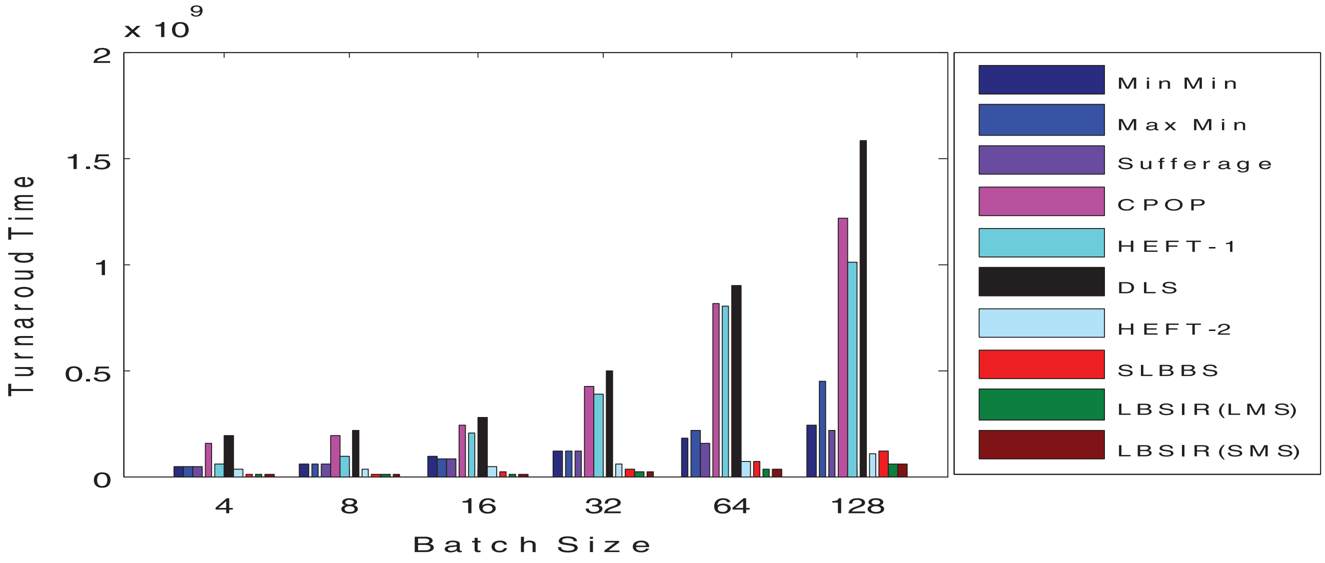

Fig. 12. Turnaround time vs batch size ($K = 16$ & $L = 16$) with increasing load.

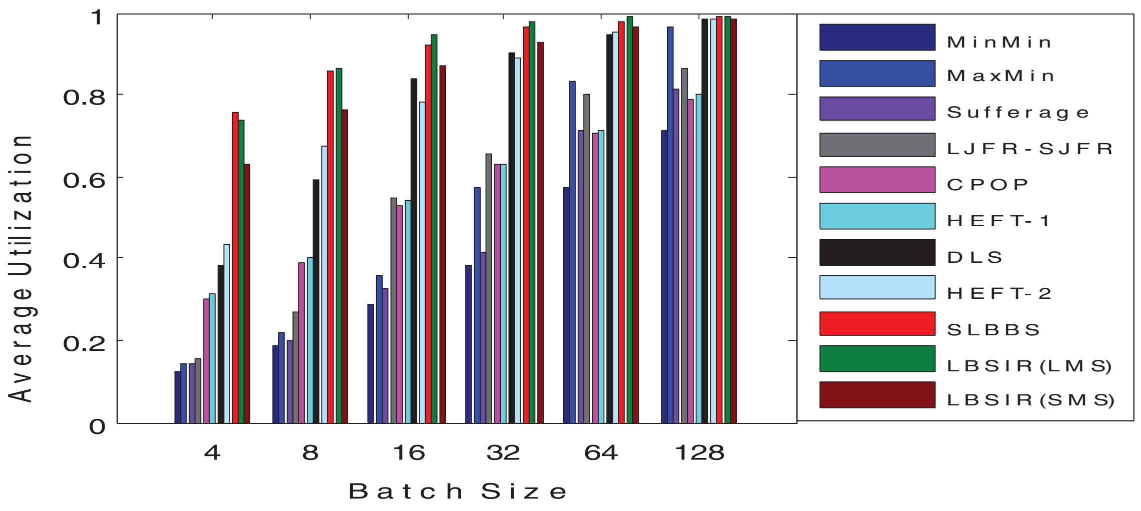

Fig. 13. Average utilization vs batch size ($K = 16$ & $L = 16$) with increasing load.

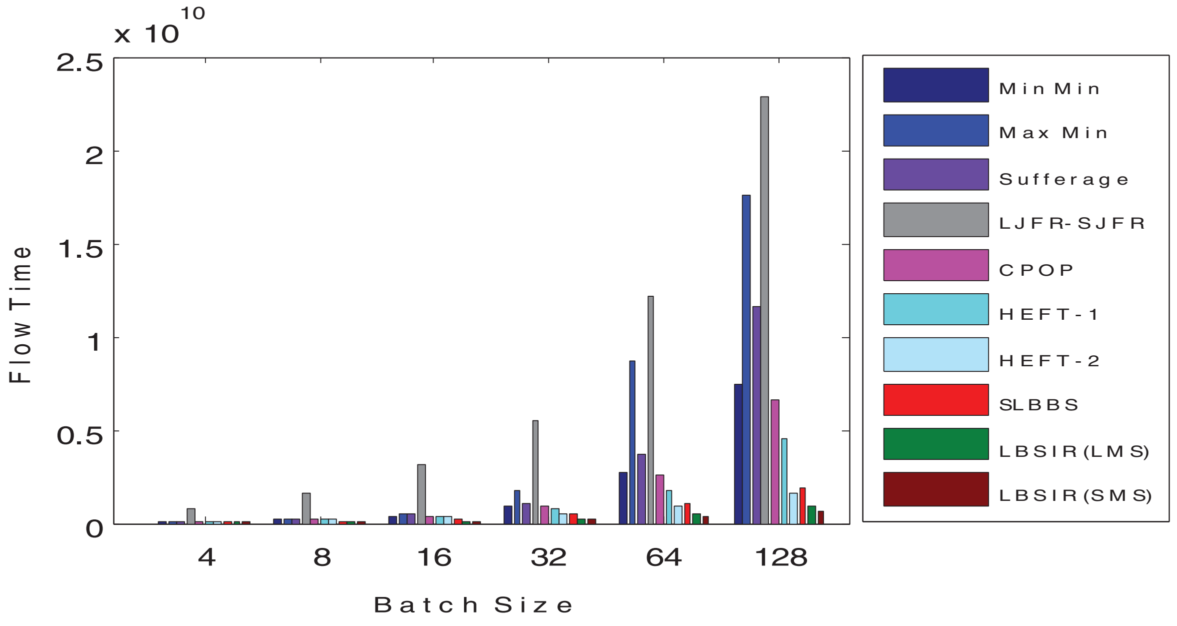

Fig. 14. Flow time vs batch size ($K = 16$ & $L = 16$) with increasing load.

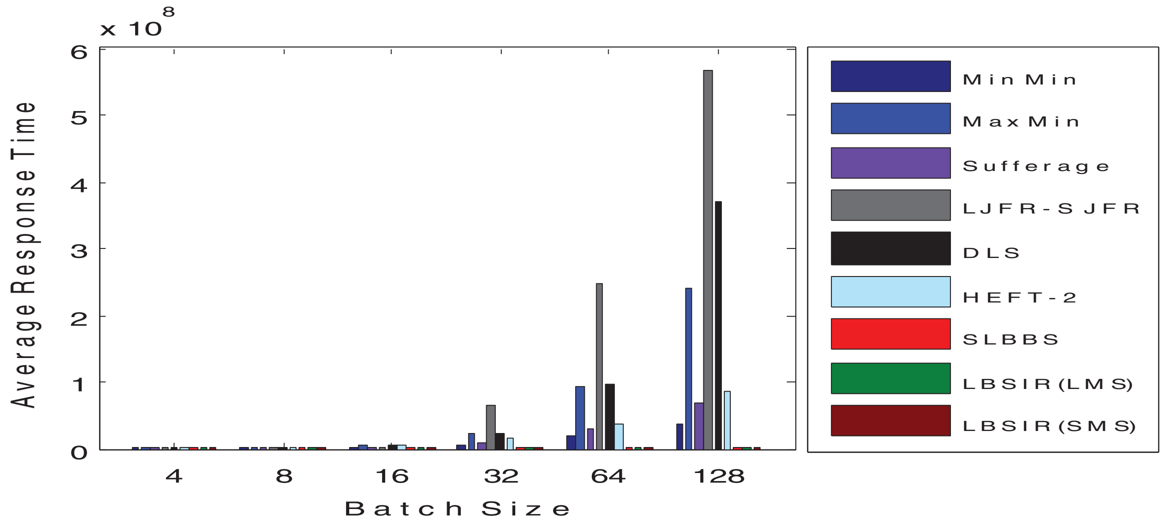

Fig. 15. Average response time vs batch size ($K = 16$ & $L = 16$) with increasing load.

in the size of the batch. The reason being that for a smaller batch, many machines remain idle as the number of jobs in the batch is smaller than the number of nodes resulting in the poor utilization for these strategies. As the batch size increases the utilization improves as more machines get occupied in the process. Unlike, Min Min, Sufferage, Max Min and LJFR-SJFR where the utilization increases many folds with the increasing batch size, LBSIR variants consistently maintain utilization in the range of 90–99% increasing only slightly with the increasing size of the batch (jobs) achieving some gain over SLBBS on average resource utilization due to idle slot reduction. Among strategies other than LBSIR, HEFT-2 and DLS perform the better with Min Min being the worst.

4. LBSIR (SMS) and LBSIR (LMS) prove to be winner being better than SLBBS even in the case of flow time with LBSIR (SMS) being the best. LBSIR (SMS) and LBSIR (LMS) using idle slot reduction achieve nearly 50% improvement over SLBBS on flow time. Among strategies other than LBSIR, HEFT-2 and HEFT-1 perform better than Min Min, Sufferage, Max Min and LJFR-SJFR while DLS being the worst is not included in the figure.
5. LBSIR (SMS), LBSIR (LMS) and SLBBS again provide the best response time as shown in Fig. 15 as they are based on exploiting the software parallelism at each depth level enabling the first few modules of every job to start simultaneously. Strategies other than LBSIR, Min Min perform better than Sufferage, HEFT-2, Max Min and LJFR-SJFR. CPOP with HEFT-1 performs the worst due to execution of the jobs in the batch in the order of one after the other.

The only instance in which LBSIR variants along with SLBBS performance lag others is when the batch size is not proportionate to the computational nodes available. In this case, parallelism in the batch becomes very low and each job gets assigned on the best single node chosen in the available set of computational nodes as the nodes are more than the total jobs.

5.2.2. *Varying the batch size with fixed load*

In this case the batch size has been increased from 4 jobs to 128 jobs along with decrease in the number of modules in each partition from 1024 to 32 in the same ratio. In this way, job level and sub job level parallelism are varying having total load fixed (total modules in the batch). The results are presented in Figs. 16–19. The other common input parameters considered here as follows:

Number of Nodes $(K) = 16$, Number of depth level in the jobs $(l) = 16$, Ready Time $(RT_k^l) = 0 - 10,000$, Inter Node distance between the nodes $=$ 1–100, Inter Module Communication $(IMC_{ipq}) =$ 1–1000, ETC $=$ 1–3,000,000, Machine Heterogeneity = high, Job Heterogeneity = high, Consistency = Consistent.

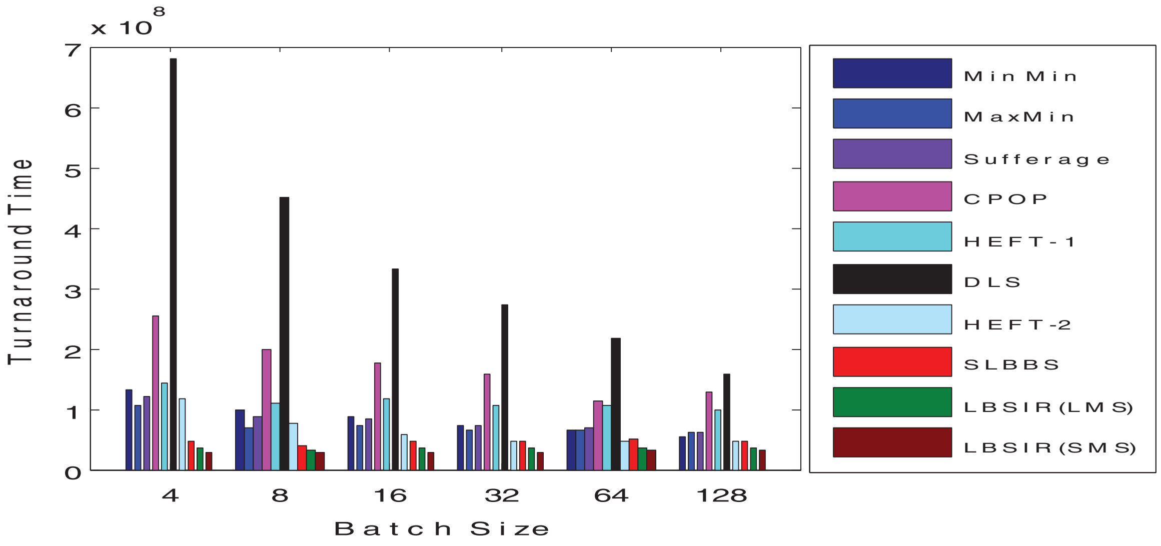

Fig. 16. Turnaround time vs batch size ($K = 16$ & $L = 16$) with fixed load.

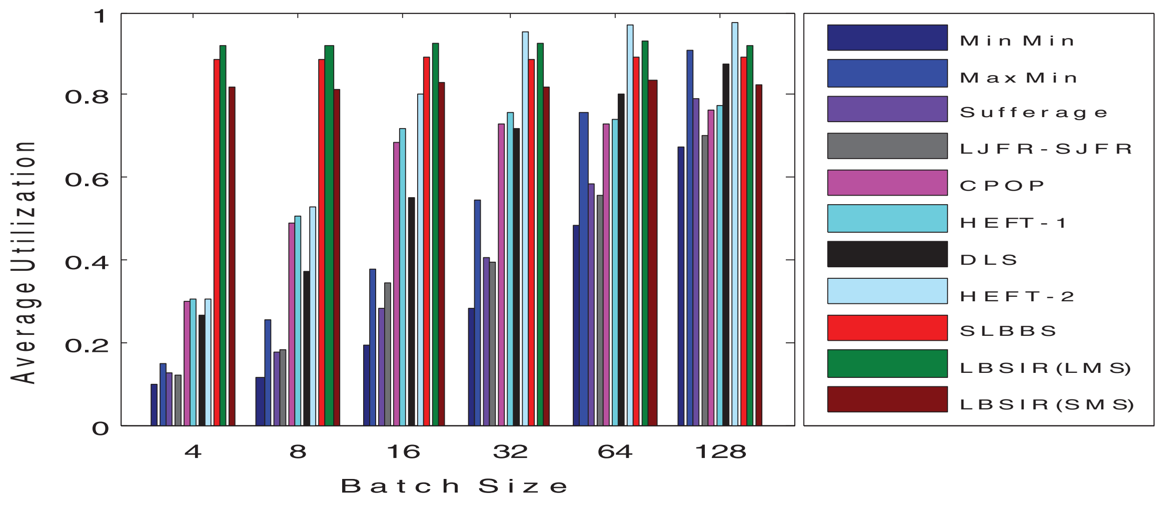

Fig. 17. Average utilization vs batch size ($K = 16$ & $L = 16$) with fixed load.

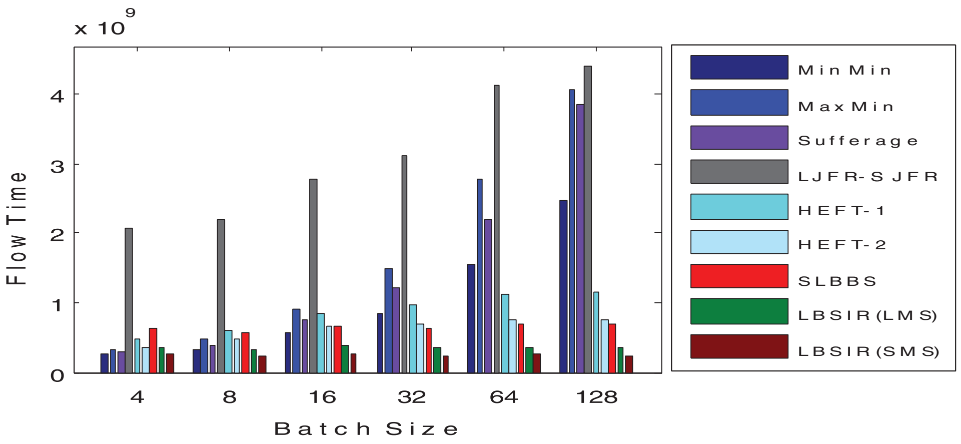

Fig. 18. Flow time vs batch size ($K = 16$ & $L = 16$) with fixed load.

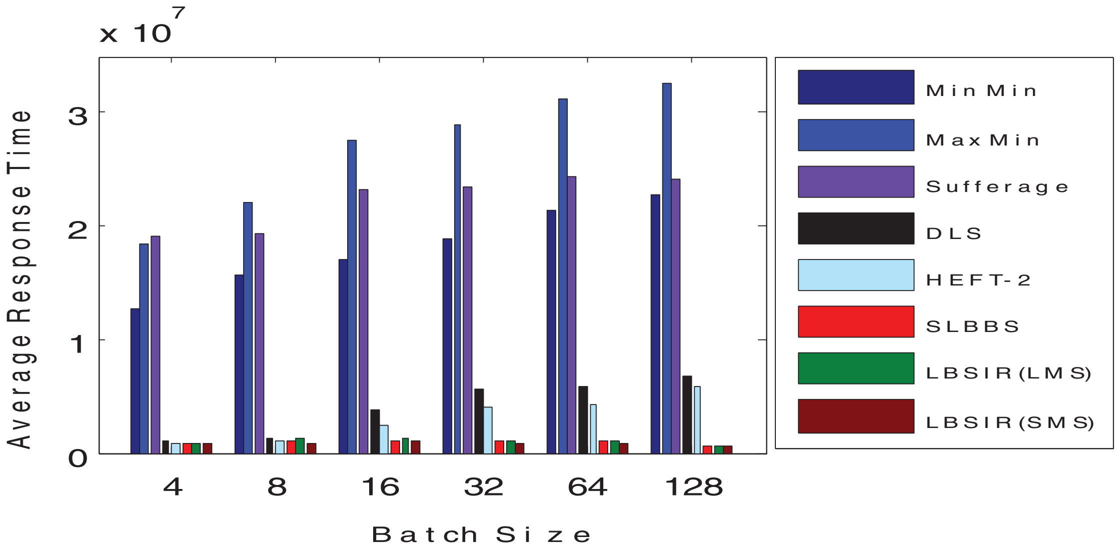

Fig. 19. Average response time vs batch size ($K = 16$ & $L = 16$) with fixed load.

Observations:

1. Turnaround time for Min Min, Sufferage, Max Min, LJFR-SJFR, HEFT-1, HEFT-2, DLS and CPOP keeps improving when the batch size is increased while keeping the number of nodes fixed. The reason behind this is that increasing the jobs in the batch will increase the job level parallelism. On other side, SLBBS, LBSIR (LMS), LBSIR (SMS) are nearly invariant to the increase in batch size with a fixed load as can be seen in Fig. 16. This is due to the fact that here the parallelism (Number of modules) in each partition remains the same maintaining the fixed load.
2. For all batch sizes from 4 to 128, LBSIR (SMS) followed by LBSIR (LMS) performs the best among all for the turnaround time. The performance of SLBBS is initially better than Min Min later becoming at par with that. HEFT-2 initially performs better than Max Min and later lags it in the performance when the batch size becomes bigger than 32.
3. Initially, average utilization for LBSIR variants is more than the DAG scheduling strategies viz. HEFT-2, HEFT-1, CPOP and DLS and even better than the batch scheduling strategies like Min Min, Sufferage, Max Min and LJFR-SJFR. The performance of LBSIR variants including SLBBS remains approximately unchanged while DAG scheduling strategies and batch scheduling strategies keep improving with increasing batch size and approach the performance of LBSIR as shown in Fig. 17 as explained earlier. Among strategies other than LBSIR, HEFT-2 performs the best with Min Min being the worst.
4. Flow time for heuristics other than LBSIR exhibits significant increase while LBSIR variants have a little increase with increase in batch size with fixed load. Performance order of the heuristics is same as in Section 5.2.1.
5. Average response time also increases with increasing batch size as response time depends on the number of jobs and their start times. LBSIR (SMS), LBSIR (LMS) and SLBBS again provide the best response time with the same reason as explained earlier. Strategies other than LBSIR, HEFT-2 followed by DLS performs better than Min Min, Sufferage, Max Min and LJFR-SJFR with CPOP and HEFT-1 is being worst.

5.2.3. *Varying the depth levels for a fixed batch size*

In this case, a batch possessing sufficient amount of parallelism has been considered for the study. Addition of more modules or jobs results in more parallelism implying better performance for LBSIR variants. The parallelism is varied by varying the number of depth levels in the jobs of the fixed batch (varying number of independent modules in the partitions) to observe the effect on the objective parameters as shown in Figs. 20–23. The input parameters considered here as follows:

Number of Nodes (K) = 32, Batch size = 256, Number of Modules in a Job (m) = 128, Ready Time (RT_k^l) = 0 − 1000, Inter Node distance between the nodes = 1–100, Inter Module Communication (IMC_{ipq}) = 1–3000, ETC = 1–3,000,000, Machine Heterogeneity = High, Job Heterogeneity = High, Consistency = Inconsistent.

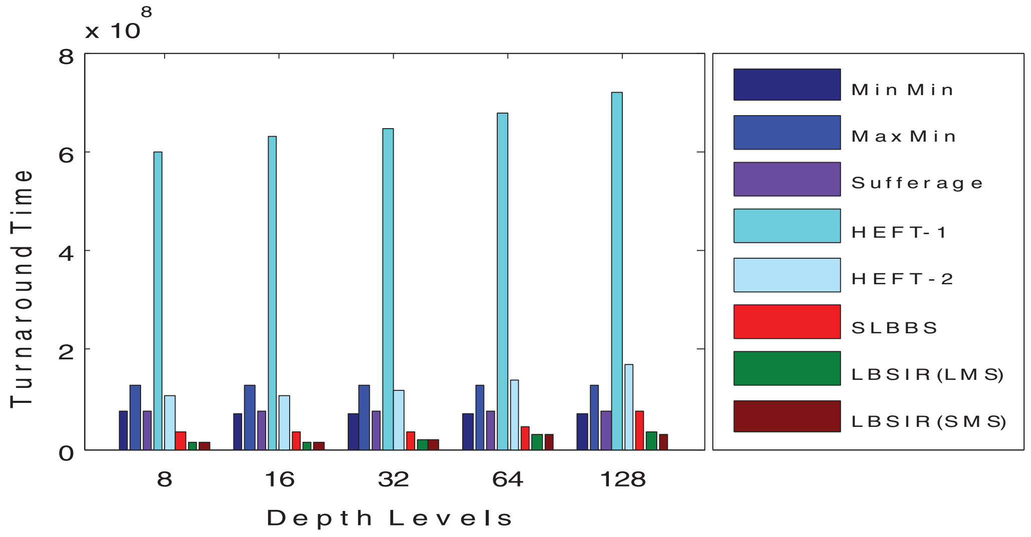

Fig. 20. Turnaround time vs depth levels ($N = 256$ & $K = 16$).

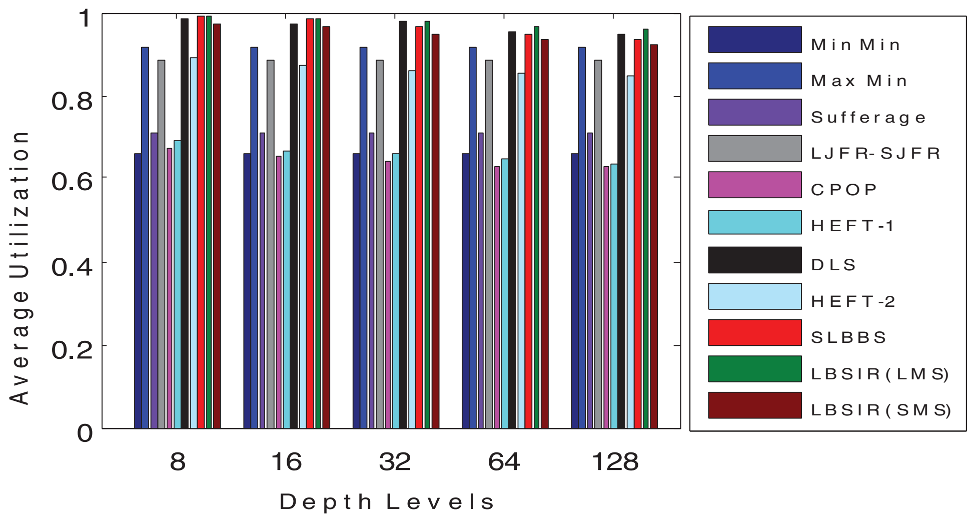

Fig. 21. Average utilization vs depth levels ($N = 256$ & $K = 16$).

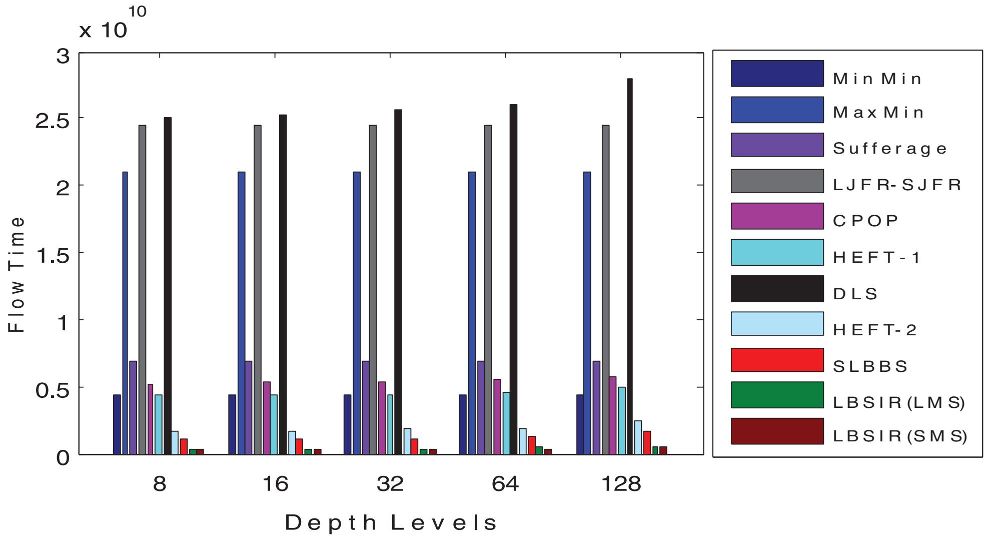

Fig. 22. Flow time vs depth levels ($N = 256$ & $K = 16$).

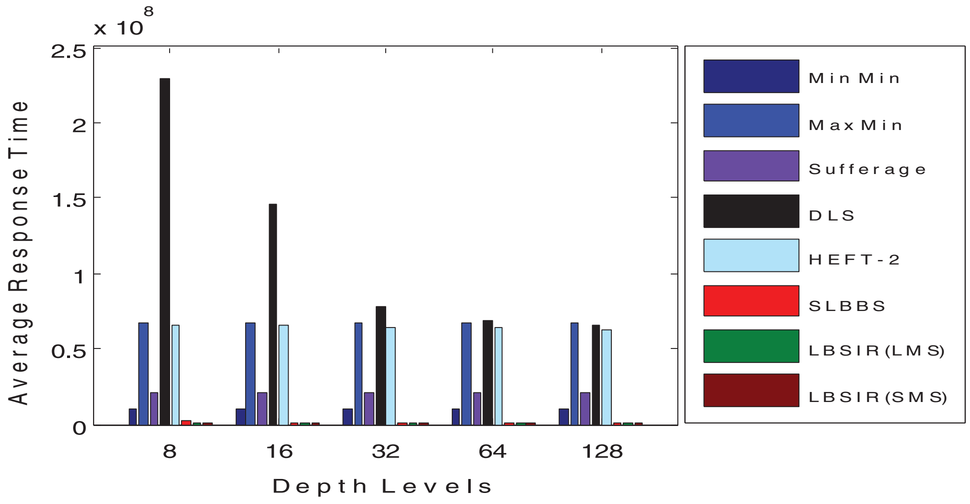

Fig. 23. Average response time vs depth levels ($N = 256$ & $K = 16$).

Observations:

1. As can be seen in Fig. 20, for decreasing parallelism for a fixed batch i.e. with the increasing number of depth levels the turnaround time for LBSIR variants and HEFT-1, HEFT-2, DLS and CPOP increases gradually. LBSIR (SMS) again proves to be the best for all depth levels on account of the turnaround time. Considering the case of depth level 128 for each job in the batch, the jobs have no parallelism as each job has 128 modules which now get executed only serially. Yet, LBSIR variants perform the best because for a batch of 256 jobs it still has 128 independent modules in each partition which can be run simultaneously. LBSIR (LMS) though slightly lagging LBSIR(SMS) performs better than Min Min, Max min, Sufferage, LLJFR-SJFR, HEFT-1, HEFT-2, DLS and CPOP for turnaround time. Performance of Min Min, Max Min, Sufferage and LLJFR-SJFR is the same as discussed earlier as they are independent of parallelism variation. DLS, CPOP and LJFR-SJFR are performing worst and not included in Fig. 20.
2. For the batch, for LBSIR (LMS), LBSIR(SMS), SLBBS, HEFT-1, HEFT-2, DLS and CPOP the average utilization slightly decreases by increasing the depth levels or decreasing parallelism with LBSIR (LMS) being the best and LBSIR (SMS) as second best as shown in Fig. 21. This degradation is approximately 1–4% for variation in the depth levels from 4 to 128 which is negligible.
3. Flow time only increases slightly with an increase in the depth levels for the strategies other than Min Min, Max Min, Sufferage and LLJFR-SJFR. LBSIR (LMS) and LBSIR (SMS) perform much better than the remaining strategies on flow time for all variations in the depth level as shown in Fig. 22.
4. As presented in Fig. 23, LBSIR (SMS), LBSIR (LMS) and SLBBS perform better than other considered strategies on average response time for all depth levels.
5. Performance of Min Min, Max Min, Sufferage and LLJFR-SJFR is same for all parameters viz. turnaround time, flow time, average utilization and average response time in all cases being independent of the parallelism variation.

It is to be noted that LBSIR variants perform extremely well when there is a proper mapping between the parallelism in the batch and the hardware parallelism. But even if the parallelism in the batch is reduced as in this case by increasing the depth levels for the same batch, it does affect the LBSIR variants due to reduction in parallelism but still if the batch size is proportional or at least equal to the number of nodes, it outperforms its peers.

5.3. Varying the number of computational nodes for a fixed batch size

In this section, the performance of LBSIR is observed by varying the number of computational nodes in the computational grid while keeping the batch size fixed. The results for the case are shown in Figs. 24–27 and the related observations are listed below. The remaining input parameters considered here as follows:

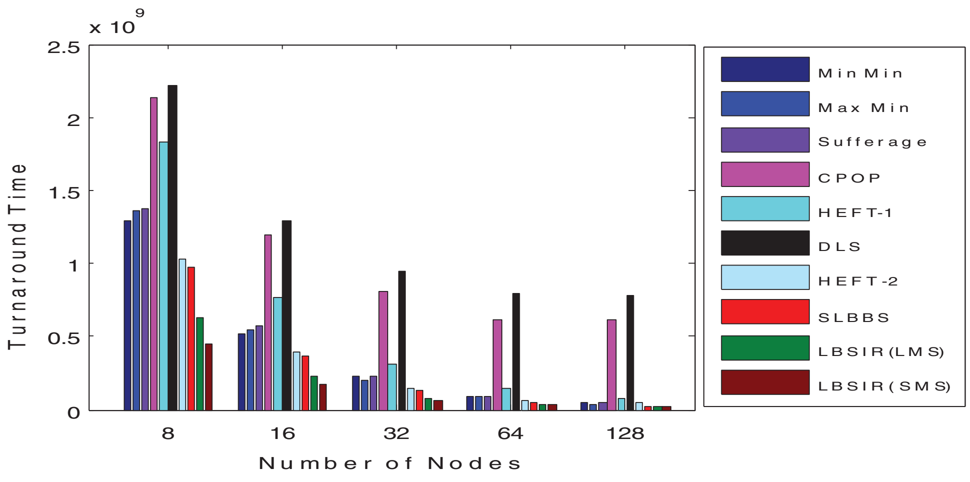

Fig. 24. Turnaround time vs number of nodes ($N = 64$ & $L = 16$).

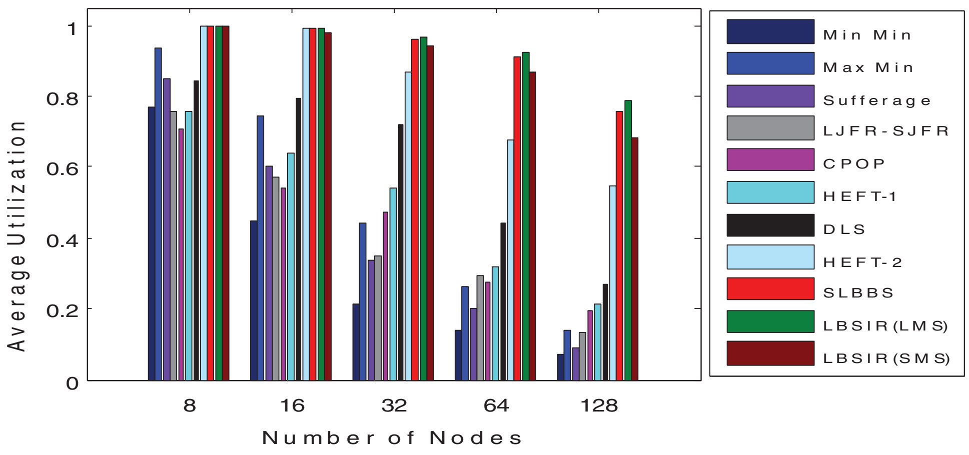

Fig. 25. Average utilization vs number of nodes ($N = 64$ & $L = 16$).

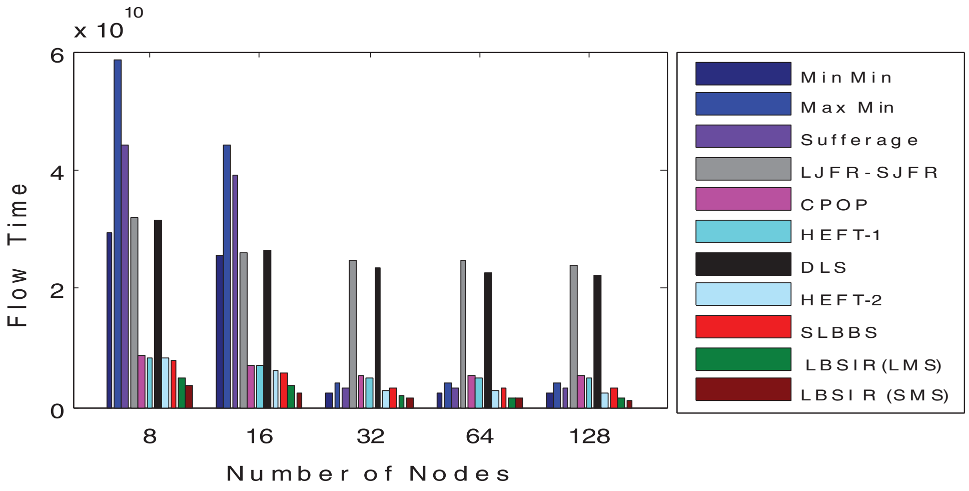

Fig. 26. Flow time vs number of nodes ($N = 64$ & $L = 16$).

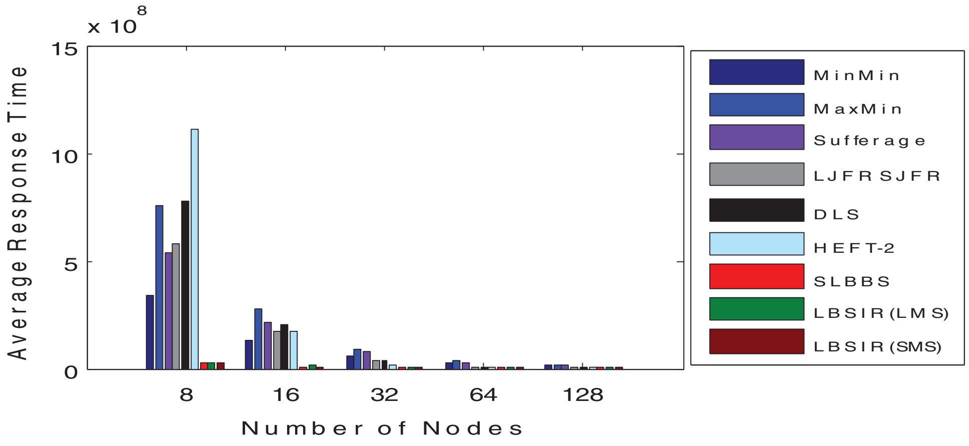

Fig. 27. Average response time vs number of nodes ($N = 64$ & $L = 16$).

Batch size = 64, Number of Modules in a Job (m) = 512, Number of depth level in the jobs (l) = 16, Number of modules of each depth level in the job = 32, Ready Time (RT_k^l) = 0 − 10, 000, Inter Node distance between the nodes = 1–1000, Inter Module Communication (IMC_{ipq}) = 1–3000, ETC = 1–3,000,000, Machine Heterogeneity = High, Job Heterogeneity = High, Consistency = Inconsistent.

Observations:

1. LBSIR (SMS) and LBSIR (LMS) perform better than Min Min, Max Min, Sufferage, LJFR_SJFR, HEFT-1, HEFT-2, DLS and CPOP for all parameters for all ranges of computational nodes for the considered batch.
2. All the parameters under consideration viz. turnaround time, average response time, flow time and average utilization of the batch

decrease by increasing the computational nodes for all considered heuristics as shown in Figs. 24–27. This reduction is very sharp on all four parameters for Min Min, Max Min, Sufferage and LJFR_SJFR. On the other hand, LBSIR (SMS) and LBSIR (LMS) too exhibit a decrease in the turnaround time and response time for an increase in the number of nodes but have a very marginal change for average utilization and flow time. Allocation strategy of LBSIR exhibits good results even using lesser number of computational nodes as compared to other strategies.

6. Conclusion

Scheduling of a batch of work flow applications represented as DAGs finds uses in many domains be it research or industry. This work proposes a centralized Level based Batch scheduling Strategy with Idle slot Reduction (LBSIR) working in two phases viz. allocation phase and idle slot reduction phase. The model considers the inter module communication constraints within the jobs of the batch consisting of a number of sub-jobs/modules for a computational grid with the objective of optimizing the turnaround time and the response time. Allocation phase begins by slicing the whole batch into a number of partitions as per the depth level followed by the assignment of sub-jobs/modules from all sorted partition to the best fit node in terms of the minimum possible execution time being done level wise. Further, after allocation of modules from each depth level, for the idle slots generated during the allocation phase an effort is made to reduce them by inserting the best fitting modules into these slots in the idle slot reduction phase. The strategy is quite suitable for use in the interactive applications as the average response time gets minimized.

Simulation study was performed to observe the benefit of idle slot reduction in LBSIR variants viz. LBSIR (LMS) and LBSIR (SMS) over various QoS parameters like turnaround time, average utilization, average response time and flow time. Further, performance evaluation has been carried out by comparing LBSIR (LMS) and LBSIR (SMS) with Min Min, Max Min, Sufferage, LJFR-SJFR, HEFT-1, HEFT-2, CPOP and DLS by varying the heterogeneity of the nodes and the jobs and consistency of the grid system using static ETC simulation benchmark followed by observing the effect of the communication requirements. Performance evaluation is also done to observe the effect of the variation in the parallelism at the depth levels and computational nodes for smaller as well as larger batch size. Simulation study reveals that the amalgamation of both phases in the LBSIR variants results in a significantly better performance than peers for all four considered QoS parameters for all cases under study.

References

Abrishami, S., Naghibzadeh, M., Epema, D.H.J., 2013. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. 29 (1), 158–169.

Arabnejad, H., Barbosa, J.G., 2014. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. 25 (3), 682–694.

Bajaj, R., Agrawal, D.P., 2004. Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. 15 (2), 107–118.

Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing system. J. Parallel Distrib. Comput. 61 (6), 810–837.

Cai, M.-C., Deng, X., Feng, H., Li, G., Liu, G., 2002. A PTAS for Minimizing Total Completion Time of Bounded Batch Scheduling, Integer programming and combinatorial optimization. Lecture Notes in Computer Science 2337, 304–314.

Canon, L.C., Jeannot, E., Sakellariou, R., Zheng, W., 2008. Comparative evaluation of the robustness of dag scheduling heuristics. In: Gorlatch, S., Fragopoulou, P., Priol, T. (Eds.), Grid Computing – Achievements and Prospects. Springer, pp. 73–84.

Daoud, M.I., Kharma, N., 2008. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. 68 (4), 399–409.

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., et al., 2014. Pegasus, a workflow management system for science automation. Future Gener. Comput. Syst. 44 (C), 17–35.

Deveci, M., Kaya, K., Uçar, B., Çatalyürek, Ü.V., 2015. Hypergraph partitioning for multiple communication cost metrics: model and methods. J. Parallel Distrib. Comput. 77, 69–83.

Foster, I., Kesselman, C., 2006. The Grid 2: Blueprint for a Future Computing Infrastructure, second ed. Morgan Kauffman, San Francisco, CA.

Garey, M.R., Johnson, D.S., 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

Hwang, J., Chow, Y., Anger, E., Lee, C., 1989. Scheduling precedence graphs in systems with inter processor communication times. SIAM J. Comput. 18 (2), 244–257.

Iverson, M., Ozguner, F., 1998. Dynamic competitive scheduling of multiple DAGs in a distributed heterogeneous environment. In: Seventh Heterogeneous Computing Workshop (HCW 98), pp. 70–78.

Iverson, M.A., Özgüner, F., Follen, G.J., 1995. Parallelizing existing applications in a distributed heterogeneous environment. In: 4th Heterogeneous Computing Workshop (HCW'95), pp. 93–100.

Izakian, H., Abraham, A., Snasel, V., 2009. Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. Neural Network World 19 (6), 695–710.

Karatza, H.D., 2001. Job scheduling in heterogeneous distributed system. J. Syst. Software 56, 203–212.

Kwok, Y.-K., Ahmad, I., 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. 31 (4), 406–471.

Leah, E., Halldórsson, M.M., Levin, A., Hadas, S., 2006. Weighted sum coloring in batch scheduling of conflicting jobs, Approximation, randomization and combinatorial optimization. Algorithms and techniques, Lecture Notes in Computer Science 4110, 116–127.

Lee, Y.C., Han, H., Zomaya, A.Y., Yousif, M., 2015. Resource-efficient workflow scheduling in clouds. Knowledge-Based Syst. 80, 153–162.

Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F., 1999a. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. 59 (2), 107–131.

Maheswaran, M., Braun, T.D., Siegel, H.J., 1999b. Heterogeneous distributed computing. Encyclopedia of Electrical and Electronics Engineering. John Wiley, pp. 679–690.

Maheswaran, M., Siegel, H.J., March 30, 1998. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In: Proceedings of the 7th Heterogeneous Computing Workshop (HCW '98), Orlando,Florida.

Melnyk, A., 2013. Multiple DAGs scheduling with deadline driven coordinator in grid. In: Second International Conference "Cluster Computing". Lviv,Ukraine, pp. 3–5.

Papazachos, Z.C., Karatza, H.D., 2010. Performance evaluation of bag of gangs scheduling in a heterogeneous distributed system. J. Syst. Software 83, 1346–1354.

Ran, M., Han, X., 2009. APTAS for minimizing total completion time of batch scheduling under tree precedence constraints. In: WASE International Conference on Information Engineering. IEEE.

Ren, F.L., Yu, J., 2012. Multiple DAGs scheduling based on lowest transportation and completion time algorithm on the cloud. In: Seventh IEEE Conference (ChinaGrid), pp. 33–35.

Ruyan, F., Tian, J., Yuan, J., 2009. On-line scheduling on an unbounded parallel batch machine to minimize makespan of two families of jobs. J. Sched. 12 (1), 91–97 Springer Science+Business Media, LLC.

Sakellariou, R., Zhao, H., 2004. A hybrid heuristic for DAG scheduling on heterogeneous systems. In: Proceedings of the 18th International IEEE Symposium on Parallel and Distributed Processing.

Saovapakhiran, B., Michailidis, G., Devetsikiotis, M., 2011. Aggregated-DAG scheduling for job flow maximization in heterogeneous cloud computing. In: IEEE GLOBECOM Proceedings.

Shahid, M., Raza, Z., 2014. Level based batch scheduling strategies for computational grid. Int. J. Grid Util. Comput. 5 (2), 135–148 Interscience Publisher.

Shi, Z., Dongarra, J.J., 2006. Scheduling workflow applications on processors with different capabilities. Future Gener. Comput. Syst. 22 (6), 665–675.

Sih, G.C., Lee, E.A., 1993. A compile time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. 4 (2), 175–186.

Topcuoglu, H., Hariri, S., Wu, M.Y., 2002. Performance effective and low complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst.

Vidyarthi, D.P., Sarker, B.K., Tripathi, A.K., Yang, L.T., 2009. Scheduling in Distributed Computing Systems. Springer ISBN 978-0-387-74480-3.

Wang, S.-D., Hsu, I.-T., Huang, Z.-Y., 2005. Dynamic scheduling methods for computational grid environments. In: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05), vol. 1, pp. 22–28.

Xhafa, F., Abraham, A., 2010. Computational models and heuristic methods for grid scheduling problems. Future Gener. Comput. Syst., 26 (4), pp. 608–621 Elsevier.

Xhafa, F., Barolliz, L., Durresi, A., 1 March 2007a. Batch mode scheduling in grid systems. Int. J. Web Grid Serv. 3 (1), 19–37.

Xhafa, F., Carretero, J., Barolli, L., Durresi, 2007b. An immediate mode scheduling in grid systems. Int. J. Web Grid Serv. 3 (2), 219–236.

Yang, J., Xu, H., Pan, L., Jia, P., Long, F., Jie, M., 2011. Task scheduling using bayesian optimization algorithm for heterogeneous computing environments. Appl. Soft Comput. 11 (4), 3297–3310.

Yang, T., Gerasoulis, A., 1994. DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Trans. Parallel Distrib. Syst. 5 (9), 951–967.

Zhang, Y., Koelbel, C., Cooper, K., 2009. Batch queue resource scheduling for workflow applications. In: IEEE International Conference on Cluster Computing and Workshops, pp. 1–10.
Zhao, H., Sakellariou, R., 2006. Scheduling multiple DAGs onto heterogeneous systems. In: 20th International, Parallel and Distributed Processing Symposium (IPDPS), pp. 125–134.
Zhu, L., Sun, Z., Guo, W., Jin, Y., Sun, W., Hu, W., 2007. Dynamic multi DAG scheduling algorithm for optical grid environment. In: Proc. of SPIE, Vol. 67841F, pp. 1–11.
Zhu, Y., Ni, L.M., 2013. A Survey on Grid Scheduling Systems Technical report SJTU_CS_TR_200309001. Department of Computer Science and Engineering, Shanghai Jiao Tong University.

Mohammad Shahid is an Assistant Professor in the Department of Commerce, Aligarh Muslim University, India. He has submitted his PhD thesis to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, India. He did his M. Tech in Computer Science and Technology from the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi and MCA in Computer Science and Applications from Aligarh Muslim University, Aligarh, India. His research interests are Parallel and Distributed Systems and Grid Computing. You can contact him at: Department of Commerce, AMU, Aligarh 202002, India.

Zahid Raza is an Assistant Professor of School of Computer and Systems Sciences, Jawaharlal Nehru University, India. He did his PhD in Computer Science from Jawaharlal Nehru University, New Delhi, India, M. Tech in Computer Science from BIT, Mesra, Ranchi, India and M.Sc. in Electronics from JNVU, Jodhpur, India. Prior to joining JNU, he served as a Lecturer in Banasthali Vidyapith University, Rajasthan, India. He has published a number of peer-reviewed papers and his research interest is in the area of Parallel and Distributed Systems, Grid Computing and Cloud Computing. You can contact him at: Lab-02, School of Computer and Systems Sciences, JNU, New Delhi 110067, India.

Mohammad Sajid received his M. Tech. in Computer Science and Technology after completion of MCA from Jawaharlal Nehru University, New Delhi, India. He is currently pursuing PhD in the School of Computer and Systems Sciences, Jawaharlal Nehru University, India. His research interests include High Performance Computing, Scheduling/Stochastic Scheduling, Evolutionary Algorithms and Multi-objective Evolutionary Algorithms. You can contact him at: Lab-02, School of Computer and Systems Sciences, JNU, New Delhi 110067, India.