Engineering Science and Technology, an International Journal 18 (2015) 256–269

HOSTED BY

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: http://www.elsevier.com/locate/jestch

Full length article

Adaptive workflow scheduling in grid computing based on dynamic resource availability

Ritu Garg*, Awadhesh Kumar Singh

Computer Engineering Department, National Institute Of Technology, Kurukshetra, Haryana, India

ARTICLE INFO

Article history:
Received 13 October 2014
Received in revised form
19 December 2014
Accepted 5 January 2015
Available online 4 February 2015

Keywords:
Grid computing
DAG grid workflow
Adaptive workflow scheduling
Re-scheduling
Resource monitoring

ABSTRACT

Grid computing enables large-scale resource sharing and collaboration for solving advanced science and engineering applications. Central to the grid computing is the scheduling of application tasks to the resources. Various strategies have been proposed, including static and dynamic strategies. The former schedules the tasks to resources before the actual execution time and later schedules them at the time of execution. Static scheduling performs better but it is not suitable for dynamic grid environment. The lack of dedicated resources and variations in their availability at run time has made this scheduling a great challenge. In this study, we proposed the adaptive approach to schedule workflow tasks (dependent tasks) to the dynamic grid resources based on rescheduling method. It deals with the heterogeneous dynamic grid environment, where the availability of computing nodes and links bandwidth fluctuations are inevitable due to existence of local load or load by other users. The proposed adaptive workflow scheduling (AWS) approach involves initial static scheduling, resource monitoring and rescheduling with the aim to achieve the minimum execution time for workflow application. The approach differs from other techniques in literature as it considers the changes in resources (hosts and links) availability and considers the impact of existing load over the grid resources. The simulation results using randomly generated task graphs and task graphs corresponding to real world problems (GE and FFT) demonstrates that the proposed algorithm is able to deal with fluctuations of resource availability and provides overall optimal performance.

1. Introduction

Recently, the rapid development of networking technology and web has led to the possibilities of using large number of geographically distributed heterogeneous resources owned by different organizations. These developments have led to the foundation of new paradigm known as *Grid Computing* [9,15]. Grid Computing is a type of parallel and distributed system that involves the integrated and collaborative use of resources depending on their availability and capability to satisfy the demands of researchers requiring large amount of communication and computation power to execute advanced science and engineering applications. Precedence constrained parallel applications (workflows) are one of the typical application models used in scientific and engineering fields requiring large amount of bandwidth and powerful computational resources. To achieve the promising potential of distributed resources, effective and efficient scheduling algorithm is important. The grid scheduler acts as an interface between user and distributed grid resources. The workflow scheduling in grid is one of the key challenges, which deals with assigning workflow tasks to the available grid resources while maintaining the task precedence (dependency) constraints and to meet the quality of service (QoS) demands of the user like minimizing the overall execution time.

In general, scheduling tasks on distributed grid resources belongs to a class of NP-hard problems [16]. So heuristics or approximations are the preferred options to obtain near optimal solutions. Many heuristics have been devoted to this problem as discussed in literature [2,7,21,30] considering that accurate prediction is available for computation cost and communication cost of resources. However, in real environment, it is difficult to accurately predict the values due to heterogeneous and dynamic

* Corresponding author.
E-mail addresses: ritu.59@gmail.com (R. Garg), aksinreck@rediffmail.com (A.K. Singh).

Peer review under responsibility of Karabuk University.

http://dx.doi.org/10.1016/j.jestch.2015.01.001

characteristics of the grid environment. The dynamicity of the grid resources is due to both the network connectivity and computational nodes.

As the grid resources are not dedicated, and can be used by the other users simultaneously, which leads to load variations on resources. The local tasks have more priority and handled first in comparison to grid tasks. The fluctuations in the resource availability (computing speed of the host and links bandwidth) due to resource's local loads and competition from other users cause the original schedule to become sub optimal. If for instance, load of the processor increases, the execution time of the tasks assigned to it will increase. Further, sudden increase in link load, increases the data transfer time between computers where dependent tasks resides. In case of grid applications especially for long running jobs (days or weeks), the performance degradation caused by load over resources is unacceptable. It leads to the necessity of relocating the tasks to other resources. Hence, it is a key challenge to maintain an application performance during its execution, if their resources suddenly receive high workload.

In order to ensure high performance in dynamic and unreliable grid environment, we considered the adaptive scheduling [19] where scheduling policy change dynamically as per the previous and current behavior of the system to cope with the variations in the resource availability. Here, initial scheduling of all the tasks is performed statically and then rescheduling of unexecuted tasks is performed when required. The ability to discover and monitor the status of resources at run time is fundamental for the adaptive operation of the grid here.

In this paper, we proposed a novel Adaptive Workflow Scheduling (AWS) algorithm for grid applications consisting of workflow tasks (dependent tasks) to meet the performance requirements based on QoS information like availability along with the accessibility of the resources as indicated by service level agreement (SLA). It considers the processors (computing speed) and network links (bandwidth) availability by monitoring the load over non-dedicated grid processing nodes and network links. The procedure involves static task scheduling, periodic resource monitoring and rescheduling the remaining unexecuted tasks in order to deal with changes/fluctuations occurring at run time and to achieve minimum execution time (makespan) of the workflow grid application. The procedure of proposed AWS differs from other approaches in literature by considering the dynamic availability of resources, both computing nodes and communication links due to existence of local load or load by other users. It considers (i) Degradation of resource performances especially computing speed of nodes and network links bandwidth as per SLA, as a source for triggering rescheduling. (ii) Evaluate the benefits of rescheduling considering cost of reevaluating the schedule and overhead due to transfer of data. (iii) Availability of newly added resources.

The AWS algorithm is efficient one as it achieves minimum execution time of the application with the help of rescheduling the computation away from: overloaded computing nodes, nodes with overloaded communication links that can slow down the computation and it also considers the addition of new nodes to increase the performance of the application. Further, algorithm also provides load balancing by supporting rescheduling of tasks form overloaded resources.

The rest of the paper is organized as follows. We briefly mention the related work in Section 2. Section 3, describes the mathematical model, including the resource model, task model along with the problem definition. Thereafter, in Section 4, we explain the proposed Adaptive workflow Scheduling algorithm. Section 5 includes the pseudo code of the algorithm and detailed example. Section 6 discusses the simulation and result analysis. Finally section 7, gives the conclusion.

2. Related work

The problem of scheduling in grid, for workflow (DAG-based) tasks has already been addressed in the literature. Most of the related work attempt to achieve the minimum execution time (makespan) on heterogeneous grid environment. To schedule scientific workflow applications, Heterogeneous Earliest Finish Time (HEFT) [29], is the most popular list based heuristic. It orders the workflow tasks based on priorities and then assign them to suitable resources to achieve high performance. Similarly another list based heuristics Min–min, Max–Min [22], Critical-Path-on-a-processor (CPOP) [29] are studied to achieve high performance. The PCH algorithm [5] uses a hybrid clustering-list-scheduling strategy, where tasks with heavy communication cost are grouped together and assigned to the same resource in a cluster. It aims to reduce the schedule length by reducing the communication cost. Further, the paper [14] describes the design, development and evolution of the Pegasus Workflow Management System, which maps abstract workflow descriptions onto distributed computing infrastructures in order to achieve reliable and scalable workflow execution.

The critical issue in list heuristics for DAG scheduling is the accurate prediction for both the computation and the communication costs. However, in a real grid environment, system is less reliable and more dynamic: individual resource capability varies over time due to internal or external factors, thus, it is difficult to estimate these values accurately. To deal with resource dynamicity, two types of approaches are proposed in literature: dynamic scheduling and adaptive scheduling. In dynamic scheduling, all tasks are scheduled at run time while in adaptive scheduling; an advance static schedule is generated using available estimates and schedules responds to changes at run time with the help of rescheduling. GrADS [4] is the typical rescheduling mechanism which schedules workflow grid tasks based on three popular heuristics of Min–min, Max-min and Suffrage. It focuses on iterative workflows, allowing system to perform rescheduling at each iteration. Rescheduling is activated by contract violation between user and resource provider. If the performance is expected by migration, then unexecuted jobs are migrated to new mapped resources.

Other rescheduling methods proposed are AHEFT [31] and SLACK [26]. In AHEFT, author proposed an adaptive rescheduling algorithm based on static strategy. Here the workflow planner adapts to grid dynamics with the help of run time executor. Rescheduling is performed on the basis of FEA, which is the earliest time when output file is available for dependent tasks, if performance increase is there. While in SLACK [26] it is using the concept of spare time, which does not affect the schedule length of the workflow. If execution time of task goes beyond the spare time then only selective rescheduling event is triggered. The major drawback of these studies is that rescheduling is performed on periodic basis on performance degradation. Moreover, during rescheduling initial information of grid resources is used, without reflecting dynamically updated information.

Another approach to deal with dynamic performance changes of grid resources is proposed in [6] which uses the path clustering heuristic (PCH) to generate the initial schedule and then round based approach is used to reschedule. On each round some of the tasks based on a criterion are sent to execute and then performance of the resource is measured in that round. If performance is below the threshold value, then the algorithm reschedules the non executed tasks. In [11] the author proposed the adaptive list scheduling service algorithm (ALSS) for workflow tasks in order to deal with dynamic nature of service oriented grid environment. Low overhead rescheduling is considered only for services on the

critical path of the workflow. In paper [27], three algorithms namely incremental algorithm, divide and conquer algorithm and genetic algorithm have been developed for deciding when and where to reschedule tightly coupled parallel applications in dynamic grid environment. A rescheduling plan consists of potential points in application execution for rescheduling and schedule of resources for execution between two consecutive rescheduling points. Both application and resource dynamics are considered in this paper. In these studies, the performance of the executing tasks is monitored continuously; if it is degraded rescheduling is performed without considering the dynamically updated resource status.

Further, there are another set of the approaches [3,28] which deals with dynamicity of resources, for executing workflow tasks in grid. They consider the effect of presence of load (local loads or load by other users) on the availability of resources (processors or links) for user's application tasks. In Ref. [28], the author schedules the tasks considering the exiting load over processors and network links. Similarly, in Ref. [3], the author considers the resource availability (especially bandwidth) at run time and initiates task migration after evaluating its benefits.

In [10], the author proposed the adaptive scoring job scheduling algorithm to schedule independent set of tasks composing of compute intensive and data intensive jobs. When the job scheduler receives the new jobs, it assigns them to the most appropriate resources according to their cluster score. Local update and global update are used to get the newest status (cluster score) of resources in Grid environment. However, it does not handle the performance degradation of the already scheduled jobs due to dynamic load. Paper [25] introduced the dynamic critical path based workflow scheduling algorithm for grid namely DCP-G that provides efficient schedule in static environment. Further, it adapts to the dynamic grid environment, where resource information is updated after fixed interval and rescheduling (Re-DCP-G) is performed if necessary. It also describes the outlining of hybrid heuristic algorithm that combines the features of the adaptive scheduling technique with meta-heuristics for optimizing execution time and cost in dynamic cloud environment. In paper [24], rescheduling approach is used for large scale distributed system (Re-LSDS) to support fault tolerance and resilience.

To the best of our knowledge, proposed AWS is different from others in considering the issues of: Effects of existing load on processors and network links on the computation and communication cost, consideration of dynamic resource availability as the source for triggering rescheduling and availability of newly added resources. Here the initial static schedule of tasks is generated by considering the availability of resources (computing speed of the hosts or links bandwidth) after considering the existing local load. Then resource discovery and monitoring component periodically monitors the changes in the availability of the resources (due to change of load or registering new resources) and triggers the rescheduling phase for remaining unexecuted tasks to escort with dynamically updated resource information.

3. Problem statement and preliminaries

In this section, we describe the formal definitions for grid resources, workflow tasks and problem considered in this study.

3.1. Resource model

In our grid computing model, we considered the groups of interconnected geographical distributed resources, where groups may be LANs, clusters, or individual nodes as shown in Fig 1. Here each resource in the group is autonomous and associated with different processing capabilities.

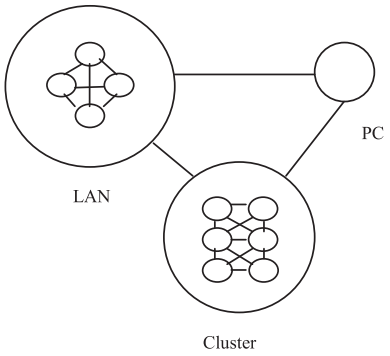

Fig. 1. Example of grid computing network model.

At each group, there is a local scheduler which is responsible for distributing tasks of workflow application among the resources of the group. Further, there is a global scheduler responsible for distributing the tasks among the different groups. The proposed algorithm distributes the tasks at the local level to all the available resources.

We consider the grid network $G_r = (R, L)$, consisting of m number of fully interconnected heterogeneous and dynamic computing nodes represented as set $R = \{r_1, r_2 \ldots r_m\}$. Fully interconnection means that a route exists between any two processors and inter-processor communication is heterogeneous in nature. The terms node, host and processor within this article, represents the computing nodes and are used interchangeably. The resource model may have different properties as in [28]:

- Processing capacities/computing speed (*CPU_speed*) of the host in terms of MIPS (Millions of instructions per second) and
- Bandwidth linkage between any two processors, where $BW_{s,j}$ is used to represents the bandwidth/data transfer rate of the link between processors r_s and r_j in terms of Mbps.
- Existing workload on processor (*load*), which is the important parameter used to decide the allocation of tasks to the processor. The more is the pre-assigned workload, the higher will be the execution time of the task.
- Existing load on the network (*linkload*), which is another important parameter affecting the communication time between dependent tasks. Increase in the network load, increases the data ready time for the task over specific host, thus increases the overall makespan of the application.

We consider that each processor has queue of grid tasks in first come first served (FCFS) fashion waiting for their turn to execute. The workflow execution manager inserts the ready tasks to this queue and task executions once started on the processor is considered to be non pre-emptive. However in the dynamic environment of the grid, the availability of the computing speed of the processor varies with time due to the existence of local workload or load by other users, sharing the processor execution with policy supported by processor operating system. Local tasks have higher priority than grid workflow tasks. Thus, the *available computing speed* (*ACPU_speed*) of the processor for grid tasks is the excess computing power of the processor which is available for current task execution.

$$ACPU\_speed_j = CPU\_speed_j * (1 - load_j) \quad (1)$$

Similarly, load over non dedicated network links varies with time [22] and it affects the communication time between dependent tasks of the workflow. Thus, load over network links affect the overall makespan of the application. So, *available bandwidth (ABW)* of the link for inter process communication of grid tasks is represented by

$$ABW_{s,j} = BW_{s,j}*(1 - linkload_{s,j}) \quad (2)$$

3.2. Workflow model

Many important grid applications in e-science and e-business fall in the category of workflow applications; a directed acyclic graph (DAG) is the standard way to represent a workflow. In this representation, parallel application/job consisting of bag of tasks with precedence constraints (dependency) and is represented as DAG [17,29], $G_t = (T, E)$ where:

- T is the set of vertices representing n different tasks $t_i \in T$, $(1 \le i \le n)$ that can be executed on any available processor.
- E is the set of directed edges $e_{ij} = (t_i, t_j) \in E$, $(1 \le i \le n, 1 \le j \le n, i \ne j)$ representing dependencies among the tasks t_i and t_j, indicating a task t_j cannot start its execution before t_i finishes and send all the required output data to task t_j.
- The weight $w(t_i)$ is assigned to task t_i represents the size/computing demand of ith task and expressed as number of instructions (MI) to be executed by the task and
- Weight $w(e_{ij})$ assigned to edge e_{ij} represents the amount of data required to be transfer from task t_i to t_j if they are not executed on the same resource.

Fig. 2 shows an example of DAG. Each node weight represents the size of task ($w(t_i)$) and each edge weight represents the inter-tasks communication data ($w(e_{ij})$). The task without any predecessor is called an *entry task* and the task without any successor is called an *exit task*. We assume that a DAG has one entry and one exit task. If a DAG has more than one entry or exit tasks then, one entry and exit task is added to DAG along with edges connecting them to original entry and exit tasks with zero weights respectively. The

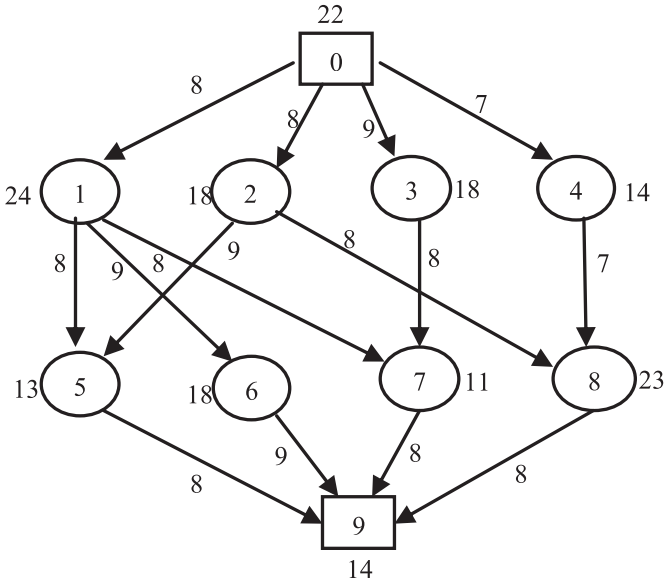

Fig. 2. An example Workflow.

Table 1
Parameters used in the mathematical models.

Parameters	Definitions
$G_t = (T, E)$	DAG representing Workflow application
n	Number of tasks
t_i	i^{th} computing tasks
$e_{ij} = (t_i, t_j)$	Dependency edge from task t_i to t_j
$w(t_i)$	Size of ith task
$w(e_{ij})$	Size of data transfer corresponding to edge e_{ij}
$G_r = (R, L)$	Grid network consisting of computing nodes and links between them
m	Number of computing nodes/processors
r_j	j^{th} computing node
CPU_speed_j	Computing speed/capacity of jth node (MIPS)
$BW_{s,j}$	Bandwidth of link between r_s and r_j nodes (Mbps)
$ACPU\_speed_j$	Available computing speed/capacity of jth node
$load_j$	Existing local load over jth node
$ABW_{s,j}$	Available bandwidth of link between r_s and r_j nodes
$linkload_{s,j}$	Existing local load over link between r_s and r_j nodes

parameters used here in the mathematical modeling are shown in Table 1.

Before giving the formal definitions of the research issues, the important parameters and notations used for scheduling (inspired by our previous work [17,18]) are formally defined as follows:

- Execution time or Computation cost of task

$$ET(t_i, r_j) = \frac{w(t_i)}{ACPU\_speed_j} \quad (3)$$

where $ET(t_i,r_j)$ represents the execution time of task t_i on resource r_j with $ACPU\_speed_j$ representing available computing capacity in MIPS for grid tasks.

- The Estimated Start Time

$$EST(t_i, r_j) = \begin{cases} RAT(r_j), & \text{if } t_i \text{ is entry task} \\ \max\{RAT(r_j), DRT(t_i, r_j)\}, & \text{otherwise} \end{cases} \quad (4)$$

where $EST(t_i,r_j)$ represents the estimated start time of tasks t_i on resource r_j. Here $RAT(r_j)$ represents the resource available time and $DRT(t_i,r_j)$ represents the data ready time for task t_i over resource r_j and is defined as

$$DRT(t_i, r_j) = \max_{t_p \in pred(t_i)} \begin{cases} EFT(t_p, r_s) & \text{if } r_s = r_j \\ \left\{EFT(t_p, r_s) + C_{p,i}^{s,j}\right\}, & \text{otherwise} \end{cases} \quad (5)$$

where $pred(t_i)$ is the set of immediate predecessor tasks of task t_i.

- The Estimated Finish Time

$$EFT(t_i, r_j) = EST(t_i, r_j) + ET(t_i, r_j) \quad (6)$$

where $EFT(t_i,r_j)$ represents the estimated finish time of a task t_i on some resource r_j.

- Overall *makespan* of grid workflow

$$makespan = EFT(t_{exit}, r_j) \quad (7)$$

where $EFT(t_{exit},r_j)$) represents the estimated finish time of the exit task on some resource r_j, considering the start time of the workflow as zero.

3.3. Problem statement

The problem addressed in this study, is the scheduling of set of precedence constraint tasks (Workflow tasks) on to the set of dynamic heterogeneous resources (hosts and network links). The scheduling performs the mapping of *n* number of workflow tasks to the set of *m* number of available grid resources/processors with the aim to achieve high and stable execution performance (i.e. minimizing the makespan) without violating the precedence constraints (dependency constraint). The makespan is the amount of time from start of the entry task to the completion of the exit task shown by Eq. (7). Since the grid resources are highly dynamic and non-deterministic in nature so it is hard to guarantee the makespan minimization when there are fluctuations in resource availability. To overcome this limitation an adaptation strategy is required. We consider two common situations and adaptation to them.

- Performance fluctuation of resources: The performance of grid resources is sensitive to the local workload as it degrades the performance of workflow application. In order to adapt to this situation, we considered the rescheduling when the local load of resources (hosts, links) increases beyond threshold. With the help of rescheduling, we are able to achieve the minimum makespan in dynamic grid environment.
- Appearance of new resource: When new resource enters into the grid system, it is likely to increase the performance of the application. To adapt to the current situation where *Grid Information Server* (*GIS*) notifies the existence of new node, we considered the rescheduling of the remaining unexecuted tasks if it is beneficial (i.e. compensates the overhead of rescheduling), thereby reducing the makespan of the application.

4. Proposed adaptive workflow scheduling (AWS) algorithm

4.1. AWS architecture and its components

Fig. 3 shows the architecture of proposed adaptive workflow scheduling strategy. There are two main components: *Resource Discovery and Monitoring, Workflow Task Scheduler.*

4.1.1. Resource discovery and monitoring

It is responsible for discovering and monitoring of grid resources in highly dynamic environment of grid. It continuously collects the information about the available grid resources consisting of computing units and the bandwidth linkages (as in [20,23] as well as discover the new ones. The information includes the type of resources, computing capacity/speed (*CPU_speed*) of the processor, available bandwidth of the links (*BW*), their internal scheduling policies and current workload (*load or linkload*) conditions etc. Further, it sends the status information to the *Grid Information Server* (*GIS*) close to the *Workflow Task Scheduler.* Hence, the *GIS* maintain the up-to-date database about the available grid resources. If the current local workload of the resources (*load* or *linkload*) increases significantly (goes beyond the specific threshold) or new resources has been added then it immediately notifies/trigger the event to the *Execution Manager.* In order to adapt to the newly updated grid information the *Execution Manager* requests the *Workflow Task Scheduler* for rescheduling the remaining unexecuted tasks of the workflow. The *Execution Manager* is responsible for getting the task input file ready and executing the task on the scheduled resources.

4.1.2. Workflow task scheduler

It is responsible for scheduling workflow tasks. It receives DAG from user. Based on the currently available information in *GIS*, it instantiates static scheduling phase with the aim of achieving optimal performance i.e. minimum execution time for the entire workflow and submits it to the *Execution Manager.* It then dispatches the ready tasks according to the execution order of workflow tasks based on their priority, to the appropriate resources as per the schedule and stores the intermediate results back. When *Resource Discovery and Monitoring* component notifies either overload or new resource event to the *Execution Manager,* it triggers the rescheduling requests for remaining unexecuted tasks to the *Workflow Task Scheduler.* The new schedule generated on rescheduling is then submitted to the *Execution Manager* for further processing, if it compensates for the rescheduling overhead.

4.2. Detailed working of AWS procedure

AWS is an adaptive scheduling strategy with the aims to minimize the execution time (makespan) of the workflow application. It

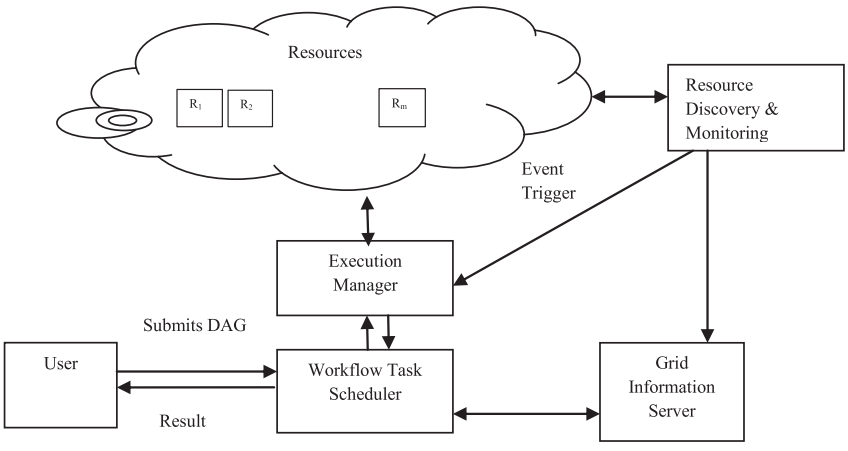

Fig. 3. Adaptive workflow scheduling architecture.

considers the available computing power of each processor and available bandwidth of network links rather than actual computing power and link bandwidth in order to reflect the impact of existing workload. Proposed AWS has three phases: (i) Resource discovery and monitoring phase, (ii) static task scheduling and (iii) rescheduling phase. First of all resource discovery and monitoring phase discovers the set of available resources and their load conditions. Further, it is a continuous phase which periodically monitors the fluctuations in the resource availability. Then initial static scheduling is performed to effectively map the workflow (DAG) tasks to the appropriate resources and submits the schedule to Execution Manager for further processing. The Execution Manger then sends the ready tasks to the scheduled resources and tasks starts executing. The proposed algorithm in dynamic grid environment periodically listens the Resource Monitor, if any abnormality of the resource happens (i.e. load increases significantly over host or link i.e. E_{load}) or new node (E_{new}) gets added. At this, with the help of triggering event, AWS adapts to the changes in the grid environment by initiating rescheduling phase for the remaining unexecuted tasks and tries to attain the optimal scheduling performance. Fig. 4 shows the flow diagram of the procedure for AWS.

4.3. Static task scheduling

The procedure for static workflow task scheduling includes two steps: (i) *assigning priorities* to the tasks in order to ensure task dependency and (ii) *mapping* the tasks to the available resources in order to minimize execution time.

4.3.1. Assigning priority or task ordering

Precedence constraints (dependency constraints) for set of parallel tasks can be guaranteed by executing predecessor tasks before the successor ones. To achieve this goal, we need to generate the ordered task sequence. Here, the ordered task sequence is generated based on b-level priority. The b-level [1] of a task is the length of longest path from the task to exit task. Thus the priority of the task is defined as

$$priority(t_i) = \left\{ \begin{array}{ll} \overline{ET(t_i)}, & \text{if } t_i \text{ is exit task} \\ \overline{ET(t_i)} + \max\limits_{t_j \in succ(t_i)} \left\{ \overline{C_{i,j}} + priority(t_j) \right\}, & \text{otherwise} \end{array} \right\} \quad (8)$$

as (9), $\overline{C_{i,j}}$ is the average communication cost of edge e_{ij} calculated as (10). The communication cost for the tasks allocated to the same resource is assumed to be zero.

$$\overline{ET(t_i)} = \frac{\sum\limits_{r_j \in R} ET(t_i, r_j)}{|R|} \quad (9)$$

$$\overline{C_{i,j}} = \frac{\sum\limits_{r_{s \in R}, r_{t \in R}, s \neq t} C_{i,j}^{s,t}}{|R||R|} \quad (10)$$

Priorities of the workflow tasks are computed upwards starting from exit task. Finally, tasks are sorted in decreasing order of their priorities and the sorted tasks represent the task scheduling sequence.

4.3.2. Mapping

We are selecting the best available resource for each task of the workflow in the task ordering sequence obtained in previous step to create an optimized schedule.

The available computing capacity of the resources is initially obtained after considering the current workload. We then selects the task form the task ordering sequence and maps it to the resource which minimizes the estimated start time (EST) of its successor tasks. In literature, popular list based heuristic HEFT [29] maps the selected task to the resource which provides minimum estimated finish time (EFT) considering the computation cost of the task only. But since communication cost with dependent tasks could severely affect the overall makespan of the workflow, so it must be considered along with the computation cost. Thus, we select the resource for the task, which minimizes the EST of its successor tasks instead of EFT of current task. To accomplish this, EFT of the task over all resources are calculated along with the average communication and computation cost with the dependent tasks.

$$map(t_i) = r_j \quad \text{where,} \quad \min_{\forall r_j \in R} \left\{ \begin{array}{ll} EFT(t_i, r_j), & \text{if } t_i \text{ is exit task} \\ EFT(t_i, r_j) + \dfrac{\sum\limits_{t_s \in succ(t_i)} \left(\overline{C_{i,s}} + \overline{ET(t_s)} \right)}{|succ(t_i)|} & \text{otherwise} \end{array} \right\} \quad (11)$$

Where $succ(t_i)$ is the set of the immediate successors of task t_i, $\overline{ET(t_i)}$ is the average computational cost/execution time calculated

As soon as the initial schedule is generated, it is submitted to the execution manager for processing of the workflow tasks.

4.4. Rescheduling

Due to dynamic environment of the grid, the status of the resources changes with time. To provide the high and stable performance, the grid scheduler must adapt to the changes in the run time environment of the grid, such as variations in resource availability (processor and link) and its performance. In order to provide adaptability to the changes in resource availability, resource discovery and monitoring component periodically monitors the resources. It makes a request to the workflow grid scheduler for initiating the rescheduling mechanism for remaining unexecuted tasks by event triggering, when either load over resources (processor or link) increases (E_{load}) beyond the threshold or if new resource (E_{new}) gets added at run time.

Based on the current status information of the processors and links, new mapping is produced for remaining unexecuted tasks. The benefits of rescheduling are balanced by the overheads, which may include transmission of output data to the dependent tasks at the newly mapped resources. The new schedule is worth considering if reduction in execution time of the new schedule compensates for these overheads.

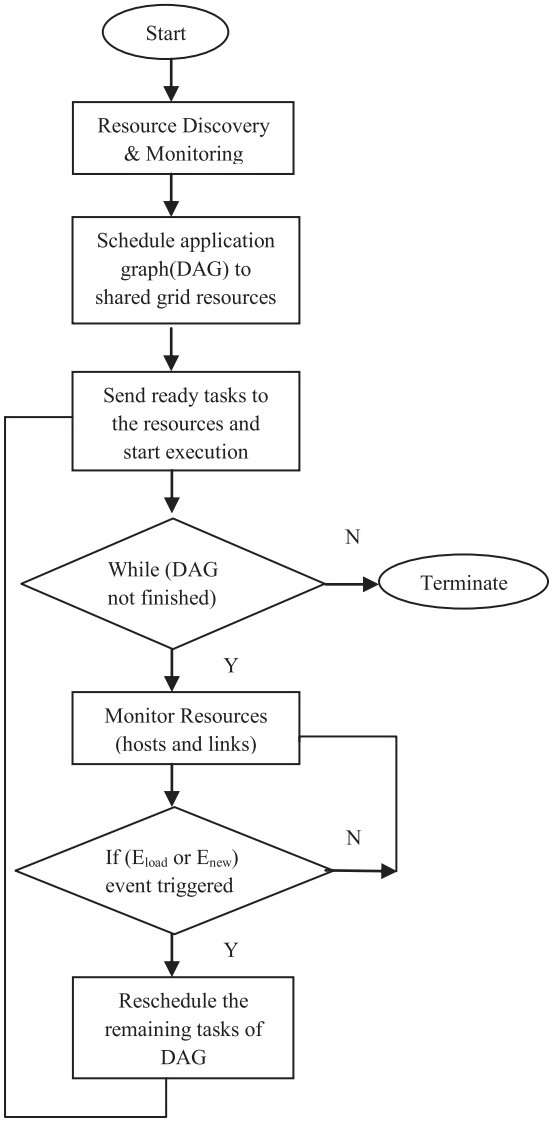

Fig. 4. The flow diagram of the procedure for AWS.

Let $makespan_{re}$ represents the makespan due to rescheduling and is calculated as the sum of elapsed time from start of the workflow (time taken by already executed tasks), makespan of the new schedule (S′) for remaining unexecuted tasks and overheads due to re-mapping.

$$makespan_{re} = elapsed\_time + S'.makespan + overhead \quad (12)$$

Further, $makespan_{pred}$ represents the predicted makespan of the original schedule, which is equal to the sum of the expected makespan of the current schedule (S) and the delay from unexecuted task due to increased workload. To calculate the delay, estimated finish time of unexecuted tasks are re-estimated as per the updated status of resources. So,

$$makespan_{pred} = S.makespan + delay \quad (13)$$

If, makespan due to rescheduling ($makespan_{re}$) is less than the predicted makespan ($makespan_{pred}$) then, new schedule is submitted for execution of remaining tasks. Rescheduling here plays important role to achieve the minimum execution time as well as it performs load balancing between grid resources.

5. Pseudo code for AWS

This section describes the pseudo code for AWS algorithm for scheduling workflow tasks in dynamic grid environment.

Algorithm 1 describes the overall adaptive workflow scheduling algorithm considering the dynamic environment of the grid. Initially, the first phase of ordering the task is performed. Line 2–7 assigns the priorities to the tasks starting from exit tasks after calculating initial parameters. Then, line 8 generates the scheduling order sequence by sorting the tasks in the non increasing order of their priorities. Lines 9–11, compute the mapping. Here it maps grid tasks to the appropriate resource with the aim to minimize the execution time. Finally the *workflow task scheduler* dispatches the current schedule(S) to the *Execution Manager*, which then dispatches the ready tasks to the mapped resources. The task list and edge list is updated accordingly shown by line 16. Periodic resource monitoring is performed here by *Resource discovery and monitoring* component. If workload of the resources increases significantly or new node gets added, then rescheduling phase is triggered for the remaining unexecuted tasks as shown in lines 17. For rescheduling, in line 19, the AWS algorithm is recursively called for unexecuted tasks and current set of resources with updated status information. If the performance gain is there with rescheduling after compensating the overheads, new schedule is submitted for processing.

The time complexity of the proposed AWS algorithm depends upon the complexity of scheduling algorithm used for making reschedule at each triggering event. The complexity of static scheduling algorithm is dominated by the basic operations of: 1) Calculating priority of the tasks $O((n + e).\ m)$ and sorting the task based on their priority $O(nlogn)$, where n is the number of tasks, m is number of processors and e is the number of directed edges. 2) Calculating the estimated start time and estimated finish time of the tasks over each processor $O(e)$. 3) Mapping is then produced with complexity of $O(nm)$. 4) The procedure repeats for unexecuted tasks when resource overload or new node event occurs. Therefore, the overall complexity of the proposed algorithm is $O(trig\_events.(n + e).\ m + nlogn))$.

Algorithm 1 Adaptive workflow scheduling(AWS) algorithm

procedure AWS (workflow $G_t(T,E)$ and grid network $G_r(R,L)$)
Input: Task graph G_t with set of tasks (T) and edges (E), grid network G_r with set of computing nodes (R) and network links (L)
Output: Scheduling the workflow tasks to minimize the makespan, considering the dynamic environment of grid

1. Compute the available computing capacity (ACPU_speed) of the nodes from set R and the available bandwidth(ABW) of the link among them after considering the existing load
2. **for** all $t_i \varepsilon T$ do
3. compute average execution time $\overline{ET(t_i)}$ //according to Eq. (9)
4. **for** all $e_{ij} \varepsilon E$ do
5. compute average communication time $\overline{C_{i,j}}$ //according to Eq. (10)
6. **for** all $t_i \varepsilon T$ do
7. compute priorities $priority(t_i)$ of each task starting from t_{exit} //according to Eq. (8)
8. sort all tasks into task ordering sequence T_{list} in non increasing order of their priorities.
9. **for** all $t_i \varepsilon T_{list}$ do
10. **for** all $r_j \varepsilon R$ do
11. find $map(t_i)$ with Eq. (11) which represents the host for task t_i and add it to schedule S along with its estimated finish time.
12. update $RAT(r_j)=EFT(t_i,r_j)$
13. compute makespan corresponding to schedule S
14. **While** ($S \neq \Phi$) begin
15. Dispatch each ready task t_i (in the order of their priorities) to its assigned resource and remove mapping from S
16. T`←T-{ $t_i$ } and E`←E-{$e_{p,i}$}
17. **if** (E_{load} or E_{new} is triggered), then
18. Update the status of resources as set R*.
19. Call S`=AWS($G_t$`(T`,E`), G_r*(R*,L*)
20. compute $makespan_{re}$ and $makespan_{pred}$ //according to Eq. (12) & (13)
21. **if** ($makespan_{re} < makespan_{pred}$) do
22. S=S` and submit new schedule
23. **end while**
24. **end**

5.1. Detailed example

To understand the proposed AWS approach, let us consider a case that there are two groups of resources in the grid each having two computing nodes and links between them. The initial status of each node is shown in Table 2a

Let the bandwidth of the link between two resources in the same group is 120 Mbps and between two resources in distinct groups is 100 Mbps. Let the variable amount of load exist over the network links, as shown in Table 2b

Consider the sample workflow DAG application shown in Fig. 1. The node weight in DAG represents the size of task on a10^6 MI scale and edge weight representing communication data in Gigabytes.

We are considering four sets of experiments corresponding to the DAG. First experiment determines the execution time of the DAG under current resource availability and load conditions. Then in the second & third experiment, the load over computing nodes and links were increased to show the performance of adaptive workflow scheduler under dynamic availability of resources. Finally, the effect of newly added node is evaluated.

As per the first experiment, the computation cost of each task over computing nodes and communication cost to transfer unit data among them considered at current workload conditions is shown in Table 3(a) and (b) respectively. Tasks ordering sequence is $t_0, t_1, t_2, t_4, t_3, t_8, t_6, t_5, t_7, t_9$ and final mapping is $t_0 \rightarrow r_4$, $t_1 \rightarrow r_4$, $t_2 \rightarrow r_3$, $t_3 \rightarrow r_1$, $t_4 \rightarrow r_2$, $t_5 \rightarrow r_2$, $t_6 \rightarrow r_3$, $t_7 \rightarrow r_1$, $t_8 \rightarrow r_4$, $t_9 \rightarrow r_4$ with makespan of 322.19 min.

Table 2a
Initial status of computing nodes.

	Group1		Group2	
	r_1	r_2	r_3	r_4
Computing speed(*CPU_speed*) (MIPS)	4000	4500	6000	6400
Load(%)	30	25	25	20
Available Computing speed(*ACPU_speed*) (MIPS)	2800	3375	4500	5120

Table 2b
Initial status of network links.

	r_1–r_2	r_3–r_4	r_1–r_3	r_1–r_4	r_2–r_3	r_2–r_4
Link Bandwidth (*BW*)(Mbps)	120	120	100	100	100	100
Link Load (%)	20	25	35	30	25	30
Available bandwidth(*ABW*) (Mbps)	96	90	65	70	75	70

Table 3a
The computation cost (min).

Tasks	Resource nodes			
	r_1	r_2	r_3	r_4
t_0	130.95	108.64	81.48	71.61
t_1	142.85	118.51	88.88	78.12
t_2	107.14	88.88	66.66	58.59
t_3	107.14	88.88	66.66	58.59
t_4	83.33	69.13	51.85	45.57
t_5	77.38	64.19	51.85	42.31
t_6	107.14	88.88	66.66	58.59
t_7	65.47	54.32	40.74	35.80
t_8	136.90	113.58	85.18	74.86
t_9	83.33	69.13	51.85	45.57

Table 3b
The communication cost (min).

Node Links	Communication cost per GB of data
r_1–r_2	1.388
r_3–r_4	1.481
r_1–r_3	2.051
r_1–r_4	1.904
r_2–r_3	1.777
r_2–r_4	1.904

In the second experiment, let the additional 25% load comes to node r_4 at 90 min, thus available computing power of it is reduced to 3520 MIPS instead of 5120 MIPS. At this time tasks list contains t_8, t_6, t_5, t_7, t_9 as unexecuted tasks. As per the AWS, the unexecuted tasks are remapped to $t_5 \rightarrow r_2$, $t_6 \rightarrow r_4$, $t_7 \rightarrow r_1$, $t_8 \rightarrow r_3$, $t_9 \rightarrow r_3$ and the new makespan is 329.63 min, which increases by 2.3% only. But if re-scheduling has not been performed, then the makespan will be 376.92 min which is 16.98% more than original scheduling.

Next, we consider the effect of load on network link. Let at 90 min, additional 50% load comes at link between nodes r_2-r_4 i.e. additional stream of 50Mbps is added. As per the original schedule, the makespan will be 366.15 min, which is 13.64% more than original schedule. But in AWS with rescheduling the unexecuted tasks, new mapping will be $t_5 \rightarrow r_4$, $t_6 \rightarrow r_4$, $t_7 \rightarrow r_2$, $t_8 \rightarrow r_3$, $t_9 \rightarrow r_3$ and the overall makespan will be 327.40 min which increases by 1.61% only.

In the final case, let us consider a new node with available computing power of 5600MIPS (*CPU_ speed* = 7000MIPS with 20% load) gets added at 90 min with available bandwidth to all the resources as 80Mbps, then as per rescheduling of unexecuted tasks by AWS, the new mapping will be $t_5 \rightarrow r_3$, $t_6 \rightarrow r_4$, $t_7 \rightarrow r_5$, $t_8 \rightarrow r_5$, $t_9 \rightarrow r_5$ with makespan of 283.62 min, which reduces by 11.98%.

6. Simulation strategy

6.1. Simulation model

To verify the correctness and effectiveness of the proposed adaptive workflow scheduling algorithm, we have conducted extensive experiments and simulation based on GridSim [8] toolkit and compares it with other popular heuristics for workflow scheduling as HEFT [29], Min–Min [22], Max-Min [22], AHEFT [31], Re-DCP-G [25] and Re-LSDS [24]. To simulate precedence constraint tasks in workflows, we used the workflow models represented by randomly generated task graphs and task graphs corresponding to real world problems such as Gaussian Elimination (GE) and Fast Fourier Transforms (FFT). We employed simulated parameters generated by set of resources and randomly varying the structure and properties of workflow task graph according to model described in Sections 3.1 And 3.2 respectively. The workflow task graph has following parameters:

- Number of tasks in the DAG (n)
- Communication to computation ratio (CCR): It indicates the characteristics of input workflow application. Higher value of CCR means data intensive application while lower value indicated computation intensive application. Size of communication data or message size can be determined from CCR values.
- Shape parameter of the DAG (shape). The height/depth of the DAG is randomly generated from uniform distribution with mean equal to $\sqrt{n}$/shape. The width of each level of the DAG is randomly generated from uniform distribution with mean equal to $\sqrt{n}$. shape.
- Out degree representing the maximum out degree of tasks in DAG. It is generated randomly from 1 to 5.
- The computation cost of each task t_i on resource r_j is selected randomly by the uniform distribution with the mean equal to the twice of specified average computation cost.
- The cost of each edge was selected randomly from the uniform distribution across the mean equal to the product of average computation cost and the communication to computation ratio (CCR).

The parameters corresponding to grid resources or grid computing environment are as follows:

- Number of processors (m) and their computing capacity/speed (MIPS)
- Bandwidth of link between processors(Mbps)
- Existing load on processors (*load*)
- Existing load on network (*linkload*)

To represents the dynamic behavior of resources, additional parameters required are as follows:

- Time interval to load change (γ): Resource load changes after fixed interval. Lower value of change interval indicates more dynamic environment while higher the value of change interval, more static is the environment.
- Amount of load change (δ)

The values for above mentioned parameters are listed in Table 4.

6.2. Performance metric

The aim of scheduling algorithm considered here is to achieve the minimum execution time (*makespan*), where *makespan* of the input workflow application (DAG) represents the total time elapsed between the start time of the first(entry) task to the completion time of last (exit) task. We considered the start time of entry task as zero and *makespan* can be calculated as the finish time of exit task. The performance metric used to evaluate the proposed algorithm in comparison to other algorithms is described below:

6.2.1. Improvement rate (IR)

It specifies the performance improvement rate of AWS algorithm with respect to other algorithms and is measured as the difference of the *makespans* over the *makespan* of AWS algorithm. Reduction in the *makespan* of AWS algorithm over other considered algorithms can be calculated by IR(%).

Table 4
Simulation parameter values.

Simulation parameters	Values
n	20,40,60,80,100(Random graphs) (Default value 60) 14,27,44,65,90(GE graphs) (Default value 44) 14,38,94(FFT graphs) (Default value 38)
CCR	0.2, 0.6, 1, 1.4, 1.8 (Default value 1)
m	5,10,15,20 (Default value 10)
Computing capacity of resources	1000–8000(MIPS)
Bandwidth of links	100–120(Mbps)
Load	0–30%
Link load	10–40%
Number of newly added resources after random interval	1,2,3,4,5
γ	20,50,70,90 (min) (Default value 50)
δ	10%, 25%, 40%, 55%(Default value 25%)

$$IR(\%) = \frac{makespan(other) - makespan(AWS)}{makespan(AWS)} \times 100 \quad (14)$$

6.3. Simulation result analysis

6.3.1. Test suit1

In this test suit, we used the workflow model represented by randomly generated task graph (Random). The size of random task graph was varied by considering the different number of tasks as 20, 40, 60, 80 and 100. The computation cost of each task t_i on resource r_j is selected randomly by the uniform distribution with the mean equal to the twice of the average computation cost. The cost of each edge was selected randomly from the uniform distribution across the mean equal to the product of average computation cost and the communication to computation ratio (CCR).

6.3.1.1. Effect of varying the size of input graph. Firstly, we consider the effect of varying the size of input graph over the makespan considering 10 fixed number of grid hosts and is shown in Fig 5. The figure clearly specifies that AWS performs best with respect to other algorithms considered. Based on the makespan the improvement rate IR as per (14) is calculated. AWS provides superior performance to AHEFT by almost 8%–15% IR, Re-DCP-G by 5%–13% IR and Re-LSDS by 9%–16% IR in every case because it considers dynamically changing host and link availability. As the number of tasks are increasing, the improvement rate (IR) increases because with large number of tasks there are more chances of rescheduling. Other considered algorithms like Min–min, Max-min, and HEFT perform badly because they does not consider dynamic load present at the resources.

6.3.1.2. Effect of varying the CCR value. In this experiment, we are considering the effect of varying the CCR value of the workflow application graph. The number of tasks considered is 60 by default with 10 numbers of processors and the results are shown in Fig. 6. When CCR is less than 1 i.e. in case of computation intensive task graph there is minimum IR of 8% over AHEFT, 6% over Re-DCP-G and 8% over Re-LSDS. But when, CCR is more than one, in case of communication intensive workflows, there is minimum improvement rate i.e. IR of 16% over AHEFT, 14% over Re-DCP-G and 17% over Re-LSDS because of consideration of the link load dynamicity in AWS. It gives 9%–18% improvement over HEFT while in case of Max-Min and Min–Min; it provides improvement of 13%–32% because they give less consideration to communication overhead while making mapping decision. Results indicate that the proposed algorithm performs efficiently at all CCR values.

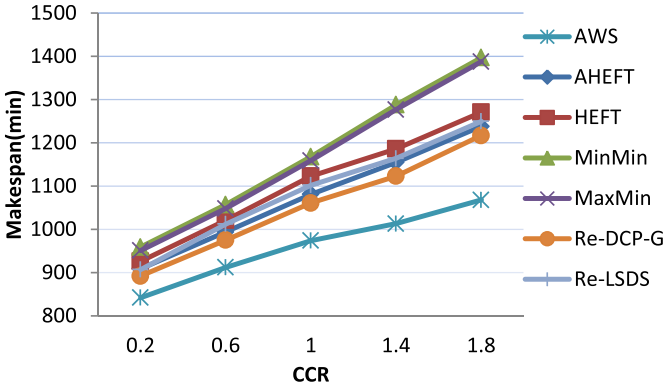

Fig. 6. Makespan under different CCR (for random task graph).

6.3.1.3. Effect of varying the number of processors. We evaluated the performance of the algorithm by varying the number of processors for fixed 60 numbers of tasks at CCR is equal to 1. The results are shown in Fig. 7, which clearly manifests that the performance of the algorithm improves, as the numbers of resources are increasing with the help of rescheduling. As depicted in Fig. 7, the performance improvement of AWS over other algorithms varies from 3% to 9% only, at less number of resources. This is due to the fact that, with less number of resources, choice of selecting better resource during rescheduling phase is quite limited. As the numbers of resources are increasing, the performance improvement of AWS over other algorithms improves. But when the large (abundant) numbers of resources are available, the IR increases slowly. It may be because of frequent rescheduling with high overheads. Thus, after compensating the overheads slow increase of IR is there with large number of resources.

Further, we considered the effect of dynamic nature of grid, by varying periodically the *load over resources* (hosts and links).

6.3.1.4. The effect of time interval to load change. The time interval of load change indicates the dynamicity of the grid environment. Results from Fig. 8, clearly manifests that at lower value (20 min) of load change interval (i.e. load changes more frequently or highly dynamic grid environment), the proposed algorithm performs better than other algorithms. The proposed AWS provides IR of 20% over AHEFT, 40% over HEFT, 38% over Min–Min, 36% over Max-Min, 15% over Re-DCP-G and 21% over Re-LSDS respectively. The performance of AWS with respect to HEFT, is more significant, as it is static algorithm and do not consider the change of load at run time. As the load change interval increases, (indicating static grid environment), the performance improvement is still there but

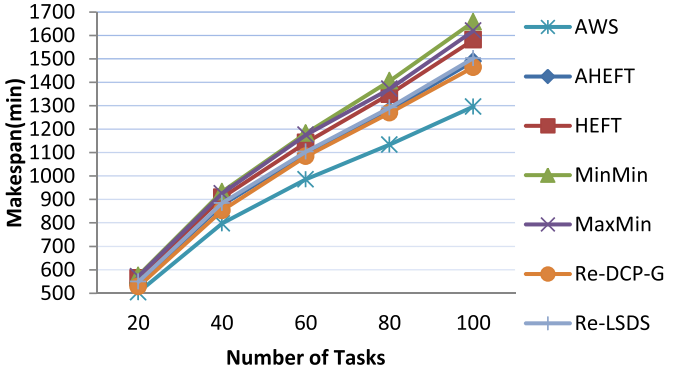

Fig. 5. Makespan under different number of tasks (for random task graph).

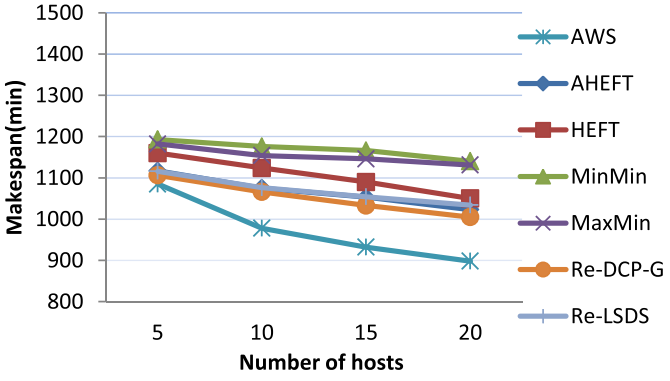

Fig. 7. Makespan under different number of hosts (for random task graph).

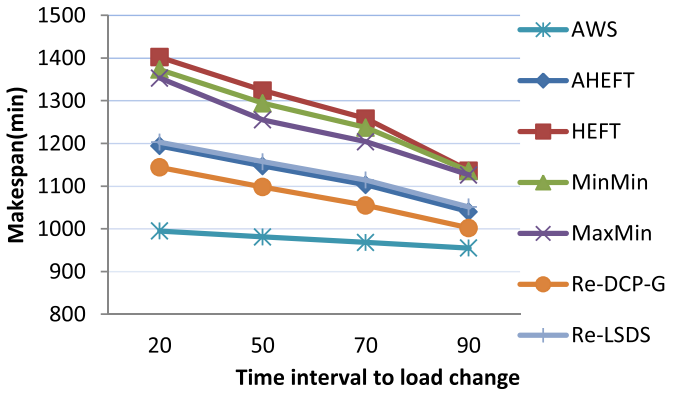

Fig. 8. Makespan under various time interval of load change or frequency of load change (for random task graph).

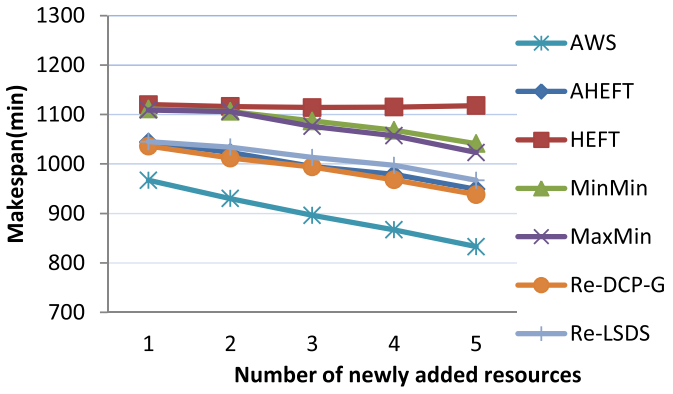

Fig. 10. Makespan under different number of newly added resources at run time (for random task graph).

comparatively less. At maximum time interval (90 min) of load change, the IR is in the range from 5% to 19% only.

6.3.1.5. The effect of amount of load change. Fig. 9 illustrates the effect of amount of load change. When the heavy amount of sudden load comes to the resources, they become inefficient. Rescheduling is highly desirable in that case, in order to provide better resources for remaining unexecuted part of the application. Thus the proposed AWS, with the help of rescheduling gives high improvement rate of 42%, over HEFT when large amount of load (i.e. 55%) changes dynamically. Compared to AHEFT, Min–Min, Max-Min, Re-DCP-G and Re-LSDS the IR is about 11%–19%, 23%–36%, 21%–33%, 8%–14% and 12%–20% respectively at different amount of load change.

6.3.1.6. Effect of number of newly added resources at run time. Lastly, we evaluated the algorithm, considering the number of newly added resources periodically to the resource pool at run time. As shown in Fig. 10, the proposed AWS algorithm provides better performance improvement, with the increase of number of newly added resources. It provides the peak performance improvements of 35% over HEFT, when number of added resources are 5. This is due to the fact that HEFT does not consider the newly added resources at all.

Overall, the results clearly specifies that our proposed algorithm outperforms the other algorithms in all the cases and is the suitable algorithm for workflow scheduling in dynamic grid environment, where the number of resources and load on them are changing dynamically.

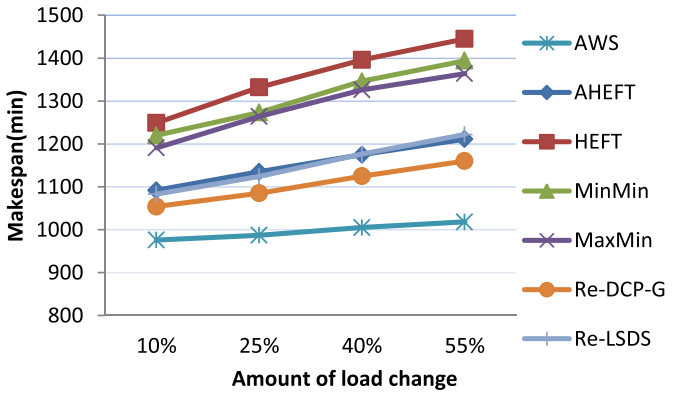

Fig. 9. Makespan under different amount of load change (for random task graph).

6.3.2. Test suit2

In this test suit, we generated the task graphs corresponding to real life problems such as Gaussian Elimination (GE) [13] and Fast Fourier Transform (FFT) [12]. The structure of these task graphs is fixed. In GE task graph, number of tasks are equal to $(m^2+m-2)\backslash 2$, where m is the matrix size, thus number of tasks chosen are 14, 27, 44, 65, 90 in GE task graph. In FFT task graph, the numbers of tasks are equal to $(2*m-1)+m*\log_2 m$, where m is any power of 2. Thus number of tasks chosen is 14, 38, and 94 in FFT task graph. The computation cost of each task t_i on resource r_j is selected randomly by the uniform distribution with the mean equal to the twice of specified average computation cost. The cost of each edge was selected randomly from the uniform distribution with mean equal to the product of average computation cost and the communication to computation ratio (CCR).

Similar set of experiments were performed for GE and FFT task graphs and the effect of (i) size of the graph, (ii) CCR, (iii) number of resources, (iv) interval of load change, (v) amount of load change and (vi) number of newly added resources were evaluated. Figs. 11 and 12 shows the experimental results for the GE and FFT tasks graphs respectively. The results trends are similar to test suit1. It can be seen that the proposed AWS performs better in comparison to other algorithms, not only for random task graphs but also for task graphs corresponding to real world problems of GE and FFT.

7. Conclusion

In this paper, we proposed the solution to address the dynamic nature of grid environment and to efficiently utilize the resources based on QoS information like availability along with the accessibility as indicated by SLA. We proposed a novel adaptive workflow scheduling (AWS) algorithm, to schedule workflow application in dynamic grid environment with the aim to achieve the minimum execution time (makespan). The variation in the availability of grid resources at run time has strong impact on the performance of the workflow application. Thus, in order to provide adaptability to the changes in resource availability in the dynamic grid environment, rescheduling is triggered if the any abnormality of the resource happens (i.e. load increases significantly over host or link) or new node gets added at run time. Further, the algorithm also provides load balancing by supporting rescheduling of tasks form overloaded resources. The procedure of adaptive workflow scheduling differs from other approaches in literature by considering the existing load over the resources both computing nodes and communication links in order to generate the availability of resources. Experiments are carried out to validate the performance of the proposed algorithm by varying the workflow and

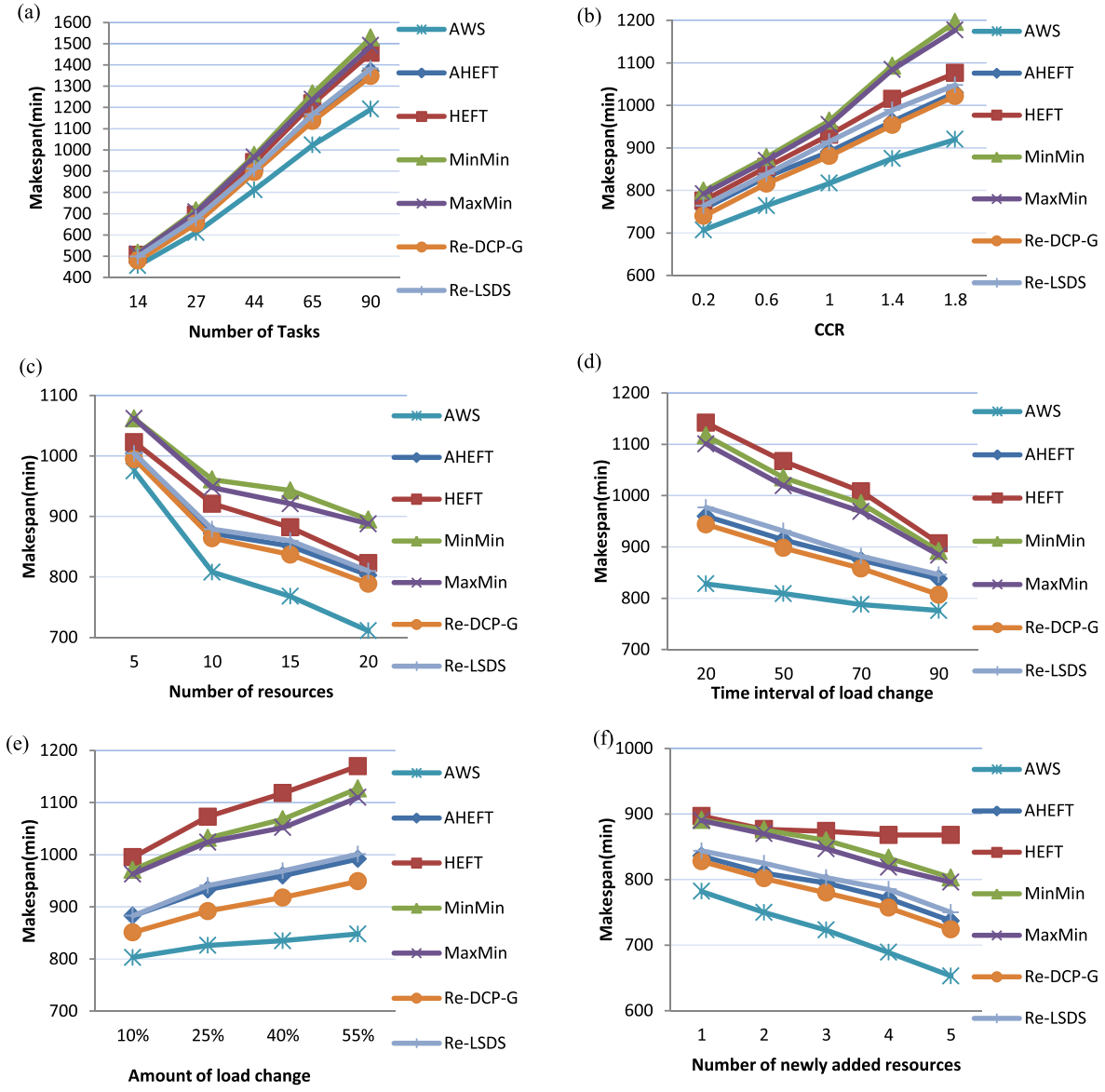

Fig. 11. Effect of various parameters on Gaussian Elimination Task graph (GE).

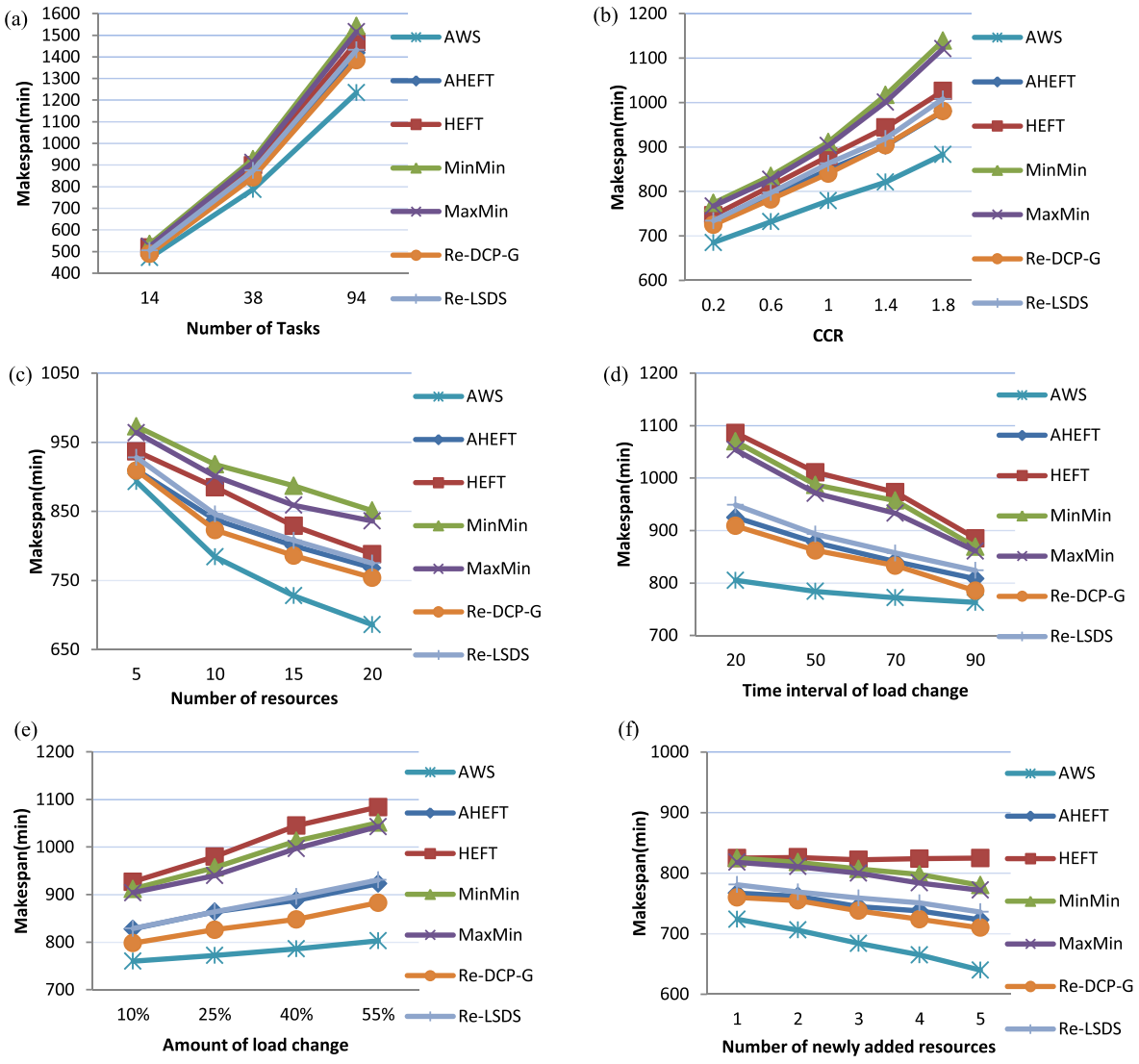

Fig. 12. Effect of various parameters on Fast Fourier Task graph (FFT).

dynamic grid system settings. The simulation results using randomly generated task graphs and task graphs corresponding to real world problems like GE and FFT demonstrates the 10%–40% performance improvement (makespan minimization) of the proposed AWS algorithm over other scheduling algorithms considered.

A good course of future research may include the development of rescheduling approach considering the effect of dynamic resource availability on the currently executing tasks along with the remaining unexecuted tasks, in order to provide high and stable performance to the workflow application.

References

[1] I. Ahmad, Y.K. Kwok, M.Y. Wu, Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors, in: ISPAN, 1996, pp. 207–213.
[2] R. Bajaj, D.P. Agarwal, Improving scheduling of tasks in a heterogeneous environment, in: IEEE Transactions on Parallel and Distributed Systems, 15(2), 2004, pp. 107–118.
[3] D.M. Batista, N.L. da Fonseca, F.K. Miyazawa, F. Granelli, Self-adjustment of resource allocation for grid applications, Computer Networks 52 (9) (2008) 1762–1781.
[4] F. Berman, H. Casanova, A. Chien, K. Cooper, et al., New grid scheduling and rescheduling methods in the GrADS project, Int J Parallel Program 33 (2–3) (2005) 209–229.
[5] L.F. Bittencourt, E.R.M. Madeira, F.R.L. Cicerre, L.E. Buzato, A path clustering heuristic for scheduling task graphs onto a grid (short paper), in: Proceedings of the 3rd ACM International Workshop on Middleware for Grid Computing, 2005. Grenoble, France.
[6] L.F. Bittencourt, E.R. Madeira, A performance oriented adaptive scheduler for dependent tasks on grids. Concurrency and Computation, Practice Exp. 20 (9) (2008) 1029–1049.
[7] T.D. Braun, H.J. Siegal, N. Beck, A comparison of Eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Dis. Comp. 61 (2001) 810–837.
[8] R. Buyya, M. Murshed, GridSim: A Toolkit for Modeling and Simulation of Grid Resource Management and Scheduling, Vol. 14, 2002, pp. 1175–1220. http://www.buyya.com/gridsim.
[9] R. Buyya, S. Venugopal, A Gentle Introduction to Grid Computing and Technologies, CSI Communications, July 2005.
[10] R.S. Chang, C.Y. Lin, C.F. Lin, An adaptive scoring job scheduling algorithm for grid computing, Information Sciences 207 (2012) 79–89.
[11] S.H. Chin, T. Suh, H.C. Yu, Adaptive service scheduling for workflow applications in service-oriented grid, J. Supercomputing 52 (3) (2010) 253–283.
[12] Y.C. Chung, S. Ranka, Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, in: Supercomputing'92., Proceedings, IEEE, 1992, pp. 512–521.
[13] M. Cosnard, M. Marrakchi, Y. Robert, D. Trystram, Parallel Gaussian elimination on an MIMD computer, Parallel Computing 6 (3) (1988) 275–296.

[14] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, K. Wenger, Pegasus, a workflow management system for science automation, Future Generation Computer Systems (2014).
[15] I. Foster, C. Kesselman, The Grid2: Blueprint for a New Computing Infrastructure, Elsevier, 2003.
[16] M. Gareym, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, WH Freeman & Co., San Francisco, 1979.
[17] R. Garg, A.K. Singh, Multi-objective workflow grid scheduling using ε-fuzzy dominance sort based discrete particle swarm optimization, J Supercomputing 68 (2) (2014) 709–732.
[18] R. Garg, A.K. Singh, Reference Point based multi-objective optimization to workflow grid scheduling, International Journal of Applied Evolutionary Computation (IJAEC) 3 (1) (2012) 80–99.
[19] E. Huedo, R.S. Montero, I.M. Llorente, Experiences on adaptive grid scheduling of parameter sweep applications, in: Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2004, pp. 28–33.
[20] B.B. Lowekamp, Combining active and passive network measurements to build scalable monitoring systems on the grid, ACM SIGMETRICS Performance Evaluation Review 30 (4) (2003) 19–26.
[21] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, R. Freund, Dynamic Matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: In 8th Heterogeneous Computing Workshop (HCW'99), IEEE, 1999, pp. 30–44.
[22] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, L. Johnsson, Scheduling strategies for mapping application workflows onto the grid, in: In Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005, pp. 125–134.
[23] F. Montesino-Pouzols, Comparative Analysis of Active Bandwidth Estimation Tools. In Passive and Active Network Measurement, Springer, Berlin Heidelberg, 2004, p. 175.
[24] A. Olteanu, F. Pop, C. Dobre, V. Cristea, A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems, Comp. Math. Appl. 63 (9) (2012) 1409–1423.
[25] M. Rahman, R. Hassan, R. Ranjan, R. Buyya, Adaptive workflow scheduling for dynamic grid and cloud computing environment, Conc. Comp. Prac. Exp. 25 (13) (2013) 1816–1842.
[26] R. Sakellariou, H. Zhao, A low-cost rescheduling policy for efficient mapping of workflows on grid systems, Scientific Programming 12 (4) (2004) 253–262.
[27] H.A. Sanjay, S.S. Vadhiyar, Strategies for rescheduling tightly-coupled parallel applications in multi-cluster grids, J. Grid Comp. 9 (3) (2011) 379–403.
[28] P.K. Tiwari, D.P. Vidyarthi, Observing the effect of inter-process communication in auto controlled ant colony optimization based scheduling on computational grid, Conc. Comp. Prac. Exp. 26 (1) (2014) 241–270.
[29] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, in: IEEE Transactions on Parallel and Distributed Systems, 13(3), 2002, pp. 260–274.
[30] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scientific workflows in the ASKALON grid environment, SIGMOD 34 (3) (2005) 56–62.
[31] Z. Yu, W. Shi, An adaptive rescheduling strategy for grid workflow applications, in: In IEEE International Parallel and Distributed Processing Symposium, 2007, IPDPS, 2007, pp. 1–8.