CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **25**:1816–1842
Published online 4 March 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3003

SPECIAL ISSUE PAPER

Adaptive workflow scheduling for dynamic grid and cloud computing environment

Mustafizur Rahman [1], Rafiul Hassan [2], Rajiv Ranjan [3,*,†] and Rajkumar Buyya [1]

[1] *Department of Computer Science and Software Engineering, The University of Melbourne, Australia*
[2] *Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Saudi Arabia*
[3] *Information Engineering Laboratory, CSIRO ICT Centre, Camberra, Australia*

SUMMARY

Effective scheduling is a key concern for the execution of performance-driven grid applications such as workflows. In this paper, we first define the workflow scheduling problem and describe the existing heuristic-based and metaheuristic-based workflow scheduling strategies in grids. Then, we propose a dynamic critical-path-based adaptive workflow scheduling algorithm for grids, which determines efficient mapping of workflow tasks to grid resources dynamically by calculating the critical path in the workflow task graph at every step. Using simulation, we compared the performance of the proposed approach with the existing approaches, discussed in this paper for different types and sizes of workflows. The results demonstrate that the heuristic-based scheduling techniques can adapt to the dynamic nature of resource and avoid performance degradation in dynamically changing grid environments. Finally, we outline a hybrid heuristic combining the features of the proposed adaptive scheduling technique with metaheuristics for optimizing execution cost and time as well as meeting the users requirements to efficiently manage the dynamism and heterogeneity of the hybrid cloud environment. Copyright © 2013 John Wiley & Sons, Ltd.

Received 11 April 2011; Revised 18 August 2012; Accepted 13 December 2012

KEY WORDS: adaptive scheduling; workflow management; grid computing

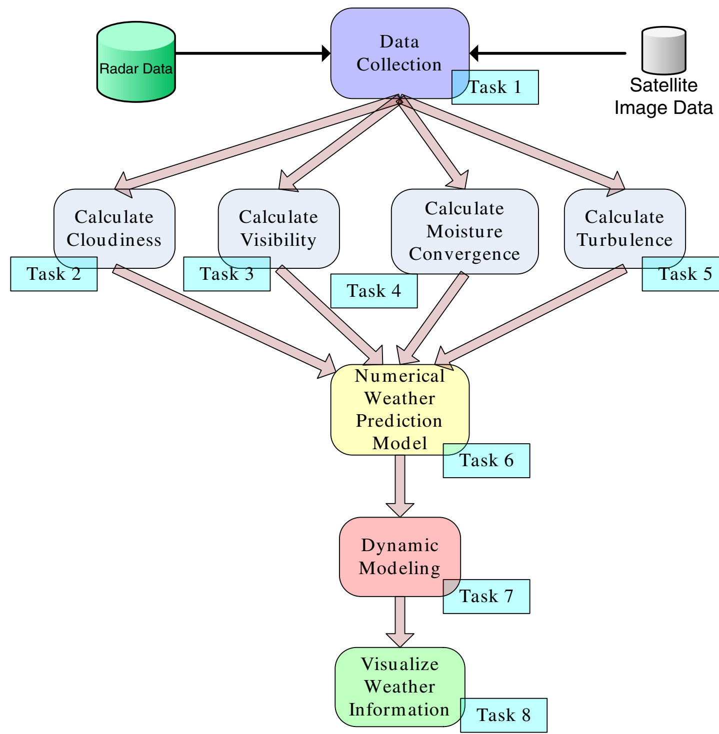
1. INTRODUCTION

Many of the large-scale scientific applications executed on present-day grids are expressed as complex e-Science workflow [1, 2], which is a set of ordered tasks that are linked by data dependencies. A workflow management system [3] is generally employed to define, manage, and execute these workflow applications on grid resources. A workflow management system uses a specific scheduling strategy for mapping the tasks in a workflow to suitable grid resources in order to satisfy user requirements. Numerous workflow scheduling strategies have been proposed in literature for different objective functions [4]. Figure 1(b) illustrates the execution of the workflow shown in Figure 1(a) on a traditional distributed computing environment.

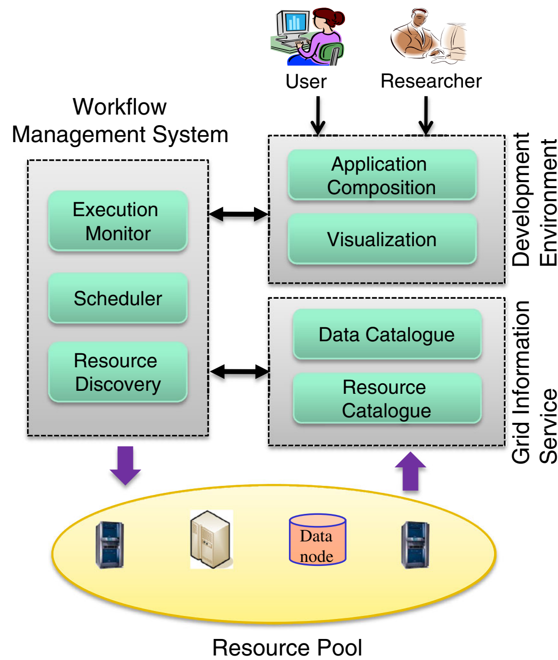
However, the majority of these scheduling strategies are static in nature. They produce a good schedule given the current state of grid resources and do not take into account changes in resource availability. On the other hand, dynamic scheduling is done on-the-fly considering the current state of the system and adaptive in nature. Thus, in this paper, we present a dynamic workflow scheduling technique that not only dynamically minimizes the workflow execution time but also reduces the scheduling overhead, which constitutes a significant amount of time used by the scheduler to generate the schedule.

*Correspondence to: Rajiv Ranjan, Information Engineering Laboratory, CSIRO ICT Centre, Camberra, Australia.
†E-mail: rajiv.ranjan@csiro.au

Copyright © 2013 John Wiley & Sons, Ltd.



(a)



(b)

Figure 1. Typical scientific workflow applications management scenario in distributed computing environment. (a) Example of workflow: weather prediction application; (b) workflow management system.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Critical path (CP) heuristics [5] have been used extensively for scheduling interdependent tasks in multiprocessor systems. These heuristics aim to determine the longest of all execution paths from the beginning to the end (or the *critical path*) in a task graph and schedule them earliest so as to minimize the execution time for the entire graph. Kwok and Ahmad [6] introduced the dynamic CP (DCP) algorithm in which the CP is dynamically determined after each task is scheduled. However, this algorithm is designed for mapping tasks on to homogeneous processors, and is static, in the sense that the schedule is only computed once for a task graph. We extend the DCP algorithm to map and schedule tasks in a workflow on to heterogeneous resources in a dynamic grid environment. To evaluate the performance of our proposed algorithm, called DCP for grids (DCP-G), we have compared it against the existing approaches, discussed in this paper for different types and sizes of workflows. The results demonstrate that DCP-G can adapt to temporal resource behavior and avoid performance degradation in dynamically changing grid environments.

The rest of the paper is organized as follows. In the next section, we provide the definition of workflow scheduling problem in grids. Section 2.2 describes the existing heuristic-based and metaheuristic-based workflow scheduling techniques utilized in distributed computing systems such as grids. The proposed DCP-G workflow scheduling algorithm is presented in Section 3. Experiment details and simulation results are discussed in Section 4. Section 5 presents a case study illustrating the usefulness of heuristic (DCP)-based dynamic and adaptive workflow scheduling scheme over metaheuristic-based static scheduling. In Section 6, we outline a hybrid heuristic leveraging proposed DCP-G for hybrid cloud computing environment. Finally, we conclude this paper in Section 7.

2. BACKGROUND OF WORKFLOW SCHEDULING

2.1. Workflow scheduling problem

In general, a workflow application is represented as a directed acyclic graph (DAG) in which graph nodes represent tasks and graph edges represent data dependencies among the tasks with weights on the nodes representing computation complexity and weights on the edges representing communication volume. Therefore, workflow scheduling problem is usually considered as a special case of the DAG scheduling problem, which is an Non-deterministic polynomial (NP)-complete problem [7]. Thus, even though the DAG scheduling problem can be solved by using exhaustive search methods, the complexity of generating the schedule becomes very high.

The overall finish/completion time of an application is usually called the schedule length or makespan. So, the objective of workflow scheduling techniques is to minimize the makespan of a parallel application by proper allocation of the tasks to the processors/resources and arrangement of task execution sequences.

Let us assume workflow $W(T, E)$ consists of a set of tasks, $T = \{T_1, T_2, \ldots, T_x, .., T_y, T_n\}$, and a set of dependencies among the tasks, $E = \{< T_a, T_b >, ..., < T_x, T_y >\}$, where T_x is the parent task of T_y. $R = \{R_1, R_2, \ldots, R_x, \ldots, R_y, R_m\}$ is the set of available resources in the computational grid. Therefore, the workflow scheduling problem is the mapping of workflow tasks to grid resources ($T \rightarrow R$) so that the makespan M is minimized.

Generally, a workflow task is a set of instructions that can be executed on a single processing element of a computing resource. In a workflow, an entry task does not have any parent task and an exit task does not have any child task. In addition, a child task cannot be executed until all of its parent tasks are completed. At any time of scheduling, the task that has all of its parent tasks finished is called a *Ready* task.

2.2. Existing workflow scheduling algorithms

As workflow scheduling is an NP-complete problem, we rely on heuristic-based and metaheuristic-based scheduling strategies to achieve near optimal solutions within polynomial time. In the following, we present some of the well-known heuristics and metaheuristics (see Table I) for workflow scheduling in grid systems.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

2.3. Heuristics

2.3.1. Myopic. Myopic is an individual task scheduling heuristic that is considered as the simplest scheduling method for scheduling workflow applications because it makes scheduling decisions on the basis of only one individual task. The myopic algorithm presented in [8] has been implemented in some grid systems such as Condor DAGMan [9]. It schedules an unmapped ready task, in arbitrary order to the resource, which is expected to complete that task earliest, until all tasks have been scheduled.

2.3.2. Min–min. A list scheduling heuristic prioritizes workflow tasks and schedules the tasks based on their priorities. Min–Min is a list scheduling heuristic that assigns the task priority on the basis of its expected completion time (ECT) on a resource. This heuristic organizes the workflow tasks into several independent task groups and schedules each group of independent tasks iteratively. In every iteration, it takes the set of all unmapped independent tasks T and generates the minimum ECTs (MCT) for each task t in T, where $\text{MCT}_t = \min_{r \epsilon R}\text{ECT}(t, r)$; R is the set of resources available, and $\text{ECT}(t, r)$ is the amount of time resource r takes to execute task t.

Where $\text{MCT}_t = \min_{r \in R}\text{ECT}(t, r)$; R is the set of resources available, and $\text{ECT}(t, r)$ is the amount of time resources r takes to execute task t.

Then, the task having minimum MCT value over all tasks is selected to be scheduled first at this iteration to the corresponding resource for this MCT (hence, the name is min–min). In this way, min–min schedules other independent tasks in T and moves to the next iteration until T is empty.

The intuition behind min–min is to consider all unmapped independent tasks during each mapping decision, whereas myopic only considers one task at a time. Min–min was proposed by Maheswaran *et al*. [10] and has been employed for scheduling workflow tasks in grid projects such as vGrADS [11] and Pegasus [12].

2.3.3. Max–min. The max–min heuristic is very similar to min–min. The only difference is the max–min heuristic sets the priority to the task that requires the longest execution time rather than the shortest execution time. In each iterative step, after obtaining the set of MCT values for all

Algorithm 1 Myopic scheduling algorithm

```
1:  PROCEDURE: Myopic
2:  Input: Workflow W(T, E), TaskDependencyList, Resource Set R
3:  begin
4:     while All tasks in TaskList are not completed do
5:        TaskList ← Get unscheduled Ready tasks from workflow W
6:        Schedule Task (TaskList, R)
7:        Update TaskDependencyList
8:     end while
9:  end
10: PROCEDURE: Schedule Task
11: Input: TaskList, Resource Set R
12: begin
13:    index ← 0
14:    while TaskList[index] ≠ null do
15:       t ← TaskList[index]
16:       r ← Get a resource from R that can complete t at earliest
17:       Schedule t on r
18:       Update status of r
19:       index ← index + 1
20:    end while
21: end
```

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

unmapped independent tasks, a task having the maximum MCT is chosen to be scheduled on the resource, which is expected to complete the task at the earliest time.

Intuitively, max–min attempts to minimize the total workflow execution time by assigning longer tasks to comparatively best resources. Max–min was also proposed by Maheswaran *et al*. [10] and has been used for scheduling workflow tasks in Pegasus [12].

2.3.4. HEFT. Heterogeneous Earliest Finish Time (HEFT) [13] is a well-established list scheduling algorithm, which gives higher priority to the workflow task having higher rank value. This rank value is calculated by utilizing average execution time for each task and average communication time between resources of two successive tasks, where the tasks in the CP have comparatively higher rank values. Then, it sorts the tasks by the decreasing order of their rank values, and the task with a higher rank value is given higher priority. In the resource selection phase, tasks are scheduled in the order of their priorities, and each task is assigned to the resource that can complete the task at the earliest time.

Let us consider $|T_x|$ to be the size of task T_x and R be the set of resources available with average processing power $|R| = \sum_{i=1}^{n} |R_i|/n$. Thus, the average execution time of the task is defined as

$$E(T_x) = \frac{|T_x|}{|R|} \tag{1}$$

Let $\overline{T}_{xy}$ be the size of data to be transferred between task T_x and T_y, and R be the set of resources available with average data processing capacity $\overline{R} = \sum_{i=1}^{n} \overline{R_i}/n$. Thus, the average data transfer time for the task is defined as

$$D(T_{xy}) = \frac{\overline{T}_{xy}}{\overline{R}} \tag{2}$$

$E(T_x)$ and $D(T_{xy})$ are used to calculate the rank of a task. For an exit task, the rank value is,

$$\text{rank}(T_x) = E(T_x) \tag{3}$$

Now, the rank value of other tasks in the workflow can be computed recursively on the basis of Equations (1), (2), and (3) and is represented as

$$\text{rank}(T_x) = E(T_x) + \max_{T_y \in \text{succ}(T_x)} (D(T_{xy}) + \text{rank}(T_y)) \tag{4}$$

Because a workflow is represented as a DAG, the rank values of the tasks are calculated by traversing the task graph in a breadth-first search (BFS) manner in the reverse direction of task dependencies (i.e., starting from the exit tasks).

The advantage of using HEFT over min–min or max–min is that while assigning priorities to the tasks, it considers the entire workflow rather than focusing on only unmapped independent tasks at each step. HEFT algorithm was proposed by Topcuoglu *et al*. [13], and it has been used in the ASKALON workflow manager [8, 14] to provide scheduling for a quantum chemistry application WIEN2K [2].

Table I. Summary of workflow scheduling algorithms.

Scheduling method	Scheduling type	Project	Organization
Myopic	Heuristic	Condor DAGMan	University of Wisconsin-Madesion, USA
Min–min	Heuristic	vGrADS	Rice University, USA
Max–min	Heuristic	vGrADS	Rice University, USA
HEFT	Heuristic	ASKALON	University of Innsbruck, Austria
GRASP	Metaheuristic	Pegasus	University of Southern California
GA	Metaheuristic	ASKALON	University of Innsbruck, Austria

HEFT, Heterogeneous Earliest Finish Time; GRASP, greedy randomized adaptive search procedure; GA, genetic algorithm.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Algorithm 2 MinMin scheduling algorithm

```
1: PROCEDURE: MinMin
2: Input: Workflow W(T, E), TaskDependencyList, Resource Set R
3: begin
4:    while all tasks in T are not completed do
5:       TaskList ← Get unscheduled Ready tasks from workflow W
6:       Schedule Task (TaskList, R)
7:       Update TaskDependencyList
8:    end while
9: end
10: PROCEDURE: Schedule Task
11: Input: TaskList, Resource Set R
12: begin
13:    while all tasks in TaskList are not scheduled do
14:       for all t ∈ TaskList do
15:          for all r ∈ R do
16:             Calculate ECT(t, r)
17:          end for
18:          MCT(t, r_t) ← min_{r∈R} ECT(t, r)
19:       end for
20:       MCT(t_m, r_m) ← min_{t∈TaskList} MCT(t, r_t)
21:       Schedule t_m on r_m
22:       Remove t_m from TaskList
23:       Update status of r_m
24:    end while
25: end
```

2.4. *Metaheuristics*

2.4.1. GRASP. Greedy randomized adaptive search procedure (GRASP) [11] is an iterative randomized search technique. In GRASP, a number of iterations are conducted to search a possible optimal solution for mapping tasks on resources. A solution is generated at each iterative step, and the best solution is kept as the final schedule. This searching procedure terminates when the specified termination criterion, such as the completion of a certain number of iterations, is satisfied. GRASP can generate better schedules than the other scheduling techniques stated previously as it searches the whole solution space considering entire workflow and available resources.

2.4.2. GA. Similar to GRASP, genetic algorithm (GA) [15] is also a metaheuristic-based scheduling technique that allows a high-quality solution to be derived from a large search space in polynomial time by applying the principles of evolution. A GA combines exploitation of best solution from past searches with the exploration of new regions of solution space. Instead of creating a new solution by randomized search as in GRASP, GA generates new solutions at each step by randomly modifying the good solutions generated in previous steps, which results in a better schedule within a less time. Jia *et al*. [16] have employed GA-based approach to schedule workflows in grids.

3. DCP-G ALGORITHM FOR WORKFLOW SCHEDULING

For a task graph, the lower and upper bounds of starting time for a task are denoted as the absolute earliest start time (AEST) and the absolute latest start time (ALST), respectively. In the DCP algorithm [6], the tasks on the CP have equal AEST and ALST values as delaying these tasks affects the overall execution time for the task graph. The first task on the CP is mapped to the processor identified for it. This process is repeated until all the tasks in the graph are mapped.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Algorithm 3 MaxMin scheduling algorithm

1: **PROCEDURE**: MinMin
2: Input: Workflow $W(T, E)$, $TaskDependencyList$, Resource Set R
3: **begin**
4: **while all** tasks in T are not completed **do**
5: $TaskList \leftarrow$ Get unscheduled $Ready$ tasks from workflow W
6: Schedule Task ($TaskList$, R)
7: Update $TaskDependencyList$
8: **end while**
9: **end**
10: **PROCEDURE**: Schedule Task
11: Input: $TaskList$, Resource Set R
12: **begin**
13: **while all** tasks in $TaskList$ are **not** scheduled **do**
14: **for all** $t \in TaskList$ **do**
15: **for all** $r \in R$ **do**
16: Calculate $ECT(t, r)$
17: **end for**
18: $MCT(t, r_t) \leftarrow min_{\ r \in R}\ ECT(t, r)$
19: **end for**
20: $MCT(t_m, r_m) \leftarrow max_{\ t \in TaskList}\ MCT(t, r_t)$
21: Schedule t_m on r_m
22: Remove t_m from $TaskList$
23: Update status of r_m
24: **end while**
25: **end**

However, this algorithm is designed for scheduling all the tasks in a task graph with arbitrary computation and communication times to a multiprocessor system with unlimited number of fully connected identical processors. But grids [17] are heterogeneous and dynamic environments consisting of computing, storage, and network resources with different capabilities and availability. Therefore, to work on grids, the DCP algorithm needs to be extended in the following manner:

- For a task, the initial AEST and ALST values are calculated for the resource, which provides the minimum execution time for the task. The overall objective is to reduce the length of the CP at every pass. We follow the intuition of the min–min heuristic in which a task is assigned to the resource that executes it fastest.
- For mapping a task on the CP, all the available resources are considered by DCP-G, as opposed to the DCP algorithm, which considers only the resources (processors) occupied by the parent and child tasks. This is because, in the latter case, the execution time is not varied for different processors, and only the communication time between the tasks could be reduced by assigning tasks to the same resource. However, in grids, the communication and computation times are both liable to change because of resource heterogeneity.
- When a task is mapped to a resource, its execution time and data transfer time from the parent node are updated accordingly. This changes the AEST and ALST of succeeding tasks.

In the following, we discuss some of the principle features of the algorithm. In the first part of the discussion, we describe the techniques used to calculate AEST and ALST that are necessary for task selection. Then, we discuss the task selection methodology followed by the resource selection strategy. The DCP-G scheduling algorithm is formalized and illustrated with an example at the end of this section. Table II provides some terms and their meanings that are used in the subsequent discussion.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Table II. Symbols and their meanings.

Symbol	Meaning
AET(t)	Absolute execution time of task t
ADTT(t)	Absolute data transfer time for task t
AEST(t, R)	Absolute earliest start time of task t on resource R
ALST(t, R)	Absolute latest start time of task t on resource R
$C_{t,t_k}(R_t, R_{t_k})$	Data transfer time between task t and t_k that are scheduled to resources R_t and R_{t_k} respectively
PC(R)	Processing capacity of resource R
BW(R)	Bandwidth of the network link that connects resource R to Global grid
DCPL	Length of a dynamic critical path in a workflow

Algorithm 4 HEFT scheduling algorithm

1: **PROCEDURE**: HEFT
2: Input: Workflow $W(T, E)$, $TaskDependencyList$, Resource Set R
3: **begin**
4: **for** all $t \in T$ of workflow W
5: Calculate execution time for t according to (1)
6: **end for**
7: **for** all $e \in E$ of workflow W
8: Calculate data transfer time for e according to (2)
9: **end for**
10: Run BFS following reverse task dependency and calculate $Rank$ value for each task according to (3) (4)
11: **while all** tasks in T are not completed **do**
12: $TaskList \leftarrow$ Get unscheduled $Ready$ tasks from workflow W
13: Schedule Task ($TaskList$, R)
14: Update $TaskDependencyList$
15: **end while**
16: **end**
17: **PROCEDURE**: Schedule Task
18: Input: $TaskList$, Resource Set R
19: **begin**
20: Sort $TaskList$ in descending order of task's $Rank$ value
21: $index \leftarrow 0$
22: **while** $TaskList[index] \neq null$ **do**
23: $t \leftarrow TaskList[index]$
24: $r \leftarrow$ Get a resource from R that can complete t at earliest
25: Schedule t on r
26: Update status of r
27: **end while**
28: **end**

3.1. Calculation of AEST and ALST in DCP-G

In DCP-G, the start time of a task is not finalized until it is mapped to a resource. Here, we also introduce two more attributes: the absolute execution time (AET) of a task, which is the minimum execution time of the task, and absolute data transfer time (ADTT), which is the minimum time required to transfer the output of the task given its current placement. Initially, AET and ADTT are calculated as

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Algorithm 5 GRASP scheduling algorithm

```
1: PROCEDURE: GRASP
2: Input: Workflow W(T, E), TaskDependencyList, Resource Set R
3: begin
4:     bestSchedule ← φ
5:     for all t ∈ T of workflow W
6:         Select r randomly from R
7:         Tuple(t, r) ← Schedule t on r
8:         bestSchedule ← bestSchedule ∪ Tuple(t, r)
9:     end for
10:    while stopping criterion is not satisfied do
11:        schedule ← Create Schedule(T, R, TaskDependencyList, bestSchedule)
12:        if makespan(schedule) < makespan(bestSchedule) then
13:            bestSchedule ← schedule
14:    end while
15: end
16: PROCEDURE: Create Schedule
17: Input: TaskList, Resource Set R, TaskDependency List D, Schedule S
18: begin
19:    TupleList ← φ
20:    schedule ← φ
21:    while TaskList is not empty do
22:        for all t ∈ TaskList
23:            for all r ∈ R
24:                I_{t,r} ← makespanIncrease(t,r,D,S)
25:                TupleList ← TupleList ∪ (t, r, I_{t,r})
26:            end for
27:        end for
28:        minI ← min_{t∈succ(T),r∈succ(R)} I_{t,r}
29:        maxI ← max_{t∈succ(T),r∈succ(R)} I_{t,r}
30:        Tuples ← Get tuples from TupleList provided I_{t,r} < {minI + α(maxI − minI)}
31:        (t̄, r̄, I_{t̄,r̄}) ← Select a tuple randomly from Tuples
32:        Remove t̄ from TaskList
33:        schedule ← schedule ∪ (t̄, r̄)
34:    end while
35:    return schedule
36: end
```

$$\mathrm{AET}(t) = \frac{\mathrm{Task_size}(t)}{\mathrm{MAX}_{k \epsilon \mathrm{ResourceList}}\{\mathrm{PC}(R_k)\}}$$

$$\mathrm{ADTT}(t) = \frac{\mathrm{Task_output_size}(t)}{\mathrm{MAX}_{k \epsilon \mathrm{ResourceList}}\{\mathrm{BW}(R_k)\}}$$

where $\mathrm{PC}(R_k)$ and $\mathrm{BW}(R_k)$ are processing capability and transfer capacity (i.e., bandwidth) of resource R_k, respectively.

Whenever a task t is scheduled to a resource, the values of $\mathrm{AET}(t)$ and $\mathrm{ADTT}(t)$ are updated accordingly. Therefore, the AEST of a task t on resource R, denoted by $\mathrm{AEST}(t, R)$ is recursively defined as

$$\mathrm{AEST}(t, R) = \mathrm{MAX}_{1 \leqslant k \leqslant p}\{\mathrm{AEST}(t_k, R_{t_k}) + \mathrm{AET}(t_k) + C_{t,t_k}(R_t, R_{t_k})\}$$

where t has p parent tasks, t_k is the kth parent task, and
$\mathrm{AEST}(t, R) = 0$ if t is an entry task.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

$C_{t,t_k}(R_t, R_{t_k}) = 0$ if $R_t = R_{t_k}$.
$C_{t,t_k}(R_t, R_{t_k}) = \text{ADTT}(t_k)$ if t and t_k are not scheduled.

Here, the communication time between two tasks is considered to be zero if they are mapped to the same resource and equal to the ADTT of the parent task if the child is not mapped yet. Using this definition, the AEST values can be computed by traversing the task graph in a breadth-first manner beginning from the entry tasks.

Once AESTs of all the tasks are computed, it is possible to calculate DCP length (DCPL), which is the schedule length of the partially mapped workflow. DCPL can be defined as

$$\text{DCPL} = \text{MAX}_{1 \leqslant i \leqslant n}\{\text{AEST}(t_i, R_{t_i}) + \text{AET}(t_i)\}$$

where n is the total number of tasks in the workflow.

After computing the DCPL, the values of ALST can be calculated by traversing the task graph in a breadth first manner but in the reverse direction. Thus, the ALST of a task t in resource R, denoted as ALST(t,R), can be recursively defined as

$$\text{ALST}(t, R) = \text{MIN}_{1 \leqslant k \leqslant c}\{\text{ALST}(t_k, R_{t_k}) - \text{AET}(t) - C_{t,t_k}(R_t, R_{t_k})\}$$

Algorithm 6 GA scheduling algorithm

```
1: PROCEDURE: GA
2: Input: Workflow W(T, E), TaskDependency List D, Resource Set R, Population size P
3: begin
4:    PopulationList ← φ
5:    for size of population P
6:       PopulationList ← PopulationList ∪ Generate Initial Population(T,D,R)
7:    end for
8:    bestMapping ← Randomly select a mapping from the PopulationList
9:    while stopping criterion is not satisfied do
10:      Perform Crossover and add new individuals to PopulationList
11:      for all individual or mapping ∈ PopulationList
12:         Perform Mutation on mapping
13:         Generate makespan for mapping using D
14:         if makespan(mapping) < makespan(bestMapping) then
15:         bestMapping ← mapping
16:      end for
17:      Reduce size of PopulationList to P using Roulette Wheel Elitism
18:   end while
19: end
20: PROCEDURE: Generate Initial Population
21: Input: TaskList T, TaskDependency List D, Resource Set R
22: Output: mapping
23: begin
24:    mapping ← φ
25:    for all t ∈ T
26:       Select r randomly from R
27:       Tuple(t, r) ← Assign t on r
28:       mapping ← mapping ∪ Tuple(t, r)
29:    end for
30:    Generate makespan for mapping using D
31:    return mapping
32: end
```

Copyright © 2013 John Wiley & Sons, Ltd.

where t has c child tasks, t_k is the kth child task, and
ALST(t, R) = DCPL − AET(t) if t is an exit task.
$C_{t,t_k}(R_t, R_{t_k}) = 0$ if $R_t = R_{t_k}$.
$C_{t,t_k}(R_t, R_{t_k})$ = ADTT(t_k) if t and t_k are not mapped.

3.2. *Task selection*

During the scheduling process, the CP in the task graph determines the schedule length of the partially scheduled workflow. Thus, while scheduling, it is necessary to give priority to the tasks in CP. However, as the scheduling process progresses, the CP can be changed dynamically, that is,

Algorithm 7 DCP scheduling algorithm

1: **PROCEDURE**: DCP
2: Input: Workflow $W(T, E)$, $TaskDependencyList$, Resource Set R
3: **begin**
4: **for** all $t \in T$ of workflow W
5: Calculate AET and ADTT for t according to Section 3.1
6: **end for**
7: **for** all $t \in T$ of workflow W
8: Calculate AEST for t running BFS according to Section 3.1
9: **end for**
10: Calculate DCPL
11: **for** all $t \in T$ of workflow W
12: Calculate ALST for t running BFS following reverse task dependency according to Section 3.1
13: **end for**
14: **while all** tasks in T are not completed **do**
15: $TaskList \leftarrow$ Get unscheduled $Ready$ tasks from workflow W
16: Schedule Task ($TaskList$, R)
17: Update $TaskDependencyList$
18: **end while**
19: **end**
20: **PROCEDURE**: Schedule Task
21: Input: $TaskList$, Resource Set R
22: **begin**
23: **while** $TaskList$ is **not** empty **do**
24: $T_{ct} \leftarrow$ Get Critical Task from all tasks of $TaskList$ according to Section 3.2
25: $T_{ct_c} \leftarrow$ Get Critical Child Task from all child tasks of T_{ct} according to Section 3.3
26: $r \leftarrow$ Get a resource from R that can provide earliest start time for both T_{ct} and T_{ct_c}
27: Schedule T_{ct} on r
28: Update status of r
29: Remove T_{ct} from $TaskList$
30: **for** all $t \in T$ of workflow W
31: Calculate AEST for t running BFS
32: **end for**
33: Calculate DCPL
34: **for** all $t \in T$ of workflow W
35: Calculate ALST for t running BFS following reverse task dependency
36: **end for**
37: **end while**
38: **end**

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

a task on a CP at one step may not be on CP at the next step because of the dynamically changing resource behavior. This is why, in dynamic environment such as grids, CP in the workflow is called DCP because it is likely to be changed at every step of scheduling.

The tasks on DCP have the same upper and lower bounds of start time, that is, have the same AEST and ALST. Therefore, a task in DCP-G is considered to be on the CP and called critical task if its AEST and ALST values are equal. To reduce the value of DCPL at every step, the task selected for scheduling is the one that is on CP and has no unmapped parent tasks, where ties are broken by choosing the critical task with lower AEST.

3.3. *Resource selection*

After identifying a critical task, we need to select an appropriate resource for that task. We select the resource that provides the minimum execution time for that task. This is discovered by checking all the available resources for one that minimizes the potential start time of the critical child task on the same resource, where the critical child task is the one with the least difference of AEST and ALST among all the child tasks of the critical task (ties are broken by choosing the critical child task with higher AEST). Finally, the critical task is mapped to the resource that provides the earliest combined start time.

3.4. *Methodology*

First, the DCP-G algorithm computes the initial AET, ADTT, AEST, and ALST of all the tasks. Then, it selects the task with the smallest difference between its AEST and ALST, where ties are broken by choosing the one with smaller AEST. According to the discussion in Section 3.2, this task is on DCP and called critical task. The critical child task of critical task is also determined in the same manner. The algorithm then computes the start time of critical task for all available resources considering the finish time of all of its parent tasks and searches for a slot starting with this start time with the duration of its execution time. The resource that gives the earliest start time for both t_{ct} and its critical child task is selected.

After selecting the suitable resource R, the algorithm calculates start time $\text{AEST}(t_{ct}, R)$ and duration $\text{AET}(t_{ct})$ for t_{ct} on this resource and updates the actual start and execution times for t_{ct} accordingly. The AEST and ALST values of other tasks are updated at the end of each scheduling step to determine the next critical task. This process continues until all the tasks in the workflow are scheduled.

3.5. *DCP-G example*

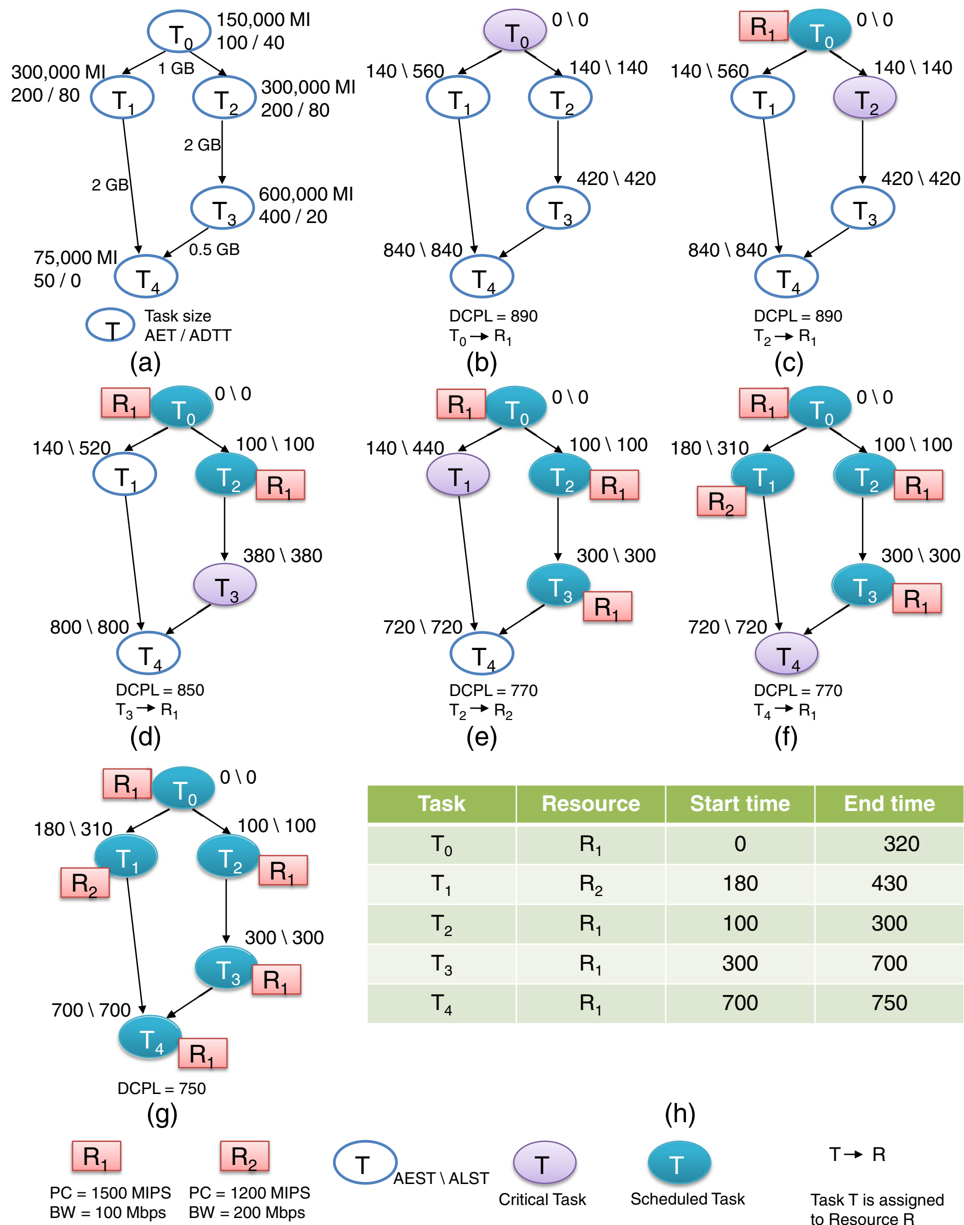
Figure 2 illustrates the DCP-G algorithm with a step-by-step explanation of the mapping of tasks in a sample workflow. The sample workflow consists of five tasks denoted as T_0, T_1, T_2, T_3, and T_4 with different execution and data transfer requirements. The length and size of the output of each task shown in Figure 2(a) are measured in million instructions (MI) and gigabytes (GB), respectively. The tasks are to be mapped to two grid resources R_1 and R_2 with processing capability (PC) and transfer capacity, that is, bandwidth (BW) as indicated at the bottom of Figure 2.

First, the AET and ADTT values for each task are calculated as shown in Figure 2(a). Then, using these values, AEST and ALST of all the tasks are calculated according to Section 3.1 (Figure 2(b)). Because T_0, T_2, T_3, and T_4 have equal AEST and ALST, they are on CP with T_0 as the highest task. Hence, T_0 is selected as the critical task and mapped to resource R_1, which gives T_0 the minimum combined start time. At the end of this step, the schedule length of the workflow (i.e., DCPL) is 890. Similarly, in Figure 2(c), T_2 is selected as the critical task and mapped to R_1. As both T_0 and T_1 are mapped to R_1 and the data transfer time of T_0 is now zero, the AEST and ALST of all the tasks are changed, and the schedule length becomes 850 (Figure 2(d)). In the next step, T_3 is mapped to R_1 as well, and the DCPL is reduced to 770 as the data transfer time for T_2 is zero.

Now, T_4 is the only task remaining on the CP (Figure 2(e)). However, one of its parent tasks, T_1, is not mapped yet, and therefore, T_1 is selected as the critical task. As T_2 and T_3 are already mapped to R_1, the start time of T_1 on R_1 is 700. Therefore, T_1 is mapped to R_2 as its start and end times on

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



Task	Resource	Start time	End time
T_0	R_1	0	320
T_1	R_2	180	430
T_2	R_1	100	300
T_3	R_1	300	700
T_4	R_1	700	750

Figure 2. Example of workflow scheduling using dynamic critical path for grids (DCP-G) algorithm.(a) Calculation of AET and ADTT values for each task; (b) Calculation of AEST and ALST values for each task; (c) Re-Calculation of AEST, ALST, and DCPL value based on new critical tasks; (d) Re-Calculation of AEST, ALST, and DCPL value based on new critical tasks; (e) Re-Calculation of AEST, ALST, and DCPL value based on new critical tasks; (f) Re-Calculation of AEST, ALST, and DCPL value based on new critical tasks; (g) Re-Calculation of AEST, ALST, and DCPL value based on new critical tasks; (h) Final schedule generated by DCP-G algorithm.

R_2 are 180 and 430, respectively. Finally, when T_4 is mapped to R_1 (Figure 2(g)), all the tasks have been mapped, the schedule length cannot be improved any further, and a schedule length of 750 is obtained. The final schedule generated by DCP-G is shown in a table in Figure 2(h).

4. PERFORMANCE EVALUATION

We evaluate DCP-G by comparing the schedules produced by it against those produced by the other algorithms described previously for a variety of workflows in a simulated grid environment.

Copyright © 2013 John Wiley & Sons, Ltd.

In this section, first, we describe our simulation methodology and setup and then present the results of experiments.

4.1. Simulation methodology

We use GridSim [18] toolkit to simulate the application and grid environment for our simulation. The GridSim toolkit allows modeling and simulation of various entities in parallel and distributed computing environment, such as systems users, applications, resources, and resource brokers (schedulers) for design and evaluation of scheduling algorithms. It provides a comprehensive facility for creating different types of heterogeneous resources for executing compute and data intensive applications. A resource can be a single processor or multiprocessor with shared or distributed memory and managed by time or space-shared schedulers. The processing nodes within a resource can be homogeneous or heterogeneous in terms of processing capability, configuration, and availability. We model different entities of GridSim in the following manner.

4.1.1. Workflow model. We implement a workflow generator that can generate various formats of weighted pseudo-application workflows. The following input parameters are used to create a workflow.

- N, the total number of tasks in the workflow.
- α, the shape parameter represents the ratio of the total number of tasks to the width (i.e., maximum number of nodes in a level). Thus, width $W = \lceil N/\alpha \rceil$.
- Type of workflow: Our workflow generator can generate three types of workflow, namely parallel workflow, fork-join workflow, and random workflow.

Parallel workflow: In parallel workflows [19], a group of tasks creates a chain of tasks with one entry and one exit task; there can be several such chains in one workflow. Here, one task is dependent on only one task, although the tasks at the head of chains are dependant on the entry task, and the exit task is dependent on the tasks at the tail of chains. The number of levels in a parallel workflow can be specified as

$$\text{Number of levels} = \left\lfloor \frac{N-2}{W} \right\rfloor$$

Fork-join workflow: In fork-join workflows [2], forks of tasks are created and then joined. So, in this kind of workflows, there can be only one entry task and one exit task, but the number of tasks in each level depends on the total number of tasks and width of that level, W. The number of levels in a fork-join workflow can be specified as

$$\text{Number of levels} = \left\lfloor \frac{N}{W+1} \right\rfloor$$

Random workflow: In random workflows, the dependency and number of parent tasks of a task, which equals to the indegree of a node in DAG representation of the workflow, is generated randomly. Here, the task dependency and the indegree are calculated as

$$\text{Maximum indegree}(T_i) = \left\lfloor \frac{W}{2} \right\rfloor$$

$$\text{Minimum indegree}(T_i) = 1$$

$$\text{Parent}(T_i) = \{T_x | T_x \epsilon [T_0....T_{i-1}]\} \text{ if } T_i \text{ is not a root task.}$$

$$\text{where } x \text{ is a random number and } 0 <= x <= \left\lfloor \frac{W}{2} \right\rfloor.$$

$$\text{Parent } (T_i) = \{\phi\} \text{ if } T_i \text{ is a root task.}$$

In Figure 3, a sample of each type of workflow is illustrated, where $N = 10$ and $\alpha = 5$. In simulation, we use MI to denote the length of tasks and megabyte (MB) to denote the output data size of each task.

Copyright © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/cpe

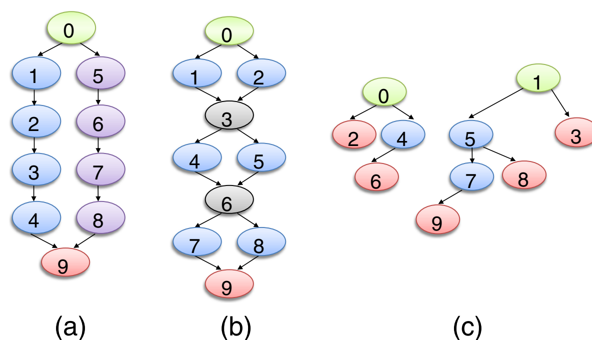


Figure 3. Three sample workflows: (a) parallel; (b) fork-join; (c) random.

Table III. Resources used for performance evaluation.

Resource name/site	Location	Number of nodes	Single PE rating (MIPS)	Mean load
RAL	UK	41	1140	0.9
NorduGrid	Norway	17	1176	0.9
NIKHEF	Netherlands	18	1166	0.9
Milano	Italy	7	1000	0.5
Torino	Italy	4	1330	0.5
Catania	Italy	5	1200	0.6
Padova	Italy	13	1000	0.4
Bologna	Italy	20	1140	0.8

PE, Processing Element

4.1.2. Resource model. As the execution environment for tasks in scientific workflows is heterogeneous, we use heterogeneous resources with different processing capabilities. Here, we choose 8 resources (Table III) from the European DataGrid 1 test bed [20] used for simulation in [21]. The processing capability of the resources is measured in million instructions per second (MIPS) and the bandwidth in Megabits per second (Mbps).

4.2. Simulation setup

The workflows for evaluation are generated using the following parameters:

- Type = parallel, fork-join, random
- $N = \{50, 100, 200, 300\}$
- $\alpha = \{10\}$

Here, the size of each task in the workflow is generated from a uniform distribution between $100,000$ and $500,000$ MI, whereas the output data size of each task is also generated from a uniform distribution between 1 and 5 GB.

For GRASP, we run 600 iterations to map tasks to resources, and then, we select the best schedule out of the generated schedules. For GA, parameters for various genetic operators, such as selection, crossover, and mutation, are set using those applied in previous studies [16]. Table IV shows the values of different parameters used for simulating GA.

4.3. Results and observations

We evaluate the scheduling heuristics on the basis of the total makespan produced and the time required for scheduling workflows. Makespan is the total time required for executing an entire workflow.

Two sets of experiments were carried out. In the first set, we consider an ideal case, where the availability and load of grid resources remain static over time. For this environment, we statically map tasks to resources according to different strategies and execute tasks accordingly. In the next set, we evaluate the strategies in a more realistic scenario, where the availability and load of grid

Copyright © 2013 John Wiley & Sons, Ltd.

Table IV. Parameters of genetic algorithm.

Parameter	Value/type
Population size	60
Crossover probability	0.7
Swapping mutation probability	0.5
Replacing mutation probability	0.8
Fitness function	Makespan of workflow
Selection scheme	Elitism Roulette wheel
Stopping condition	300 iterations
Initial individuals	Randomly generated

resources vary over time. In this case, the instantaneous load (i.e., number of PEs occupied) for each resource during the simulation is derived from a Gaussian distribution, as performed in [21].

4.3.1. Execution time in static environment. The graph in Figure 4 plots the execution time of parallel, fork-join, and random workflows of 50, 100, 200, and 300 tasks for seven workflow scheduling strategies namely, Myopic, min–min, max–min, HEFT, DCP-G, GRASP, and GA in static environment.

For random workflow (Figure 4(c)), DCP-G can generate schedules with up to 13% less makespan than HEFT, which generates better schedule than myopic, min–min, and max–min. Because from any task in the random workflow there can be multiple paths to an exit node, dynamically assigning priorities to tasks helps DCP-G to generate better schedules. As GRASP and GA search the entire solution space for the best schedule, they generate $20-30\%$ better schedule than DCP-G.

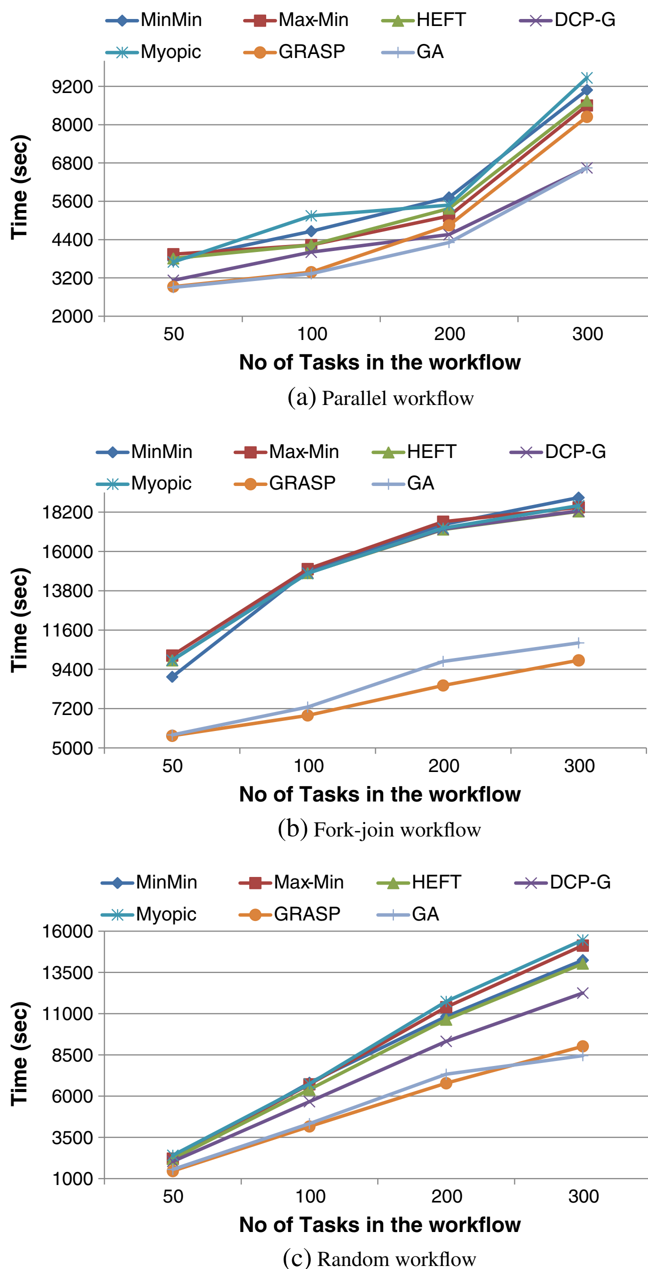
However, the execution time of fork-join workflows (Figure 4(b)) shows a significant difference between heuristic-based and metaheuristic-based approaches. During the process of task selection for mapping, heuristic-based approaches do not consider the impact of mapping child tasks. Thus, all the heuristic-based techniques generate similar schedule with DCP-G being marginally better. However, in a fork-join workflow, a join task depends on the output of all the forked independent tasks that precede it. If this join task is assigned to a resource with low bandwidths to other resources, increase in data transfer time impacts the makespan adversely. However, metaheuristics (GA, GRASP) consider the impact of mapping not only the parent fork tasks but also the child join tasks and are therefore able to generate $40-50\%$ better schedule than DCP, which is the best among heuristic-based methods.

According to Figure 4(a), the execution time of parallel workflow exhibits a slow exponential growth with the increase in workflow size. The reason is that, unlike fork-join workflows, the number of unmapped ready tasks at every step of scheduling in a parallel workflow is always equal to W, and a task becomes ready as soon as its parent finishes. Thus, when the available resources are less than unmapped ready tasks, the time spent by some of these tasks in waiting to be scheduled results in an increase in the total execution time. In case of parallel workflows, DCP-G and GA generate better schedules than others, and the makespan is reduced by at least 20%. Here, the execution time of GRASP rises beyond that of DCP-G as the number of candidate solutions for task mapping increases exponentially with the workflow size. This will be explained further in Section 4.3.3.

4.3.2. Execution time in dynamic environment. As the resource availability changes over time in dynamic environment, the resource availability information needs to be continuously updated after a certain period of time, and the tasks have to be remapped if necessary, depending on the updated availability of resources. Here, we compare the performance of rescheduling using DCP-G and other heuristic-based approaches against the static schedules generated by the metaheuristics.

Figure 5 shows the execution time of different scheduling techniques in dynamic environment, where resource information is updated every 50 s. The number of available processing elements and hence the number of tasks that can start execution in a resource varies with the load on the resource. However, for GA and GRASP, if a resource is heavily loaded and unavailable, the tasks mapped to that resource have to wait to be executed. This waiting time consequently impacts the start time of

Copyright © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/cpe



(a) Parallel workflow

(b) Fork-join workflow

(c) Random workflow

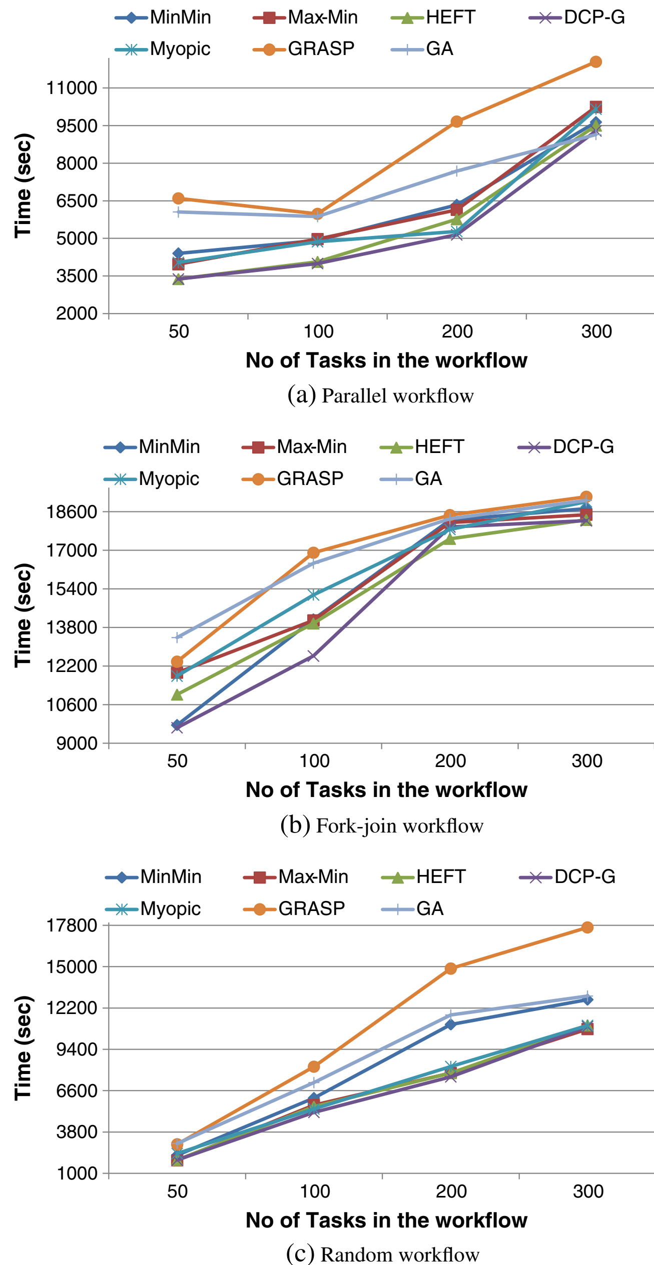
Figure 4. Execution time of different types of workflows for static environment. (a) parallel workflow; (b) fork-join workflow; (c) random workflow. GRASP, greedy randomized adaptive search procedure; HEFT, Heterogeneous Earliest Finish Time; GA, genetic algorithm; DCP-G, dynamic critical path for grids.

other dependent tasks and increases the makespan. This is reflected in the poor performance of GA and GRASP in the graphs in Figure 5. Moreover, heuristic-based approaches are able to generate up to 30% better schedules than these two metaheuristic-based approaches. Among the heuristics, DCP-G is able to achieve up to 6% better makespan than the others. This is because, in DCP-G, tasks on the CP waiting to be executed on a heavily loaded resource are rescheduled to resources with available PEs. This reduces the CP length; thus, the makespan for workflow execution is also reduced.

It can also be seen that heuristic-based approaches perform better in dynamic than static environments for the same workflows and experimental setup. This can be attributed to the fact that not only

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



(a) Parallel workflow

(b) Fork-join workflow

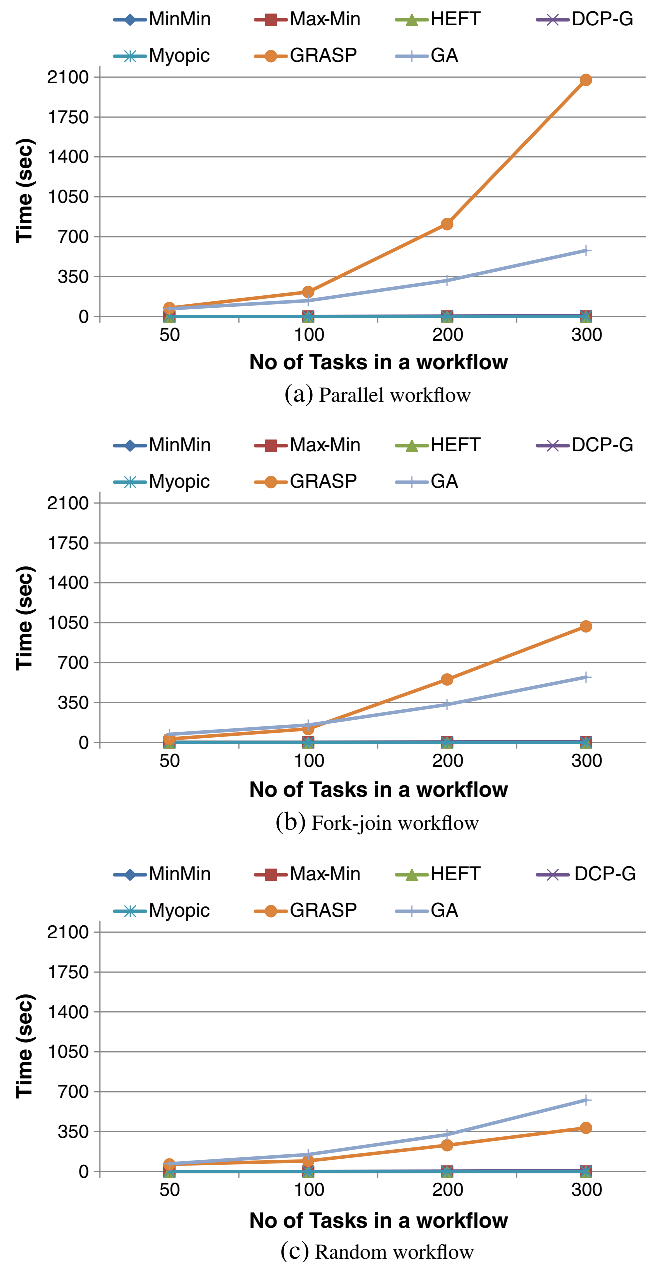
(c) Random workflow

Figure 5. Execution time of different types of workflows for dynamic environment. (a) Parallel workflow; (b) fork-join workflow; (c) random workflow. GRASP, greedy randomized adaptive search procedure; HEFT, Heterogeneous Earliest Finish Time; GA, genetic algorithm; DCP-G, dynamic critical path for grids.

the load but also the resource availability in dynamic environments is updated regularly. This means that the heuristics are able to adapt to resources that are more frequently available and therefore produce better schedules.

4.3.3. Scheduling time. Figure 6 shows the total scheduling time of a workflow for different scheduling techniques in the case of three types of workflow. Scheduling time is considered as the scheduling overhead, which constitutes a significant amount of time used by the scheduler to generate the schedule.

Copyright © 2013 John Wiley & Sons, Ltd.



(a) Parallel workflow

(b) Fork-join workflow

(c) Random workflow

Figure 6. Scheduling time of different scheduling approaches for various types of workflows. a) Parallel workflow; (b) fork-join workflow; (c) random workflow. GRASP, greedy randomized adaptive search procedure; HEFT, Heterogeneous Earliest Finish Time; GA, genetic algorithm; DCP-G, dynamic critical path for grids.

For the convenience of discussion, the average scheduling time (in milliseconds) for one task of parallel, fork-join, and random workflows to generate a single schedule for different scheduling techniques is presented in Table V. To generate a single schedule, myopic, min–min, max–min, and HEFT require nearly 1 ms for each task irrespective of the workflow size and type, whereas the average scheduling time of one task for DCP-G is 16–17 ms and does not vary with the workflow type as the task selection procedure is independent of workflow structure.

Scheduling time using GRASP increases exponentially not only with the increase of tasks in a workflow but also with the change in workflow structure. In each iteration, GRASP creates restricted

Copyright © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/cpe

Table V. Average scheduling time per task.

Scheduling strategy	Random workflow(ms)	Fork-join workflow(ms)	Parallel workflow(ms)
Myopic	1	1	1
Min–min	1	1	1
Max–min	1	1	1
HEFT	1	1	1
DCP-G	17	16	16
GRASP	1180	2840	5720
GA	1940	1780	1750

GRASP, greedy randomized adaptive search procedure; HEFT, Heterogeneous Earliest Finish Time; GA, genetic algorithm; DCP-G, dynamic critical path for grids.

candidate list (RCL) for each unmapped ready task and then selects a resource for the task randomly. When the number of tasks increases, RCL increases exponentially resulting in increased scheduling time. But the size of RCL is also dependent on workflow structure. For example, when a workflow consists of 300 tasks, parallel and fork-join structures contain 30 tasks in each level, whereas the random structure contains random number of levels as well as random number of tasks in each level. Thus, at every step, a parallel workflow has 30 ready tasks, fork-join workflow has maximum 30 ready tasks, and the average number of ready tasks in each level of random workflow is less than 30. Therefore, the scheduling time for random workflow is the lowest, and parallel workflow is the highest in this case.

However, scheduling time for GA does not change much with the workflow type because it executes the same number of genetic operations irrespective of workflow structure. But the size of each individual in the solution space is equal to the number of tasks in workflow. Thus, scheduling time increases with the increase in the size of workflow.

Although it is possible to reschedule at regular intervals in GA and GRASP, Table V shows that the scheduling times for these are at least 100 times as high as DCP-G and increase with the size of workflow as well. Hence, we did not incorporate rescheduling for GA and GRASP in the experiments for the dynamic environment.

4.4. Discussion

From Figure 4, it is evident that among the heuristic-based scheduling techniques, DCP-G can generate better schedule by up to 20% in static environment, especially for random and parallel workflows, irrespective of workflow size. GA and GRASP can generate more effective schedule than DCP-G for random and fork-join workflow, but they suffer from the problem of higher scheduling time. In our simulation, for parallel workflow of 300 tasks, DCP-G takes 6 s to map the tasks to resources, whereas GA and GRASP take 580 and 2076 s, respectively.

In dynamic environment, heuristics-based techniques adapt to the dynamic nature of resources and can avoid performance degradation. But metaheuristic-based techniques perform worse in this situation because of the unavailability of mapped resources at certain intervals. However, in dynamic environment, DCP-G can generate better schedule than other approaches, irrespective of workflow type and size.

5. CASE STUDY

This section provides a case study of adaptive scheduling and execution of a sample workflow application in dynamic grid environment under various temporal resource behaviors. To incorporate the dynamism and heterogeneity of the environment, we consider different types of resources with different processing capability measured in terms of millions of instructions per second (MIPS) and network connectivity measured in Mbps, where the size of the tasks and data transfer between the tasks in the workflow are measured in MI and GB, respectively.

Copyright © 2013 John Wiley & Sons, Ltd.

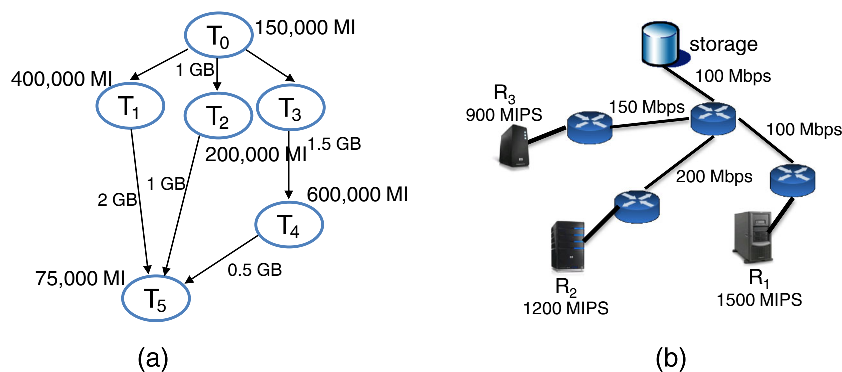


Figure 7. Testbed setup: (a) sample fork-join workflow; (b) resource connectivity.

We generate the mapping of workflow tasks to grid resources using DCP-G heuristic and GA metaheuristic, and calculate the total execution time for the following scenario: (i) static environment, where the status of resources do not change during workflow execution; (ii) dynamic environment, where the processing capability of resources changes during execution; and (ii) dynamic environment, where the network connectivity to resources change during execution.

5.1. Testbed setup

Figure 7(a) presents a sample fork-join workflow with 5 tasks as described by workflow model in Section 4.1.1. The size of the tasks in workflow and the amount of data to be transferred between the tasks are also shown in the figure for corresponding tasks and dependencies.

We assume that the resources are volatile in nature, and to improve the reliability, after execution of a task t, the corresponding resource transfers the generated intermediate data to the centralized storage or repository, and the resources scheduled to execute dependant tasks of t are required to download the data from the repository before they can start executing the task. However, if the same resource executes both t and its dependant tasks, then it does not need to download the data from the repository.

For this case study, we consider three grid resources distributed globally. Figure 7(b) shows the connectivity among these resources, where bandwidth of the network connections of these resources to the centralized storage or repository is also stated in the figure.

5.2. Schedule generation

The total execution time of all the tasks in the workflow for adaptive and nonadaptive scheduling techniques are illustrated in Figure 8. Figure 8(a) shows the processing capability of R_1, R_2, and R_3 over time, whereas Figure 8(b) shows the bandwidth of network connections to these resources. As we can see from these figures, the status of the resources does not change over time because the environment is static.

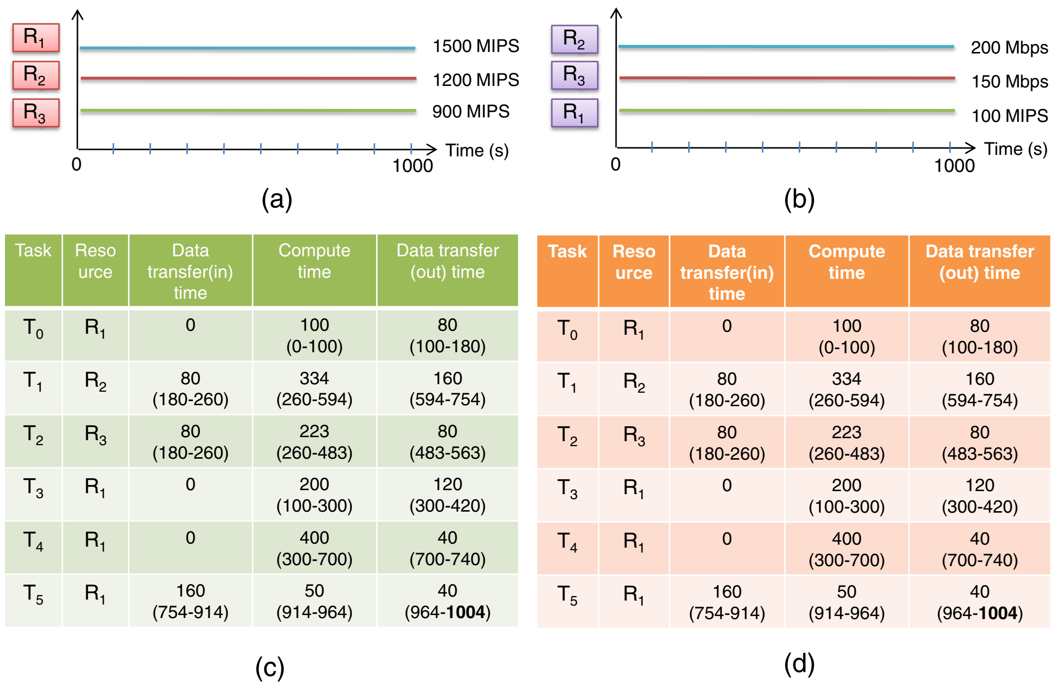
On the other hand, Figure 9 presents the scheduling in dynamic environment, where the processing capability of R_1 drops from 1500 to 750 MIPS for the time duration of 100–800 s. However, the network connectivity to all resources remains the same for the whole duration of workflow execution.

Next, Figure 10 illustrates the scheduling in dynamic environment, where network connectivity to R_2 drops from 200 to 50 Mbps for the time duration of 100 to 800 s. However, the processing capability of all resources remains the same for the whole duration of workflow execution.

5.3. Discussion

This is evident from Figure 8 that if the status of the grid environment does not change, the performance of both adaptive and nonadaptive scheduling techniques is not degraded for workflow

Copyright © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/cpe



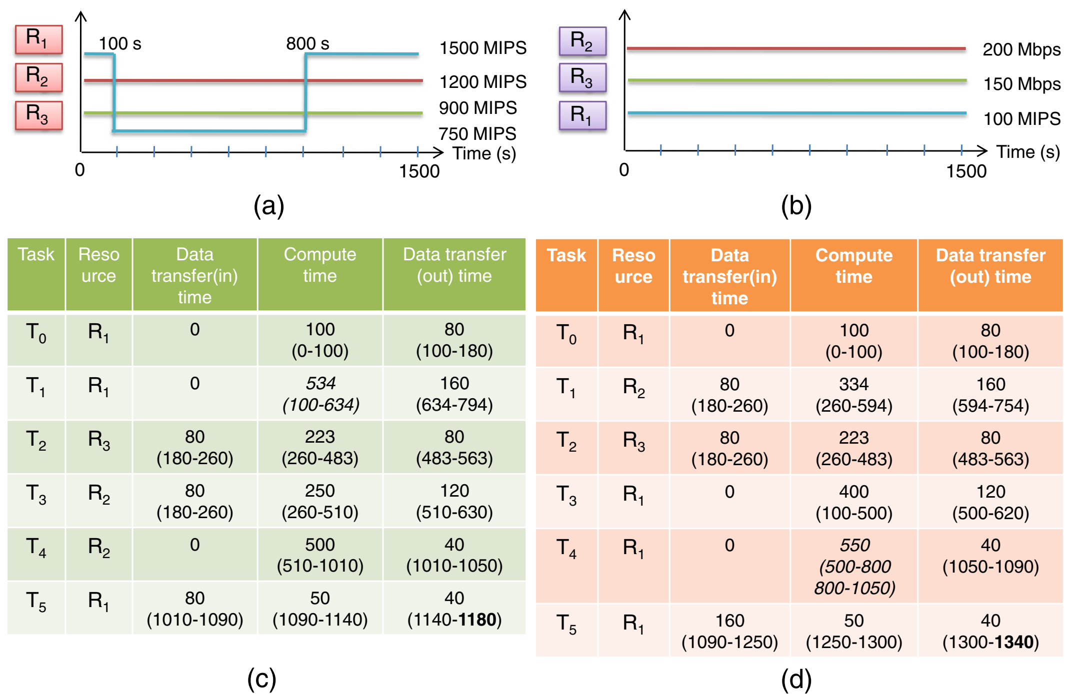
Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_2	80 (180-260)	334 (260-594)	160 (594-754)
T_2	R_3	80 (180-260)	223 (260-483)	80 (483-563)
T_3	R_1	0	200 (100-300)	120 (300-420)
T_4	R_1	0	400 (300-700)	40 (700-740)
T_5	R_1	160 (754-914)	50 (914-964)	40 (964-**1004**)

(c)

Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_2	80 (180-260)	334 (260-594)	160 (594-754)
T_2	R_3	80 (180-260)	223 (260-483)	80 (483-563)
T_3	R_1	0	200 (100-300)	120 (300-420)
T_4	R_1	0	400 (300-700)	40 (700-740)
T_5	R_1	160 (754-914)	50 (914-964)	40 (964-**1004**)

(d)

Figure 8. Workflow execution at static environment, where the status of resources does not change during workflow execution: (a) processing capacity of resources over time; (b) network connectivity of resources over time; (c) adaptive scheduling scheme (dynamic critical path for grids); (d) nonadaptive scheduling scheme (genetic algorithm).



Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_1	0	*534 (100-634)*	160 (634-794)
T_2	R_3	80 (180-260)	223 (260-483)	80 (483-563)
T_3	R_2	80 (180-260)	250 (260-510)	120 (510-630)
T_4	R_2	0	500 (510-1010)	40 (1010-1050)
T_5	R_1	80 (1010-1090)	50 (1090-1140)	40 (1140-**1180**)

(c)

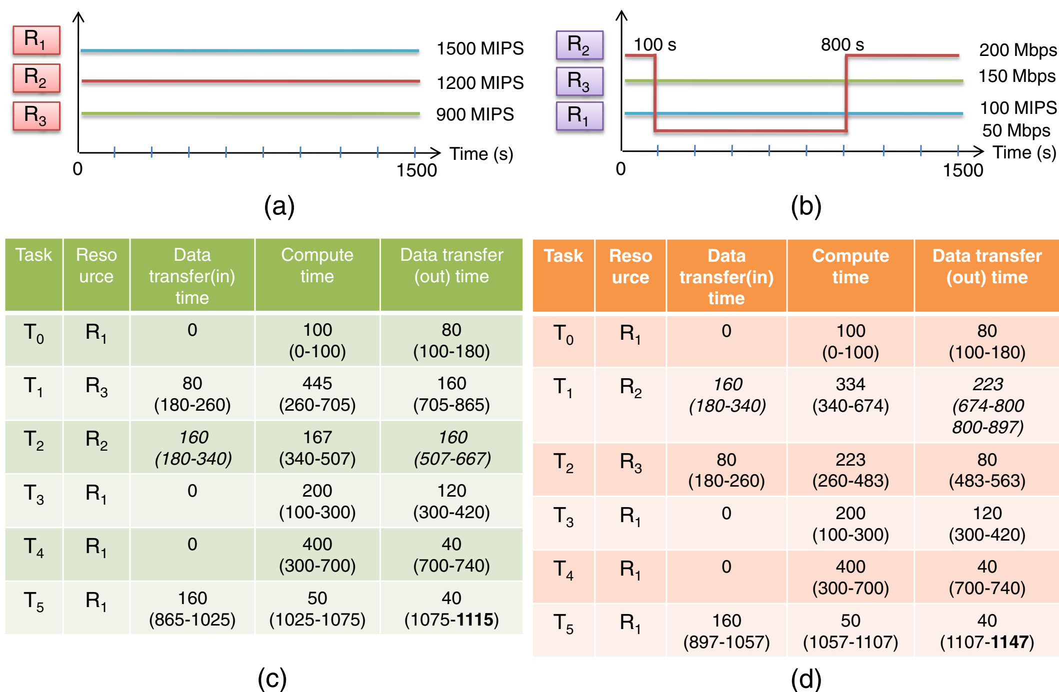
Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_2	80 (180-260)	334 (260-594)	160 (594-754)
T_2	R_3	80 (180-260)	223 (260-483)	80 (483-563)
T_3	R_1	0	400 (100-500)	120 (500-620)
T_4	R_1	0	*550 (500-800 800-1050)*	40 (1050-1090)
T_5	R_1	160 (1090-1250)	50 (1250-1300)	40 (1300-**1340**)

(d)

Figure 9. Workflow execution at dynamic environment, where the processing capability of resources changes during execution: (a) processing capacity of resources over time; (b) network connectivity of resources over time; (c) adaptive scheduling scheme (dynamic critical path for grids); (d) nonadaptive scheduling scheme (genetic algorithm).

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe



Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_3	80 (180-260)	445 (260-705)	160 (705-865)
T_2	R_2	*160 (180-340)*	167 (340-507)	*160 (507-667)*
T_3	R_1	0	200 (100-300)	120 (300-420)
T_4	R_1	0	400 (300-700)	40 (700-740)
T_5	R_1	160 (865-1025)	50 (1025-1075)	40 (1075-**1115**)

(c)

Task	Resource	Data transfer(in) time	Compute time	Data transfer (out) time
T_0	R_1	0	100 (0-100)	80 (100-180)
T_1	R_2	*160 (180-340)*	334 (340-674)	*223 (674-800 800-897)*
T_2	R_3	80 (180-260)	223 (260-483)	80 (483-563)
T_3	R_1	0	200 (100-300)	120 (300-420)
T_4	R_1	0	400 (300-700)	40 (700-740)
T_5	R_1	160 (897-1057)	50 (1057-1107)	40 (1107-**1147**)

(d)

Figure 10. Workflow execution at dynamic environment, where the network connectivity to resources change during execution: (a) processing capacity of resources over time; (b) network connectivity of resources over time; (c) adaptive scheduling scheme (dynamic critical path for grids); (d) nonadaptive scheduling scheme (genetic algorithm).

execution. Thus, both DCP-G and GA (Figure 8(c) and (d), respectively) generate the same schedule (i.e., task to resources mapping) and take the same time (1004 s) for executing the whole workflow.

However, when processing capability is changed during execution (Figure 9(a)), nonadaptive scheduling scheme such as GA is not able to address the changing resource behavior (slowdown of R_1's processing capability for 700 s) into schedule as it sticks to the task-to-resource mapping generated initially. Thus, the execution completion time is increased to 1340 s (Figure 9(d)). On the other hand, adaptive scheduling scheme, such as DCP-G, takes scheduling decision dynamically based on the current resource condition. Therefore, it assigns the critical task T_3 to the fastest resource R_2 at 100th second and able to complete execution of all the tasks by 1180 s (Figure 9(c)).

Further, if network connectivity to the resources is changed during execution (Figure 10(b) for R_2), GA cannot adapt to the changes because of the aforementioned reason. Whereas DCP-G assigns T_1 to R_3 instead of R_2 when the bandwith of R_2 is reduced at 100th second and avoids performance degradation. Thus, the execution completion time for DCP-G (1115 s) is less than that for GA (Figure 10(c) and (d)) in this case as well.

6. HEURISTIC FOR ADAPTIVE WORKFLOW MANAGEMENT IN HYBRID CLOUDS

Cloud computing [22] has emerged as the next generation platform for hosting business and scientific applications. It offers infrastructure, platform, and software as services that are made available as on-demand and subscription-based services in a pay-as-you-go model to users.

Clouds are mainly deployed in two ways: public cloud and private cloud. Public or external cloud describes cloud computing in the traditional main stream sense, where services are dynamically provisioned on demand and self-service basis over the Internet and charged by the third party providers on the basis of utility. Whereas private or internal cloud refers to emulating cloud computing services on private networks through virtualization. As shown in Figure 11, a hybrid cloud is a combination of a public and a private cloud aiming at serving the organizational demand of baseline computing

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

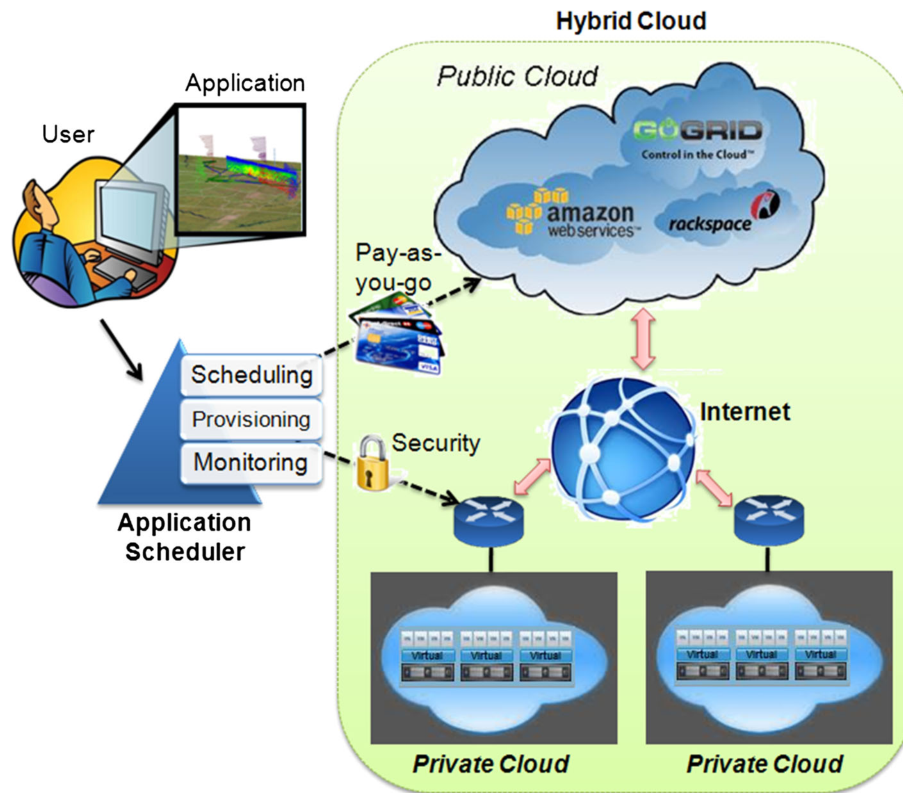


Figure 11. Hybrid cloud computing environment.

through private cloud and peak computing using public clouds. For example (Figure 11), an organization may use the private cloud computing services deployed in local clusters for mission-critical and security-concerned applications, whereas utilize a public cloud service, such as Amazon Simple Storage Service (Amazon S3), for archiving data as backup.

Cloud computing environments are not only dynamic but also heterogeneous with multiple types of services (e.g., infrastructure, platform, and software) offered by various service providers (e.g., Amazon). Scheduling data analytics workflow applications in such environment (Figures 12 and 13) requires to address a number of issues, including minimizing cost and time of execution, satisfying user's QoS constraints, and considering the temporal behavior of the environment. The majority of scheduling techniques [23, 24] proposed to solve these issues are based on metaheuristics, which produce a good schedule given the current state of cloud services and reserve the services in advance accordingly, thus lack the ability to adapt to the changes in the services during execution.

On the other hand, the heuristic-based scheduling techniques as discussed in this paper are dynamic in nature and map the workflow tasks to services on-the-fly but lack the ability of generating schedule considering workflow-level optimization and user QoS constraints, such as deadline and budget. Thus, it is necessary to develop a hybrid heuristic that can effectively integrate most of the benefits of both heuristic and metaheuristic-based approaches to optimize execution cost and time as well as meet the user's requirements through an adaptive fashion in order to efficiently manage the dynamism and heterogeneity of the hybrid cloud environment.

Therefore, we propose Adaptive Hybrid Heuristic scheduling algorithm, which is designed to first generate a task-to-service mapping with minimum execution cost using GA within user's budget and deadline as well as satisfying the service and data placement constraints specified by the user. This initial schedule is then utilized to distribute the workflow-level budget and deadline to task levels. Finally, the DCP-G heuristic presented in this paper is employed to dynamically schedule the ready tasks level-by-level on the basis of the initial schedule, t_{budget} and $t_{deadline}$, as well as changed status of services. Figure 14 presents the flowchart that illustrates various segments and flow of logics in the algorithm.

Copyright © 2013 John Wiley & Sons, Ltd.

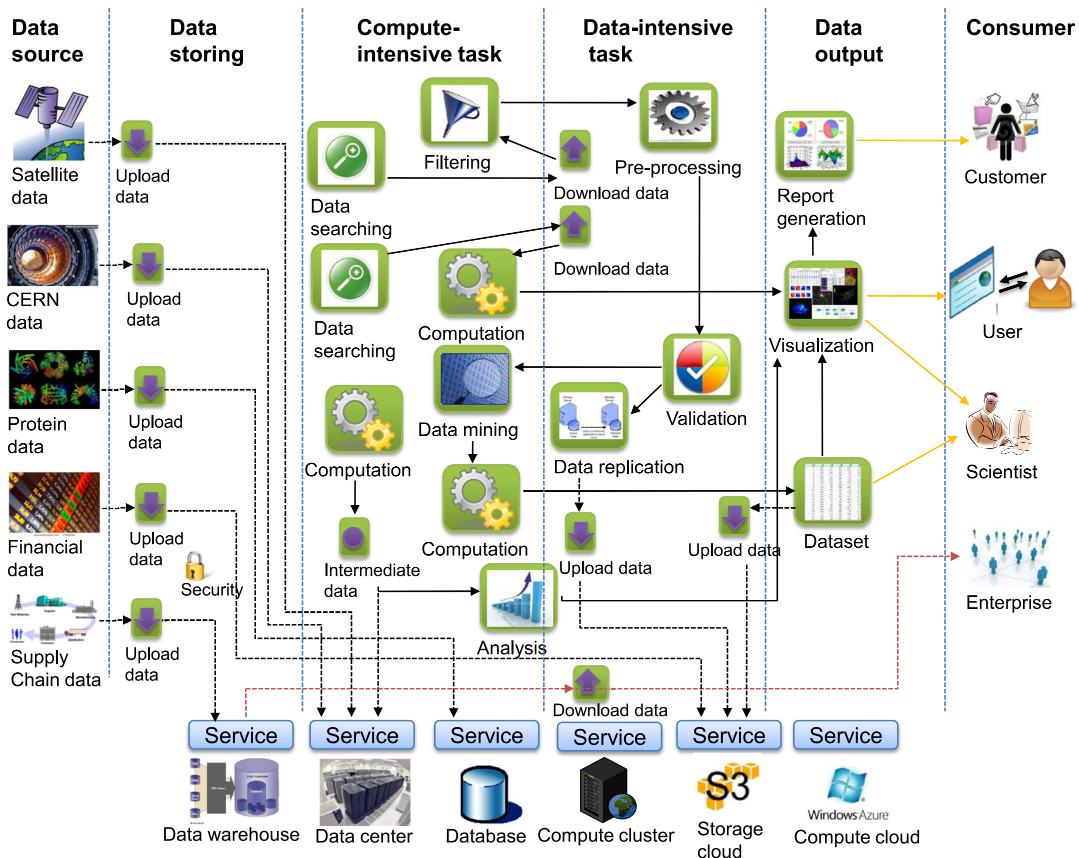


Figure 12. An example of data analytics workflow execution in cloud.

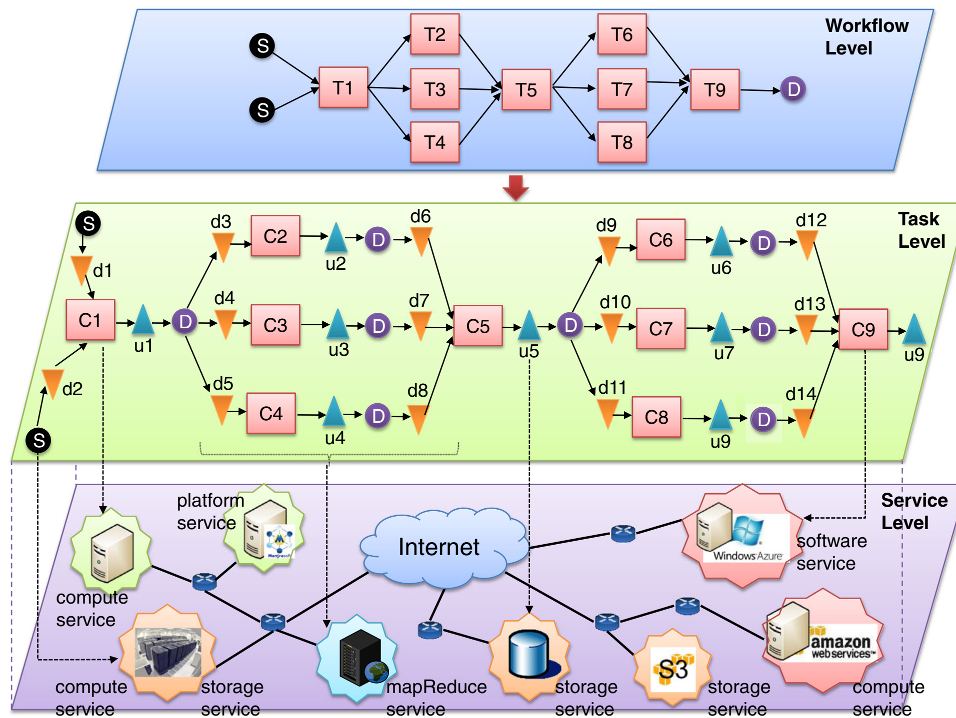


Figure 13. Layered architecture for workflow execution in hybrid cloud.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

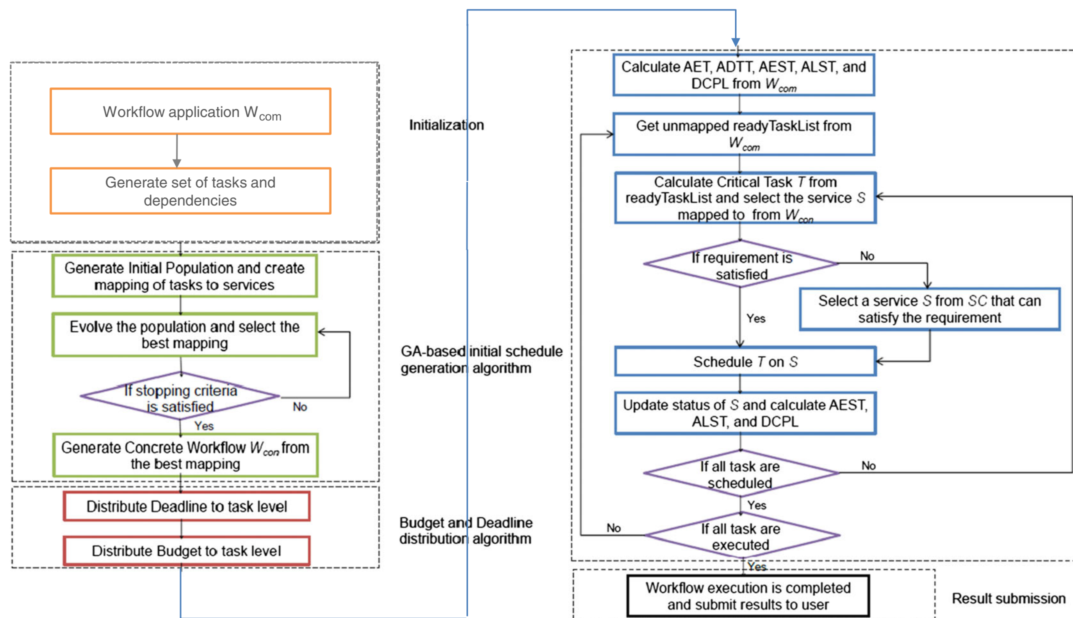


Figure 14. Flowchart for Adaptive Hybrid Heuristic scheduling algorithm.

7. CONCLUSIONS

In this paper, we have defined the workflow scheduling problem in grid computing environment and discussed the existing well-known workflow scheduling techniques. Nevertheless, these techniques are static in nature and do not take into account dynamic resource behavior. Therefore, we have proposed a dynamic and adaptive scheduling approach, named DCP-G for scheduling grid workflows. DCP-G determines an efficient mapping of workflow tasks to grid resources by calculating the CP in the workflow task graph at every step and assigns priority to a task in the CP that is estimated to complete earlier. We have compared the performance of DCP-G with other existing heuristic-based and metaheuristic-based scheduling strategies for different types and sizes of workflows. The results show that DCP-G can generate better schedule for most of the workflow types, irrespective of their sizes particularly when the resource availability changes frequently.

In summary, this paper identifies that dynamic scheduling approaches can adapt to temporal behavior of heterogeneous grid resources and are able to avoid performance degradation by generating efficient schedules, which is demonstrated by the case study in Section 5.

In the future, we endeavor to evaluate the performance of proposed hybrid heuristic through extensive simulation and real-world prototype implementation. We plan to devise specific policies for different workload and particular user requirements in order to better utilize the features of hybrid cloud computing environment through this evaluation. Moreover, we intend to incorporate other QoS parameters, such as reliability of a service for task-to-service mapping, by integrating a multi-objective optimization technique into our proposed algorithm.

REFERENCES

1. Laity AC, Anagnostou N, Berriman GB, Good JC, Jacob JC, Katz DS, Prince T. Montage: an astronomical image mosaic service for the nvo. In *Proceedings of the 14th Annual Conference on Astronomical Data Analysis Software and Systems (ADASS'XIV)*, USA, October, 2004.
2. Blaha P, Schwarz K, Madsen G, Kvasnicka D, Luitz J. Wien2k—an augmented plane wave plus local orbitals program for calculating crystal properties. *Technical Report*, Vienna University of Technology, Austria, 2001.
3. Yu J, Buyya R. Taxonomy of workflow management systems for grid computing. *Journal of Grid Computing* 2005; **3**(3-4):171–200.
4. Yu J, Buyya R, Ramamohanarao K. *Workflow Scheduling Algorithms for Grid Computing*, Metaheuristics for Scheduling in Distributed Computing Environments, Xhafa F and Abraham A (eds.) Springer: Germany, 2008.

Copyright © 2013 John Wiley & Sons, Ltd.

5. Kim S, Browne J. A general approach to mapping of parallel computation upon multiprocessor architectures. In *Proceedings of the 27th IEEE International Conference on Parallel Processing (ICPP'98)*, USA, August, 1988.
6. Kwok Y, Ahmad I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 1996; **5**(7):506–521.
7. Garey MR, Johnson DS. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.: USA, 1990.
8. Wieczorek M, Prodan R, Fahringer T. Scheduling of scientific workflows in the ASKALON grid enviornment. *ACM SIGMOD Record* 2005; **34**(3):56–62.
9. The directed acyclic graph manager, condor project. (Available from: http://www.cs.wisc.edu/condor/dagman/) [Accessed on 18 August 2012].
10. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund R. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*, Puerto Rico, April, 1999.
11. Blythe J, Jain S, Deelman E, Gil A, Vahi K. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05)*, UK, May, 2005.
12. Mandal A, *et al.* Scheduling strategies for mapping application workflows onto the grid. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, USA, July, 2005.
13. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(3):260–274.
14. Fahringer T, Jugravu A, Pllana S, Prodan R, Seragiotto C, Truong HL. ASKALON: a tool set for cluster and grid computing. *Concurrency and Computation: Practice and Experience* 2005; **17**(2-4):143–169.
15. Goldberg DE. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley: USA, 1989.
16. Yu J, Buyya R. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, France, June, 2006.
17. Foster I, Kesselman C (eds). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers: USA, 1999.
18. Buyya R, Murshed M. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13-15): 1175–1220.
19. Sulzer S, Kinzelbach W, Rutschmann P. Flood Discharge Prediction using 2D Inverse Modelling. *ASCE Journal of Hydraulic Engineering* 2002; **128**(1):46–54.
20. Bell WH, Cameron DG, Capozza L, Millar AP, Stockinger K, Zini F. Simulation of dynamic grid replication strategies in optorsim. In *Proceedings of the 3rd IEEE/ACM International Workshop on Grid Computing (GRID'02)*, November, 2002.
21. Venugopal S, Buyya R. A set coverage-based mapping heuristic for scheduling distributed data-intensive applications on global grids. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing (GRID'06)*, Spain, September, 2006.
22. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 2009; **25**(6):599–616.
23. Pandey S, Wu L, Guru S, Buyya R. A particle swarm optimization (pso)-based heuristic for scheduling workflow applications in cloud computing environments. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10)*, Australia, April, 2010.
24. Wu Z, Liu X, Ni Z, Yuan D, Yang Y. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *Journal of Supercomputing* 2013; **63**(1):256–293.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe