Journal of Scientific & Industrial Research
Vol. 79, June 2020, pp. 517-525

Multi-Workflow Concurrent Scheduling in a Heterogeneous Computing Cluster

Rintu Nath[1]*, A Nagaraju[2] and M N Raghavendra Sreevathsa[3]

[1]Department of Computer Science and Engineering , Central University of Rajasthan, Kishan Garh, Rajasthan, India

[2]Department of Computer Science, Central University of Rajasthan, Kishan Garh, Rajasthan, India

[3]National Centre for Medium Range Weather Forecasting (NCMRWF), NOIDA, Uttar Pradesh, India

Received 11 September 2019; revised 18 February 2020; accepted 19 April 2020

Scheduling scientific workflow in a distributed computing resource is a challenging job. It involves heterogeneous resource allocation to concurrent tasks in order to achieve the desired scheduling goal. In this paper, we are presenting a multi-workflow Earliest Cycle Time algorithm, mECT. The objective of the algorithm is to reduce schedule length while ensuring no deadlock situation occurs due to dependency violation. The algorithm is suitable for multi workflow applications that deal with large data sets and run in a cyclic order. We have carried out extensive simulations to compare the proposed algorithm with well known existing algorithms. We have also tested our algorithm for a low resolution (N48) weather Unified Model (UM-10.2) in a simulated environment developed by the Met Office, United Kingdom. Results show that mECT performs better in terms of shorter makespan and lesser deadlocks.

Keywords: Distributed computing, Heterogeneous cluster, Multi-workflow, Scheduling

Introduction

Execution of any scientific workflow in a cluster of computers involves mapping of tasks in a distributed heterogeneous computing environment. Scheduling is required in order to optimise some of the performance criteria, such as system utilisation, throughput, efficiency, and reliability. Multiple task assignment in a distributed computing system is an NP-complete problem, Kumar *et al*.[1], Shahul *et al*.[2] Hence, a single scheduling solution is not possible. Concurrent scheduling of multiple applications is a well-researched topic, and the references[3–9] present several heuristics. However, scientific applications may be data-intensive or computation-intensive. Some applications need to optimise throughput, whereas another application may require fast response time. As a result, different algorithms are required to schedule tasks in a distributed system.

Weather forecasting models are multi-workflow applications that deal with large data sets and need extensive computation time. Forecast models typically depend on their own most recent previous forecast. Thus, some models are dependent on external data like real-time observational data or output data from another model. One or more models need to wait for these data before proceeding to further downstream tasks. One way to represent Numerical Weather Prediction (NWP) is to group dependent tasks and introduce a common cycle point based on start time. A forecast cycle point spawns its successor when external driving data is available. A real-time operation consists of a series of forecast cycle points.

The present work involved scheduling weather forecasting workflows in a distributed heterogeneous computing platform. The workflow involves different Weather Models (WM) that share data between them. WMs process input and output temporal files sequentially. During execution, some of these files are accessed several times and shared between different tasks. Oliver *et al*.[10] reported that a batch of workflows could be merged into a meta-scheduler. The authors have developed 'cylc' - a workflow engine for running suites of inter-dependent jobs. In weather forecasting, multiple weather models run concurrently. Hence we need to modify scheduling solutions for concurrent execution of multiple workflows in a cyclic order. Some references are available on concurrent workflow scheduling methods. Hwang *et al*.[11] proposed grouping of individual applications into an application pool and apply list scheduling heuristics. Different resource allocation policies for high throughput computing are

*Author for Correspondence
E-mail: rintu2013_csephd@curaj.ac.in

presented by Arabnejad *et al*.[12] In this paper, we have proposed a concurrent scheduling solution of WMs in a heterogeneous platform.

Multi-workflow forecast cycles

Weather forecasting has multiple iterative processes. The workflow engine needs to orchestrate distributed suits of interdependent tasks. The dependency relationship for a single forecast cycle point may be represented by a Directed Acyclic Graph, as shown in Fig. 1(a). Each node in the DAG represents a task. Within a single forecast cycle point, the dependency between tasks needs to be resolved. Nodes *a*, *b*, and *c* in the DAG represents three Weather Models (WMs). Nodes *e* and *f* represent two post-processing or model output tasks. WMs are dependent on their own most recent previous state, called warm cycling, and may have inter-cycle dependence between different tasks. In a weather forecast suite, WMs do observation processing and data-assimilation tasks for the next cycle point. In Fig. 1(a), *u* represents external data. The scheduler must be able to branch model output data to multiple downstream tasks without dependency violation.

We may consider a scenario 1, where, in real-time, forecast cycle points run consecutively in a serial manner. In Fig. 1(b) two such cycles having a time gap between them is shown. Task *a* needs to wait for external data, whereas task *b* and *c* need model output from *a*. Post-processing is possible only when WM *b* and *c* are completed. In this scenario, each task in a cycle is dependent on the cycle preceding it. As a result, a new cycle begins only after completion of the previous cycle.

Scenario 1 is rather simplistic and will introduce cycle wait time. A new cycle will start only when the last task of the previous cycle is completed. If we assume that the external driving data are available in advance, we may consider scenario 2, where task *a* is started without waiting for the completion of the cycle preceding it. However, in that case, there is a possibility of a dependency violation. Scenario 2 with a dependency violation for task *c* is depicted in Fig. 1(c). Task *a* is dependent on external driving data as well as its previous instance, task *b* and *c* are dependent on their previous instances. As shown in Fig.1(c), task *c* in cycle 2 started before completion of task *c* in the upstream cycle. Hence, it may be concluded that starting a whole new cycle without completion of the previous cycle is not possible unless inter-cycle dependencies are handled.

Scenario 3, a possible multi-cycle workflow, where each task starts the moment its previous instance is completed is depicted in Fig. 1(d). Warm cycled tasks *a*, *b,* and *c* are dependent on their previous instances, whereas tasks *d* and *e* are the model outcome. In scenario 3, each task starts as soon as its previous instance is complete. However, in this scenario, it is assumed that there is no delay in the external data source. The scheduler, handling such workflow, must adapt dynamic external conditions like delay in receiving external data, else; dependency violation will bring the system down.

Platform

Our heterogeneous computing platform has clusters of compute nodes connected by the high-speed backbone, *InfiniBand*, shown in Fig. 2. Processors and bandwidth of switches of different clusters may be different. However, within the same cluster, all processors are identical, i.e., we have $C = \sum_{i=1}^{N} c_i$ clusters. For each c_i, we have P_j *{ j=1 …m}*, identical processors. Processors execute tasks in a sequential

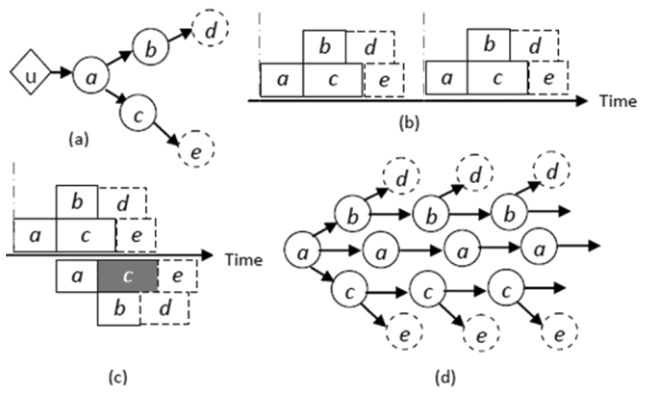

Fig. 1 — Job scheduling of consecutive forecast cycles

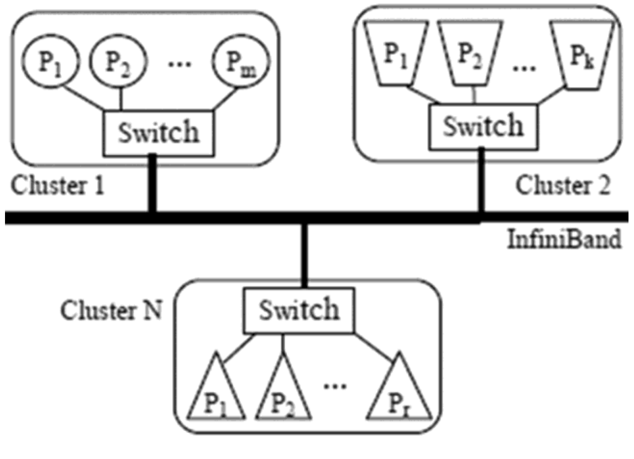

Fig. 2 — Heterogeneous clusters connected by *InfiniBand*

manner as it is done in a space sharing framework. Processors in each cluster are connected to the backbone by a switch. Sheikh *et al*.[13] presented a computational model to measure communication overhead for such a cluster.

For a cluster having P_j processors, parallel execution time T_p of a task t_i can be calculated using Amdahl's law:

$T_p(t_i, P_j) = (\alpha + \frac{1-\alpha}{Pj}). T_p(t_i, 1)$, where $T_p(t_i, 1)$ is the execution time of t_i in single processor and, α is the fraction of t_i that cannot be parallelised. We assume that parallel tasks can be deployed in any number of clusters without increasing communication overhead.

Topcuoglu *et al*.[14] proposed Heterogeneous Earliest Finish Time (HEFT) algorithm for a fixed number of heterogeneous processors. The HEFT algorithm selects a task based on the highest rank, and insertion based assignment is done for processor selection that ensures the earliest finish time. A multi-workflow application for heterogeneous distributed platform is presented by Acevedo *et al*.[15]

Problem formulation

Formally, in a multiple workflow (MF) situation, each workflow in a set of MF_{set} is represented as a graph $MF_i = (V_i, E_i)$, $i = 1, 2, \ldots N_w$, where N_w, is the number of workflows in MF_{set}. $V_i=\{1,2,\ldots,N_i\}$ is a set of vertices or nodes representing N_i tasks in the i-th workflow. and $E_i = \{ e_{i,j} \mid (i, j) \in \{1, \ldots , V \}\times\{1, \ldots , V \}\}$ represents is a set of weighted edges between vertices. Each directed edge $e_{i,j}$ represents communication between task v_i to task v_j, whereas the weight of each edge represents the volume of data transmitted. A different number of processors may execute tasks v_i and v_j. We have assumed all tasks are non-preemptive.

A heterogeneous cluster has N resources having N_t types. Each resource can have any one of the types, ranging from 1 to N_t. Here resource signifies single cluster of multi-core processors. We have assumed zero communication costs between processors in a cluster.

For any application model represented by DAG, Let c_{ik} is computation cost of i^{th} task in k^{th} processor, e_{ij} is communication cost between i^{th} task to j^{th} task, d_{ij} is data transfer between i^{th} to j^{th} task. σ is a vector of size *n* that maps tasks to processor. Φ is a set of all the mappings. Then the objective function of a task scheduling problem is expressed as:

$$\sigma = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{i,j}(d_{\sigma[i]\sigma[j]}) + \sum_{i=1}^{n} c_{i\sigma[i]} \;\; \forall\sigma \in \Phi$$

where $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} e_{i,j}(d_{\sigma[i]\sigma[j]})$ is the communication cost between interacting tasks and $\sum_{i=1}^{n} c_{i\sigma[i]}$ is the computation cost for all mappings between tasks to processors. Task scheduling algorithms try to minimize the objective function.

We are proposing a multi-workflow scheduling algorithm mECT. The objective of the algorithm is to reduce makespan by ensuring no dependency violation between inter-cycle tasks. The algorithm has to ensure the concurrent processing of multiple tasks from different workflows. All the workflows run concurrently and in a cyclic manner, i.e., after completion of the first cycle of the i^{th} workflow, MF_i, the second cycle of MF_i starts, and the process continues. In order to accommodate continuous cycles, we have introduced *Cycle Time* T_c in the list scheduling algorithm. For each workflow, *Completion Time* (*CT*) has to be lesser than T_c .

For a given MF_i, i = 1 ... N_w we need to maximise $\{T_{ci} - \sum_{j=1}^{n} CT_j \}$, where T_{ci} is the cycle time of i^{th} workflow and $\sum_{j=1}^{n} CT_j$ is the completion time of all the *n* tasks.

Multi-workflow Earliest Cycle Time Algorithm

HEFT is one of the preferred algorithms to improve schedule length in any workflow. However, HEFT is applies to a single workflow. Our problem is to schedule all the workflows in a cyclic order. Hence we are extending the scope of HEFT to schedule multiple workflows having inter-cycle dependencies. Similar to HEFT, mECT also prioritise tasks and builds a queue of tasks. As mentioned in the previous section, we have introduced a *Cycle Time* T_c in the scheduling solution. The introduction of T_c will create a priority list of tasks as well as estimated execution time for a particular workflow.

The algorithm has two phases. Phase one computes *Cycle Time* (T_c) of each workflow; phase two is the mapping phase, where a priority queue based on precedence constraints and *Cycle Time* is created. Tasks from the queue are assigned to processors that provide earliest finish time.

```
procedure mECT
{$\sigma_0$ = initialization
while(1)
1.        Compute Cycle Time $T_c$
2.        Task setup
    Compute communication and computation cost
    σ = candidate solution
3.  Priority queue and mapping }
```

In phase 1, T_c is computed from the task graph based on maximum parallel execution time. Pseudocode to calculate cycle time is given below:

```
1. While(1)
2. For all i=1 to N
3.   For all j = 1 to $m_T$( $level_i$ )
4.   For all k = 1 to $N_t$
5.   $T_c \leftarrow T_c + max(c_{jk})$
6. End while
```

N is the total number of tasks in a DAG, $m_T(level_i)$ is the number of tasks in level *i*, N_t is the number of resource type, $max(c_{jk})$ is the maximum computation cost of jth task in kth resource.

Two workflows, each having four tasks is shown in Fig. 3. There are three resource; one is of resource type 1, R1, and two are of resource type 2, R2. For each task, the execution time for both types of resources is given. For both workflow1 and workflow 2, $N = 4$, $N_t = 2$. T_c for workflow 1 is 4 + 6 + 8 = 18 and for workflow 2, T_c is 5 + 9 + 10 = 24.

Phase 2 of the algorithm is based on satisfiability checking from a candidate solution based on T_c. In each iteration, configuration checking is done for a better solution. A candidate solution is generated based computation cost, and processor availability (PA) and *n* tasks are assigned among *p* processors.

Algorithm: *mECT*

Input: $priority(n_i)$, T_{ci}

Output: $n \times p$ Look-up matrix with task assignment $PA(n_i,p_j)$

```
1. Initialization of vector σ
2. Initialize computation cost $c_{i\sigma[i]}$, cycle time $T_c$
3. Compute priority of tasks
   $priority(n_i) = average(c_i) + max\sum[\sum d_{ij} + priority(n_{i-1})]$
4. Setup processor availability
   $PA(n_i,p_j) = MAX\{ avail[j], CT(n_{i-1},p_j)\}$
5. While(1)
   i.   Re-compute completion time of task $n_{i-1}$ on processor $p_j$ : $CT(n_{i-1},p_j)$
   ii.  Check candidate solution $\sigma \in \Phi$
   iii. If ( $T_{ci} - \sum_{j=1}^{n} CT_j$ ) ≥ 0 then
   iv.  $PA(n_i,p_j) = LPT\{ EST[j], CT(n_{i-1},p_j)\}$
   v.   else
   vi.  compute $priority(n_i)$
6. compute $PA(n_i,p_j)$
7. return
```

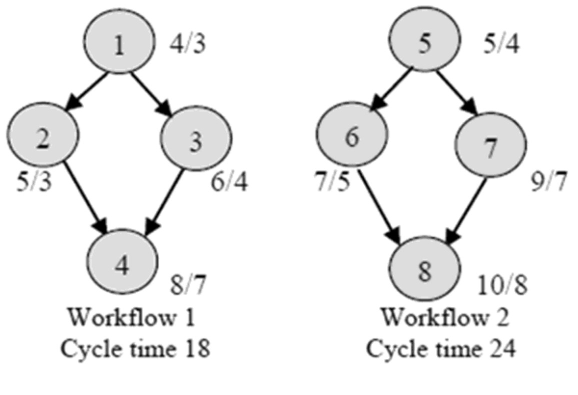

Fig. 3 — Two workflows. Notation: a/b , a:computation cost in resource 1, b: computation cost in resource 2

Step 3, $average(c_i)$ calculates the average computation cost of task n_i in p processors and $\sum d_{ij}$ is communication cost between task i and j. Priority assignment is done by traversing DAG upwards, starting from a leaf node. The start time of processor p_j is calculated based on the completion time of p_j of the last assigned task.

Steo 4 determines processor availability based on the computational capacity that minimizes ready task n_i. $PA(n_i,p_j)$ is available time of processor p_j for task n_i; *avail* [j] the time when processor p_j is available for execution of task n_i. $CT(n_{i-1},p_j)$ is the Completion Time of task n_{i-1} on processor p_j. .

Step 5 (i to vi) schedules the ready tasks in decreasing order and assigns processors based on the lowest processing time *LPT*. At line 6, the processors are mapped to tasks. When the lowest processing times are calculated, tasks are placed in the priority queue in ascending order. Makespan should be less than the cycle time.

In order to explain the introduction of cycle time to schedule multiple workflows, we consider two workflows given in Fig. 3. We try to schedule both the workflows. The computation cost of each task is set as the mean value. The rank of each task is then computed by traversing the graph upwards in a recursive way, starting from the exit task.

The priority of each task is calculated as:

$priority(n_i) = average(c_i) + max\sum[\sum d_{ij} + priority(n_{i-1})]$, where $average(c_i)$ is the mean computation cost of each task and $max\sum[\sum d_{ij} + priority(n_{i-1})]$ is the recursive calculation of priority for each task while traversing towards the node starting from exit node. The path that gives maximum cost is considered. For example, to calculate the priority of task 1 in workflow 1 (Fig. 3), traversing starts from exit node 4 and continues until node 1. Before traversing, the mean computation costs of nodes are calculated. $\overline{w_i}$ for node 4, $\overline{w_4} = \frac{8+7}{2} = 7.5$. Similarly, the mean computation costs of other nodes of workflow 1 are calculated. Priority of task 1 is for the traversal $4 \rightarrow 2 \rightarrow 1$ is 7.5 + 4 + 3.5 = 15, while for the traversal $4 \rightarrow 3 \rightarrow 1$ is 7.5 + 5 + 3.5 = 16. Hence maximum, i.e. 16 will be the rank of task 1 of workflow 1. Ranks of all the tasks are given in Table 1.

Resource allocation is done based on priority value and precedence constraints. The task having the highest priority is scheduled first provided the earliest start time of the task EST_i is more than or equal to earliest finish time EFT_j of the task preceding it. In the example shown in Fig. 3, task 5 is assigned to R2 as it has the highest priority value (21.5), and processing cost in R2 is lesser compared to R1. The next priority value is for task 7 (rank 17); however, $EST7 < EFT_5$. As the condition is not fulfilled, the next highest priority 16 (task 1) is selected and scheduled. Scheduling is shown in Fig. 4(a). Makespan is 17. Clearly, makespan for workflow1 is more than the cycle time. Hence the first candidate solution is modified based on the cycle time, and the final task queue is prepared, shown in Fig. 4(b). Makespan of both the workflow is less than the respective cycle times. For the given example (Fig. 3), mECT fulfills both the cycle time conditions.

Experimental methodology

The performance of the proposed scheduling algorithm mECT is evaluated using an extensive simulation setup. We have explored wide ranges of scenarios like the number of workflows, the number of tasks in each workflow, heterogeneity of resources. Statistically significant numbers of experiments were carried out in a repeatable manner. We have studied our algorithm on a Unified Model virtual machine (VM) developed by Met Office, United Kingdom. VM comes with access to the Met Office Science Repository Service (MOSRS) and provides a simple environment to test run scheduler. VM also has a built-in test suite called rose-stem. The capability of VM depends upon the allocated resources before launch. Although VM is not capable of running an HPC scale simulation, most modern laptops can run low-resolution weather forecasting global model.

Through this simulation process, we wanted to compare different task scheduling algorithms under different numbers of workflows and resources. This approach helped us in understanding the performance of different algorithms under varied resources. For simulation, we have generated synthetic test data, like the number of workflows, the number of tasks per workflow, and DAG parameters. We have used the Pegasus workflow generator reported by Deelman *et al*.[16] to generate workflows. Parameters used to define DAG are width, density, regularity, and jump. Width signifies the maximum number of tasks across all levels, i.e., the maximum number of tasks that can be processed concurrently. Density signifies the number of edges between two levels. Uniformity of a

Table 1 — Task scheduling queue

Work flow	Task number	Initial *priority*	Modified *priority*
1	1	16	17
1	2	11.5	16
1	3	12.5	15
1	4	7.5	11
2	5	21.5	21.5
2	6	15	11
2	7	17	16
2	8	9	8

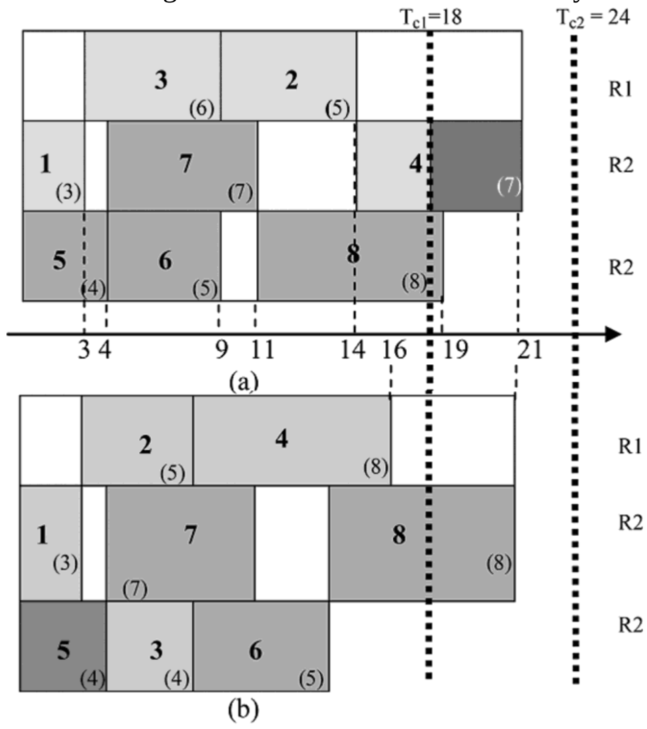

Fig. 4 — a) Scheduling of workflow 1 and 2 with initial priority, This results in violation of cycle time of workflow 1, b) Scheduling with modified priority

DAG is measured with regularity value, a low value of regularity signifies a wide variation of tasks at different levels. Jump signifies jumping from level L to (L+ jump) level. Width, density, and regularity can have values between 0 – 1. Workflow and DAG parameters are listed in Table 2.

We have considered resources having 1, 2, 4, or 8 clusters; each cluster can have only one of the two types of resources (resources R1 and R2). The number of processors in each type of resources varies between 16 and 128. Processors in a cluster are connected to switch by 10 Gigabit (bandwidth 10 GBPS, latency 50 μsec) Ethernet. Switches are having the same bandwidth and latency characteristics as the network links. Backbone network connecting all the clusters is having bandwidth 100 GBPS and latency 50 μsec.

In our experiments with a given set of clusters, we have kept network characteristics fixed and varied only workflow and DAG parameters. Heterogeneity factor (β) is the same as the number of clusters, and minimum processor speed is calculated as the speed of the processor multiplied by β.

Results and Discussion

For performance analysis of mECT, we have compared it with two algorithms, 1) Parallel Heterogeneous Earliest Finish Time (pHEFT), an implementation of HEFT for dynamic load conditions, proposed by Barbosa *et al.*[17], and 2) Heterogeneous Critical Path and Area Based Scheduling (HCPA), an implementation of CPA in a heterogeneous cluster proposed by N'Takpe *et al.*[18] Each set of experiments was carried out with a different number of tasks (10 or 20), we use this value with the algorithm; for example, HCPA10, mECT20.

pHEFT schedules multiple DAGs having different arrival times. It has a scheduling strategy to define scheduling instants based on new job arrival and change in hardware availability. HCPA uses a novel virtual cluster methodology in order to handle resource heterogeneity. It has different task placement steps such that heuristics designed for homogeneous resources can be adapted for a heterogeneous environment.

Table 2 — Workflow and DAG parameters

Number of workflows	2, 5, 8, 11, 15
Cost of computation (GFLOP)	10000 - 50000
Tasks per workflow	4, 10, 20
DAG width	0.1, 0.2, 0.8
DAG density	0.2, 0.4
DAG regularity	0.2, 0.8
DAG jump	1, 2, 4, 8

The performance of the algorithm was evaluated based on makespan, machine idle time, system utilisation, and cycle time violation. The average values of test run parameters are listed in Table 3, and graphs are shown in Fig. 5.

Makespan

Makespan is considered as one of the significant comparison matrices for any task scheduling algorithm. It is also called schedule length, i.e., finish time of the last task in a DAG. One of the objectives of mECT algorithm is to minimise makespan. However, there are different numbers of tasks in different workflows. The makespan of a DAG having more number of tasks will always be larger than the one having a lesser number of tasks. Hence, we have evaluated the schedule length ratio (SLR) as proposed by Topcuoglu *et al.*[14] It is defined as follows:

Schedule length ratio (SLR)

SLR is defined as the ratio of makespan to minimum serial computation cost, i.e.

$$\text{SLR} = \frac{makespan}{\sum_{n \in N} \min\{c_{ij}\}}, \text{SLR} \leq 1 \qquad \ldots (1)$$

$\sum_{n \in N} \min\{c_{ij}\}$ represents the summation of minimum computation cost considering all the tasks in a DAG are computed in series. SLR is a more suitable comparison matrix for multiple workflows scheduling algorithms.

For workflow 1 given in Fig. 3 and the schedule given in Fig. 4, $\sum_{n \in N} \min\{c_{ij}\} = 3 + 3 + 4 + 7 = 17$ and makespan = 21, therefore from Eq. (1), SLR = $\frac{16}{17} = 0.94$. For a task scheduling algorithm, lower the SLR, better is the algorithm. The variation of SLR with the number of workflows is shown in Fig. 5(d). In terms of SLR, mECT performed better with 20 tasks compared to 10 tasks, having SLR 0.6 and 0.7,

Table 3 — Average values of test run parameters

	mECT10	*mECT20*	*pHEFT10*	*pHEFT20*	*HCPA10*	*HCPA20*
Avg. idle time fraction	0.20	0.24	0.20	0.27	0.24	0.29
System utilisation	0.54	0.63	0.54	0.64	0.43	0.44
Cycle time violation	2.42	4.35	3.50	5.07	4.28	5.92
SLR	0.70	0.60	0.77	0.71	0.83	0.81

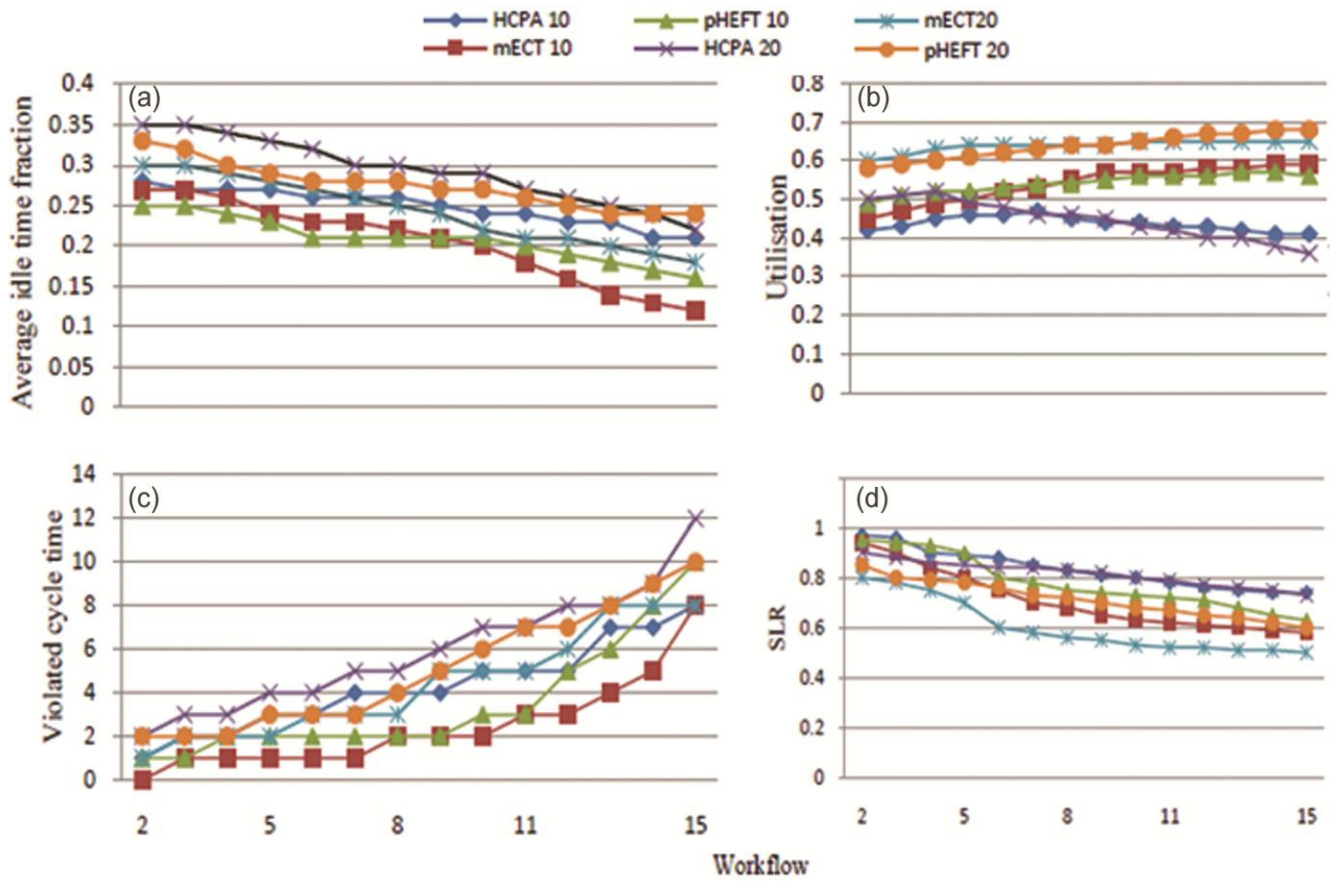

Fig. 5 — Performance evaluation (a) Average idle time fraction (b) Utilisation (c) Violated cycle time (d) SLR

respectively. SLR of mECT is 10% and 18% lower compared to pHEFT and HCPA in 10 task category, whereas, in case of 20 task category, SLR improvement is 16% and 35%.

To understand resource utilisation, apart from the comparison matrices based on makespan, we have studied behaviour of our algorithm in terms of average idle time fraction ($\boldsymbol{\delta}_{idle}$) and system utilisation (η).

Average idle time fraction ($\boldsymbol{\delta}_{idle}$)

For any workflow, total idle time is measured from the width of idle slots. For the example workflow 1 given in Fig. 3, and schedule sequence given in Fig. 4, total idle time is calculated as 6. However, total idle time will keep increasing as the number of tasks or workflow increases. Hence total idle time cannot be considered to be a performance measurement criterion. Therefore we calculate average idle time fraction, which is defined as the ratio of total idle time to the width of busy interval.

$$\boldsymbol{\delta} idle = \sum_{i=1}^{N} \frac{idle\ \ time_{\ i}}{width\ (interval\ \ busy\ )},\ 0 \le \boldsymbol{\delta}_{idle} < 1 \quad \dots (2)$$

System utilisation (η)

System utilisation is the ratio of busy time intervals to resource reserve time. Resource reserve time is different from the makespan. From Fig. 4, makespan for workflow 1 is 16; whereas, resource reserve time is 21. In the case of cyclic scheduling of multiple workflows, resource reserve time tends to be equal to makespan. Resource reserve time is equal to makespan when no cycle time violation takes place.

$$\eta = \sum_{i=1}^{N} \frac{width\ (interval\ \ busy\ )}{Resource\ \ reserve\ \ time},\ \eta \le 1 \quad \dots (3)$$

For the two workflows described in Fig. 3, we calculate $\boldsymbol{\delta}_{idle}$ and η using mECT scheduling given in Fig 4(a). $\sum_{i=1}^{2} idle\ time_i$ = 7, $\sum_{i=1}^{2} width\ (interval\ busy)_i$ = 45, $\sum_{i=1}^{2} Resource\ reserve\ time\ _i$ = 63. Therefore, $\boldsymbol{\delta}_{idle} = 0.15$, $\eta = 0.71$

Now we calculate the same parameters from Fig. 4(b). $\sum_{i=1}^{2} idle\ time_i$ = 6, $\sum_{i=1}^{2} width\ (interval\ busy)_i$ = 44, $\sum_{i=1}^{2} Resource\ reserve\ time\ _i$ = 63. Therefore, from

Eq. (2) we get $\boldsymbol{\delta}_{\text{idle}} = 0.13$ and from Eq. (3) we get $\eta = 0.69$.

Cycle time T_c is one of the key criteria for scheduling multiple workflows that run concurrently and in a cyclic order. If the makespan of any workflow exceeds T_c, the next cycle of the same workflow will not be able to start, and resource utilisation will suffer in the long run. This was established during simulation with more number of workflows having a different number of tasks (10 and 20). From the graph shown in Fig. 5(c), it is clear that mECT performed better in terms of system utilisation with more number of workflows. In both 10-task and 20-task categories, mECT and pHEFT show almost similar system utilisation, both performed 20% better than HCPA.

We have evaluated violated cycle time in each set of experiments with a different number of workflows. From Fig. 4(a), violated cycle time is 1 (workflow 1, cycle time 20, makespan 21), whereas, it is zero in the case of 4(b). However, the number of cycle time violation increases as the number of workflows is increased. The violated cycle time with the number of workflows is shown in Fig 5(b). Cycle time violation with mECT is 44% lesser compared to pHEFT and 75% lesser compared to HCPA in 10 task category. For 20 task category, mECT performance is 31% and 37% better compared to pHEFT and HCPA, respectively.

For each set of experiments, the simulation was carried out twice. Before the first run, the estimation of the cycle time of each workflow is calculated based on parallel execution time in the slowest resource. For example, cycle time for workflow 1, given in Fig. 3 is calculated as 4 + 6 + 8 = 18, and for workflow2, cycle time is 24. Based on the results of the first run, as shown in Fig. 4(a), cycle time is modified to 16 and 21.

Conclusions

In this paper, we have presented the multi workflow Earliest Cycle Time (mECT) algorithm to schedule workflows concurrently in a heterogeneous computing cluster. The algorithm enables cyclic processing of multiple tasks without any deadlock. We have done extensive simulation to evaluate the performance of mECT and compared it with two other algorithms. We have tested the algorithm to run a low resolution (N48) Weather Model in a simulated environment, Met Office Virtual Machine Box, developed by the Met Office, United Kingdom. Results indicate that mECT performs better in most of the performance evaluation criteria, except resource utilisation. We want to extend our work to address QoS parameters like fair resource sharing and resource on-demand, without increasing time complexity.

Acknowledgment

We have used *cylc* workflow engine to run N48 model in Virtual Machine Box. *Cylc* is developed by Oliver *et al.*[10] It is a workflow engine for orchestrating complex distributed suites of inter-dependent cycling (repeating) tasks, as well as common non-cycling workflows.

References

1 Kumar S, Dubey G & Tiwari S, Advances in Data and Information Sciences (Springer Singapore) 2020, 351–359.

2 Shahul S & Sinnen, Optimal Scheduling of Task Graphs on Parallel Systems, in *Int Conf on Parallel and Distributed Computing Applications and Technologies* (Otago, New Zealand) 1–4 December 2008.

3 Canon L, Sayah M, & Hе́am P, Euro-Par 2019: Parallel Processing (Springer International Publishing) 2019, 61–73.

4 Zhao H & Sakellariou R, Scheduling Multiple DAGs onto Heterogeneous Systems, in *Int Conf Parallel and Distributed Processing Symposium* (Research Academic Computer Technology Institute, Greece) 25–29 April 2006

5 N'Takpe T & Suter F, Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations, in *IEEE Int Symposium on Parallel & Distributed Processing*, (IPDPS, Rome, Italy), 25–29 May 2009.

6 Iverson M A & Özgüner F, Hierarchical, Competitive Scheduling of Multiple DAGs in a Dynamic Heterogeneous Environment, *Distrib Syst Engng*, **6(3)** (1999) 112–120.

7 Bochenina K, Butakov N & Boukhanovsky A, Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments, *Future Gener Comput Syst*, **55** (2016) 51–61.

8 Sun W, Zhang Y, & Inoguchi Y, Dynamic Task Flow Scheduling for Heterogeneous Distributed Computing: Algorithm and Strategy, *IEICE Trans Inf Syst*, **E90–D** (2007) 736–744.

9 Niyom A, Sophatsathit P & Lursinsap C, A fast predictive algorithm with idle reduction for heterogeneous system scheduling, *Simulat Model Pract Theor*, **63** (2016) 83–103.

10 Oliver H J, Shin M, & Sanders O, Cylc: A Workflow Engine for Cycling Systems, *J Open Source Softw*, **3** (2018) 737–738.

11 Hwang E, Kim S, Yoo T, Kim J, Hwang S, & Choi Y, Resource Allocation Policies for Loosely Coupled Applications in Heterogeneous Computing Systems, *IEEE Trans Parallel Distrib Syst*, **27** (2016) 2349–2362.

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

13 Sheikh M M, Khan A M, Thangavelu R P & Sinha U N, On scalability aspects of cluster computing, *J Sci Ind Res*, **43** (2014) 642–645.

14 Topcuoglu H & Hariri S, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Trans Parallel Distrib Syst*, **13** (2002) 260–274.

15 Acevedo C, Hernández P, Espinosa A, & Méndez V, A Critical Path File Location (CPFL) algorithm for data-aware multiworkflow scheduling on HPC clusters, *Future Gener Comput Syst*, **74** (2017) 51–62.

16 Deelman E, Vahi K, Rynge M, Mayani R, da Silva R F, Papadimitriou G & Livny M, The Evolution of the Pegasus Workflow Management Software, *Comput Sci Eng*, **10** (2019) 1–11.

17 Barbosa J G & Moreira B, Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters, *Parallel Comput*, **37** (2011) 428–438.

18 N'Takpe T & Suter F, Critical Path and Area Based Scheduling of Parallel Task Graphs on Heterogeneous Platforms, in *Int Conf on Parallel & Distributed Processing*, (Rhodes Island, Greece), 25–29 April 2006.