2017 IEEE International Parallel and Distributed Processing Symposium Workshops

Dynamic Mapping of Application Workflows in Heterogeneous Computing Environments

Muhammad Qasim[1], Touseef Iqbal[1], Ehsan Ullah Munir[1], Nikos Tziritas[2], Samee U. Khan[3], and Laurence T.Yang[4]
[1] COMSATS Institute of Information Technology, Pakistan
[2] Chinese Academy of Sciences, China
[3] North Dakota State University, USA
[4] St.Francis Xavier University, Canada
qasim.std@ciitwah.edu.pk; tusifpk.std@ciitwah.edu.pk; ehsanmunir@comsats.edu.pk; nikolaos@siat.ac.cn;
samee.khan@ndsu.edu; ltyang@stfx.ca

***Abstract*—Performance of a Heterogeneous Computing Environment (HCE) mainly depends on the efficiency of application workflow scheduling algorithms. Achieving high efficiency of application workflow scheduling algorithms in HCE is an NP-Complete problem. A novel application workflow scheduling algorithm called Heterogeneous Dynamic List Task Scheduling (HDLTS) for HCE is proposed in this paper. The functionality of HDLTS majorly relies on the following three pillars; first, duplicate the entry task only if it helps to reduce the overall application execution time; second, for mapping, consider only those tasks that have all the necessary input conditions to start the execution and find out the heterogeneity of their execution time on the computational resources; third, select the task that has higher execution time heterogeneity, and map it to a resource that takes minimum time to execute the task. The HDLTS task selection and mapping policies dynamically consider the resource utilization and task assignment that makes it more efficient and enables it to produce good quality schedules. The performance of the HDLTS is evaluated against popular list scheduling algorithms on randomly generated application workflows and real world application workflows. Experimental results prove that the HDLTS outperforms well-known list scheduling algorithms, such as in terms of schedule length and efficiency.**

***Keywords*—Heterogeneous computing, Workflow scheduling, List scheduling, Makespan**

I. Introduction

High-speed networks made it possible to interconnect the diverse set of low powered computational resources, such as Personal Computers, Tablets, and Cell Phones, to form a single apparent resource that can provide powerful computational capabilities. This powerful computational resource is termed as the Heterogeneous Computing Environment (HCE) and can be used to achieve High-Performance Computing (HPC) [1]. With the evolution of big data and big data analytics, more powerful computational resources are required to process the large volumes of big data. The HCE is an economical solution to achieve HPC for processing big data applications. An application for HCE consists of the number of tasks that can be executed in parallel on a set of computational resources. An efficient task assignment and resource management policy also called task scheduling is required to exploit the advantage of parallel execution of tasks among the set of heterogeneous computing resources.

Task scheduling is the process of assigning the application tasks to available computational resources such that it minimizes the communication overhead incurred due to data transfer among the execution of the tasks on different computational resources and speed up task execution by mapping tasks to a computational resource that complete task execution in minimum time. Task scheduling takes into account the heterogeneity of computational resources and communication overhead between computational resources. An efficient task scheduling policy can increase the performance of a HCE. The aim of task scheduling policy is to reduce the overall execution time of the application also called makespan of the application. Applications for a heterogeneous computing environment are modeled as Application Workflow represented by Directed Acyclic Graph (DAG). Application workflows are classified into two categories, namely Static application workflows and Dynamic application workflows [2], [3]. In a static application workflow model, nodes represent the tasks of application and edges represent the dependency of the tasks. Static application workflow scheduling is an NP-Complete problem [4]–[7].

Heuristic methods are well known for solving the application workflow scheduling problem in polynomial time. These methods generate near optimal schedules in polynomial time. To solve the static application workflow scheduling problem, many heuristic methods exist in the literature. Broadly, these methods are classified into four categories, namely cluster-based scheduling heuristics, list-based scheduling heuristics, task-duplication based scheduling heuristics, and genetic-based scheduling heuristics. List-based scheduling heuristics are more popular because of their low complexity and higher efficiency. Popular list-based scheduling heuristics are Heterogeneous Earliest Finish Time ($HEFT$) [8], Performance Effective Task Scheduling ($PETS$) [9], Critical-Path on Processor ($CPOP$) [8], Predict Earliest Finish Time ($PEFT$) [10], and Standard Deviation Based Task Scheduling ($SDBATS$) [11]. List-based scheduling usually consists of two phases. In the first phase, a ready list of precedence-constrained tasks is generated using some pre-defined criteria. The second phase assigns the tasks based on their pre-defined priority order to suitable machines. The complexity of list-based scheduling heuristics is least than any other scheduling heuristic methods. Genetic-based scheduling heuristics [12]–

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.129

[17] go through intensive search and produce a good quality schedule, but their time complexity is higher. Clustering-based scheduling heuristics [2], [18]–[20] are usually used for the unbounded number of computational resources. However, their schedule generation process is more complex to make them impractical to use. The Task-duplication based scheduling heuristics [13], [21], [22] have the highest time complexity hence, they are inefficient to deploy.

The major contribution of the research presented in this paper is a novel Heterogeneous Dynamic List Task Scheduling Heuristic ($HDLTS$) to solve the static application workflow scheduling problem in a HCE. We consider that all computing resources in the system are fully connected to each other and the number of computing resources are fixed for any given evaluation. Central Processing Unit (CPU) and computing resource are the alternative terms that we use to represent the set of computing resources in HCE. The motivation behind this work is to select only those tasks for mapping to computing resources in HCE that are ready to be executed at that time instance. This approach is more practical and generates a good quality schedule with low time complexity, therefore $HDLTS$ is a low-cost solution. The priority criteria for the execution of any task from the set of tasks is the Penalty Value (PV) of the task (defined in Section IV). The tasks with the highest PV will be executed first. We make our approach realistic and practical by considering the finish time of each task on a set of computational resources during each assignment. Therefore, our algorithm can be used for both types of static application workflows and dynamic application workflows. However, in this work we consider only the static application workflows. We build a synthetic task graph generator as part of this work to compare $HDLTS$ with well-known list scheduling algorithms in literature, such as Heterogeneous Earliest Finish Time ($HEFT$), Performance Effective Task Scheduling ($PETS$), Critical-Path on Processor ($CPOP$), Predict Earliest Finish Time ($PEFT$), and Standard Deviation Based Task Scheduling ($SDBATS$).

II. Related Work

A plethora of task scheduling techniques exists to solve the application workflow scheduling problem in a HCE. These algorithms are broadly categorized into two main categories, namely dynamic application workflow scheduling algorithms and static application workflow scheduling algorithms. Dynamic application workflow scheduling algorithms make scheduling decisions at run-time. However, in static application workflow scheduling algorithms, scheduling decisions are taken before deploying the application based on prior knowledge of task dependencies and resource capacities. This enables task scheduling at compile time. Static application workflow scheduling algorithms are further segregated into guided arbitrary-search based and the heuristic based group of algorithms. List-based scheduling heuristics, task-duplication based scheduling heuristics, and cluster-based scheduling heuristics are sub-categories of heuristic-based scheduling algorithms. An overview of heuristics based scheduling algorithms is discussed in the following section.

A. List Scheduling Heuristics

List scheduling heuristics work in two phases. In the first phase also called the task prioritization phase, a given set of precedence-constrained tasks are prioritized using some predefined criteria of prioritization. In the second phase, tasks are selected based on their pre-defined priority order. The execution time of the selected task is computed from the set of computational resources. The selected task is assigned to the machine that completes its execution in minimum time. The popular list based heuristics are $HEFT$, $PETS$, $CPOP$, $PEFT$, and $SDBATS$.

B. Task Duplication Based Heuristics

Scheduling a task from a given application workflow on all available processors is called task duplication. Task duplication may reduce the overall makespan, but with the cost of complexity and cost of higher energy consumption. The Duplication Based Heterogeneous Earliest Finish Time ($DHEFT$) introduces the concept of duplication in $HEFT$ algorithm that reduces the makespan significantly [23].

C. Clustering Heuristics

Clustering algorithms work by mapping a given application workflow to an unbounded number of clusters [18]. Iterative assignment of the task to the clusters is performed until the number of available computational resources become equal to the number of clusters. In each iteration, a random task is selected for mapping to a cluster and the clusters formed iteratively merge to form more optimum clusters. Tasks in the same cluster are executed on the same computational resource. Additionally, cluster mapping algorithm or the clustering algorithm is required in cluster-based heuristics to schedule the tasks within one cluster. Few of the main clustering algorithms for application workflows are Linear Clustering Method (LCM), Dominant Sequence Clustering (DSC), and Clustering and Scheduling System ($CASS$) [2], [18]–[20].

D. Selected List Scheduling Heuristics

This section briefly describes a few of the most cited lists based task scheduling heuristics which are $HEFT$, $PETS$, $CPOP$, $PEFT$, and $SDBATS$.

1) Heterogeneous Earliest Finish Time: The $HEFT$ algorithm computes upward rank of each task in application workflow using their mean computation time across a diverse set of computational resources. After that, tasks are sorted in the descending order of their rank values such that the task with a higher rank value will have a highest priority. Prioritized tasks are then mapped to the available computational resources using their Earliest Finish Time (EFT) value. A processor that gives the minimum EFT result will be selected for the task execution. Insertion based policy is adopted to fully utilize the idle time slots if exists. The $HEFT$ has the time complexity of $O(V^2 \times P)$ for the V number of tasks in the application and P number of CPUs in the HCE.

2) Performance Effective Task Scheduling: The $PETS$ algorithm, models the communication cost between tasks as Data Transfer Cost (DTC) and Data Receiving Cost (DRC). Priorities of the tasks are calculated in the task prioritization phase on the basis of their ranks that are calculated by adding their DTC, DRC, and Average Computation Cost (ACC). The CPU assignment phase maps the prioritized tasks on available CPUs based on their minimum EFT value. The $PETS$ also take in account the insertion base policy to address the possibility of inserting a process in the free time slot. The $O(V+E)(P+logV)$ is the time complexity of $PETS$ algorithm for V number of tasks in the application, E number of edges between tasks in application and P number of CPUs in a HCE.

3) Predicted Earliest Finish Time: The basic idea behind the $PEFT$ algorithm is the formulation of the Optimistic Cost Table (OCT) on the basis of which the task prioritization and processor selection are performed. Each element in the OCT table annotates maximum cost for a task to the exit task through its immediate child tasks. The time complexity of $PEFT$ is $O(V^2 \times P)$ for V number of tasks of application and P number of CPUs in a HCE.

4) Standard Deviation Based Task Scheduling Heuristic: The $SDBATS$ algorithm uses standard deviation of the execution time of a given set of tasks on the set of the computational resources as the key value for calculating the upward rank of tasks. The value of upward rank determines the priority of tasks and their execution order. The $SDBATS$ uses entry task duplication to enhance the efficiency of the algorithm. The time complexity of $SDBATS$ is also $O(V^2 \times P)$ for V number of tasks in the application and P number of machines in a HCE.

III. Problem Formulation

In this paper, we adopt a static application workflow model for the task scheduling problem. A static application workflow model is represented by a direct acyclic graph (DAG) $G = (V, E)$, where V is the set of tasks $V = \{v_1, v_2, ..., v_n\}$ that shows n number of tasks of application workflow and E is the set of edges $E = \{e_1, e_2, ..., e_m\}$ that represents the dependency constraint between the tasks of an application workflow. Moreover, the tasks in the application workflow are distributed into the k levels in DAG. Tasks on the same level are the independent tasks and can be executed in parallel. An example static application workflow model is shown in the Fig. 1. The task with no parent task is representing the entry task (V_{entry}) of the application workflow and the task with no child task is representing the exit task (V_{exit}) of the application workflow. A task graph may have more than one entry and exit tasks. We use a pseudo task to model the multiple entry and exit task graphs into a single entry and exit task graphs. This pseudo task has zero computation cost and is connected with its child tasks with zero communication cost. An HCE consists of the $M = \{m_1, m_2, ..., m_p\}$ set of p heterogeneous computational resources. We also assume that all the computational resources are fully connected and there is no network contention between them. Moreover, the assignment of tasks to the computational resources is non-preemptive. The common attributes of the task scheduling problem are discussed in the following section:

A. Common Attributes of Task Scheduling Problem

Definition 1: The execution time of a task on a CPU is calculated by dividing the number of instructions in the task by the clock frequency (Hz) of the CPU. The W is an $n \times p$ matrix that represents the computation time of the task $v_i \epsilon V$ on each CPU $m_p \epsilon M$. The mean execution time of a task $v_i \epsilon V$ on set of p computational resources $M = \{m_1, m_2, ..., m_p\}$ is calculated using the following relationship:

$$W(v_i) = \sum_{j=1}^{p} W(v_i, m_j)/p. \quad (1)$$

Definition 2: The communication time is the overhead that generates due to the data transfer between the different computational resources. Let the volume of the data transfer between the two tasks v_i and v_j is $Data(v_i, v_j)$. The Bandwidth of the channel that connects the two computational resources m_i and m_j is $B(m_i, m_j)$ that is representing the maximum data transfer rate of the channel. The communication cost is the amount of time taken to send the $Data(v_i, v_j)$ on the channel $B(m_i, m_j)$. A communication cost matrix C is an $n \times n$ matrix that shows the task dependencies and the communication cost between the tasks and is expressed as:

$$Comm\_Cost(v_i, v_j) = Data(v_i, v_j)/B(m_i, m_j). \quad (2)$$

If two tasks v_i and v_j are computed on the same computing resource then the communication cost will be considered zero.

Definition 3: The time in which a CPU finishes the execution of a task and becomes ready to execute a new task is called the available time of that CPU. If a task $v_i \epsilon V$ is assigned to a CPU $m_p \epsilon M$, then the available time of m_p will be the finish time of v_i on m_p. The available time of a CPU can be calculated as:

$$Avail(m_p) = Avail(m_p) + W(v_i, m_p). \quad (3)$$

Definition 4: The finish time, also called the Actual Finish Time (AFT) of a task $v_i \epsilon V$, is the time taken by a CPU to successfully complete the task. It is calculated using the following relationship:

$$AFT(V_i) = Finish(V_i). \quad (4)$$

Definition 5: The ready time of a task $v_i \epsilon V$ on a CPU $m_p \epsilon M$, is the time when the input conditions of the task have met and the task is ready to be assigned to a CPU for the execution. A task becomes ready only when its parent tasks have finished their execution and the results from all the parent tasks have become available as the input of the task. The ready time of the entry task v_{entry} is always assumed to be zero. If a task has a single parent and both the parent and child task assigned to the same computational resource then the ready time of the child task will be the finish time of the parent task. Following relationship represents the ready time:

$$Ready(v_i, m_p) = AFT(v_j) + Comm\_cost(v_i, v_j). \quad (5)$$

Definition 6: The earliest start time (EST) of a task v_i on the CPU m_p depends on the ready time of the task v_i on the

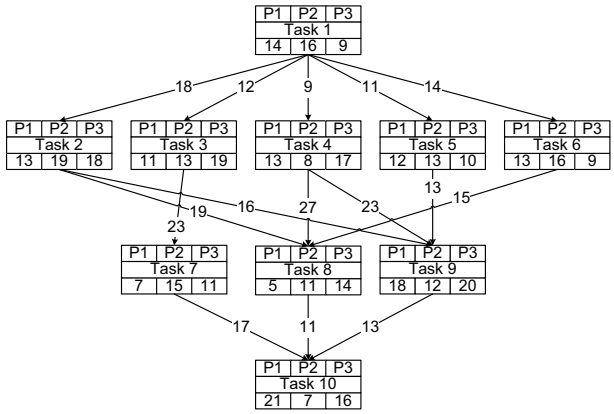

Fig. 1. A sample application workflow of ten processes.

CPU m_p and the available time of the CPU m_p, whichever is later. The EST of a task is calculated as:

$$EST(v_i, m_p) = max\{Ready(v_i, m_p), Avail(m_p)\}. \quad (6)$$

Definition 7: The earliest finish time (EFT) of a task $v_i \epsilon V$ on the computing resource m_p is the least amount of time a computing resource $m_p \epsilon M$ takes to process v_i. The EFT of a task is calculated as:

$$EFT(v_i, m_p) = EST(v_i, m_p) + W(v_i, m_p). \quad (7)$$

Definition 8: The penalty value (PV) of a task $v_i \epsilon V$ is the standard deviation σ of the EFT values of the task v_i on a set of the computing resources M. It represents the heterogeneity of the task v_i, which is executed on the different computational resources in a HCE. A task with a higher PV value may increase the overall finish time of an application significantly if not handled on the priority. The PV of a task is calculated as:

$$PV_{v_i} = \sigma\{EFT(v_i, m_p)\}. \quad (8)$$

Definition 9: The makespan of an application workflow is the AFT of the v_{exit} of the application. It is the maximum time taken to complete an application workflow.

$$Makespan = AFT(v_{exit}). \quad (9)$$

IV. Heterogeneous Dynamic List Task Scheduling Heuristics (HDLTS)

List scheduling algorithms usually consist of two phases, namely task prioritization phase, and CPU assignment phase. The task duplication and the insertion base assignment are the two optimization techniques to reduce the makespan in list scheduling. Our proposed algorithm HDLTS consists of three phases, namely *Effective Entry Task Duplication* phase, *Task Prioritization phase*, and *CPU Selection* phase.

Effective Entry Task Duplication Phase: This phase is the optimization phase of the HDLTS. To avoid the overhead incurred due to the task duplication, we propose an effective technique to duplicate only the entry task of an application. The proposed Algorithm 1 duplicates the entry task only if it results in reducing the overall makespan. Our proposed technique checks the actual finish time (AFT) and the communication time ($Comm\_Cost$) of the entry task to all of its child tasks on a particular CPU. If sum of the AFT and $Comm\_Cost$ is greater than the earliest start time (EST) of the entry task on that particular CPU, then the entry task will be duplicated only on that particular CPU. The Algorithm 1 ensures that there is no unnecessary duplication of the entry task on all the CPUs.

Algorithm 1: Effective Entry Task Duplication

Input: Application workflow graph $G = (V, E)$
Output: Entry Task Duplication(true|false)
begin
  for *k in CPUs* **do**
    if $EST(V_i, k) < AFT(V_i) + Comm\_Cost(V_i, V_j)$ **then**
      Duplicate the entry task on the CPU k

Task Prioritization Phase: The task prioritization phase in the list scheduling generates a static list of the ready tasks using some prioritization criteria and assign the priorities of the tasks to schedule them in a specific order. Once this static ready list is computed, then the tasks are mapped according to their priorities in the task assignment phase. The approach is simple, but it does not allow the scheduler to take into account the status of the computing resources during the task assignment phase. In an application workflow, the term independent task refers to a task whom parent tasks have finished their execution. Our task prioritization phase generates a dynamic ready list of the independent tasks only. We named this dynamic ready list as the Independent Task Queue (ITQ). The standard deviation of the execution time of a task on the set of CPUs represents the heterogeneity of the task execution in a HCE. The prioritization criterion of the HDLTS is the penalty value (PV) of the tasks in the ITQ. The PV of the task represents its heterogeneity and is computed using the Equation 8. Our algorithm prioritizes the tasks in the ITQ and updates the ITQ after mapping the task to a computing resource. An independent task is allowed to run on any of the available CPUs in a HCE. The task with a higher PV will have a highest priority of the execution and is mapped to a CPU according to the Algorithm 2. Once a task is mapped to a CPU it will be removed from the ITQ. A new independent task if produced, will be added to the ITQ and the priorities of the tasks will be re-calculated. This process continues until all the tasks of the application are successfully prioritized as described in the Algorithm 2.

CPU Selection Phase: In this phase, the $HDLTS$ maps the highest priority task in the ITQ to the computing resource that can complete its execution at the earliest among all the CPUs in a HCE. The $HDLTS$ computes the earliest finish time (EFT) of a selected task for all the CPUs using the Equation (7). The $HDLTS$ selects the CPU that gives the minimum EFT value and assigns the task to selected CPU. The selected task is then removed from the ITQ. This process continues until all the tasks of an application are mapped.

To illustrate the working of the $HDLTS$ we use a task

Algorithm 2: Heterogeneous Dynamic List Task Scheduling (HDLTS)

Input: Scientific application workflow graph $G = (V, E, W, C)$
Output: Makespan of the application workflow.
begin
  for *all tasks in the application workflow* **do**
    Add independent tasks into the independent task queue (ITQ)
    while ITQ *is not empty* **do**
      for *all tasks in the* ITQ **do**
        Duplicate the entry task using the Algorithm 1.
        Compute the EST and EFT values of each independent task using the Eq. (6) and Eq. (7).
        Assign priorities based on heterogeneity of the tasks using the Eq. (8).
        Remove the highest priority task from the ITQ and assign it to its ideal CPU define in the Section 4.
        Update the ITQ.

graph shown in the Fig. 1. Each box represents a task and P_1, P_2, and P_3 represent three CPUs in a HCE. The value in the each column of the box represents the execution time of the task on that CPU. The task graph shown in the Fig. 1, has 10 tasks and 3 heterogeneous computing resources. A matrix W is a (10×3) matrix represents the execution time of each task on the corresponding CPU. The edges in the task graph represent the dependency constraints and show the communication cost incurred due to the data transfer between two tasks. Therefore, the input of $HDLTS$ is an application task graph $G = (V, E, W, C)$ and N computing resources in a HCE. For the above, V represents the number of tasks in the application, E is the set of edges between tasks, W is an $(V \times N)$ matrix showing the execution time of V tasks for N computing resources, and C is an $(V \times V)$ matrix showing the communication cost between tasks and their dependencies. The output of the Algorithm 2 is the makespan of the application.

Initially, V_{entry} is the only independent task and will be added to ITQ. The execution cost of the V_{entry} on the CPUs P_1, P_2, and P_3 is 14, 16, and 9, respectively. The penalty value of the V_{entry} is 7.0 and is calculated using the Equation (8). Since P_3 takes the minimum execution time, therefore, the V_{entry} will be assigned to P_3. The value of the AFT of the V_{entry} will be updated to 9. Now the tasks T_2, T_3, T_4, T_5, and T_6 become the independent tasks and will be added to ITQ. The EFT values for all the independent tasks T_2, T_3, T_4, T_5, and T_6 are calculated using the Equation (7) that are $[27, 35, 27]$, $[25, 29, 28]$, $[27, 24, 26]$, $[26, 29, 19]$, and $[27, 32, 18]$ respectively. The penalty values of the tasks T_2, T_3, T_4, T_5, and T_6 are calculated using the Equation (8), which are 4.6, 2.0, 1.5, 5.1, and 7.0, respectively. The ITQ is then sorted in the descending order of their penalty values and T_6 is removed from the ITQ because of its highest PV. The Algorithm 1 checks for the possible entry task duplication of T_6. The AFT value of T_6 is calculated using the Equation (3) and T_6 is assigned to P_3. This process is repeated and on every iteration, the tasks which become the independent task will be added to the ITQ and the task with the highest PV will be removed from ITQ and mapped to the suitable CPU. Once all the tasks have finished their execution, the makespan of the task graph is obtained using the Equation (9). The Table I shows the scheduled of an application task graph shown in the Fig. 1 produced by the $HDLTS$ algorithm. The makespan of the $HDLTS$ for the given application task graph is 73, which is least among all the other list scheduling algorithms described in the Section II. While, the makespan of the $HEFT$, $PETS$, $PEFT$, and $SDBATS$, are 80, 77, 86, and 74, respectively.

TABLE I. HDLTS SCHEDULE PRODUCED AT EACH STEP

Step	Ready Task	Penalty Values	Selected Task	EFT P1 P2 P3
1	$\mathbf{T_1}$	**7.0**	$\mathbf{T_1}$	14 16 **9**
2	T_2, T_3, T_4, T_5,$\mathbf{T_6}$	4.6, 2.0, 1.5, 5.1, **7.0**	$\mathbf{T_6}$	27 32 **18**
3	T_2, $\mathbf{T_3}$, T_4, T_5	4.9, **6.1**, 5.6, 1.5	$\mathbf{T_3}$	**25** 29 37
4	T_2, T_4, T_5, $\mathbf{T_7}$	1.5, 7.3, 4.9, **16.8**	$\mathbf{T_7}$	**32** 63 59
5	T_2, $\mathbf{T_4}$, T_5	5.5, **10.5**, 8.9	$\mathbf{T_4}$	45 **24** 35
6	T_2, $\mathbf{T_5}$	4.7, **8.0**	$\mathbf{T_5}$	44 37 **28**
7	$\mathbf{T_2}$	**1.5**	$\mathbf{T_2}$	45 **43** 46
8	T_8 , $\mathbf{T_9}$	11.0, **13.3**	$\mathbf{T_9}$	77 **55** 79
9	$\mathbf{T_8}$	**5.5**	$\mathbf{T_8}$	67 **66** 76
10	$\mathbf{T_{10}}$	**13.2**	$\mathbf{T_{10}}$	98 **73** 93

In the $HDLTS$, we update the priority of the tasks in the ITQ on each iteration and takes into account the CPUs availability before assignment of the task. This approach is beneficial if any of the CPU in the underlying HCE is malfunctioning, the $HDLTS$ will still be able to efficiently assign the tasks to the remaining available CPUs. Therefore, the $HDLTS$ has the higher efficiency and load balancing. The complexity of the $HDLTS$ for mapping of v tasks that are distributed over the k levels of an application workflow is $O((v^2) \times (v/k) \times p))$, where p represents the number of computing resources available in a HCE.

V. EVALUATION

This section presents the evaluation of the $HDLTS$, $HEFT$, $PETS$, $PEFT$, and $SDBATS$ algorithms. For the evaluation, we develop a synthetic DAG generator with properties as discussed in the [8]. We also use the real world example scenarios like Fast Fourier Transform (FFT), Molecular Dynamics (MD), and Montage application workflows for evaluation of the $HDLTS$ and selected list scheduling algorithms. Our simulation testbed includes a single PC with Intel Quad-Core Xeon CPU running at 2.93 GHz and with 12GB memory.

A. Comparison Metrics

The comparison metrics used in the evaluation are the scheduling length ratio (SLR), $speedup$, and $efficiency$.

1) Scheduling Length Ratio: The makespan is the overall execution time of the application as defined in the Equation (3). However, the makespan does not represent how much the generated makespan is better than the minimum possible makespan of the application. The scheduling length ratio (SLR) is a common metric that represents the ratio between the makespan of the application to the minimum possible makespan of the application. The SLR is defined as follows:

$$SLR = \frac{makespan}{\sum_{n \epsilon CP_{MIN}} min_{P_j \epsilon (W_{(i,j)})}}. \quad (10)$$

where CP_{min} represents the critical path of an application workflow. The denominator value shows the lower bound of the makespan and it is equal to sum of the minimum execution time of the critical path tasks. A Higher value of SLR means the poor performance of the algorithm.

2) Speedup: This metric numerically measures in the unit time that how quickly an application workflow execution is completed by using the parallel execution. It is the mathematically equivalent to the ratio of the sequential computation time on single CPU to the parallel execution time on the set of CPUs.

$$Speedup = \frac{min \sum_{P_j \epsilon M} W(v_i, p_j)}{makespan}. \quad (11)$$

The numerator value shows the minimum value of the sequential execution time of all the tasks on a set of CPUs. $Speedup$ value shows the improvement in the overall execution time of the application due to the parallel execution of tasks.

3) Efficiency: The term efficiency describes how much for a certain number of CPUs a system will $Speedup$. For a given application workflow, increase in the total number of CPUs in a HCE increase the system efficiency to a certain point after that point increase in the number of CPUs may decrease the system performance or efficiency. The efficiency of a system can be calculated using the following relationship:

$$Efficiency = \frac{Speedup}{NumberofCPUs}. \quad (12)$$

B. Random Task Graph Generator

We develop a synthetic task graph generator for the evaluation of the $HEFT$, $PETS$, $PEFT$, $SDBATS$, and $HDLTS$. Our task graph generator uses the same parameters for the generation of random task graphs as described in the [8], [11], [24] for a fair evaluation. However, our task graph generator has the ability to generate the multiple entry and exit tasks graphs. In our evaluation, we used a pseudo task that models the multiple entry and exit task graphs into a single entry and exit task graphs. Our task graph generator is more scalable and effectively generates a range of task graphs that contain tasks as low as few tasks to tens of thousands of tasks. The parameters used in our task graph generator controls the different characteristics of the generated task graphs and are defined in the following section:

1) V: This parameter defines the total number of tasks in the generated task graph.

TABLE II. PARAMETERS USED TO GENERATE RANDOM TASK GRAPHS

Parameters	Values
Tasks (V)	100, 200, 300, 400, 500, 1000, 5000, 10000
Alpha (α)	0.5, 1.0, 1.5, 2.0, 2.5
$Density$	1, 2, 3, 4, 5
CCR	1.0, 2.0, 3.0, 4.0, 5.0
Number of CPUs	2, 4, 6, 8, 10
W_{dag}	50, 60, 70, 80, 90, 100
Beta (β)	0.4, 0.8, 1.2, 1.6, 2.0

2) α: The shape parameter (α) is also called the fitness of a task graph. It determines the parallelism between the different tasks in an application workflow. The height of application workflow is equal to the $\sqrt{v}/\alpha$, a lower value of α generates the taller and thin task graphs with low parallelism. While the width of the task graph is equal to $\sqrt{v} \times \alpha$. Therefore, a higher value of α represents the fatty task graph with a high parallelism between tasks.

3) $Density$: This parameter defines the outdegree of a task and represents the dependencies among the tasks. A higher value of the density generates the task graphs with more edges hence more dependency.

4) Communication to Computation Ratio (CCR): This parameter controls the type of task graph generated. A higher value of CCR represents the data intensive task graphs and a low value of CCR represents the computation intensive task graphs.

5) Number of CPUs: This parameter defines the number of computing resources used for evaluation of the generated task graphs.

6) Mean Computation Time of DAG (W_{dag}): This parameter controls the overall computational time of all the tasks in the task graphs. The value of W_{dag} is distributed uniformly for all the tasks in the task graphs.

7) β: The value of the β controls the heterogeneity factor of the task execution among the set of CPUs. A Higher value of the β shows a higher variation in task execution time among CPUs.

The average computation time of each task v_i is selected randomly over the uniform distribution of range $[0, 2 \times W_{dag}]$. The computation time of each task v_i for the set of CPUs is selected using the following relationship:

$$\{w_i \times (1 - \beta/2)\} <= w(i,j) <= \{w_i \times (1 + \beta/2)\}. \quad (13)$$

The communication cost of the edges is calculated by using the following relationship:

$$Comm\_Cost(v_i, v_j) = w_i \times CCR. \quad (14)$$

The combination of the parameters used in our simulation study to generate random application task graphs are given in the Table. II

The combination of the parameters shown in the Table. II can generate $125K$ unique application workflow graphs. We run the simulation for each combination of the parameters for 1000 time to get an average value of the results of performance metrics define in the Section V-A. The results of our evaluation show that the $HDLTS$ outperforms the other list scheduling heuristics used in the evaluation.

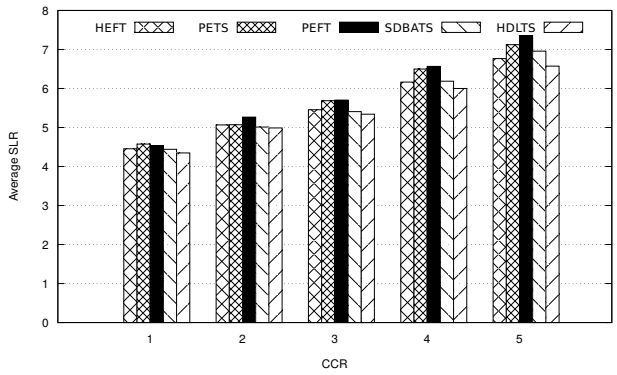

Fig. 2. Average SLR of random application workflow Vs CCR.

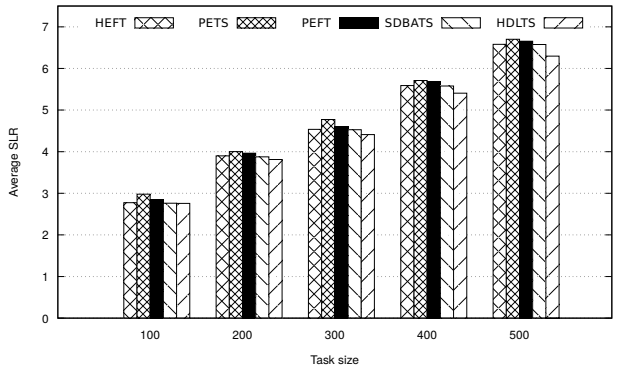

Fig. 3. Average SLR of random application workflow Vs Task size.

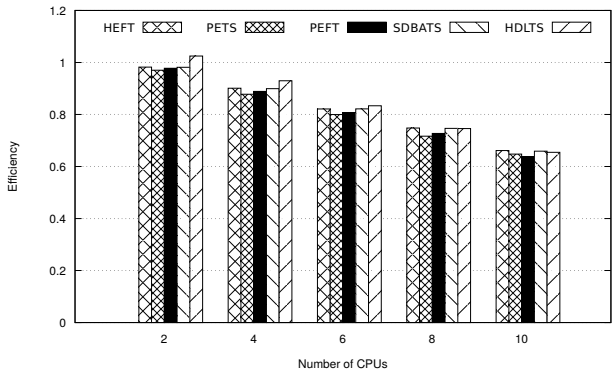

Fig. 4. Efficiency of random application workflow Vs Number of CPU.

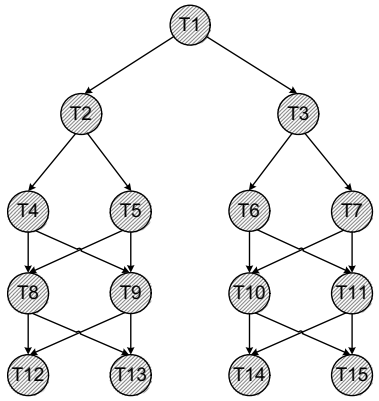

Fig. 5. An FFT application workflow of four input points.

The SLR obtained as a function of the different values of CCR and task size is shown in the Fig. 2 and Fig. 3, respectively. The results show that for the lower values of CCR, the SLR of the $HDLTS$ is equal to the SLR of $HEFT$ and $SDBATS$, but the increasing value of CCR shows that for communication intensive task graphs, the $HDLTS$ outperforms the other list scheduling algorithms. The Fig. 3 shows that the $HDLTS$ performs better for application workflows that have the higher number of tasks. The evaluation of efficiency as a function of the number of CPUs for synthetic task graphs is shown in the Fig. 4. The results show that for a less number of computing resources, the efficiency of the $HDLTS$ is higher significantly, but with an increasing number of CPUs, $HEFT$ and $SDBATS$ outperform the $HDLTS$. A decrease in the efficiency of the $HDLTS$ for the higher number of CPUs is because the $HDLTS$ takes into account the heterogeneity of the independent tasks only and does not take a look at the overall structure of the application and the impact of a CPU assignment for a task to its child tasks.

C. Real World Applications Workflows

In addition to the Random Task Graph Generator, we evaluate the $HDLTS$ for the real world application workflows such as, Fast Fourier Transform application workflow [8], Montage application workflow [25], and Molecular Dynamic application workflow [8].

1) Fast Fourier Transform application workflow: The FFT based application workflows are structured and designed on the basis of recursive and butterfly algorithms. A FFT application workflow for 4 data points based structure is shown in the Fig. 5. The total number of tasks in the recursive part of the application are equal to $(2 \times (m-1)+1)$, and the total number of tasks in butterfly operation are equal to $(m \times \log^2 m)$, where the size of a matrix is denoted by m. We can change the value of m to the power of 2, to obtain the different number of tasks in an application.

To analyze $HDLTS$ for different types of the FFT based application workflow models, we change the value of m from 4 to 32, which generates the FFT application workflows that contain the task ranges from 15 tasks to the 223 tasks, respectively. The evaluation of the SLR as a function of the different values of input points for the FFT application workflows is shown in the Fig. 6. The average SLR as a function of the CCR values is shown in the Fig. 7. The results show that the $HDLTS$ outperforms with the lowest SLR for the various input values of the CCR. The evaluation of efficiency of the $HDLTS$ and selected list scheduling algorithms, we set the size of the input points to 16, and perform the simulation for the different number of CPUs. The results are shown in the Fig. 8. The results show that, the HDLTS outperforms all the other selected list scheduling algorithms in terms of efficiency.

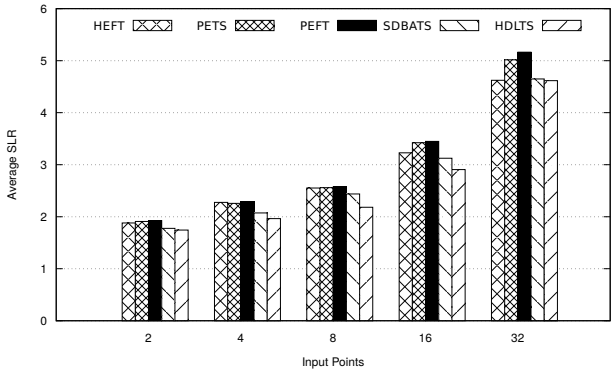

Fig. 6. Average SLR of FFT application workflow Vs Input points.

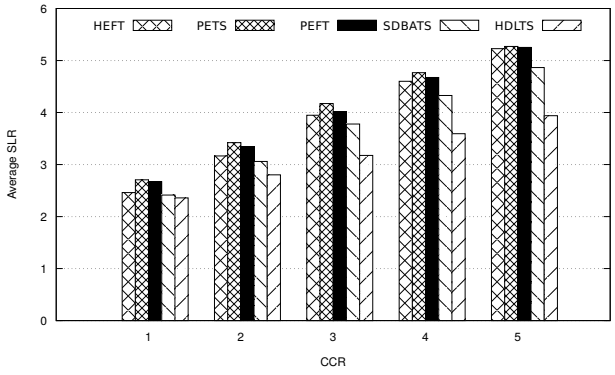

Fig. 7. Average SLR of FFT application workflow Vs CCR.

2) Montage application workflow: The Montage [26] is another real world application based on a specific type of the application workflow structure. It is used in space industry for the mosaics of an astronomical image of the sky or any galaxy. A typical model of the 20 nodes montage application is shown in the Fig. 9. The structure of the montage application workflows are well defined so we can change the heterogeneity factor of computation time of each task in the application, the CCR value of the application and the number of CPUs for the evaluation of the $HDLTS$. In our evaluation, we use the montage workflows with fixed structure, that contains 50 and 100 nodes, and varies the different CCR values in the range between 1 to 5, while keeping the number of CPUs to 5, and obtained the mean SLR value of 1000 runs for each value of the CCR. We run the simulation, while keeping the CCR at 3 and the varying number of CPUs from 2 to 10 and get the efficiency of the $HDLTS$ and other algorithms. The comparison results of both SLR and $efficiency$ are shown in the Fig. 10 and Fig. 11, respectively. The results for the average SLR against the CCR values, and the efficiency against the number of CPUs show that the $HDLTS$ outperforms in the both cases. This shows the HDLTS is more efficient for scheduling the scientific application workflows, such as the Montage.

3) Molecular Dynamics Code: An application workflow model of a modified molecular dynamic code [27] is shown in the Fig. 12. The structure of the molecular dynamics code

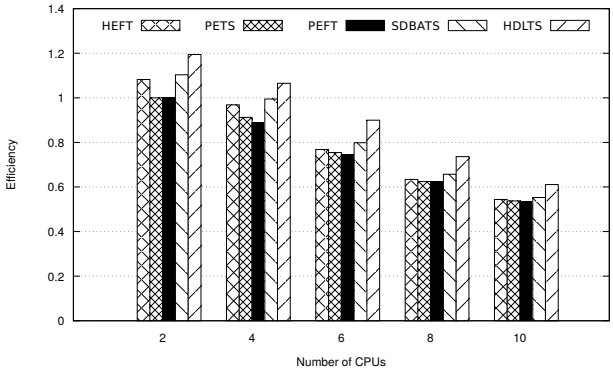

Fig. 8. Efficiency of FFT application workflow Vs Number of CPU.

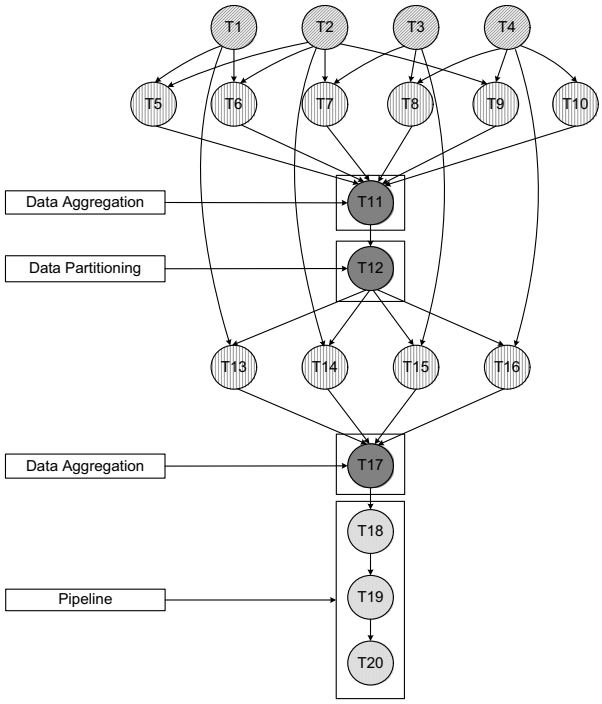

Fig. 9. A sample Montage workflow.

is fixed, therefore our evaluation is based on varying values of the CCR, β, and the number of computing resources. The Fig. 13 represents the performance of the $HDLTS$ for the average SLR as a function of different values of the CCR. The results show the $HDLTS$ outperforms other algorithms significantly. We also evaluate the efficiency of the $HDLTS$ for a different number of CPUs while keeping the CCR value at 3. We take different values of the CPUs from the range 2 to 10 and measure the efficiency of the $HDLTS$. The results are shown in the Fig. 14. The $HDLTS$ performs better in all scenarios with a different number of CPUs. This represents that for the real world application based on the Molecular Dynamic,

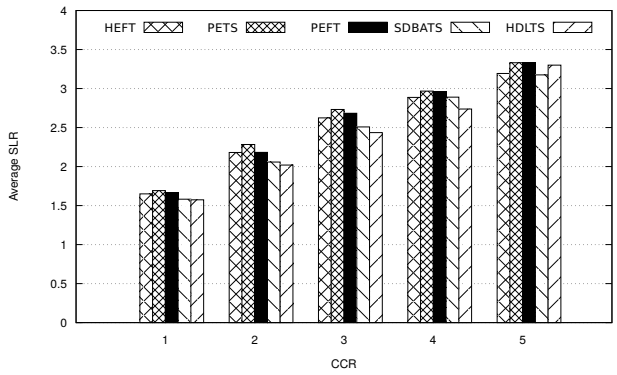

Fig. 10. Average SLR of Montage application workflow Vs CCR.

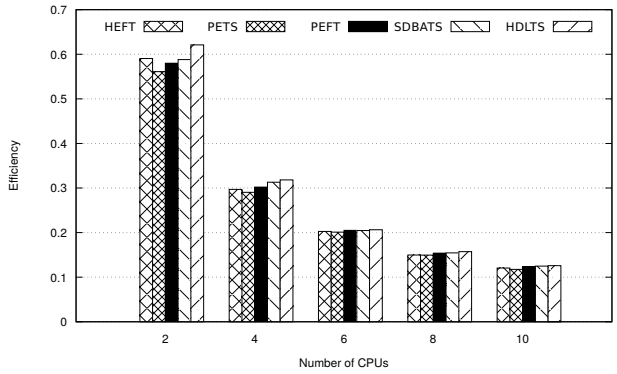

Fig. 11. Efficiency of Montage application workflow Vs Number of CPU.

our algorithm outperforms the $HEFT$, $PETS$, $PEFT$, and $SDBATS$. Moreover, the $HDLTS$ is more efficient and effective algorithm for these type of applications.

VI. Conclusion

In this paper, we proposed a novel task scheduling algorithm called the $HDLTS$. This algorithm produces better schedules as compared to the schedules generated by the popular list scheduling algorithms $HEFT$, $PETS$, $PEFT$, and $SDBATS$. The algorithm is based on the dynamic list of ready tasks instead of the static list. This dynamic ready list contains only the independent tasks at a given instance. We called this dynamic list as independent task queue (ITQ). Tasks in the ITQ are prioritized using the heterogeneity of their execution times on the set of computing resources. The assignment of a computing resource to a task is based on the EFT value of the highest priority task in the ITQ. The $HDLTS$ also takes into account the resources status before prioritizing the tasks for mapping. Therefore, the $HDLTS$ is efficient in load balancing also. Our approach is straightforward and can be applied for both static and dynamic application workflows. The complexity of the $HDLTS$ for mapping of v tasks that are distributed over the k levels of an application workflow is $O((v^2) \times (v/k) \times p))$, where p represents the number of computing resources available in a HCE.

The HDLTS mainly focuses on reducing the scheduling

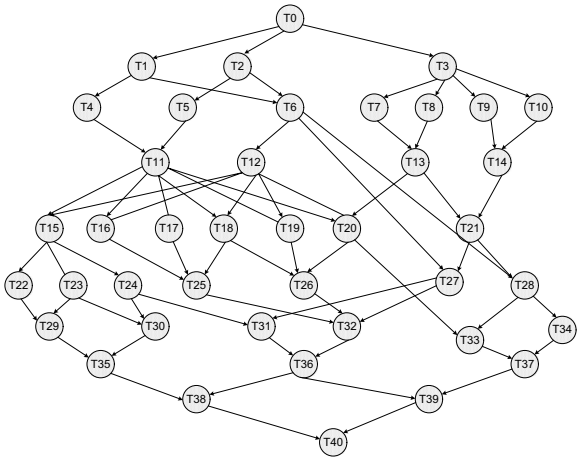

Fig. 12. A sample Molecular Dynamic workflow.

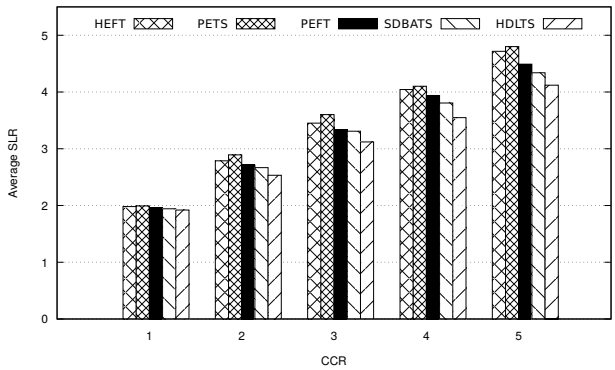

Fig. 13. Average SLR of Molecular Dynamic application workflow Vs CCR.

length and efficient utilization of resources. The optimization criterion used in the $HDLTS$ is the duplication of the entry task. We proposed an efficient solution to duplicate the entry task only if it is helpful to reduce the makespan of an application. The generation of the static list of all the tasks of an application in advance restricts the scheduler to make the scheduling decisions at runtime. The $HDLTS$ allows the scheduler to assign the priorities to the tasks at runtime, therefore, the $HDLTS$ can increase the efficiency of scheduling for uncertain conditions in a HCE.

The experimental evaluation of the $HDLTS$ is based on two types of application workflows, namely random application workflows and real world application workflows. The random application workflows are produced using a synthetic task graph generator. We develop this synthetic task graph generator as part of this research and for the evaluation purpose, we produced the $125K$ unique random task graphs. The real world application workflows used in evaluation are Fast Fourier Transform application workflow, Montage application workflow and Molecular Dynamics application workflow. The evaluation is performed to obtain the results of SLR and $efficiency$ of the $HDLTS$, $HEFT$, $PETS$, $PEFT$, and $SDBATS$. The results represent the $HDLTS$ outperforms rest of the algorithms in terms of both performance metrics.

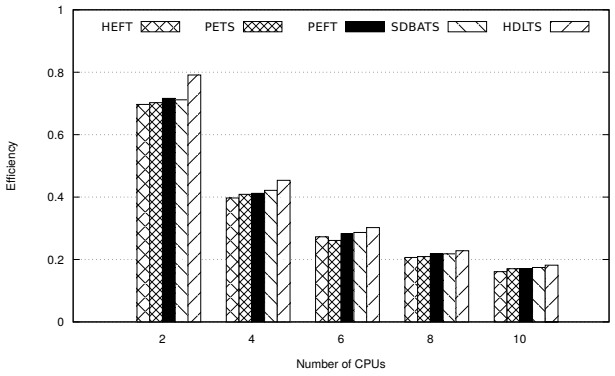

Fig. 14. Efficiency of Molecular Dynamic application workflow Vs Number of CPU.

In future, we will use the $HDLTS$ in an uncertain heterogeneous computing environment and network conditions, to test the efficiency of the $HDLTS$. We will also propose the application of the $HDLTS$ in dynamic application workflow and test its performance.

VII. Acknowledgments

Samee U. Khan's work was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[3] A. Masood, E. U. Munir, M. M. Rafique, and S. U. Khan, "Hets: Heterogeneous edge and task scheduling algorithm for heterogeneous computing systems," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 1865–1870.

[4] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang, and J. Wang, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, no. 4, pp. 177–188, 2013.

[5] Q.-K. Pan and R. Ruiz, "An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem," *Omega*, vol. 44, pp. 41–50, 2014.

[6] O. Sinnen, "Reducing the solution space of optimal task scheduling," *Computers & Operations Research*, vol. 43, pp. 201–214, 2014.

[7] T. K. Ghosh, S. Das, S. Barman, and R. Goswami, "A comparison between genetic algorithm and cuckoo search algorithm to minimize the makespan for grid job scheduling," in *Advances in Computational Intelligence: Proceedings of International Conference on Computational Intelligence 2015*. Springer, 2017, pp. 141–147.

[8] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, 2002.

[9] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, "Performance effective task scheduling algorithm for heterogeneous computing system," in *Parallel and Distributed Computing, 2005. ISPDC 2005. The 4th International Symposium on*. IEEE, 2005, pp. 28–38.

[10] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 3, pp. 682–694, 2014.

[11] E. U. Munir, S. Mohsin, A. Hussain, M. W. Nisar, and S. Ali, "Sdbats: a novel algorithm for task scheduling in heterogeneous computing systems," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 43–53.

[12] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014.

[13] C. Fan, H. Deng, F. Wang, S. Wei, W. Dai, and B. Liang, "A survey on task scheduling method in heterogeneous computing system," in *Intelligent Networks and Intelligent Systems (ICINIS), 2015 8th International Conference on*. IEEE, 2015, pp. 90–93.

[14] Y. Xu, K. Li, L. He, L. Zhang, and K. Li, "A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems," *IEEE Transactions on parallel and distributed systems*, vol. 26, no. 12, pp. 3208–3222, 2015.

[15] M. Oussalah, D. Professor Ali Hessami, N. Jafari Navimipour, A. Masoud Rahmani, A. Habibizad Navin, and M. Hosseinzadeh, "Job scheduling in the expert cloud based on genetic algorithms," *Kybernetes*, vol. 43, no. 8, pp. 1262–1275, 2014.

[16] S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 87, pp. 80–90, 2016.

[17] S. G. Ahmad, E. U. Munir, and W. Nisar, "Pega: A performance effective genetic algorithm for task scheduling in heterogeneous systems," in *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, June 2012, pp. 1082–1087.

[18] A. Deldari, M. Naghibzadeh, S. Abrishami, and A. Rezaeian, "A clustering approach to scientific workflow scheduling on the cloud with deadline and cost constraints," *Amirkabir International Journal of Modeling, Identification, Simulation & Control*, vol. 46, no. 1, pp. 19–29, 2014.

[19] L. Wang, S. U. Khan, D. Chen, J. KołOdziej, R. Ranjan, C.-Z. Xu, and A. Zomaya, "Energy-aware parallel task scheduling in a cluster," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1661–1670, 2013.

[20] G. Wu, J. Liu, M. Ma, and D. Qiu, "A two-phase scheduling method with the consideration of task clustering for earth observing satellites," *Computers & Operations Research*, vol. 40, no. 7, pp. 1884–1894, 2013.

[21] Q. Tang, S.-F. Wu, J.-W. Shi, and J. Wei, "Optimization of duplication-based schedules on network-on-chip based multi-processor system-on-chips," *IEEE transactions on parallel and distributed systems*, 2016.

[22] J. Singh and N. Auluck, "Dvfs and duplication based scheduling for optimizing power and performance in heterogeneous multiprocessors," in *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 2014, p. 22.

[23] Y. Zhang, Y. Inoguchi, and H. Shen, "A dynamic task scheduling algorithm for grid computing system," in *Parallel and Distributed Processing and Applications*. Springer, 2004, pp. 578–583.

[24] G. Wang, Y. Wang, H. Liu, and H. Guo, "Hsip: A novel task scheduling algorithm for heterogeneous computing," *Scientific Programming*, vol. 2016, 2016.

[25] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[26] D.-K. Kang, S.-H. Kim, C.-H. Youn, and M. Chen, "Cost adaptive workflow scheduling in cloud computing," in *Proceedings of the 8th International conference on ubiquitous information management and communication*. ACM, 2014, p. 65.

[27] J. Mei, K. Li, and K. Li, "Energy-aware task scheduling in heterogeneous computing environments," *Cluster Computing*, vol. 17, no. 2, pp. 537–550, 2014.