E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing

Samadi Yassir
National School of Computer science and Systems Analysis
Mohamed V University
Rabat, Morocco
yassir.samadi@um5s.net.ma

Zbakh Mostapha
National School of Computer science and Systems Analysis
Mohamed V University
Rabat, Morocco
m.zbakhg@um5s.net.ma

Tadonki Claude
Mines ParisTech-PSL
Centre de Recherche en Informatique (CRI)
Paris, France
claude.tadonki@mines-paristech.fr

***Abstract*—Cloud computing is one of the most widely spreaded platforms for executing tasks through virtual machines as processing elements. However, there are various issues that need to be addressed in order to be efficiently utilized for workflow applications. One of the fundamental issues in cloud computing is related to task scheduling. Optimal scheduling of tasks in cloud computing is an NP-complete optimization problem, and many algorithms have been proposed to solve it. Furthermore, the existing algorithms fail to either meet the user's Quality of Service (QoS) requirements such as minimizing the makespan and satisfying budget constraints, or to incorporate some basic principles of cloud computing such as elasticity and heterogeneity of computing resources. Among these algorithms, the Heterogeneous Earliest Finish Time (HEFT) heuristic is known to give good results in short time for tasks scheduling in heterogeneous systems. Generally, the HEFT algorithm yields good tasks execution time, but its drawback is that there is no load balancing. In this paper, an enhancement of Heterogeneous Earliest Finish Time (E-HEFT) algorithm under a user-specified financial constraint is proposed to achieve a well balanced load across the virtual machines while trying to minimize the makespan of a given workflow application. To evaluate the performance of the enhancement algorithm, we compare our algorithm with some existing scheduling algorithms. Experimental results show that our algorithm outperforms other algorithms by reducing the makespan and improving load balance among virtual machines.**

***Index Terms*—scientific workflows; task scheduling; data dependency; cloud computing; load balancing**

I. Introduction

Cloud computing is a kind of Internet-based computing, where virtually unlimited resources are provided on demand and as services (i.e., IaaS, PaaS and SaaS, respectively). Cloud computing has also emerged as a platform for executing scientific workflows. Under this new paradigm, services and storage resources are now delivered on a pay-as-you-go basis [1], [2], where clients only pay for resources they actually used and for the time they own them. Cloud clients are consuming and producing a huge volume of data that should be analyzed and processed, thus being considered as data-intensive experiments. This large volume of data can be found in many areas like astronomy [3], high-energy physics [4] and bio-informatics [5], etc. Thus, scientists need to analyze terabytes of data either from existing data resources or collected from physical devices. In order to manage the execution of these complex experiments, scientific workflows can be a prominent solution. A scientific workflow [8] describes the automation of scientific or general process and the set of rules (dependencies). In other words, scientific workflows are sets of elementary tasks and their dependencies. In addition, a given client is interested in reducing the makespan. So, the scheduler should efficiently fully utilize the available resources and make sure that the load is globally well balanced.

In a scheduling algorithms, scientific workflow can be viewed as a directed acyclic graph (DAG), where nodes (or tasks) represent the computation and edges represent the communication between them. The scheduler assigns a weight for each node in the DAG, which represents the computation cost, and assigns weights for edges corresponding to communication cost between nodes. In addition, scientific workflow applications have many computations and tasks that generate many intermediate large datasets with dependencies among them. So, the scheduler should also take care of precedence constraints between the set of tasks.

In its most general form, the problem of tasks scheduling of a graph onto a set of different resources is an NP-Complete problem [9]. As a result, over several years, a number of heuristic algorithms suitable for DAG scheduling on heterogeneous resources have been suggested [10] that attempt to strike a good balance between running time, complexity and schedule quality [11], but still a lot of work needs to be done to make scheduling in clouds more effective. The Heterogeneous Earliest Finish Time heuristic (HEFT) [6] has been one of the most often cited and used, with the advantage of simplicity and generating generally good schedules with a short makespan. However, HEFT algorithm lacks load balancing among the machines of the cluster. In this paper, we extend our previous work [7], which balances the load between datacentres and afterwards minimizes the overhead of data exchanges, and

propose an enhancement of HEFT algorithm. In other words, our scheduling algorithm aims to minimize the total execution time of tasks as well as to achieve a well-balanced load across all VMs in cloud environment, always obeying both budget and precedence constraints imposed by the DAG. That is, there are two objectives considered here. The first one is the minimization of the tasks execution time. The second one is to evenly distribute the workload among the virtual machines of the entire cluster.

In this paper, in order to reduce the total execution time of the workflow, we propose a scheduling approach that takes into account the variety and heterogeneity of virtual machines in a cloud computing cluster (e.g. different bandwidths, transfer rates, and processing capacities). It also takes into account the data distribution and data constraints all together within the same solution, i.e. tasks and data transfers are scheduled together by the E-HEFT. Thus, the scheduler defines the distribution of tasks and data among the virtual machines so it minimizes the total execution time and the unnecessary data transfers. Then, our scheduling algorithm is simulated using the CloudSim toolkit package [12], [13]. To evaluate the performance of the enhancement algorithm, a comparative study is done among some state-of-the-art algorithms.

The rest of the paper starts with a review of related works in section 2. In section 3, we describe the task scheduling problem, its formulation and our objective function. Section 4 presents the proposed algorithm. Section 5 assesses the applicability of our proposed solution and illustrates its performance using Cloudsim simulator. Section 6 outlines a conclusion and some future works.

II. Related Work

In this section, we conduct a survey of related work on different task scheduling techniques in cloud computing. Task scheduling or module mapping for workflows has been investigated extensively in the literature in the past decade [14].

Many task scheduling algorithms have been proposed to minimize the workflow makespan in heterogeneous environment such as Opportunistic Load Balancing (OLB) [15], Minimum Execution Time (MET) [16] and Heterogeneous Earliest Finish Time (HEFT) [6]. OLB assigns randomly each task to the machine that is expected to be available, regardless of the tasks expected execution time on that machine [17]. This algorithm aims to keep all machines as busy as possible. One advantage of OLB is its simplicity. However, OLB does not consider expected task execution times, the mappings it finds can result in very poor makespan. In contrast to OLB, Minimum Execution Time (MET) assigns randomly each task to the machine with the best expected execution time for that task, regardless of that machines availability [15], [16]. MET aims to give each task to its best machine. This can cause a severe load imbalance across machines. Earliest Finish Time (HEFT) algorithm aims to minimize the overall execution time of a DAG application in a heterogeneous environment. While being effective at optimizing makespan, the HEFT algorithm does not consider the budget constraint and load balancing among the machines of the cluster when making scheduling decisions. In [18], Chase et al. constructed analytical models to quantify the network performance of scientific workflows using cloud-based computing resources. In this work, they designed a critical-greedy algorithm to minimize the workflow end-to-end delay by defining a global budget level (GBL) parameter and preassigns tasks with the budget-level execution cost. However, this algorithm is designed for homogeneous cloud environments, where the communication time between tasks is assumed zero, which is not the case on heterogeneous cloud computing systems.

In addition to single parameter optimization scheduling, the task scheduling problem becomes more challenging when two QoS parameters (i.e., time, cost and load balancing) are considered simultaneously. The authors in [19] have studied the problem of budget- constrained schedules of bag-of-tasks problems on clouds. Authors have presented a scheduler which is used to calculate the cost and performance of a scientific workflow on multiple different clouds under budget-constraints and offer viable options to the user. The work have focused on a statistical method for estimating costs and application makespan for a given bag on different clouds. Their work however does not consider the problem of load balancing discussed in this paper. Another multi-objective algorithm named Revised Discrete Particle Swarm Optimization (RDPSO) algorithm was proposed by Wu et al. [20]. The algorithm suggested by the authors, either optimizes cost or makespan based on the budget and deadline constraints. For evaluating cost, the data transmission costs and the computation costs are taken into account. Different sets of workflows are used for experimentation and comparisons are made with the standard PSO and the BRS algorithms. This multiobjective model is efficient and the results show that the RDPSO algorithm can attain much more cost savings and reduces the makespan than the existing PSO and BRS (Best Resource Selection) algorithms.

A common drawback of the aforementioned approaches is that they do not consider the load balancing among the cluster machines. The proposed E-HEFT algorithm covers all of these deficits and presents an approach that achieves well balanced load across the machines and minimizes the makespan of a given workflow application under a user-specified budget constraint.

III. System model and Problem formulation

A. System model

We start by refining the problem definition and then present a background about HEFT algorithm. Recall that the problem we are interested in this paper is performing the tasks of a scientific workflow in a cloud computing environment while taking into account the quality of service criteria, namely the overall execution time and load balancing.

As illustrated in Fig. 1, there are three layers in the system model for the workflow scheduling problem in cloud environments [18]: the task graph layer which is comprised of tasks with precedence constraints, the resource graph layer

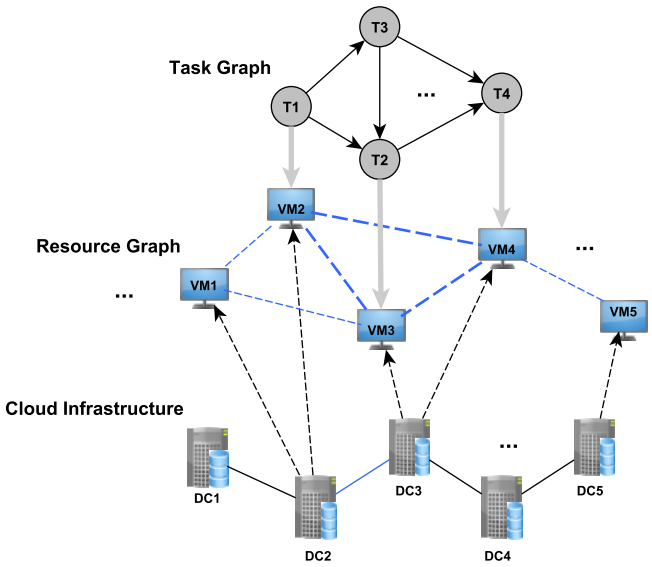


Fig. 1: System model of workflow application scheduling on cloud Computing.

which represents a network of virtual machines, and the cloud infrastructure layer consisting of a set of data centers connected by network links. According to [21] the problem of task scheduling in heterogeneous systems is finding proper assignment of tasks to machines in order to optimize some performance metrics such as resources utilization, load balancing and execution time. A popular representation of a scientific workflow application is the directed acyclic graph (DAG): $G(T, E)$, where $T = \{t_1, t_2, \cdots, t_n\}$ is the set of tasks and E is the set of directed edges between tasks. An edge e_i^j of the form (t_i, t_j) of graph G represents the precedence constraint between these tasks, case in which t_i is said to be the parent task of t_j and t_j is said to be the child task of t_i. Based on this constraint, a child task can not be executed until all of its parent tasks are completed. If there is data transmission from t_i to t_j, the t_j can start only after all the data from t_i has been received. A task without parent is called an *entry* task, denoted t_{entry}, whereas a task without child is called an *exit* task, denoted t_{exit}. A sample workflow is shown in Fig 2.

The data exchanged between the virtual machines are represented by a $n * n$ matrix which is denoted by D^{out}, where $D^{out}[i, j]$ represents the amount of transmitted data from the virtual machine $VM(t_i)$ that executes the task t_i to the virtual machine $VM(t_j)$ that executes the task t_j. We assume that the size of the output data $D_{t_i}^{out}$ produced by task t_i is known in advance. Let β be a matrix of dimension $m * m$ representing the bandwidth between the different virtual machines that execute the workflow tasks. $\beta[i, j]$ is the bandwidth between the virtual machine VM_i and the virtual machine VM_j. Note that this matrix is not necessarily symmetric. Indeed, the bandwidth between VM_i and VM_j is not necessarily the same as that between VM_j and VM_i. Fig 3 shows an example of resources graph with transfer speeds (bandwidth) between different virtual machines of a cloud environment. The transfer time between two virtual machines executing the tasks t_i and t_j, that is $VM(t_i)$ and $VM(t_j)$, is determined by the amount of transferred data and bandwidth between these virtual machines. Note that the transfer time between two tasks running on the same VM equals 0. The transfer time TT is calculated as:

$$TT(t_i, t_j) = \frac{D^{out}[i, j]}{\beta[i, j]} \tag{1}$$

Let ET be a $n*m$ matrix, where $ET(t_i, VM_j)$ is the estimated execution time of the task t_i on the virtual machine VM_j. The real execution time of a task on a given virtual machine also depends on the amount of its input and output data.

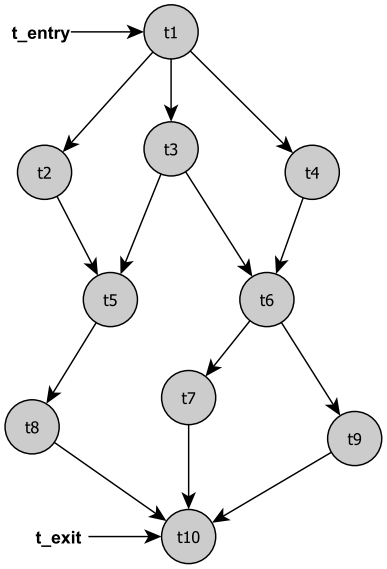


Fig. 2: An example of workflow application.

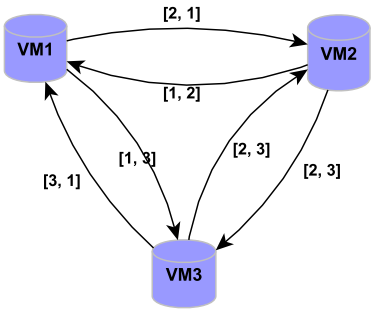


Fig. 3: Architectural model.

B. Background

Among the scheduling algorithms for heterogeneous system, the Heterogeneous Earliest Finish Time (HEFT) algorithm has been one of the most frequently cited and used because of both its simplicity and good performance [6]. HEFT is a natural extension of the classical list scheduling algorithm for homogeneous systems to cope with heterogeneity, outperforms other comparable algorithms in terms of minimization of the execution time.

There are two phases for the algorithm. The prioritizing phase for giving a priority to each task and a machine selection phase for selecting a suitable machine that minimizes the execution time. If two or more tasks have equal priority, then the tie is resolved by selecting a task randomly. The last phase is repeated until all tasks are scheduled on suitable machines. This algorithm has relatively low complexity and is very efficient when compared to other algorithms. Thus, HEFT assigns first a weight to each node and edge of the DAG based on the average computation and communication costs, respectively [22]. $Rank_u$ represents the length of the longest path from task t_i to the exit node, including its computational. It is recursively defined as:

$$rank_u(t_i) = \overline{W}_i + \max_{t_j \in succ(t_i)} (\overline{c}_{i,j} + rank_u(t_j)) \quad (2)$$

Where $succ(t_i)$ is the set of immediate successors of the task t_i, $\overline{c}_{i,j}$ is the average communication cost of edge e_i^j, and $\overline{W}_i$ is the average execution cost for task t_i. Then, the graph is traversed upwards and $rank_u$ value is assigned to all nodes. For the exit task, $rank_u(t_{exit})$ = $\overline{W}_{exit}$. Tasks are then scheduled, in descending order of their rank value, on the machine which gives the smallest estimated finish time.

C. Objective function

Before presenting the objective function, it is necessary to define the ST and FT attributes, which are derived from a given partial schedule. $ST(t_i)$ and $FT(t_i)$ are the earliest starting time and the earliest finishing time of task t_i, respectively. For the input task t_{entry}:

$$ST(t_{entry}) = 0 \quad (3)$$

$$FT(t_{entry}) = ET(t_{entry}, VM_j)) \quad (4)$$

For the other graph tasks, the values of ST and FT are calculated recursively, starting from the entry task. In order to compute the FT of a task t_i, all immediate predecessor tasks of t_i must have been assigned and scheduled with the consideration of the transfer time as shown in equations 5 and 6:

$$FT(t_i) = ST(t_i) + \min_{VM_j}\{ET(t_i, VM_j)\} \quad (5)$$

$$ST(t_i) = \max_{t_j \in pred(t_i)}\{FT(t_j) + TT(t_j, t_i)\} \quad (6)$$

Where $pred(t_i)$ is the set of predecessors of the task t_i.
We propose a scheduler S defined by a set of resources, a set of tasks to schedule, the total execution time, and the total execution cost. $S = (T, R, ET(t_i, VM_j), PC(t_i, VM_j))$

- $T = \{t_1 \cdots t_n\}$ is the set of tasks compose the Workflow W.
- $R = \{VM_1 \cdots VM_m\}$ is the set of virtual machines.
- $ET(t_i, VM_j)$ is the total execution time of the task t_i on VM_j.
- $PC(t_i, VM_j)$:(Processing cost) is the total execution cost of the task t_i on VM_j, which equals:

$$PC(t_i, VM_j) = Cost(VM_j) * ET(t_i, VM_j) \quad (7)$$

Where $Cost(VM_j)$ is the cost of data processing per hour in the virtual machine VM_j, and $ET(t_i, VM_j)$ is the execution time of the task t_i on VM_j.
The objective function of the overall execution time can be defined as follows:

$$Makespan = \min\{FT(t_{exit})\} \quad (8)$$

Therefore, the objective function is to efficiently assign tasks of a given workflow application to virtual machines such that its makespan is minimized.

Subject to:

$$\sum_{t_i \in T} PC_{t_i} \leq Budget(W) \quad (9)$$

$$\forall t \in T \mid \exists!\, VM_j \in R\ ,\ \mu(t_i) = VM_j \quad (10)$$

$$\sum_{t_i \in r_j} RAM(t_i) \leq RAM(VM_j) \quad (11)$$

$$\sum_{t_i \in r_j} IS(t_i) + OS(t_i) \leq SC(VM_j) \quad (12)$$

The associated constraints are given in Eq. 9 to 12.

Constraint 9 ensures that the workflow processing cost must be less or equal to the budget dedicated to this workflow.

Constraint 10 ensures that each task is scheduled on a single virtual machine. Where μ is a matching function between the set of tasks and the set of virtual machines.

Constraint 11 ensures that the total RAM required by all tasks running on VM_j should be less than the available RAM on this virtual machine.

Constraint 12 ensures that the total disk space (input size and output size) required by all tasks assigned to VM_j should be less than the storage capacity available on VM_j.

IV. TASK SCHEDULING ALGORITHM

In this section, an algorithm to resolve the above mentioned problems is proposed. The algorithm is consisted of four phases. In the first phase, we specify the load threshold of each machine based on both processing speed and storage capacity. In the second phase, we define the datasets dependencies in order to cluster the datasets into different datacenters based on these dependencies. In the third phase, we group the set of tasks by level, then we assign a value (Rank) for each task based on the algorithm HEFT. Finally, we schedule these tasks on its best machine bases on Matching Game theory.

A. Attribution of VMs threshold phase

The threshold represents the utilization storage percentage for each machine that should not be exceeded. We will attribute a threshold for each machine on the cluster based on its storage capacity and data processing speed. For quantifying and evaluating the performance of each machine for the scientific workflow execution, we implement the HPC Challenge (HPCC), which is benchmark designed to give a picture of overall supercomputer performance including floating point computer power, memory subsystem performance and global network issues. It consists of seven separate workloads building on the success of the TOP500 list Linpack HPL based workload [23].

In this paper, we first use the HPL (High-Performance Linpack Benchmark) [24] measurement as a performance value to know the real datacenter performance on the HPCC benchmark. Then, we record the execution time of each machine based on the output of HPL benchmark. After that, the shortest execution time is used as a reference to normalize the execution time measurements, in such a way we attribute randomly the highest threshold to the machine that has the shortest execution time. Then, the normalized values are used to distribute an appropriate threshold to each machine on the cluster. Thus that a high-speed machine will handle tasks more than low-speed machine.

B. Specifying datasets dependencies phase

In our strategy, we initially adapted a dependency matrix to represent the affinity between the datasets. Cloud workflows can be complex, the execution of one task might require many datasets. Furthermore, one dataset might also be required by many tasks. So, we say that the datasets d_i and d_j have a dependency if there are tasks that will use d_i and d_j together. The dependency degree is the total number of tasks that use both d_i and d_j. We use $dependency_{ij}$ to denote the dependency between the datasets d_i and d_j and the quantity of this dependency is the number of tasks that use both d_i and d_j . It can be calculated by counting the tasks in common between the task sets of d_i and d_j, which are denoted as T_i and T_j.

$$dependency_{ij} = card(T_i \cap T_j) \tag{13}$$

The dependency matrix is defined by: $DM = \{C_{i,j}, \quad (1 \leqslant i, j \leqslant m)\}$. For the elements in the diagonal of DM, each value means the number of tasks that will use this dataset. DM is a symmetric matrix of dimension $m * m$, where m is the total number of existing datasets. We store the datasets that have the highest dependency degree in the same datacenter based on the total number of tasks that use these datasets as shown in formula 14.

$$Max(dependency_{ij}) \tag{14}$$

As an example in Fig 4 (left side), we assume that we have four datasets (d_1, d_2, d_3 and d_4), and we have also five tasks to be processed (t_1 to t_5), the dependency matrix DM is shown in Fig. 4 (right side).

C. Task sorting phase

The objective of this phase is to define the different levels of a given workflow (DAG). Therefore, tasks belonging to the same level can be execute concurrently. This is because two tasks at the same level do not exchange data (or they are not linked by precedence constraints). Then, a weight is assigned to each node and edge of the graph, depending on processing length of each task in all clusters machines and communication length between tasks. Then, the graph is traversed upwards and a value (rank) is assigned to each node at each level. The level 1 and the last level contains respectively the exit task t_{exit} and the entry task t_{entry}. Tasks are sorting in descending order in the list based on their rank value.

For calculating rank value to give priority to sort tasks to be executed, we use the rank function of the HEFT algorithm. **Algorithm 1** shows the pseudo code of the ranking method of HEFT algorithm.

Algorithm 1: HEFT ranking algorithm

```
for each task t_i in the graph (DAG) do
    calculate average execution time on all VMs
    if task t_i is the last task then
        rank value of t_i = its average execution time
    end
    else
        rank_u(t_i) =
          W̄_i + max_{t_j ∈ succ(t_i)} (c̄_{i,j} + rank_u(t_j))
    end
end
```

D. Virtual machine selection phase

The final phase of the proposed approach is to choose an "optimal" virtual machine to run each of the workflow tasks. We use game theory [25], which is a theoretical approach of the game that provides appropriate solid mathematical tools to study the considered issue and to allocate resources to the workflow tasks on the basis of low cost, load balancing and improved tasks execution time. So far, we have modeled the problem of virtual machines selection as a many-to-one matching game where the players are the tasks and the VMs. Each task has the right to choose one VM. By contrast, the

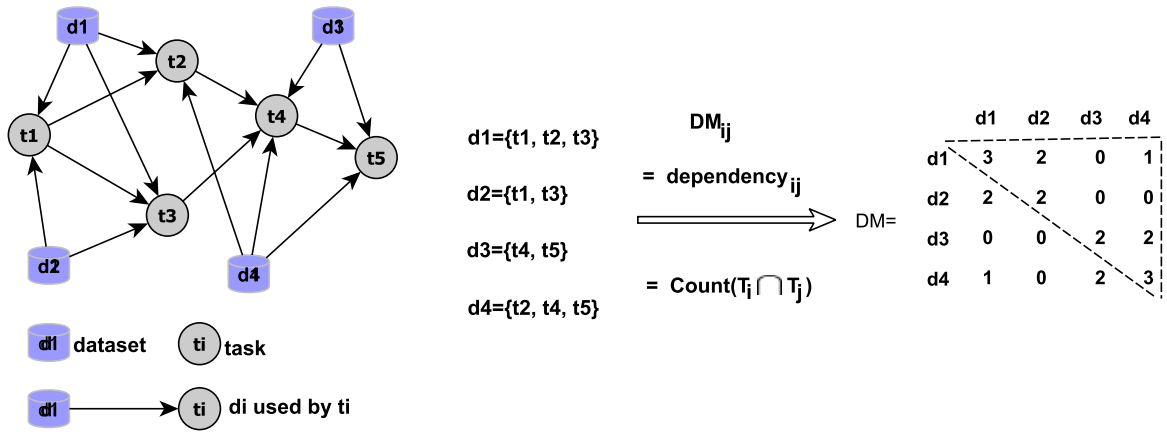


Fig. 4: An example of dependecy matrix.

VMs can host one or more task respecting the maximum number of quota q_{vm}.

This choice based on the preferences list for all tasks and virtual machines. The preference lists of each task on the workflow $PL(t_i) = \{VM_{j*}, \cdots, \}$ contains the set of VMs, which are selected like potential sites. The elements in preference list $PL(t_i)$ are put in an ascending order according to the processing power of datacenters, their cost processor, memory and storage capacity. So, that the one having the best values of these parameters depending on the requirement of the client has the highest rank. On the other hand, The preference list of the j^{th} virtual machine VM_j is $PL(VM_j) = \{t_{i*}, \cdots, \}$. These tasks are sorted in ascending orders according to different parameters. The tasks are initially prioritized according to the impact of these tasks on the use of the virtual machine in terms of CPU, RAM, etc, such that one having least impact has highest rank. Thus, the key factor for prioritizing tasks is their CPU utilization ratio. As a solution of virtual machines selection phase we used the matching algorithm as shown in **algorithm 2** based on the deferred acceptance algorithm introduced in [26], which is a well-known approach to solving the standard matching games.

An overview of our approach is given by **algorithm 3** which includes all the aforementioned phases.

Algorithm 2: Task-VM matching algorithm

Input: Tasks' preference lists, VMs' preference lists .

while *($\exists$ a free task t_i that still has a VM_j to propose to)* **do**
  Task t_i proposes to all VMs on its preference list.
  if *($q_{vm_j} \geq 0$)* **then**
    t_i is assigned to the waiting list of VM_j
    else
      Compare t_i with the current tasks on the waiting list for the VM_j and reject the task that has least preference.
    end
  end
  The rejected tasks re-apply to their next best choice.
  Each VM update its waiting list based on its preference list, and rejects the rest.
end

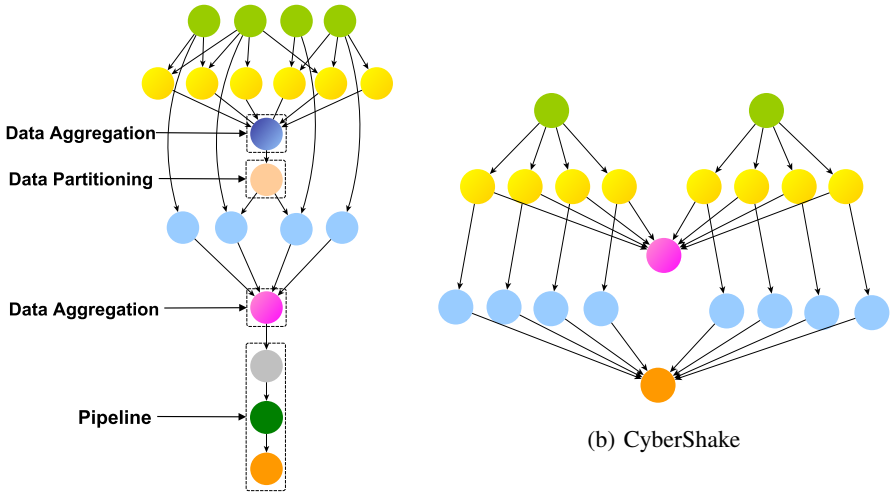
Output: Every t_i is on a waiting list of one of the VMs, and,thus, we have convergence to a stable matching.

V. Performance evaluation

In this section, we present the experiments conducted in order to evaluate the performance of the proposed E-HEFT algorithm. The performance metrics, real world workflows used in our experiments, experimental setup and experimental results are shown in the following subsections.

A. Experimental setup

To evaluate a workflow scheduling algorithm, we should measure its performance on some sample workflows. So, there is a need for a good simulator for experimental purposes. One such a simulator is CloudSim [27], which has been widely adopted for the modeling and the evaluation of cloud-based solutions. Particularly, CloudSim provides a generic broker modeled as a class named DataCenterBroker, we extended this class to support DAG-structured workflows and to model the behavior of this component and its particular placement policies. On the other hands, CloudSim is an extensible simulation toolkit that enables modeling and simulation of cloud computing systems and application provisioning environments. The Cloudsim could implement generic application provisioning techniques that can be extended easily with limited efforts. According to the implementation using Cloudsim, the VMs are considered as the cloud resources and Cloudlets as tasks/jobs. We ran our experiment on one datacenter that contains 20 VMs, and the configuration of virtual machine is shown in Table I. All experiments are performed on a Pentium(R) Dual-Core processor with a speed of 2.8GHz and memory of 8GB.



(a) Montage

(b) CyberShake

Fig. 5: The structure of two realistic scientific workflows.

Algorithm 3: Task scheduling algorithm

Input: Workflow W, set of virtual machines VMs
Attribute a threshold for each Virtual machine
Defining Datasets Dependencies() by using the dependency matrix
Read the DAG
Break W into set of levels L by traversing the DAG upward.
$K \leftarrow 1$; // first level
while $(K \leq L)$ **do**
  for *each task in level k* **do**
    calculate rank value applying the HEFT ranking algorithm
  end
  Sort the tasks in a scheduling list by descending order of $rank_u$ values
  Check tasks and virttual machines Preferences Lists(PL)
  Assign task t_i to the best virtual machine VM based on preferences list.
  Apply Task-VM matching algorithm()
  Remove t_i from the level K
  $K \leftarrow K + 1$
end
return Set of VMs with the mapping tasks.

TABLE I: Initial parameters of virtual machine

parameters of virtual machine	Value
Number of VM cores	4
Mips of each core	2000MI
RAM	1GB

B. Experimental Workflows

In order to make the results more realistic, it is important to conduct experiments using workload traces from a real system. Bharathi et al. [28] investigate the characterizations of six realistic workflows from diverse scientific applications, two of which are used in our experiments, which are Montage for astronomy and CyberShake for earthquake science. Montage [29] is an astronomy application that was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to construct large image mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. The CyberShake [30] workflow is a seismology application that calculates Probabilistic Seismic Hazard curves for geographic sites in the Southern California region. Four different sizes of these workflows are chosen, small (around 30 tasks), medium (around 100 tasks), large (1000 tasks) and x-large (10000 tasks). Fig. 5 shows the approximate structure of a small instance of each workflow used in our experiments. For each workflow, tasks with the same color belong to the same type. It can be seen that these two workflows have different structures, data and computational requirements.

C. Performance metrics

The scheduling problem aims to minimize the total execution time of tasks as well as to achieve a well-balanced load across all VMs in cloud. That is, there are two factors considered here. The first one is the minimization of the tasks completion time. The second one is the evenly distribution of the workload among virtual machines. So, the two metrics used to evaluate the scheduling approaches are makespane and degree of imbalance (DI).

- **Makespan**:is the maximum needed time to complete the execution of all tasks.

$$makespan = \max_{i \in \{1,\cdots k\}} \{FT(t^i_{exit})\} \tag{15}$$

Where t^i_{exit} , $i \in \{1, \cdots k\}$ are the independent tasks that should to run in parallel to complete the execution of the workflow.

- **Degree of imbalance (DI):** to measure degree of imbalance (DI) of all virtual machines on the cluster, we use the following formula:

$$DI = \frac{L_{max} + L_{min}}{L_{avg}} \tag{16}$$

Where L is a load of a VM on the cluster, and equals:

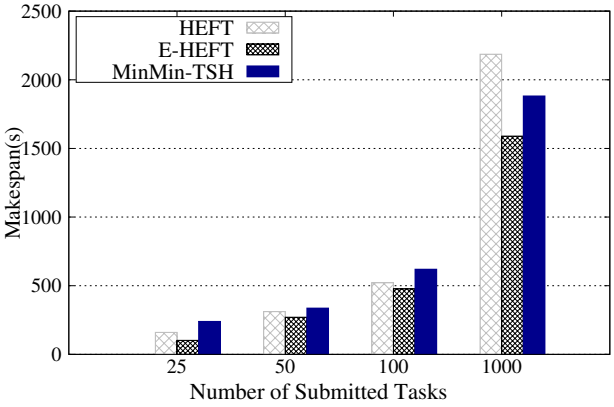
$$L = \frac{tasklength(VM_j)}{C(VM_j)} \tag{17}$$

Where $tasklength(VM_j)$ is the total length of tasks submitted to the VM_j and $C(VM_j) = pe_{numj} * pe_{mipsj}$ is the capacity of a VM_j, with pe_{numj} is the number of processors in VM_j and pe_{mipsj} is the million instructions per second of all processors in VM_j.

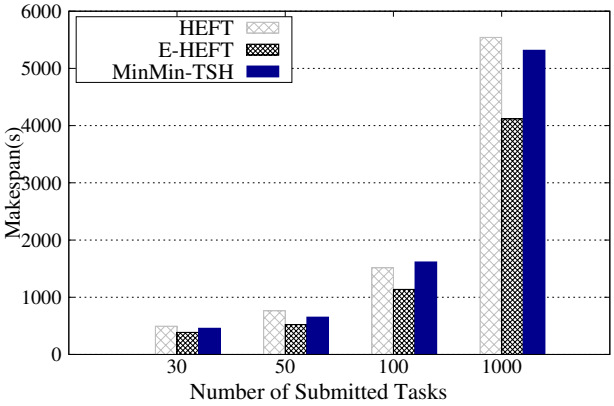
D. Result analysis

We compared our E-HEFT algorithm with the Heterogeneous Earliest Finish Time (HEFT) [6] and MinMin Task Scheduling Heuristic (MinMin-TSH) [31]. The MinMin-TSH algorithm is a widely-used scheduling algorithm. This is a task-based greedy algorithm that allocates each ready-to-run task to a machine based only on the local information of that task. There are two phases in MinMin-TSH algorithm. In the first phase it finds the minimum execution time of all tasks. Then in the second phase it select the tasks with the minimum execution time among all the tasks and assign that task on that resource. The same procedure is repeated by MinMin-TSH algorithm until all tasks are scheduled. We have chosen MinMin-TSH [31] algorithm and HEFT [6] because these heuristics run in polynomial time, produce efficient schedules and have been used as baselines in many related works.

The makespan results (the average of 5 executions) for the Montage and Cybershake DAGs are shown in Fig 6, while Fig. 7 shows the comparison of degree of imbalance between E-HEFT, HEFT and MinMin-TSH Algorithms. According to the experimental results in Fig 6, it can be seen that the total running time of scientific workflows lengthens with increase of the number of submitted tasks. It is clearly evident from the graph that E-HEFT is more efficient when compared with other two algorithms based on both workflows. According to Montage workflow, our proposed algorithm (E-HEFT) outperforms HEFT and MinMin-TSH algorithms in terms of the average makespan of the submitted tasks by 21.37%, and 28.98% respectively as shown in Fig 6a. On the other hand, The E-HEFT showed an average improvement of 26.07% in relation of HEFT and 21.93% in relation of MinMin-TSH based on Cybershake workflow as shown in Fig 6b. It is worth mentioning that makespan is closely pertinent to the volume of data movement for data intensive workflow. So, we can state that those improvements variations are mainly due to the volume of data that is transferred among activities in each workflow. Compared with two other algorithms, our algorithm incurs less data movement during scientific workflow execution due to the datasets dependencies specified in the second phase of our algorithm.



(a) Montage workflows



(b) CyberShake workflows

Fig. 6: Average makespane of the three algorithms based on Montage and CyberShake workflows.

From Fig 7, it can be seen that our algorithm (E-HEFT) can effectively balance the load among all virtual machines and has a lesser degree of imbalance when compared with other two algorithms. This is due to the attribution of VMs utilization threshold specified in the first phase of our algorithm.

VI. Conclusion and Future Work

Task scheduling is one of the main issues in cloud computing. Efficient task scheduling is essential for achieving good system performance. In this paper, we have proposed an enhancement of Heterogeneous Earliest Finish Time algorithm (E-HEFT) for achieving tasks scheduling with load balancing among virtual machines. The main idea of the enhancement algorithm is first attribute a utilization threshold for each VM that should not to be exceeded. Then, it specifies the datasets dependencies to store the dependent datasets on the same datacenter. After that, the set of tasks are sorted based on the rank value of the HEFT algorithm. Finally, we have used matching game theory for selecting virtual machines to execute the submitted tasks. To evaluate the performance of

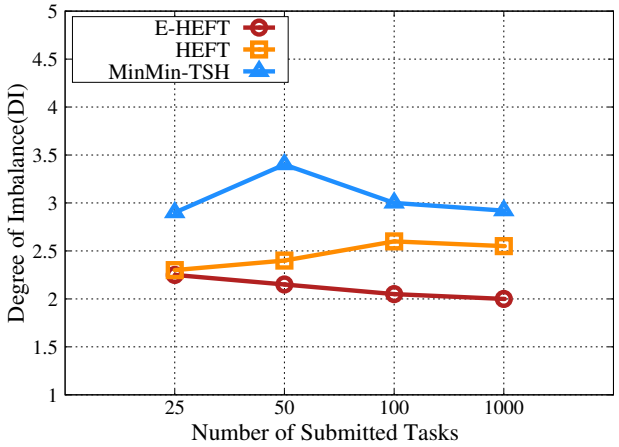


Fig. 7: Comparison between algorithms (E-HEFT, HEFT and MinMin-TSH) based on degree of imbalance.

our algorithm, we have compared our proposed algorithm with some existing techniques based on the execution time (i.e., makespan) and degree of imbalance (DI). The experimental results show that our algorithm not only balances the load, but also takes less time to execute tasks compared to the two other algorithms. In the future, we plan to extend the proposed work to take into account others criteria like execution cost. We also aim to improve our algorithms to run in real cloud environments, such as Amazon so that it can be utilized for deploying applications in real life environments.

REFERENCES

[1] L. Youseff, M. Butrico, and D.D. Silva, "Toward a unified ontology of cloud computing," Grid Computing Environments Workshop, 2008, pp. 110.

[2] D. Oliveira, F.A. Baio, and M. Mattoso, "Towards a taxonomy for cloud computing from an e-Science perspective," Cloud Computing. Springer, London, 2010. p. 47-62.

[3] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," rid Computing: Second European AcrossGrids Conference (AxGrids 2004). 2004. p. 11-26.

[4] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, and E.A. Lee, "Scientific workflow management and the Kepler system," Concurrency and Computation: Practice and Experience, vol. 18, no 10, pp. 10391065, 2005.

[5] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," Bioinformatics, vol. 20, no. 17, p. 3045-3054, 2004.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260274, 2002.

[7] Y. Samadi, M. Zbakh, and C. Tadonki, "Graph-based Model and Algorithm for Minimizing Big Data Movement in a Cloud Environment", Int. J. High Performance Computing and Networking., in press.

[8] J. Yu, and R. Buyya, "A taxonomy of workflow management systems for grid computing," J. Grid Comput, vol. 3, no. 3-4, p. 171-200, 2005.

[9] M. L. Pinedo, "Scheduling: theory, algorithms, and systems," Springer Publishing Company, 2008.

[10] L. F. Bittencourt, R. Sakellariou, and E. R. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," Parallel, Distributed and Network-Based Processing (PDP), 18th Euromicro International Conference on. IEEE, 2010. p. 27-34.

[11] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," ACM Computing Surveys, vol. 31, no. 4, pp. 406471, 1999.

[12] R. Buyya, R. Ranjan, and R.N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudSim toolkit: challenges and opportunities," High Performance Computing & Simulation. HPCS'09. International Conference on. IEEE, 2009. p. 1-11.

[13] R. N. Calheiros, R. Ranjan, C. A. F. De Rose, and R. Buyya, "CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services," technical report, GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory, the University of Melbourne, Australia, 2009.

[14] R. Bajaj and D. Agrawal. "Improving scheduling of tasks in a heterogeneous environment," IEEE Trans. Parallel Distrib. Syst., vol. 15, no 2, p. 107-118, 2004.

[15] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," 7th IEEE Heterogeneous Computing Workshop (HCW'98), pp. 79-87, 1998.

[16] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," in 7th IEEE Heterogeneous Computing Workshop (HCW'98), pp. 184-199, 1998.

[17] R. F. Freund and H. J. Siegel, "Guest editor's introduction: Heterogeneous processing," Computer, vol. 26, no 6, p. 13-17, 1993.

[18] C.Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," IEEE Trans. Cloud Comput, vol. 3, no 2, p. 169-181, 2015.

[19] A. M. Oprescu, T. Kielmann, and H. Leahu, "Budget estimation and control for bag-of-tasks scheduling in clouds," Parallel Processing Letters, vol. 21, no. 2, pp. 219243, 2011.

[20] Wu Z, Ni Z, Gu L, Liu X, "A revised discrete particle swarm optimization for cloud workflow scheduling," Computational Intelligence and Security (CIS), International Conference. IEEE. 2010, p. 184-188.

[21] D. M. Abdelkader, and F. Omara, "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system," Egyptian Informatics Journal, vol. 13, no 2, p. 135-145, 2012.

[22] H. Zhao, and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," Euro-Par Parallel Processing, 2003, p. 189-194.

[23] Top 500 list. https://www.top500.org/lists/ [November 2017].

[24] T. Y. Chao and H. C. Lung, "Implementation of a performance-based loop scheduling on heterogeneous clusters," International Conference on Algorithms and Architectures for Parallel Processing. Springer, Berlin, Heidelberg, 2009. p. 44-54.

[25] A. E. Roth, "Deferred acceptance algorithms: history, theory, practice, and open questions," Int. J. Game Theory, vol. 36, pp. 537-569, 2008.

[26] D. Gale, and L. S. Shapley, "College admissions and the stability of marriage," The American Mathematical Monthly, vol. 69, no 1, p. 9-15, 1962.

[27] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities," High Performance Computing & Simulation, HPCS'09. International Conference on. IEEE, 2009. p. 1-11.

[28] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of scientific workflows," Workflows in Support of Large-Scale Science. Third Workshop on. IEEE, 2008. p. 1-10.

[29] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M. Su. "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand," Optimizing Scientific Return for Astronomy through Information Technologies. International Society for Optics and Photonics, 2004. p. 221-233.

[30] R. Graves, T. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi, "CyberShake: A Physics-Based Seismic Hazard Model for Southern California," Pure and Applied Geophysics, vol. 168, no 3-4, p. 367-381, 2011.

[31] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy," Task scheduling strategies for workflow-based applications in grids," Cluster Computing and the Grid, CCGrid 2005. IEEE International Symposium on. IEEE, 2005. p. 759-767.