Egyptian Informatics Journal 19 (2018) 33–55

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Full length article

An extended Intelligent Water Drops algorithm for workflow scheduling in cloud computing environment

Shaymaa Elsherbiny [a], Eman Eldaydamony [a], Mohammed Alrahmawy [b,*], Alaa Eldin Reyad [c]

[a] Department of Information Technology, Faculty of Computer and Information, Mansoura University, Egypt
[b] Department of Computer Science, Faculty of Computer and Information, Mansoura University, Egypt
[c] Department of Information System, Faculty of Computer and Information, Mansoura University, Egypt

ARTICLE INFO

Article history:
Received 19 September 2016
Revised 17 May 2017
Accepted 2 July 2017
Available online 11 July 2017

Keywords:
Cloud computing
Scheduling algorithms
Resource management
Intelligent Water Drops
Workflow scheduling
Natural-based algorithms

ABSTRACT

Cloud computing is emerging as a high performance computing environment with a large scale, heterogeneous collection of autonomous systems and flexible computational architecture. Many resource management methods may enhance the efficiency of the whole cloud computing system. The key part of cloud computing resource management is resource scheduling. Optimized scheduling of tasks on the cloud virtual machines is an NP-hard problem and many algorithms have been presented to solve it. The variations among these schedulers are due to the fact that the scheduling strategies of the schedulers are adapted to the changing environment and the types of tasks. The focus of this paper is on workflows scheduling in cloud computing, which is gaining a lot of attention recently because workflows have emerged as a paradigm to represent complex computing problems. We proposed a novel algorithm extending the natural-based Intelligent Water Drops (IWD) algorithm that optimizes the scheduling of workflows on the cloud. The proposed algorithm is implemented and embedded within the *workflows* simulation toolkit and tested in different simulated cloud environments with different cost models. Our algorithm showed noticeable enhancements over the classical workflow scheduling algorithms. We made a comparison between the proposed IWD-based algorithm with other well-known scheduling algorithms, including MIN-MIN, MAX-MIN, Round Robin, FCFS, and MCT, PSO and C-PSO, where the proposed algorithm presented noticeable enhancements in the performance and cost in most situations.

1. Introduction

Cloud computing is emerging as a hot topic that gains a lot of attention in both academia and industry as it enables data and services to be stored remotely but can be accessed from everywhere. Every day many companies and organizations transfer their data and applications to the cloud, due to its flexible and dynamic infrastructures, which offers QoS guaranteed computing environments in addition to configurable software services that can be rapidly provisioned and released with minimal management effort. Hence, large number of researcher~~s~~ focuses on how to improve the performance and reduce the obstacles that may lead to degradation in the performance or unsafe use. Therefore, active research fields in cloud computing includes security, scheduling, privacy and policy, cloud storage, cloud performance, deployment, energy management, etc.)

Task scheduling is an important issue which greatly influences the performance of cloud computing environment. Scheduling is the process of mapping tasks to available resources on the basis of tasks' characteristics and requirements. Efficient scheduling is essential requirement for the cloud, as it can optimize the use of the cloud resources to provide highly efficient services with high quality. As more users begin to use clouds for deploying complex applications and store remote data, there is importunate need to have strong scheduling algorithms that can allocate tasks to data centers.

The focus in this paper is on the problem of workflow scheduling in the cloud computing environment. The problem of workflow scheduling in the cloud is a special case of the general problem of

* Corresponding author.
E-mail address: mrahmawy@gmail.com (M. Alrahmawy).
Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2017.07.001

task scheduling in the cloud, where some precedence execution dependencies exist among some of the schedulable tasks; hence, workflow scheduling has, in addition to its own dependencies characteristics, all the characteristics and features of the task scheduling problem.

There are many existing workflow scheduling algorithms for heterogeneous computing environments, these algorithms implement suitable compromise, or apply combination of scheduling algorithms according to different applications requirements. However, these algorithms have difficulties in being directly applied to the Cloud environments; as the Cloud, unlike traditional heterogeneous environment and the Grid, offers extremely large-size resource pools; however, using these pools is restricted by the complex processing cost scheme offered by the cloud provider. This adds an extra objective to be satisfied by the cloud scheduler; this objective is to optimize resource utilization of the rented cloud resources in order to minimize the processing cost of executing the workflows [1]. In this paper, we propose a cloud workflow scheduling algorithm based on the meta-heuristic IWD algorithm in order to satisfy the required objectives of minimizing both makespan and the cost of executing workflows on the cloud. In order to evaluate the proposed scheduler, we used workflowSim simulation environment to use the proposed algorithm to execute a set of common workflow in different simulated data centers with different configurations and cost models.

The rest of the paper is organized as follows: In Section 2, background and concepts of workflow scheduling in cloud computing are presented. In Section 3, the basic Intelligent Water Drops (IWD) algorithm is overviewed. In Section 4, some of the related work is presented. In Section 5, we present our proposed workflow scheduling algorithm and explain its different phases; then, in section 6, we give a simple detailed case study to explain the operation of the proposed algorithm. Then a set of performance and cost evaluation experiments and comparisons of the proposed algorithm with other algorithms are presented in Section 7, using various datacenters configurations and cost models. Finally conclusion and future work is presented in section 8.

2. Workflow scheduling in cloud computing

A Workflow is an attractive paradigm that helps scientists orchestrate complex, multistep simulations and analyses. Workflows are commonly used in distributed computing environments, e.g. grids and clouds for their powerful capabilities in modeling a wide range of applications, including scientific computing, multiprocessors system [2], multi-tier Web [3], and big data processing applications [4]. The problem of workflow scheduling in cloud has become an important research topic due to the development of cloud technology.

A Workflow is represented graphically using *Directed Acyclic Graph* (DAG) to reflect the interdependency among the workflow's tasks, where the nodes represent computation tasks to be executed on the available resources and directed edges represent data flow or control flow dependencies among the tasks. In general, the execution time of a task on a specific resource is inversely proportional to the speed of that resource; therefore, the optimized mapping of individual workflow tasks to the available computational cloud resources is an essential requirement in the cloud scheduler. In addition to the extensive workflows execution time requirements, due to the processing of significant amounts of data, workflow applications often involve dependencies that exist amongst tasks, which enforce the scheduler to respect these precedence requirements. The emergence of cloud computing has introduced a utility-type market model, where computational resources with varying capacities can be procured on demand, in a pay-per-use fashion. In general, the two most important objectives of workflow schedulers are the minimization of both cost and makespan. The cost involves not only the computational costs incurred from processing individual tasks, but also data transmission costs, where potentially large amounts of data must be transferred between compute and storage sites.

The problem of mapping various tasks of services to a set of resources is categorized as an NP-hard problem [5]. Finding solutions to NP-hard problems, using known algorithms is impractical, which means that it is difficult to build an optimum scheduler that works with a reasonable computation speed. In general, NP-hard problems can be solved by enumeration method, heuristic method or approximation method. However, building an optimum workflow scheduler using enumeration method requires first building all the possible schedulers and comparing them one by one to select the best one, which is not feasible for cloud workflow scheduling, as there are exhaustive number of possible schedulers due to the large number of workflows to be processed. Hence, schedulers built based on heuristics can be a suboptimal method to find reasonably good schedulers that are reasonably fast. For that, we propose a scheduler that extends the meta-heuristic IWD algorithm, to schedule tasks connected to each other in workflows and we evaluate this scheduler using the WorkflowSim simulator, which has been chosen as it provides a framework that takes into consideration heterogeneous system overheads and failures and it supports widely used workflow optimization techniques such as task clustering.

3. Intelligent Water Drops (IWD) algorithm

In recent years, IWD had been used in many fields to solve optimization and complex scientific problems such as Travelling salesman problem (TSP [6]), Job shop scheduling [7], Steiner tree problem [8], Code coverage [9], Graph Coloring [10], Optimizing routing protocol [11], Multidimensional Knapsack problem (MKP) [12], Vehicle routing problem [13], Air Robot Path Planning [14], Automatic multilevel three sholding using a modified Otsu's criterion [15], Optimal data aggregation tree in wireless sensor networks [16], Economic Load Dispatch [17], etc.

IWD algorithm *was first introduced by Shah-Hosseini, H. in 2007* [6]. The idea of IWD algorithm is inspired by the flow of natural water drops in lakes, rivers, and seas, where a water stream goes through an optimum path to reach its final destination, which is often a lake or a sea. In their way to the destination, the water drops react with the surrounding environment (river beds) and influence it. The river beds can be changed by the water drops and they can influence directions of the water drops. The gravity forces the water drops to move toward the destination. If there are no barriers or obstacles, the moving of water drops goes in a straight path to the destination. But actually, in real scenarios, there are different types of obstacles like twists, rocks and turns. In the original IWD algorithm, the Natural water drops have two properties: the velocity of the water drops and the amount of soil in their path. The water drops' soil collection and movement is negatively affected by the high amount of soil in their path, while they move faster along the path with less soil and can attain a higher speed and erode more soil from that path. The velocity of the water drops enables them to transfer soil from one place to another. More soil from the river beds can be gathered and transferred by faster water drops, where the velocity of the water drops is affected by the path properties.

In the IWD algorithm, the water drops move from the source to the destination in finite-length time steps. The water drops' velocity proportionally increases nonlinearly with the inverse of the soil of the path it goes through. Also, soils of the water drops increase

as they collect some soil from the path; this increase is inversely proportional to their travel time, where the travel time duration depends on both the velocity of the stream and the distance travelled. The amount of soil in the path is a major concern when deciding the chosen path to go through; where water drops always prefer the easier path, i.e. the path with less soil. The IWD algorithm constructs probabilistic solutions for the best path(s), where the parameters of the algorithm are updated iteratively in order to converge to high-quality solutions. Optimum approximation algorithms are used for finding approximate solutions to optimized solution. Authors of [18] identified three reasons for the significance of the IWD algorithm: (I) It converges to the solution faster than other techniques, (II) It converges to high quality solutions using average values, (III) it is flexible with the dynamic environment and incorporates pop-up threats easily.

4. Related work

Extensive research has taken place on the scheduling of workflow applications onto distributed resources in cloud computing. In these studies, several algorithms have been used for scheduling, and these algorithms apply different scheduling policies and parameters. Among those, there are various workflow scheduling algorithms. For example, authors of [19] proposed a scheduler that uses Ant Colony Optimization (ACO), to satisfy all user-specified QoS constraints, this scheduler uses a set of heuristics and workflow experiments to calculate the pheromone values. In the scheduler proposed in [20], the authors used Particle Swarm Optimization (PSO) for scheduling workflows on cloud resources; the algorithm takes into account both the computation cost and the data transmission cost. The Immune Particle Swarm Optimization (IPSO) is used in [21], IPSO is a model of grid task scheduling that addresses multi-objective optimization problems of task scheduling in dynamic and heterogeneous grid environments according to an objective function based on the satisfaction rate. In [22], a Multiple QoS Constrained Scheduling Strategy of Multiple Workflows (MQMW) was proposed; this strategy can schedule multiple workflows that start at any time and the QoS requirements are taken into account. Authors in [23] presented another algorithm based on the PSO, this algorithm takes both data transmission cost and computation cost into account to schedule applications among cloud services; they conducted their experiments with a set of workflow applications in which they vary data communication costs and computation costs according to a cloud price model. This scheme showed a great performance, where it achieves much more cost savings and better performance on both makespan and cost optimization. The SHEFT workflow scheduling algorithm was presented in [24]. The preliminary experiments showed that it does not only outperform several representative workflow scheduling algorithms in optimizing workflow execution time, but also it enables resources to scale elastically during workflow execution.

Authors in [25] proposed a market-oriented hierarchical scheduling, which considers and optimizes both the Task-to-Service assignment, and the Task-to-VM assignment in local cloud data centers. The Cat Swarm Optimization (CSO) was used by the authors of [26] for workflow scheduling in the Cloud. The algorithm aims at reducing wastage of energy, where its policy is based on updating positions of cats. The algorithm proved to be more efficient than PSO as it converges to the optimal solutions in less number of iterations. In [27], a resource-aware hybrid algorithm was proposed to schedule both batch jobs and workflows in the cloud, where tasks are first assigned to group of cloud resources; then classical scheduling is applied for each group to schedule its assigned tasks.

From the above, we conclude that a lot of research work has been done on workflow scheduling; however, there are still more research work needed in several areas, e.g. satisfying the QoS in the elastic and heterogeneous cloud environments.

5. The proposed workflow scheduler

In this paper, we propose a novel IWD-based Cloud scheduling algorithm (IWDC), which optimizes the scheduling of the execution of different workflows tasks in the cloud computing environment. We have developed an implementation of the proposed algorithm and embedded it within the WorkflowSim simulator to be its main scheduling algorithm. In this section, we show how we extended the IWD to schedule workflows in the cloud environment. The proposed algorithm is divided into three phases, see Fig. 1, where the first two phases are identical to the modified IWD algorithm presented [9], and the third phase represents an extension that enables the IWDC algorithm to schedule workflows in the cloud environment. The following subsections present the phases of this novel algorithm in details.

5.1. Preparation and parameter initialization phase

The first step in the proposed algorithm is the initialization of all the static parameters; these parameters include the soil of each edge and the velocity of each water drop (IWD). The definitions of the algorithm parameters including the assigned static values are shown in Table 1.

5.2. Paths construction phase

In this stage, the initial soil on each edge of the graph is calculated according to the fitness function defined in Equation (1).

$$\mathrm{Soil}(i,j) = \max((a * \mathrm{subgraph}(j) + b * \mathrm{condition}(j)), 1) \quad (1)$$

where soil (i, j) refers to the quantity of soil on the directed edge from node i to node j. Condition (j) identifies whether the node j has child node(s) or no. If it has one or more child; then, it is assigned a value of 1; otherwise, it is assigned 0. The method subgraph (j) counts the number of nodes below that particular node (j). The max() method is used to ensure that soil (i,j) = 1; if subgraph (j) && condition (j) == 0. Finally, a, b has the constant values a = 2, b = 1.

After initializing the parameters, steps (A-D), explained next, are repeated to update the soil values, as long some of the paths have not been traversed.

STEP A: Select Next visited node: for the WD, choose the next node to visit in the schedule operation list according to the probability calculated (a conditional probability computation scheme is designed to further improve the diversity of the solution space

$$\mathrm{Probability}(i,j) = \frac{\mathrm{Soil}(i,j)}{\Sigma k} \neq \mathrm{vcSoil}(i,k) \ \& \ \Sigma k \neq 0 \quad (2)$$

where

- Soil (i, j): refer to soil between the two nodes
- Σ k ≠ vc Soil (i, k) refer to summation of the soils of all the paths which can be visited from the current node (i).

STEP B: Calculate Δ Soil of the WD: The local soil update is responsible for the control of the convergence rate of finding a path. So, in order to fully adapt and control the convergence rate, the soil carried by WD is bounded by using the following formula:

Algorithm : IWDC Workflow Scheduling

```
Begin
  //Phase 1:Preparation and static Parameters Initialization (Section 5-1)
  Set the initial parameters: av, bv, cv, as, bs, cs, P, etc.
  Set the initial velocity of each water drop: Vel_WD (i,j)
  //Phase 2:  WDs' Paths Construction (Section 5-2)
  WHILE The Workflow graph is not Empty DO
        //Discover one possible full path of the generated WD
          Calculate Soil on the Edge Between each 2 Remaining Nodes in the Workflow Graph   Equation (1)
          Repeat Until A Full WD Path from the Sink to Source is Discovered
              Set to the Top-Most node as the Current Visited Node
              Calculate All Possible Edges Probabilities                                    Equation (2)
              Use the Calculated Probabilities to Select the Next Node to Visit
              Calculate Soil carried by the WD to the Select Next Node                      Equation (3)
              Calculate Time of the WD travel to the Visited Node                           Equation (4)
              Calculate  the Velocity of the WD;                                            Equation (5)
              Local Soil updating of the Edge to the Selected Node                          Equation (6)
          IF the Selected Visited Node is a Leaf Node
              Delete the Leaf Node
              IF the All the Children of the Parent of the Leaf Node are deleted (or do not exist)
                  Delete  the Parent of the Leaf Node
              END IF
          END IF
          Add the discovered WD path to the best discovered paths List
          Reinitialize the dynamic WD Parameters
  END WHILE
  Search in the List of Best Discovered Paths for the best path P_IWD_best
  //Phase 3: Tasks Assignment Phase (Section 5-3)
  WHILE  The list of discovered best paths is not empty
     Get the best allocated Path P_IWD_best from the list of best WD paths
     WHILE (P_best has more unscheduled tasks)
        Get next top-most unscheduled task t_next in P_IWD_best
        IF t_path  is at the root of the path P_IWD_best
            Search the virtual machine list for the fastest free virtual machine VM_selected
            IF  VM_selected  exists
                Assign t_path to VM_selected
            ELSE
                Search the virtual machine list for the VM that will be free  and set it as VM_selected
                Assign t_path to VM_selected
            END IF
        ELSE
            Assign t_path  to the same virtual machine assigned for  its parent task in the path P_IWD_best
        END IF
        FOR EACH task in the same level of t_path
            Get the left-most unscheduled task t_level in this level
            Search the virtual machine list for the fastest free virtual machine VM_selected
            IF  VM_selected  exists
                Assign t_level to VM_selected
            ELSE
                Search the virtual machine list for the VM that will be free  and set it as VM_selected
                Assign t_level to VM_selected
            END IF
     END WHILE
  END WHILE
END
```

Fig. 1. IWDC workflows scheduling algorithm.

$$\Delta \mathrm{Soil}(i,j,wd) = \frac{as}{(bs + cs * \mathrm{Time}(i,j))} \quad (3)$$

whereas, bs, and cs are the positive parameters declared in Table 1 and

$$\mathrm{Time}(i.j) = (\mathrm{subgraph}(i) - \mathrm{subgraph}(j))/\mathrm{velocity}(wd) \quad (4)$$

where (subgraph (i) – subgraph (j)) represents the between node i and node j, and velocity(wd) refers to the velocity of moving WD from node i to node j.

A bounded local soil update is proposed to make full use of the guiding information and control the convergence rate of finding a path

STEP C: Calculation of the updated velocity of the WD: After moving from node i to node j, the velocity of the WD is affected by the amount of soil existing between node i and node j. Hence, the change of the velocity of WD can be calculated and bounded using the formula in Equation (5).

$$\Delta \mathrm{vel}(i,j,wd) = \mathrm{vel}(wd) + (av/(bv + cv * \mathrm{soil}(i,j))) \quad (5)$$

Table 1
Definitions of IWD parameters.

Symbol	Definition
N	Number of jobs
M	Number of virtual machines
Probability (i, j)	The probability of the WD to move from node i to node j
Time (i, j)	The time taken by the WD to move from node i to node j
Δ Soil (i, j, wd)	The soil collected by the WD on the path between node i and node j
Vel (i, j, wd)	The velocity of the WD when it goes from node i to node j
av, bv, cv	WD velocity updating parameters Those parameters are assigned arbitrarily chosen constant values as follows [6], av = bv = cv = 0.1
as, bs, cs	WD soil updating parameters as = bs = cs = 0.1
P	The local soil updating parameter This parameter is assigned an arbitrarily chosen constant as follows ρ = 0.1
Vel_WD	The initial velocity value is set to Vel_WD = 100

where
- soil (i, j) is the amount of soil on the path between nodes i and j before the WD traverses this path.
- av, bv, cv are user defined positive parameters as specified earlier.
- vel(WD) is the previous velocity of the WD.

STEP D: Local soil updating: Since, the WD carries some amount of soil with it; the soil along the path from node i to node j is reduced. Therefore, the updated amount of soil on the path from i to j, Soil (i, j), is calculated using the following equation:

$$\mathrm{Soil}(i,j) = (1-\rho)\mathrm{Soil}(i,j) - \rho * \Delta\mathrm{soil}(i,j,\mathrm{wd}) \quad (6)$$

where
- ρ is a constant parameter.
- Δsoil (i, j, wd) is the soil removed from the path by the WD while moving from the node i to node j. In the IWD algorithm, updating the global soil is proposed to retain good information of the obtained results.

After updating values of the algorithm for the current node, the same procedure (Steps A-D) is repeated for all successor nodes along the selected path in sequence until it reaches a leaf node. After it reaches the leaf node, that node is deleted, and its parent is also checked. If all leaves of a certain node have been deleted; then, it is also deleted, as seen in Fig. 2, to prevent visiting it again.

5.3. Tasks assignment phase

This phase represents an extension to the original IWD algorithm, it starts after finishing the second phase, in which the whole workflow is traversed and all the IWD paths are discovered. In this phase, in order to assign the workflow tasks to the cloud VMs, the scheduler assigns the workflow tasks level-by-level guided by the best discovered IWD paths in order. As shown in the algorithm, the scheduler traverses all the tasks t_{path} on the best discovered path P_{IWD_best} from its root to its leaves one by one. For the task at the root, i.e. first level, of the P_{IWD_best}, the algorithm searches for the fastest idle machine $VM_{selected}$ and selects it to be assigned it to this task; if no idle VM exists, the algorithm finds the next VM to be free and selects it for executing this task. In the next levels, all descendent tasks in the same path are assigned by default to the same $VM_{selected}$ of the task at their root, as it remains always the fastest free VM. At each level of the best path, after assigning the task t_{path} on the P_{IWD_best} at this level, the algorithm assigns all tasks t_{level} on its level belonging to other connected paths as well from left to right, where for each task t_{level}, the next fast idle VM is assigned to it; otherwise, if no free VM is available, the schedule finds the next VM to be free and assigns it to this task. This important as it ensures the respect of the precedence relationship among the tasks, i.e. no task at any level is assigned to be executed by a VM before its parent task. It is possible that after finishing the above steps some tasks remains unscheduled in the workflow, e.g. if the workflow has several unconnected paths, once the whole P_{IWD_best} is completely traversed, the same steps are repeated on the next best IWD path containing unscheduled tasks in the same workflow.

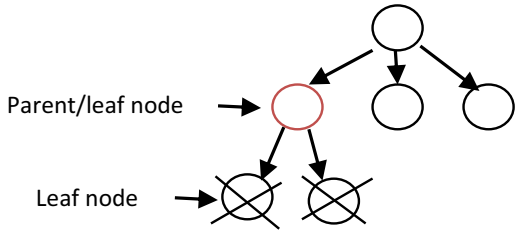


Fig. 2. Parent/leaf node.

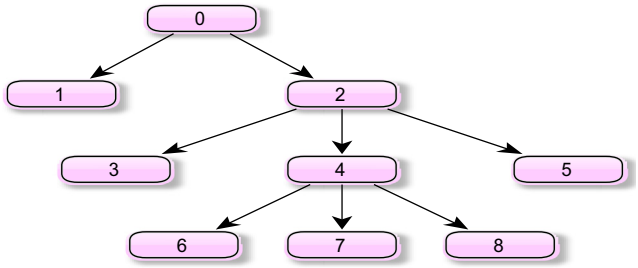


Fig. 3. A case study workflow.

6. A case study of applying the proposed cloud workflow scheduler

In this section, we present a case study of how to apply our proposed algorithm on an example workflow. We study its detailed executed scenario and analyze its behavior. This case study extends the case study presented in [28], it has a workflow composed of a pool of 9 tasks represented as nodes in a DAG numbered from 0 to 8, as shown in Fig. 3. In the graph, each node represents a computational task and each directed edge between the corresponding nodes represents a data or control dependency between tasks. The sequence is important because workflow applications respect the precedence relationship.

Applying our algorithm steps on the graph goes through the following phases:

Initialization and preparation phase: In this phase, all static parameters are assigned and a Sub-graph value is calculated for each node; this value for a certain node indicates the number of all child and grand-child nodes connected to this node, where the sub-graph is traversed using the depth first traversal algorithm, i.e. in the Example workflow, Sub-graph value for node (0) = 8, for node (2) = 6, for node (4) = 3 and for nodes (3, 5, 6, 7, 8) the sub-graph is zero. The decision condition *Condition* (*j*) required in Eq. (1) is checked and tagged for each node. Hence, only nodes (0, 2, 4) are tagged a value of 1 as they have non-zero sub-graph values, while all other nodes (3, 5, 6, 7, 8) are tagged a value of 0.

Paths construction phase: In this phase, the algorithm starts processing from the root, i.e. from node (0), and traverses through the tree to apply Eqs. (1)–(6). In this example, the water drop at

node (0) has two paths to travel through; either go to node (1) or go to node (2). Then, we calculate the soil along both paths as indicated in Eq. (1). The calculation of the soil values along the two paths is as follows:

$$\text{Soil}(0,2) = \max((2*6+1*1),1) = 13$$

$$\text{Soil}(0,1) = \max((2*0+1*0),1) = 1$$

Next we calculate the probability for each of the two paths using Equation (2) in order to find out which path to be visited next as follows:

$$P(0,1) = \text{Soil}(0,1)/(\text{Soil}(0,1)+\text{Soil}(0,2)) = 1/14$$

$$P(0,2) = \text{Soil}(0,2)/(\text{Soil}(0,1)+\text{Soil}(0,2)) = 13/14$$

The path with higher probability is chosen to be visited first. So, node (2) is chosen to be visited first before node (1). Then, in order to calculate the change of the soil amount between node (0) to node (2), we first calculate the time taken to go through from node (0) to node (2) by the WD as indicated in Eq. (3), where the initial velocity is 100 as assumed in Table 1.

$$\text{Time}(0,2) = \max\{(8-6),1\}/100 = 2/100 = 0.02 \text{ sec}$$

According to the IWDC algorithm, some soil is carried by the WD when it travels along the selected path, this amount can be calculated according to Eq. (3) as follows:

$$\Delta\text{Soil}(0,2) = 1/(1+1*0.02) = 0.98$$

This results in a change of both the WD velocity and the soil amount remaining along the selected path. The new values of both the WD velocity and the remaining soil amount can be calculated as follows according to Eqs. (5) and (6) respectively as follows:

$$\Delta\text{Velocity}(0,2) = 100 + 1/(1+1*13) = 100.07$$

$$\text{Updated Soil}(0,2) = 0.9*13 - 0.1*0.98 = 11.6$$

$$\text{Soil(wd)} = 0.98$$

After reaching node (2) the WD has three nodes available for selecting the next node: node (3), node (4), node (5), to decide which node among them to visit, the IWDC algorithm recalculates the both the soil amount at each node of them; then it uses the calculated soil amounts, to calculate the probability of visiting each of them as done before, as shown in the following:

$$\text{Soil}(2,3) = 1, \quad \text{Soil}(2,4) = 2*3+1*1 = 7, \quad \text{Soil}(2,6) = 1$$

$$P(2,4) = \text{Soil}(2,4)/\{\text{Soil2},4) + \text{Soil}(2,4) + \text{Soil}(2,4)\} = 7/9 = 0.77$$

$$P(2,3) = \text{Soil}(2,3)/\{\text{Soil2},4) + \text{Soil}(2,3) + \text{Soil}(2,6)\} = 1/(7+1+1) = 1/9 = 0.11$$

$$P(2,6) = \text{Soil}(2,6)/\{\text{Soil2},4) + \text{Soil}(2,3) + \text{Soil}(2,6)\} = 1/(7+1+1) = 1/9 = 0.11$$

It is clear from these values, that node (4) is the next chosen node to be visited, as it has the greatest probability among the three nodes.

As done with the previous path from node (0) to node (2), the IWDC parameters are calculated for the WD travel between node (2) and node (4), the calculated/updated values are:

$$\text{Time}(2,4) = (6-3)/100.07 = 0.03$$

$$\Delta\text{Soil}(2,4) = 1/(1+1*0.03) = 0.97$$

$$\Delta\text{Velocity}(2,4) = 100.07 + 1/(1+1*7) = 100.195$$

$$\text{Updated Soil}(2,4) = 0.9*7 - 0.1*0.97 = 6.2$$

$$\text{Soil(wd)} = 0.98 + 0.97 = 1.95$$

At node (4), we have three child nodes only: node (6), node (7) and node (8), and because all of them are leaf node with no child, the soil amount along the paths to them from node 4 does not change, while the three nodes gets the same probability values to go to each of them.

$$\text{Soil}(4,6) = \text{Soil}(4,7) = \text{Soil}(4,8) = 1$$

$$P(4,6) = P(4,7) = P(4,8) = 0.33$$

Hence, one of these leaf nodes is selected randomly to be the next node to visit. We assume, here that node (6) is randomly chosen to be next node, and the updated parameters for the path from node 4 to 6 are calculated as follows:

$$\text{Time}(4,6) = (3-0)/100.195 = 0.03$$

$$\Delta\text{Soil}(4,6)1/(1+1*0.05) = 0.97$$

$$\Delta\text{Velocity}(4,6) = 100.195 + 1/(1+1*1) = 100.695$$

$$\text{Updated Soil}(4,6) = 0.9*1 - 0.1*0.97 = 0.803$$

$$\text{Soil(wd)} = 1.95 + 0.97 = 2.92$$

From the above steps, the best path to be discovered by the IWDC algorithm, P_{IWD_best}, is: $0 \to 2 \to 4 \to 6$. Next, the same above calculations is repeated in a second iteration, where the IWDC restart again from Node (0) after purging the pre-visited paths. This results in the second best path which is: $0 \to 2 \to 4 \to 7$. Then the process is repeated again until all paths in the workflow graph is discovered, which results in the following best paths in order: $0 \to 2 \to 4 \to 8$, $0 \to 2 \to 3$, $0 \to 2 \to 5$ and finally $0 \to 1$.

Tasks assignment phase: In this phase, the assignment is made level by level to 4 virtual machines with different speeds shown in Table 2, where the scheduler assign the task at node (0) which lies at the root of the best discovered path P_{IWD_best} to the fastest idle VMs; then as there are no other tasks at the level of node (0), the scheduler goes to the next level, where it searches first for the task at this level that belongs to P_{IWD_best}, which is the task at node (2), and assigns it to the same VM assigned to the task at its parent node, node (0). After that, it checks if there are more unassigned tasks at this level to assign them, from left to right in order, to the next free VM. As only the task at node (1) is not assigned, the scheduler assigns it to the second fastest VM. Then, the same process is repeated in next levels. Hence, we note that the VM3, which is the fastest VM, is chosen to execute all the tasks t_{path} (0, 2, 4, 6) which lies on the P_{IWD_best}. During the execution of each of these tasks, the scheduler assigns tasks t_{level} lying on the same level, but they belong to other paths, to idle VMs, level by level and respecting the dependency among them. In this workflow, see Table 3, since node 1 lies at the same level as node (2) it is assigned to the next fast idle VM, i.e. VM2 in this case. As there are no more nodes on this level, the scheduler turns to the next level, where it finds tasks at nodes 3 and 5, as they both depend on the task at node (2), they have to be scheduled for execution

Table 2
Speeds of VMs used in the case study.

VM	VM0	VM1	VM2	VM3
Speed (MIPS)	500	625	750	875

Table 3
Timing properties of the scheduled workflow tasks.

Task ID	Depth level	Execution TIME	Assigned VM	Start TIME	Finish time
0	1	6.4	VM3	0	6.4
2	2	9.03	VM3	6.4	15.43
1	2	13.73	VM2	6.4	20.13
4	3	7.66	VM3	15.43	23.09
6	4	7.66	VM3	23.09	30.75
3	3	15.68	VM1	15.43	31.11
5	3	16.6	VM0	15.43	32.03
7	4	9.87	VM2	23.09	32.96
8	4	7.66	VM3	30.75	38.41

after node (2) finishes at 15.43. So, the scheduler assigns them, in order from left to right, to the fastest idle virtual machines at this moment, which are VM1 and VM0 respectively, as VM3 and VM2 are busy executing tasks at nodes 4 and 1 respectively at this moment. After that, the scheduler checks the unassigned tasks in the final level, i.e. tasks at nodes 7, 8 in order. As both tasks are dependent on the task at node 4, they both wait for it to finish its execution at the moment 23.09. At this moment, only VM2 is idle; hence task at node 7 is assigned to it; then, the task at node 8 is assigned to the first fastest VM to be idle, which is VM3 that becomes available for assignment after it finishes at 30.75 the execution of the last task along the best path, i.e. task at node 6. Finally, the algorithm checks the next best path(s) in order; if they exist, to check if there are any tasks on them not assigned yet, where the scenario of the best path is repeated after excluding the assigned tasks. In this workflow, no other tasks remain unassigned after finishing the processing of the best path; hence, the algorithm terminates.

For the studied workflow, as seen in Table 3, the makespan, which is the time when the last task finishes its execution, equals 38.41, which is the moment when the task at node 8 finishes its execution. This value is compared with the makespan values of some other common algorithms, FCFS, MCT, RR, MIN-MIN and MAX-MIN, see Table 4. It is clear that our algorithm outperforms all of them.

7. Experimentations and results

The CloudSim toolkit is a java-based discrete event Cloud simulation toolkit. It is a well-known framework for modeling and simulating cloud computing infrastructures and services, it enables the simulation of large scale cloud computing environments. However, CloudSim supports only the execution of single-job workloads and lacks the support of scheduling the scientific workflows. Hence, in order to evaluate the proposed workflow scheduling algorithm, the WorkflowSim simulation tool has been used. The WorkflowSim runs atop of the CloudSim and works as an extended layer to handle the processing of workflows above the cloud. WorkflowSim adds functionalities required to support simulation of scientific Workflows. WorkflowSim does not support only the evaluation of scheduling mechanisms, but it also enables the analysis of various task scheduling/execution overheads and failures. So, we implemented our IWDC algorithm in WorkflowSim 1.1, and evaluated it against a set of well-known scheduling algorithms (explained in details later). We ran our experiments over a system running on a PC with Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz, a RAM of 4.00 GB and running windows 64-bit operating system.

We divided our experiments into 2 sets; the experiments of the first set have been simulated using configurations of a proposed non-realistic environment that provides a proposed non-realistic configurations and cost models of a proposed cloud datacenter, while the experiments in the second set have been simulated using a realistic environment that follows exact configurations adopted from the commercial Amazon EC2 cloud.

7.1. Evaluation using a proposed Non-realistic cloud environment

In the following, we proposed a non-realistic cloud environment with pre-assigned simulation parameters using WorkflowSim, to represent the cloud data center in which the workflows are supposed to be executed. Then, we explain the different workflows that we used in our experiments and finally, we present the scenarios of the proposed experiments and explain the obtained results. The simulator is assigned the set of parameters shown in Table 5, where the number of VMs varies according to the experiment. Also, the bandwidths values and the speed of VMs are either fixed for all VMs in the case of using homogeneous configuration or vary in the ranges shown in case of using heterogeneous configuration.

7.1.1. Performance evaluation using generic workflows

To evaluate the proposed scheduler, we generated five generic workflows of varying sizes and applied the proposed IWDC scheduling algorithm on each. Then, we compared the results with the results obtained by applying other well known algorithm, MIN-MIN, MAX-MIN, Round robin, FCFS and MCT, on the same workflows. For each workflow, we repeated all the experiments on it in CloudSim using virtual machines of different number/configuration as indicated in each experiment.

7.1.1.1. Experiments on Generic Workflow I. In these experiments, we apply our proposed algorithm on the generic workflow shown in Fig. 4. This workflow has 20 tasks, where each task has the same length (T = 10 MI). We conducted the experiments over 3 homogeneous virtual machines (MIPS = 1000, BW = 1000). Then, we repeated the same experiments on three heterogeneous virtual machines (MIPS & BW = (500, 666.66, 833.33). The results obtained of running these algorithms are shown in Figs. 5 and 6. We can see that the proposed IWDC algorithm scored the minimum makespan, while FCFS scored the worst makespan in the both configurations. Also, we can see that MIN-MIN, MAX-MIN, MCT scored nearly the

Table 5
Simulation parameters.

Parameter	Value
Number of data centers	1
Number of VM	3, 5, 10, 20
PEs number (CPU)	1
Band width	Homogeneous: 1000, Heterogeneous: [0–1000
MIPs of PE per VM	Homogeneous: 1000, Heterogeneous: [0–1000]
RAM per VM	512 MB

Table 4
Evaluation of the proposed algorithm with other common algorithms.

Algorithm	FCFS	MCT	MIN-MIN	RR	MAX-MIN	IWDC
Makespan	110.70	39.50	40.77	46.27	41.06	38.41

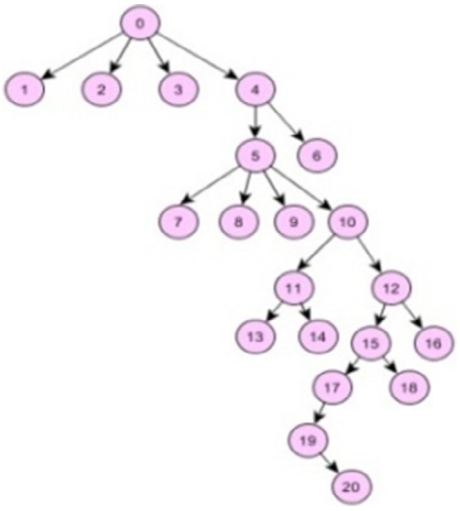


Fig. 4. The structure of Generic Workflow I.

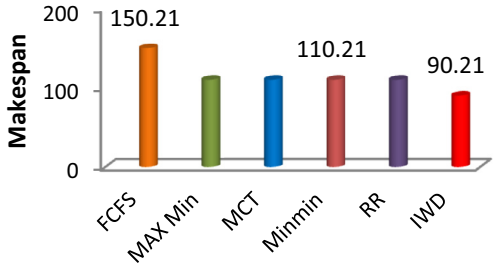


Fig. 5. Makespan values of scheduling Generic Workflow I (20 Tasks) on 3 Homogeneous VMs.

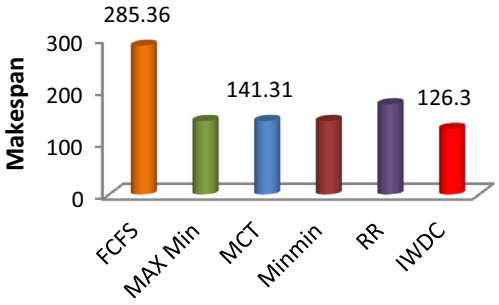
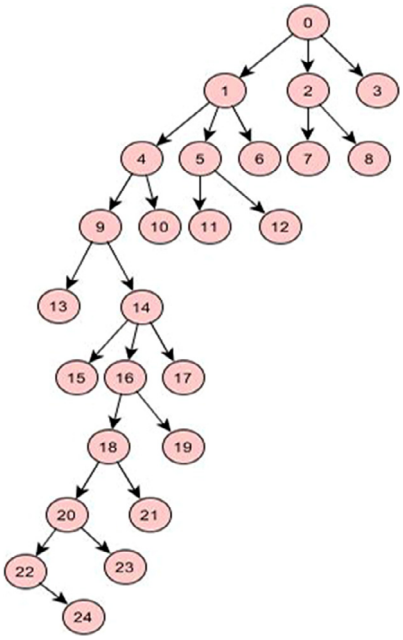


Fig. 6. Makespan values of scheduling Generic Workflow I (20 Tasks) on 3 Heterogeneous VMs.

same makespan, which is expected, as the task–to-VM assignment policies of all these algorithms depend on the task length, which is the same for all tasks in this workflow; this makes no difference in the assignments made by these algorithms, while the IWDC policy defines an optimized path of execution that is not completely dependent on task length.

7.1.1.2. Experiments on Generic Workflow II. In these experiments, we apply our proposed algorithm on the custom generated workflow shown in Fig. 7a. Unlike the workflow in the previous section, this workflow has 25 tasks, where each task has a different length; the lengths of the tasks are shown in Fig. 7b. The experiments were conducted in WorkflowSim, using two main configurations: 5 homogeneous VMs (MIPS = 1000, BW = 1000), and 5 heterogeneous VMs (MIPS $\leq$ 1000, BW $\leq$ 1000). The results obtained of running these experiments using these two configurations are shown in Figs. 8 and 9 respectively. From the graphs, we can see that in the case of using homogeneous configuration, the proposed IWDC scheduler had a makespan equal to RR, MIN-MIN and MCT and lower than FCFS, MAX-MIN. However, in the heterogeneous case, the IWDC scored a makespan value less than all the other algorithms.



-a-Generic Workflow II

Node	length
1	620
2	350
3	100
4	260
5	620
6	670
7	540
8	420
9	750
10	780
11	750
12	324
13	590
14	290
15	489
16	354
17	679
18	579
19	465
20	420
21	564
22	350
23	450
24	390

-b-Tasks Lengths

Fig. 7. Generic Workflow II.

7.1.1.3. Experiments on Generic Workflow III. In this set of experiments, we apply our proposed algorithm on a bigger custom generated Workflow that has 50 tasks, see Fig. 10. All the experiments are conducted in WorkflowSim over 5, 10 then 20 VMs. The experiments on this workflow were done first assuming all its tasks are assigned the same length T = 15, see Fig. 11; then, the same experiments are repeated after assigning random non-equal lengths to the tasks (T $\leq$ 15), see Fig. 12. From the results, we can see that the proposed IWDC scheduler had a Makespan lower than all the other algorithms, where it is significantly better than FCFS and RR, but slightly better than the three other algorithms.

7.1.1.4. Experiments on Generic Workflow IV. In these experiments, we apply our proposed algorithm on a much bigger generated Workflow that has 100 tasks as shown in Fig. 13. We divided the experiments in four sets according to the following four configurations:

- All tasks have equal lengths (T = 10), and all VMs are homogeneous (MIPS = 1000, BW = 1000)
- All tasks have non-equal lengths (T $\leq$ 15), and all VMs are homogeneous (MIPS = 1000, BW = 1000)
- All tasks have equal lengths (T = 10), and all VMs are heterogeneous (MIPS $\leq$ 1000, BW $\leq$ 1000)
- All tasks have non-equal lengths (T $\leq$ 15), and VMs are heterogeneous (MIPS $\leq$ 1000, BW $\leq$ 1000)

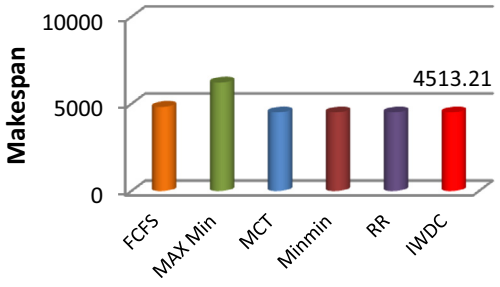


Fig. 8. Makespan values of scheduling Generic Workflow II (25 Tasks) on 5 Homogeneous VMs.

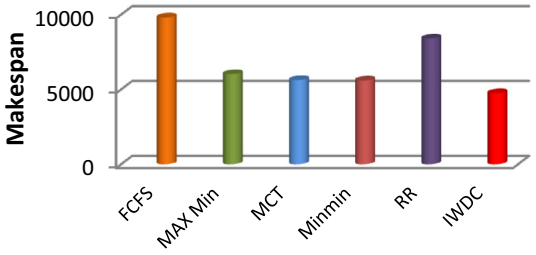


Fig. 9. Makespan values of scheduling Generic Workflow II (25 Tasks) on 5 Heterogeneous VMs.

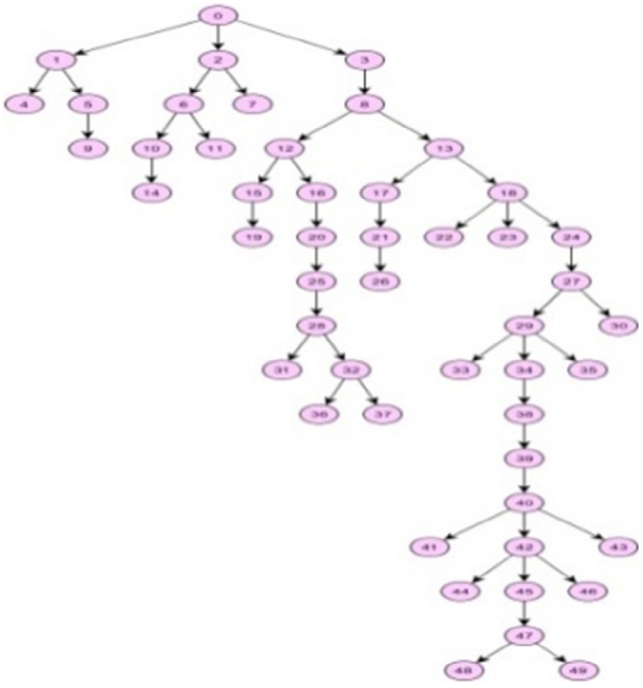


Fig. 10. Generic Workflow III.

Each of the four sets of experiments are repeated over Cloudsim datacenter that has (5, 10, 20) VMs. The results obtained of running these four set of experiments are shown in Figs. 14–17 respectively. We can see that in the four sets of experiments, our proposed IWDC scheduler scored the lowest makespan when compared with all the other algorithms.

The results of all experiments presented in this section is collected and summarized in Table 6, where the lowest values are shown in red color. It is clear from the table that our proposed algorithm beats, in the tested cases, all the other algorithms as it provided the lowest makespan values in all the experiments.

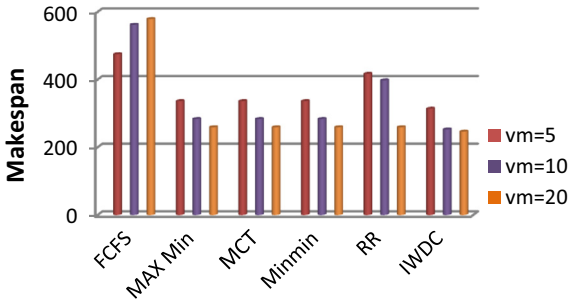


Fig. 11. Makespan values of scheduling Generic Workflow III (50 Tasks of equal lengths) on 5, 10, 20 Heterogeneous VMs.

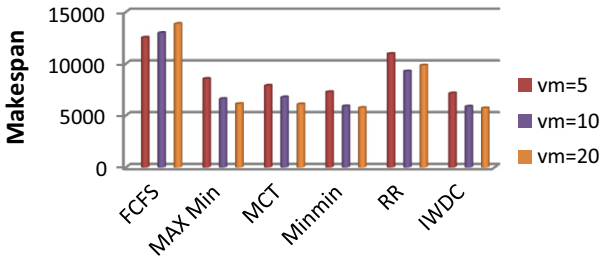


Fig. 12. Makespan values of scheduling Generic Workflow III (50 Tasks of non-equal lengths) on 5, 10, 20 Heterogeneous VMs.

7.1.2. Performance evaluation using common workflows

In this section, we consider the five common workflow structures shown in Fig. 18, which were characterized and profiled in [29]; these workflows are Ligo(Inspiral) [30], Montage [28], Epigenomics [31], CyberShake [32] and Sipht [33]; these five workflows have different structures and parallelism. The Laser Interferometer Gravitational Wave Observatory (**Ligo/Inspiral**) is an application used to detect gravitational-waves; it is used to search for gravitational wave signatures in data collected by large-scale interferometers. The LIGO workflow is a data-intensive workflow. This workflow is characterized by having CPU intensive tasks that consume large memory. **Montage** is an astronomical application widely used as a Grid and parallel computing benchmark. It is used to construct large image mosaics of the sky. Input images are re projected onto a sphere and overlap is calculated for each input image. Most of its tasks are characterized by being I/O intensive while not requiring much CPU processing capacity. **Epigenomics** is a data processing pipeline of various genome sequencing operations. It uses the Pegasus Workflow Management System [34] to automate the execution of the various genome sequencing operations. This workflow is being used by the Epigenome Center in the processing of production DNA methylation and histone modification data. Epigenomics is CPU-intensive. **SIPHT** is used in bioinformatics in the National Center for Biotechnology Information1 database to automate the process of searching for sRNA encoding-genes for all bacterial replicons. Most of the tasks in this workflow have a high CPU and low I/O utilization. **CyberShake** is used to characterize earthquake hazards by generating synthetic seismograms. It calculates synthetic seismograms for each rupture variance from where peak intensity measures are extracted and combined with the original rupture probabilities to produce probabilistic seismic hazard curves for the site and it is classified as a data intensive workflow with large memory and CPU requirements. Datasets of all the mentioned workflows are provided with the WorkflowSim simulator software in the form of DAX. The DAX files are XML files that can be converted to DAG-based workflows by using workflow management system framework tools such as Pegasus [34].

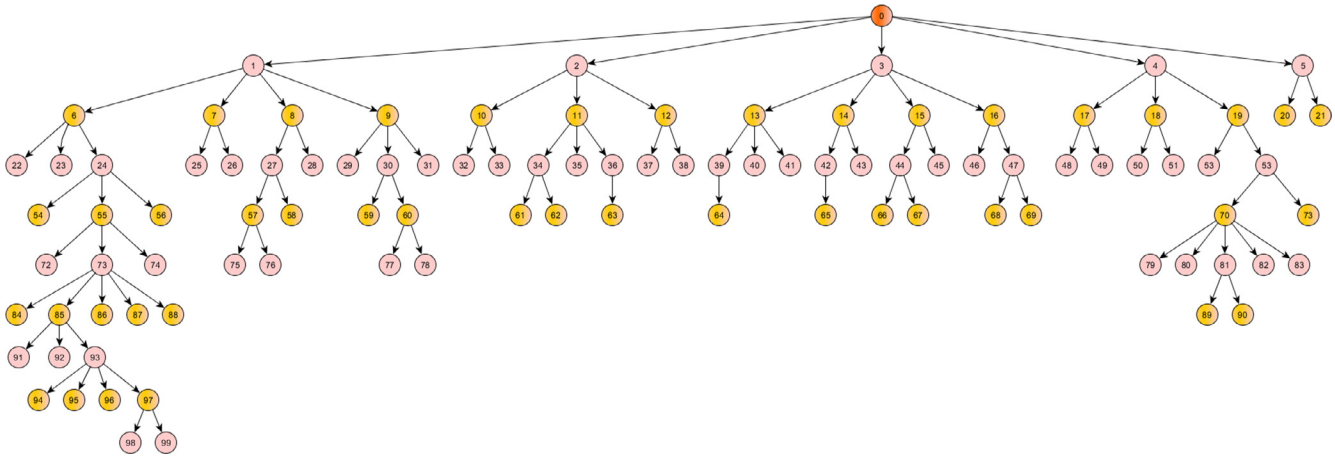


Fig. 13. Generic Workflow IV.

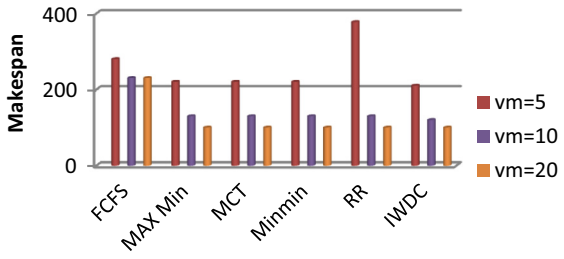


Fig. 14. Makespan values of scheduling Generic Workflow IV (100 Tasks of equal lengths) on 5, 10, 20 Homog. VMs.

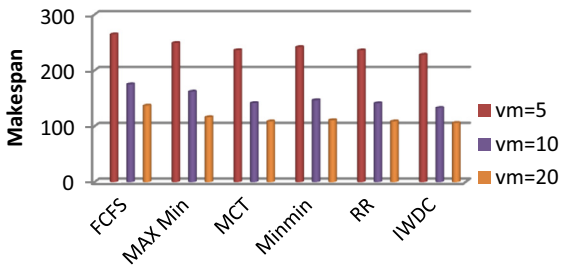


Fig. 15. Makespan values of scheduling Generic Workflow IV (100 Tasks of Non-equal lengths) on 5, 10, 20 Homog. VMs.

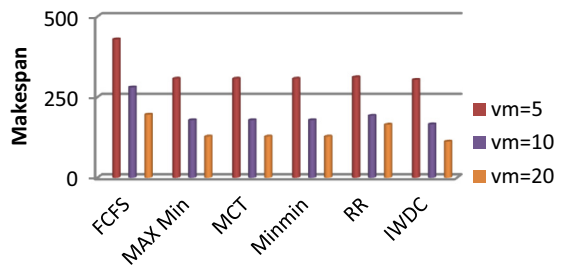


Fig. 16. Makespan values of scheduling Generic Workflow IV (100 Tasks of equal lengths) on 5, 10, 20 Heterog. VMs.

In this section, we use these common workflows to evaluate the application of the proposed IWDC scheduler on them. The evaluation is done by running these workflows on data centers with either homogeneous VMs (MIPS=1000, BW=1000), or heterogeneous VMs with different processor speeds and /or bandwidths. The evaluation is done by comparing the use of the proposed IWDC

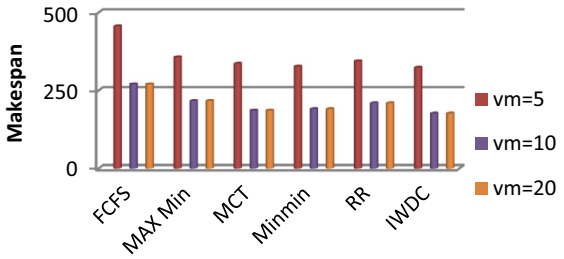


Fig. 17. Makespan values of scheduling Generic Workflow IV (100 Tasks of Non-equal lengths) on 5, 10, 20 Heterog. VMs.

scheduling algorithm on these data centers; then, compare the resulting makespan values with the makespan values achieved by running other well known scheduling algorithm MIN-MIN, MAX-MIN, Round robin, FCFS, MCT for scheduling the same workflows on the same data center configurations.

7.1.2.1. Scenario I. In this scenario, we apply our proposed algorithm on the **Epigenomics100** flow that has 100 tasks. We ran our IWDC scheduling algorithm and all the other algorithms and repeat them on a data center with a number of 5, 10, and 20 VMs. The results obtained of running these algorithms for both the homogeneous and heterogeneous configurations are shown in Figs. 19 and 20 respectively. We can see that the IWDC algorithm scored in both configurations the lowest Makespan in comparison with the other five algorithms.

7.1.2.2. Scenario II. In this scenario, we apply our proposed algorithm on the **Montage100** workflow that has 100 tasks. The results obtained of running the algorithms on the homogeneous and heterogeneous VMs are shown in Figs. 21 and 22 respectively. We can see that in the homogeneous configuration, only MAX-MIN is slightly better than IWDC algorithm, as the IWDC algorithm is behind MAX-MIN by 0.06, 0.01 and 0.04 in the cases of 5, 10, 20 VMs respectively. When using the heterogeneous configuration, the proposed IWDC algorithm was always better than FCFS, MCT, MIN-MIN and RR, but it was behind MAX-MIN with only 0.02 in the case of running the workflow on 5 VM; however, the proposed IWDC algorithm beats it when the workflow runs on 10, 20 VMs.

7.1.2.3. Scenario III. The **Inspiral-100** workflow was used in this scenario. Fig. 23 shows the results obtained of scheduling it on 5, 10 and 20 homogeneous VMs. we see that IWDC algorithm is

Table 6
Summary of Makespan values of the Generic Workflows by all algorithms.

Scenarios	(n)VMs	FCFS	MAX – MIN	MCT	MIN-MIN	RR	IWDC
Workflow I on Homog. VMs	3	285.36	141.31	141.31	141.31	173.38	126.3
Workflow I on Hetrog. VMs	3	150.21	110.21	110.21	110.21	110.21	90.21
Workflow II on Homog. VMs	5	9806.31	6022.46	5619.8	5592.36	8396.51	4750.74
Workflow II on Hetrog. VMs	5	4828.21	6229.21	4513.21	4513.21	4513.21	4513.21
Workflow III (All Task Lengths = 15)	5	474.78	336.46	336.46	336.46	417.16	313.96
	10	562.26	283.82	283.82	283.82	398.16	252.84
	20	579.14	259.42	259.42	259.42	259.42	246.36
Workflow III (All Task Lengths ≤ 15)	5	12538.92	8561.29	7896.23	7280.63	10963.41	7143.29
	10	12972.49	6609.76	6769.46	5909.92	9282.17	5868.63
	20	13866.88	6140.17	6102.13	5763.88	9858.82	5718.15
Workflow IV on Homogeneous VMs & (All Task Lengths = 10)	5	456.03	357.36	336.19	327.61	344.23	323.65
	10	270.43	216.87	186.06	190.92	209.71	176.98
	20	270.43	216.87	186.06	190.92	209.71	176.98
Workflow IV on Homogeneous VMs & (All Task Lengths ≤ 10)	5	264.52	249.38	236.23	242.11	235.99	228.29
	10	175.37	161.97	141.58	146.74	141.21	133.06
	20	137.09	116.37	108.86	110.97	108.91	105.91
Workflow IV on Heterogeneous VMs & (All Task Lengths = 10)	5	429.41	307.48	307.48	307.48	312.3	304.04
	10	280.42	178.61	178.61	178.61	192.75	165.7
	20	195.88	128.01	128.01	128.01	164.6	112.24
Workflow IV on Heterogeneous VMs & (All Task Lengths ≤ 15)	5	280.21	220.21	220.21	220.21	377.61	210.21
	10	230.21	130.21	130.21	130.21	130.21	120.21
	20	230.21	100.21	100.21	100.21	100.21	100.21

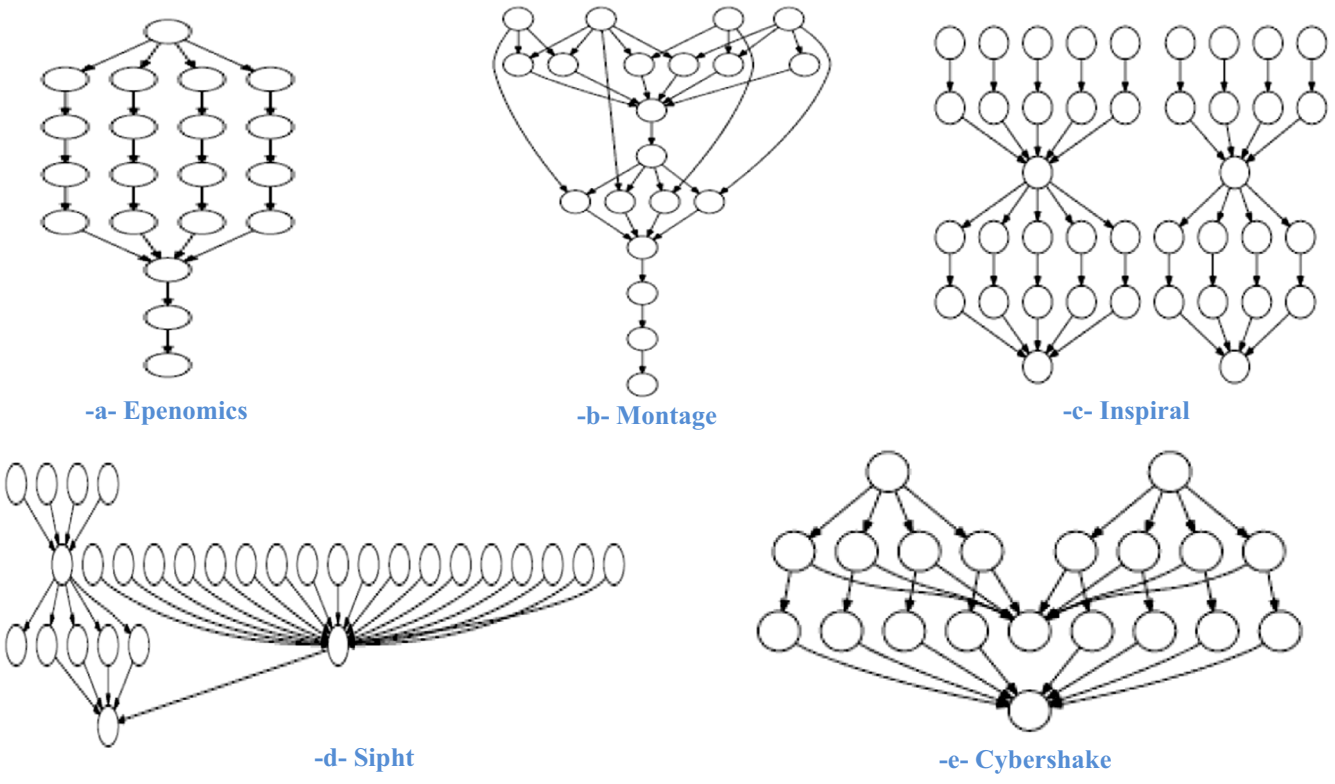


Fig. 18. Common workflows used for evaluation.

better than all other algorithms, except when 5 virtual machines were used (MAX-MIN was better). When 10 virtual machines were used both the IWDC and MAX-MIN were equal and better than all the other algorithms. When heterogeneous VMs are used, see Fig. 24, IWDC algorithm was always better than the other algorithms, except in two configurations: when 5 virtual machines are used, MCT was better than the IWDC algorithm, and when 10 virtual machines are used RR was better.

7.1.2.4. Scenario IV. We ran the **Sipht100** workflow that has 100 tasks of equal execution times. The results obtained of running the scheduling algorithms on homogenous virtual machines are

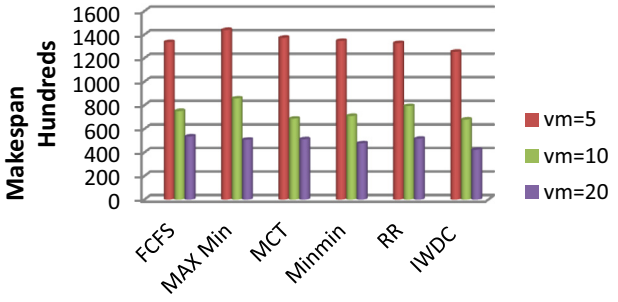


Fig. 19. Makespan values of scheduling Epigenomics-100 Workflow on 5, 10, 20 Homogeneous VMs.

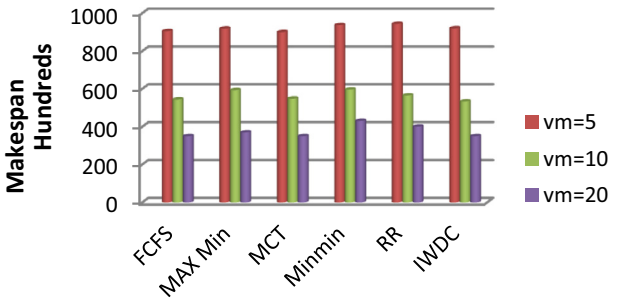


Fig. 20. Makespan values of scheduling Epigenomics-100 Workflow on 5, 10, 20 Heterogeneous VMs.

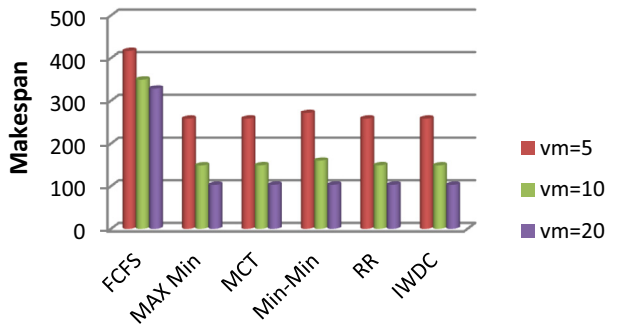


Fig. 21. Makespan values of scheduling Montage-100 Workflow on 5, 10, 20 Homogeneous VMs.

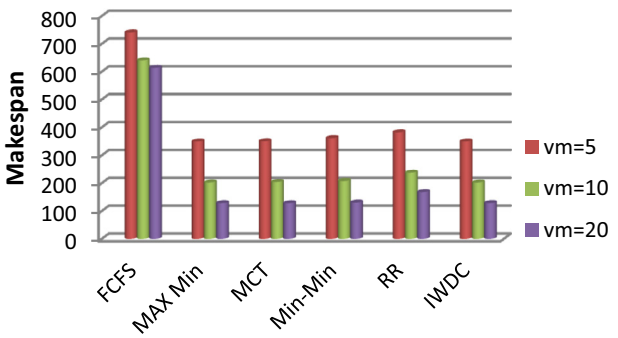


Fig. 22. Makespan values of scheduling Montage-100 Workflow on 5, 10, 20 Heterogeneous VMs.

shown in Fig. 25, where we see that IWDC algorithm scored the lowest makespan. The results of using heterogeneous VMs are shown in Fig. 26; these results show that the IWDC algorithm is better than all other algorithms, except when 10 and 20 virtual machines were used; only the MAX-MIN was better than the IWDC

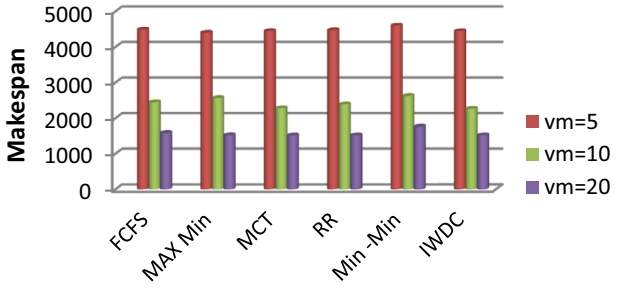


Fig. 23. Makespan values of scheduling Inspiral-100 Workflow on 5, 10, 20 Homogeneous VMs.

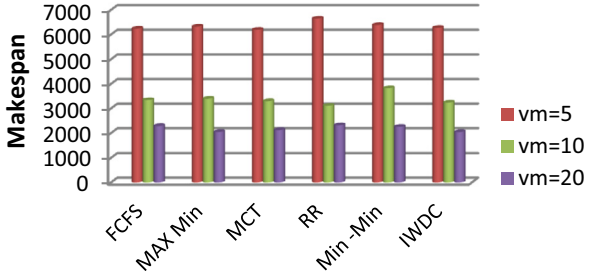


Fig. 24. Makespan values of scheduling Inspiral-100 Workflow on 5, 10, 20 Heterogeneous VMs.

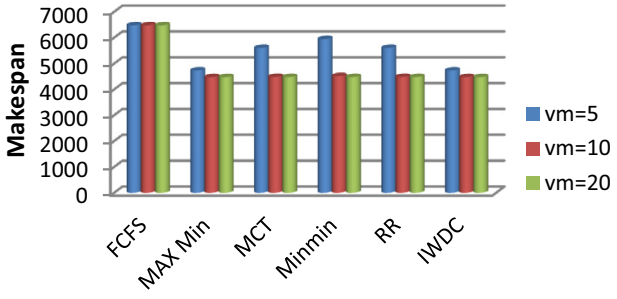


Fig. 25. Makespan values of scheduling Sipht-100 Workflow on 5, 10, 20 Homogeneous VMs.

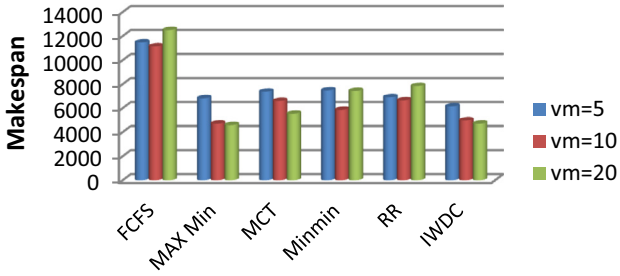


Fig. 26. Makespan values of scheduling Sipht-100 Workflow on 5, 10, 20 Heterogeneous VMs.

7.1.2.5. Scenario V. **CyberShake-100** workflow is used in this scenario. We see in Fig. 27, that when using 5 homogeneous VMs, only MAX-MIN was slightly better than IWDC, but when the number of VMs increased to be 20, both IWDC and MAX-MIN result in similar makespan values better than the others. In contrary, in heterogeneous case in Fig. 28, MAX-MIN was always better followed by the IWDC algorithm. Table 7 shows a summary of the Makespan of the five algorithms and the IWDC Makespan, where the lowest values are shown in red color.

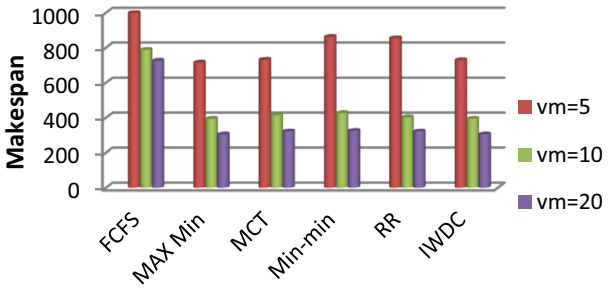


Fig. 27. Makespan values of scheduling Cypershake-100 Workflow on 5, 10, 20 Homogeneous VMs.

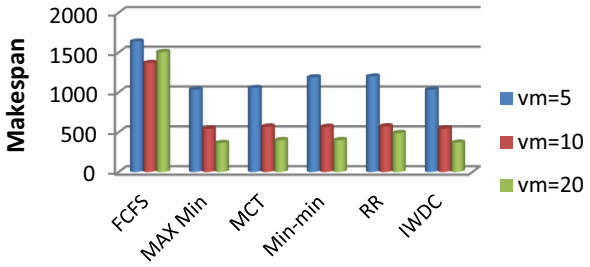


Fig. 28. Makespan values of scheduling Cypershake-100 Workflow on 5, 10, 20 Heterogeneous VMs.

7.1.3. Performance evaluation against other meta heuristic algorithms

In the previous sections, we compared our algorithm with a set of common heuristic algorithms. In this section, we compare it with schedulers based on the meta heuristic optimization techniques PSO and its variant C-PSO [36]. PSO and C-PSO are workflow cloud-scheduling algorithms that rely on the social behavior of a swarm of particles and C-PSO a variant of PSO that is inspired by the catfish effect observed by Norwegian fishermen when catfishes were introduced into a holding tank of sardines. We used WorkflowSim to run our scheduler to schedule 16 different workflows on a cloud environment with 6 VMs similar to the environment presented in [37] in order to compare with their results. Table 8 shows the makespan values obtained by our algorithm and the published results of PSO and C-PSO. The results are drawn in Fig. 29, which shows clearly that our algorithm outperforms the other two algorithms. In addition to its efficiency, out algorithm has faster execution time over the two algorithms as the two algorithms requires generating many solutions before reaching the final solution, this makes them have higher complexity times in comparison with our algorithm.

7.1.4. Cost evaluation of using the resources of the proposed non-Realistic cloud

Cost reduction is a main concern when developing a cloud scheduler. Different cloud cost plans can be used for the rented resources of the data center, where variable charges may exist

Table 7
Summary of Makespan values of the Common Workflows by all algorithms.

Evaluation using common Workflows								
Scenarios		nVM	FCFS	Max -min	MCT	MIN-MIN	RR	IWDC
Scenario I Epigenomics100	Heterogeneous	5	133796.83	144004.08	137360.23	134718.98	132860.8	**125585.7**
		10	75137.1	85872.76	68764.84	71174.05	79376.2	**68065.07**
		20	53674.18	50930.02	51414.92	47843.51	51760.39	**42313.65**
	Homogeneous	5	90419.9	91836.45	89970.84	93639.32	94309.18	**91945.82**
		10	54341.3	59264.61	54841.96	59588.64	56441.39	**53266.13**
		20	34991.24	36972.03	34988.53	43044.26	40036.51	**34987.83**
Scenario II Montage100	Heterogeneous	5	360.07	741.26	349.9	350.74	361.51	**349.92**
		10	215.38	640.65	202.54	204.12	207.94	**202.29**
		20	166.95	613.78	128.29	128.03	130.42	**128.44**
	Homogeneous	5	258.19	416.85	258.08	258.26	271.42	**258.14**
		10	148.94	349.62	148.79	149	160	**148.8**
		20	103.34	328.51	103.19	103.45	103.53	**103.23**
ScenarioIII Inspiral100	Heterogeneous	5	6236.97	6320.62	6195.44	6647.27	6385.32	**6270.64**
		10	3343.08	3397.18	3301.43	3110.95	3828.35	**3240.31**
		20	2288.3	2047.13	2125.17	2316.09	2257.65	**2043.79**
	Homogeneous	5	4489.7	4405.03	4447.38	4473.18	4600.47	**4444.89**
		10	2450.19	2567.88	2278.85	2390.7	2625.93	**2264.2**
		20	1587.21	1523.58	1519.72	1519.9	1762.86	**1519.72**
Scenario IV Siph100	Heterogeneous	5	11465.05	6823.43	7368.04	7484.71	6906.59	**6150.09**
		10	11136.18	4711.19	6603.67	5859.09	6653.49	**4972.79**
		20	12503.6	4590.39	5531.14	7437.26	7843.72	**4711.12**
	Homogeneous	5	6477.06	4733.52	5603.27	5944.19	5603.44	**4731.59**
		10	6477.06	4475.62	4482.22	4516.49	4482.22	**4475.61**
		20	6477.06	4475.62	4478.22	4479.07	4478.22	**4475.61**
Scenario V CyberShake100	Heterogeneous	5	1636.73	1033.1	1054.09	1187.63	1198.41	**1031.86**
		10	1368.78	544.11	570.78	567.95	573.3	**546.98**
		20	1505.72	365.64	399.44	399.77	490.57	**369.22**
	Homogeneous	5	999.39	716.16	732.53	862.59	854.29	**729.59**
		10	787.5	394.37	416.46	426.82	404.01	**394.08**
		20	726.49	305.19	321.92	325.56	321.92	**305.19**

Table 8
A comparison of Makespan values of IWDC, PSO and C-PSO.

Workflow size	Makespan			Workflow size	Makespan		
	PSO	C-PSO	IWDC		PSO	C-PSO	IWDC
Montage 100	471.68	353.71	216.52	Montage 300	1221.73	1114.52	657.28
Inspiral 100	5214.21	4718.31	3804.26	Inspiral 300	13891.87	11911.5	11197.34
Cybershake100	977.67	910.44	776.77	Cybershake300	2402.08	2227.7	1296.46
Epigenomics100	69339.78	62194.96	51276.15	Epigenomics300	40109.54	35336.74	41213.43
Montage 200	810.13	709.95	432.37	Montage 400	2450.92	1817.19	898.18
Inspiral 200	11263.47	9290.73	7958.96	Inspiral400	20251.18	17525.57	16007.68
Cybershake200	1833.95	1589.41	870.29	Cybershake 400	3029.09	2799.77	1649.73
Epigenomics200	74773.56	69140.07	41192.1	Epigenomics400	204505.02	174331.88	147727.12

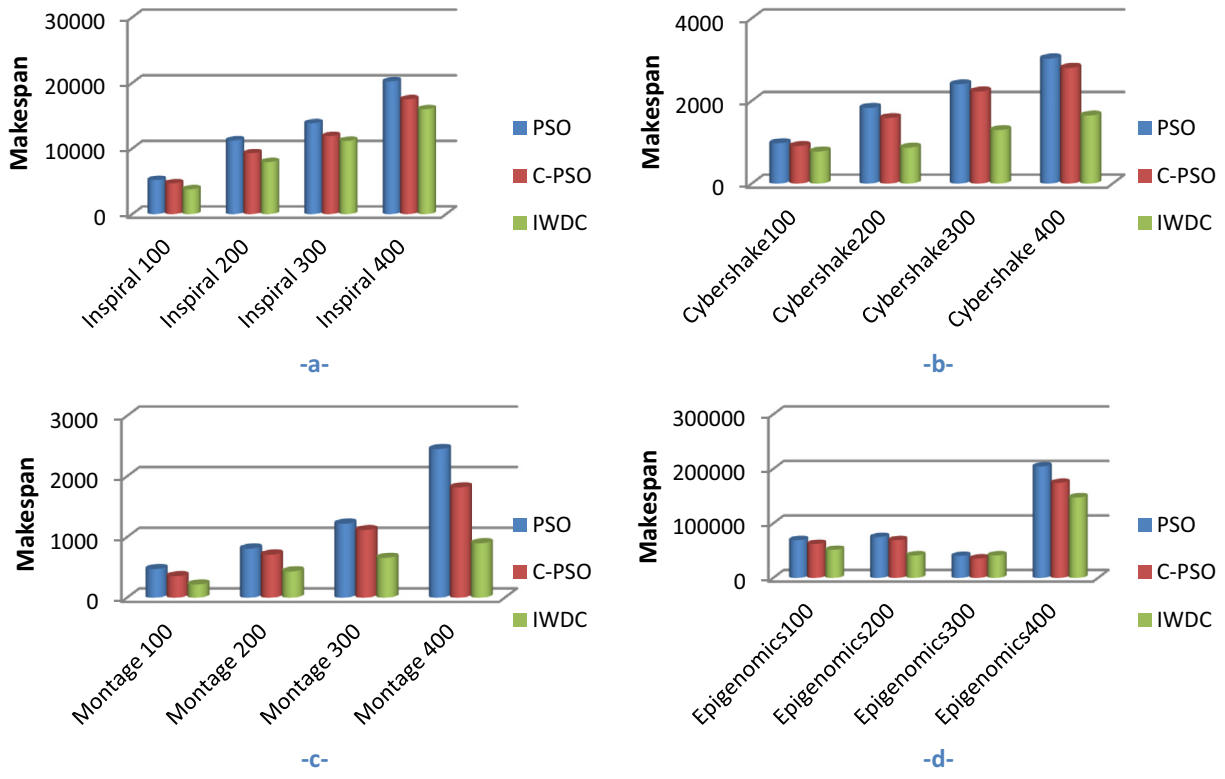


Fig. 29. A Comparison of Makespan values of IWDC, PSO and C-PSO.

based on resource sizing, but the pricing doesn't always scale linearly based on performance. In our proposed Cloud data center, to evaluate the economic usage of the proposed IWDC scheduler, we proposed the applications of two different cost plans. These two cost plans and the results of using them when using the proposed scheduler and some other algorithms are discussed next.

7.1.4.1. Cost Plan 1. This proposed cost plan charges the customers only for the use of datacenter irrespective of the types of virtual machines used. This cost plan involves the processing cost, the memory usage cost, the storage cost and the bandwidth cost shown in Table 9.

By applying the proposed scheduler and the other 5 schedulers on the Montage 100 workflow, we can see that IWDC algorithm provides a schedule with a cost less than MCT, RR, MIN-MIN when VM = 5. Also, the proposed algorithm was better than MCT, MAX-MIN and RR when VM = 10 and when VM = 20 the cost was lower than MCT, MAX-MIN, RR and MIN-MIN which mean that IWDC is stable algorithm as shown in Table 10.

Table 9
Cost plan 1.

Resource	RAM	Processor	Bandwidth	Storage
Size	512 MB	3 × 1000MIPS	1000 bps	No limit
Cost	0.05 per MB	3 per processor	0.1	0.1 per MB

7.1.4.2. Cost plan 2. Unlike the cost plan 1, this cost plan charges the customers depending on the types of virtual machines used. It is calculated according to the following formula:

$$\begin{aligned}\text{Cost2} &= (\text{communicationcost} + \text{computationcost}) \\ &= (\text{data}(\text{bothinputandoutput}) * \text{unitcostofdata} + \text{runtime} * \text{cpu.cost})\end{aligned}$$

In order to evaluate the effect of using the proposed IWDC scheduler on the cost paid by the customer, we conducted a set of experiments in which we embedded the proposed scheduler in the workflowSim and calculated the cost of scheduling 4 versions of different sizes, i.e. different number of tasks, of each of the 5 common workflows: Montage, Cypershake, Epigenomics, Sipht

Table 10
Results of using cost plan 1.

	FCFS	MCT	MAX-MIN	RR	MIN-MIN	IWDC
5 VMs	3439.4	3441.15	3440.44	3440.77	3441.15	3440.9
10 VMs	3444.76	3446.71	3447.02	3447.45	3446.47	3446.98
20 VMs	3448.64	3451.96	3452.34	3451.71	3453.14	3449.74

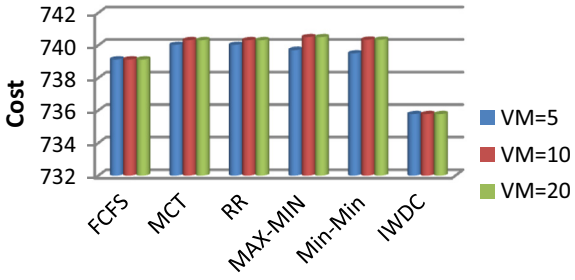


Fig. 30. Costs of scheduling Montage 25.

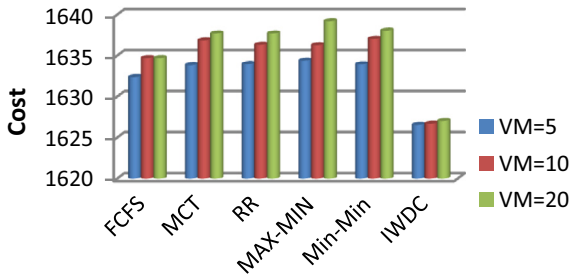


Fig. 31. Costs of scheduling Montage 50.

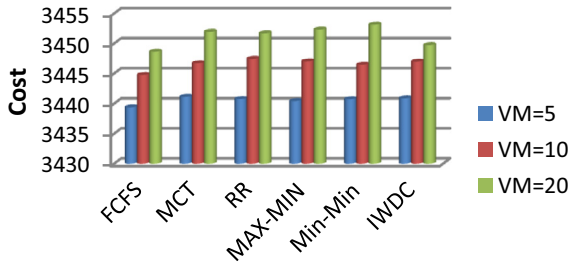


Fig. 32. Costs of scheduling Montage 100.

and Inspiral. Then, we compared the calculated cost with the cost calculated by applying 5 other scheduling algorithms: FCFS, MCT, RR, MAX-MIN and MIN-MIN, where each algorithm is applied on a data center that has 5, 10 and 20 VMs The results of these experiments are discussed next.

7.1.4.2.1. Evaluating the cost of scheduling Montage workflows. The 4 versions of the Montage workflow have 25, 50, 100 1000 tasks are scheduled in this experiment. As indicated in Figs. 30–33, for the Montage-25 and Montage-50, the proposed IWDC algorithm had the lowest cost significantly in comparison with all the other five algorithms. For Montage-100 and Montage-1000 the differences among the algorithms in costs were not significant.

7.1.4.2.2. Evaluating the cost of scheduling Cybershake workflows. In this experiment, we examined the 4 versions of the **Cybershake** workflow have 30, 50, 100 1000 tasks. As indicated in the Figs. 34-37 In the case of CyberShake-30, FCFS led to the lowest

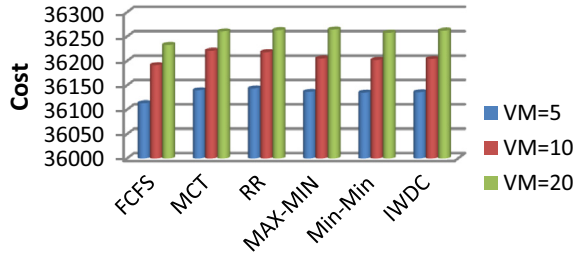


Fig. 33. Costs of scheduling Montage 1000.

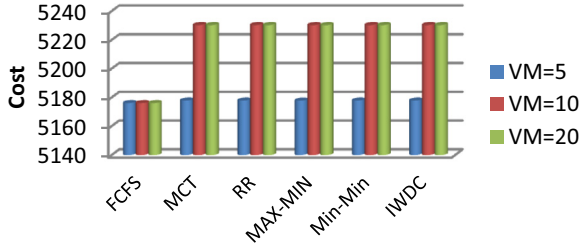


Fig. 34. Costs of scheduling Cypershake-30.

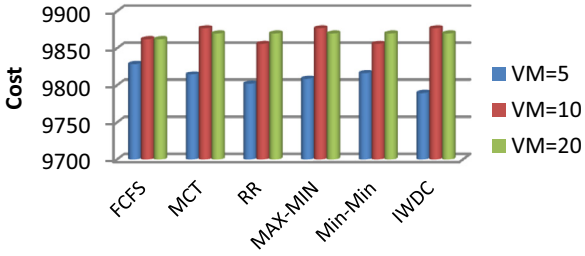


Fig. 35. Costs of scheduling Cypershake-50.

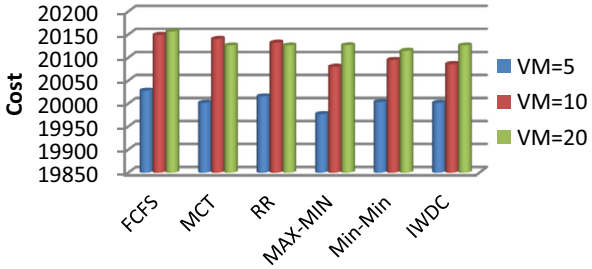


Fig. 36. Costs of scheduling Cypershake-100.

cost, while the IWDC algorithm scored costs nearly equal to the other algorithms. For CyberShake-50, when 5 VMs were used, IWDC was the best while in case of using 10 VMs FCFS, MIN-MIN and RR were the best while MCT, MAX-MIN and our proposed algorithm scored slightly around 2% higher costs. In case of 20 VMs, all algorithms scored nearly equal costs except FCFS which scored slightly around 3% less costs. For CyberShake-100, In the case of

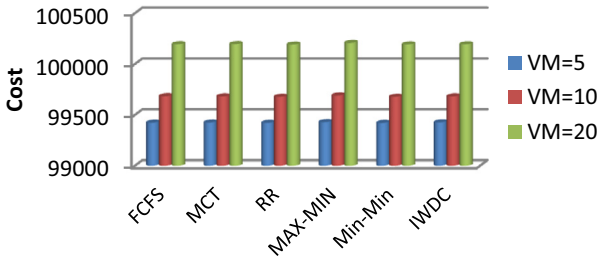


Fig. 37. Costs of scheduling Cypershake-1000.

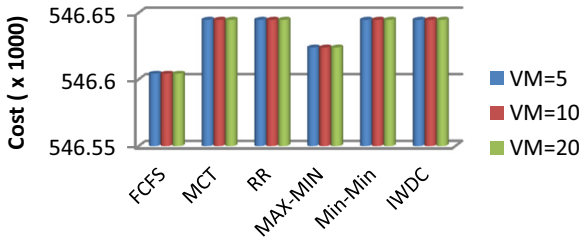


Fig. 38. Costs of scheduling Epigenomics-24.

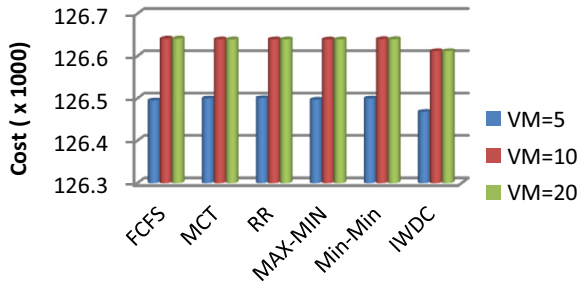


Fig. 39. Costs of scheduling Epigenomics-46.

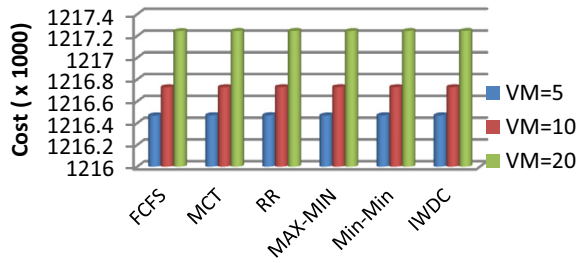


Fig. 40. Costs of scheduling Epigenomics-100.

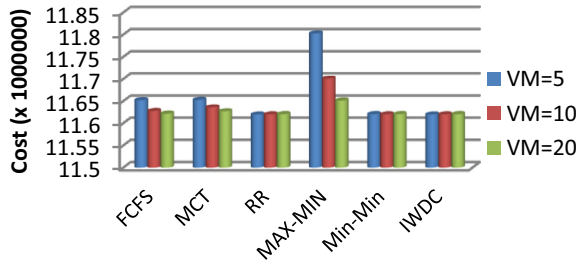


Fig. 41. Costs of scheduling Epigenomics-997.

using 5VMs IWDC was slightly better than FCFS, RR and MIN-MIN, but MAX-MIN had the lowest cost, while with 10 VMs, RR was the best followed by our IWDC algorithm, with 20VMs MIN-MIN was

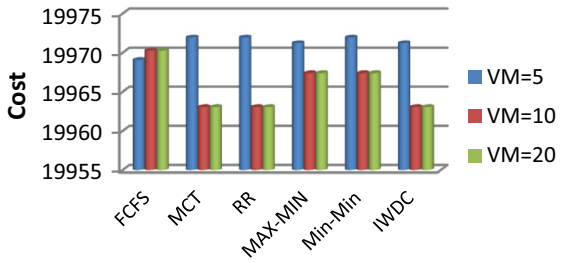


Fig. 42. Costs of scheduling Inspiral-30.

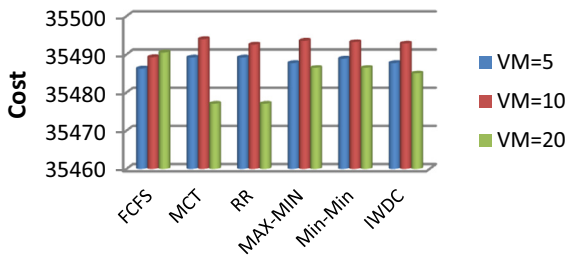


Fig. 43. Costs of scheduling Inspiral-50.

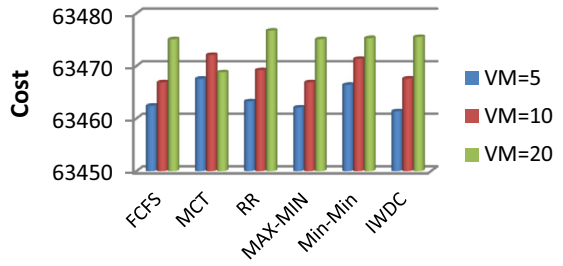


Fig. 44. Costs of scheduling Inspiral-100.

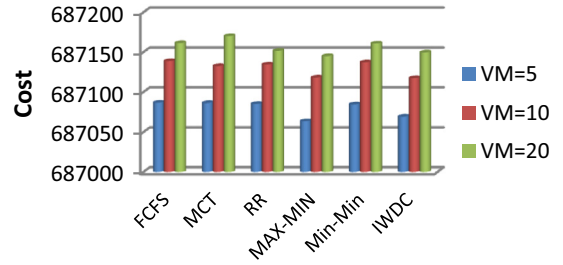


Fig. 45. Costs of scheduling Inspiral-1000.

the best followed by IWDC. For CyberShake-1000 workflow, there was no significant difference in the calculated cost among all the algorithms.

7.1.4.2.3. Evaluating the cost of scheduling Epigenomics workflows. In this experiment, there are 4 versions of the Epigenomics workflows that have 24, 46, 100 and 997 tasks per workflow. As shown in Figs. 38–41, for the Epigenomics-24, FCFS and MAX-MIN had significantly better costs than the other four algorithms that nearly have the same cost. For Epigenomics-46, see Fig. 39, our IWDC algorithm has a slightly less cost than the other algorithms. In the case of Epigenomics-100, there were no significant differences in the costs among all the algorithms. For Epigenomics-997 IWDC, RR and MIN-MIN had the lowest cost.

7.1.4.2.4. Evaluating the cost of scheduling Inspiral workflows. In these experiments, we examined 4 versions of the Inspiral workflow have 30, 50, 100 1000 tasks. For Inspiral-30, RR, MCT and

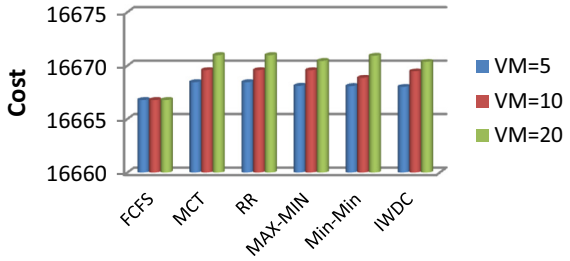


Fig. 46. Costs of scheduling Sipht-30.

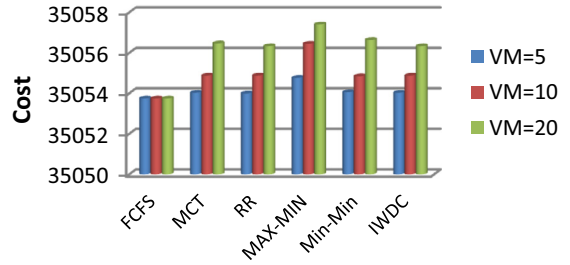


Fig. 47. Costs of scheduling Sipht-60.

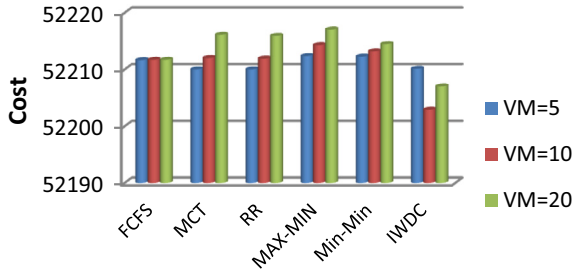


Fig. 48. Costs of scheduling Sipht-100.

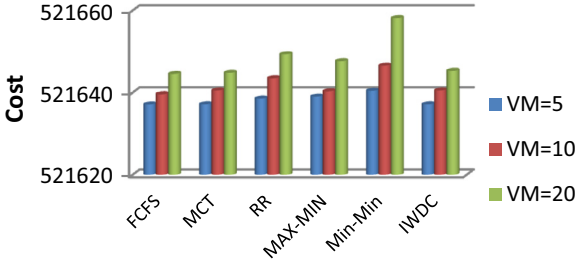


Fig. 49. Costs of scheduling Sipht-1000.

our algorithm had significant lower costs than the other algorithms as shown in Fig. 42. For Inspiral-50, IWDC came with costs higher than MCT and RR but nearly equal to the other algorithms as shown in Fig. 43. For Inspiral-100, IWDC came with costs similar to the costs of the other algorithms, except the MCT which had higher cost when 5 and 10 VMs were used and it had worse cost when 20 VMs were used as shown in Fig. 44. Finally, in the case of Inspiral-1000, both IWDC and MAX-MIN had slightly lower costs than the other four algorithms as shown in Fig. 45.

7.1.4.2.5. Evaluating the cost of scheduling Sipht workflows. In these experiments, there are 4 versions of the Sipht workflow have 30, 60, 100 1000 tasks per workflow. From Figs. 46 and 47 we see that Sipht-30 and Sipht-60 the proposed IWDC algorithm had the lowest cost after the FCFS algorithm. While, in the case of Sipht-100, it had costs significantly better than all the other algorithms as shown in Fig. 48, and in the case of Sipht-1000, it had, with FCFS and MCT the lowest costs as shown in Fig. 49.

From all the above cost-evaluation experiments, it is clearly that our proposed IWDC algorithm has always reasonable costs, in comparison with other algorithms, when applied in cloud that have different number of VMs and for scheduling different types and sizes of workflows, which means that IWDC is stable algorithm and leads to providing reasonable costs for cloud tasks and services.

7.2. Evaluation using configurations of a commercial environment

In order to evaluate the efficiency of our algorithm commercially, we examined both the performance and the cost of running some common scientific workflow applications scheduled by our algorithm and by some other common cloud scheduling algorithms using configurations from the commercial Amazon's EC2 cost model [35]. The cost model of Amazon EC2 was used for running the experiments because it is currently the most popular, feature-rich, and stable commercial cloud available. Amazon EC2 has several cloud pricing models including On Demand, Reserved and Spot Instances models. We choose On-Demand instances model as it makes the client pay for the compute capacity he actually uses by the hour with no long-term commitments, where the usage is rounded up to the nearest hour, so any partial hours are charged as full hours.

The virtual machines offered by Amazon EC2 are called instances, where the user can select the required instance from several preconfigured instances called *instance types*. Each instance type is configured with a specific amount of memory, CPUs, and local storage. In our work we examined the impact of 4 different Linux-based instance types on the performance and cost of some common workflow applications. The specifications and the on-demand cost of these instances are summarized in Table 11.

In order to evaluate our algorithm in different working conditions, we divided the evaluation experiments into two types of experiments; experiments to evaluate our algorithm when it schedules the workflows on a set of homogenous instances and other experiments to evaluate it when it schedule workflows on a set of heterogeneous instances. These two types of experiments are presented next.

7.2.1. Cost-Makespan tradeoffs on commercial homogeneous VM instances

The aim of these experiments is to evaluate the efficiency of IWDC algorithm (in terms of both makespan and cost by comparing its performance with the performance of a set of common cloud

Table 11
The specifications of the instance types used in the experiments.

Instance types	API name	Compute units (ECU)	Memory	Network performance	Cost (Linux)
High-CPU Extra large	c1.xlarge	2.5 ECU	7.0 GB	High	$0.520 hourly
Extra large instance	c4.large	2 ECU	3.75 GB	Moderate	$0.100 hourly
Small instance	m1.small	1 ECU	1.7 GB	Low	$0.044 hourly
Micro instance	t1.micro	0.5 ECU	0.613 GB	Very low	$0.020 hourly

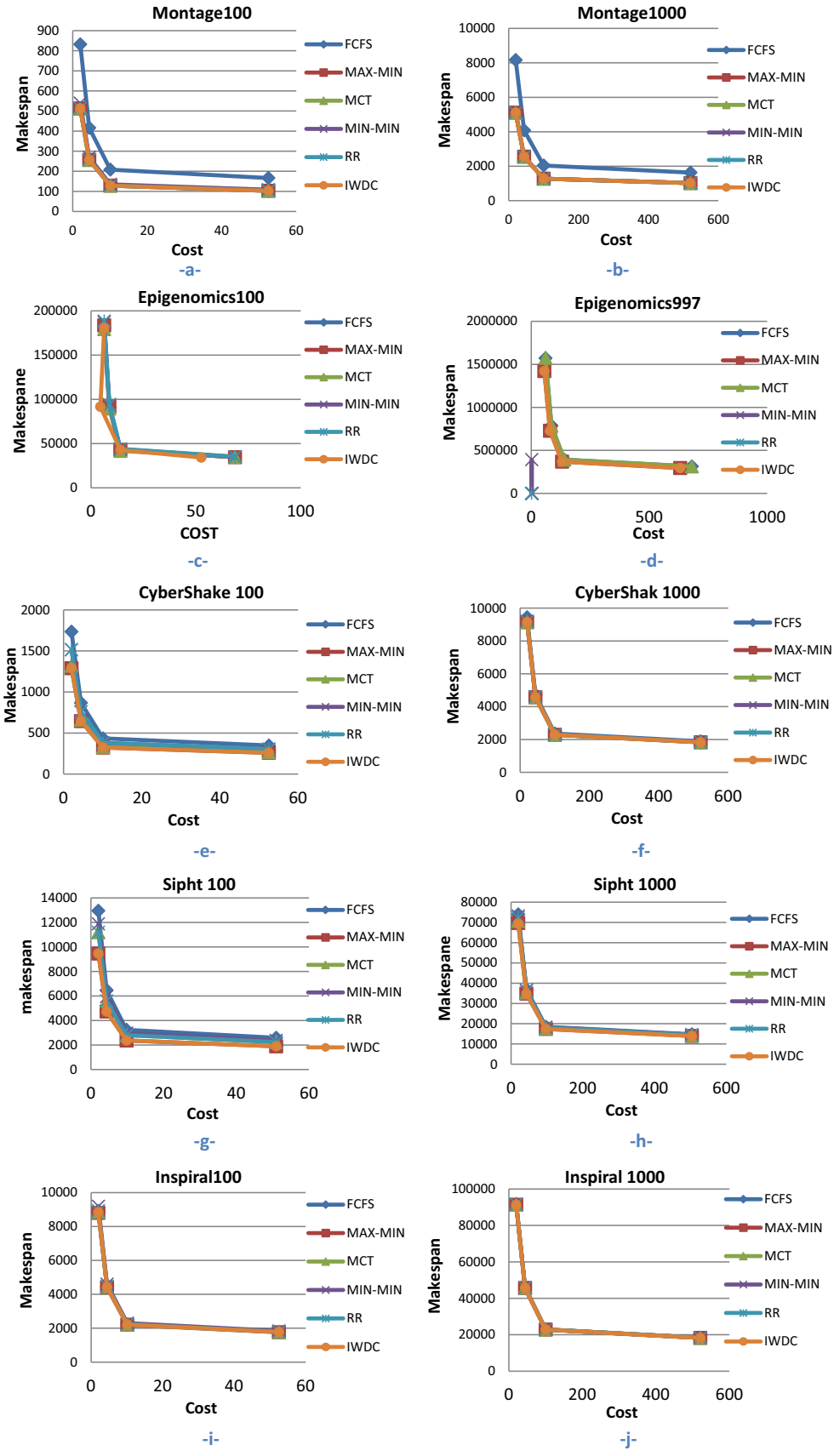


Fig. 50. Makespan-time trade-offs for some real scientific workflows.

Table 12
Summary instant types specifications of the 6 groups used in the experiments.

	Montage		Epigenomics		CyberShake		Sipht		Inspiral	
	Cost	Makespan	Cost	Makespan	Cost	Makespan	Cost	Makespan	Cost	Makespan
c1.xlarge	52.52	102.7	52.52	34218.33	52.52	259.45	50.96	1892.6	52.52	1779.22
c4.large	10.1	128.29	14	42772.84	10.1	324.06	9.8	2365	10.1	2212.66
m1.small	4.44	256.45	4.44	91670.95	4.44	647.12	4.31	4731.4	4.44	4425.2
t1.micro	2.02	512.81	6.28	180607.94	2.02	1293.23	2.04	9462.7	2.02	8850.27

scheduling algorithms. In these experiments, the common scientific workflows, mentioned earlier in 7–1–2, are scheduled on a set of 5 identical instances using each of the evaluated algorithms, the same experiments are repeated four times, where in each time 5 identical instances of one of the instance types defined in Table 11. Fig. 50(a)–(j) shows the tradeoffs between the makespan and cost results coming of running these experiments. The results plots indicate that IWDC either outperforms the five other algorithms in most cases, or at least equals to them. However; the enhancements are not clearly significant as the IWDC algorithm adopts the principle of choosing the best path among a set of paths and assigns it to the most efficient VM. Since, in these experiments, we use identical VMs, there will be no enough chance for the algorithm to show its efficiency.

7.2.2. Experiment of scheduling workflows on heterogeneous instances

The aim of these experiments is to evaluate the efficiency of IWDC algorithm when it is used to schedule workflows on a set of heterogeneous VMs. In these experiments, we first defined 6 different groups of VMs (G1–G6) with instances types as defined in Table 12. Each of these groups represents a heterogeneous set of rented VMs instances on which the evaluated algorithms schedule the workflows. These groups, as indicated in the table, are different in both the number and the types of the VM instances. For evaluation, all the algorithms are applied over each individual group, where in each experiment the makespan is measured and the cost is calculated. Next, we analyze these collected values to evaluate the effect of the different VMs groups on the efficiency of the scheduling algorithms and to evaluate the tradeoffs between cost and makespan of the IWDC and compare it with the other algorithms.

7.2.3. Effect of the VMs instances configurations on the IWDC and the other algorithms

The radial charts drawn in Fig. 51(a)–(h) show the makespan and cost values resulted from applying all the 6 different algorithms on all the 6 different heterogeneous configurations (G1–G6) used in the experiments, where each radial chart has 6 axes; one for each configuration (G1–G6). These charts show that, for most configurations, IWDC data points are the closest to the center, which indicates that IWDC outperforms or at least equal to the best of the other algorithms in scheduling the tested workflows in both the cost and the makespan; especially for the configurations G1, G2 and G3, This stable behavior is not the same for the other algorithms, e.g. MAX-MIN algorithm causes the worst makespan values for the Epigenomics workflow, whereas it causes the best makespan values for the sipht100 workflow. This shows that the IWDC keeps its high efficiency in comparison with the other algorithms irrespective of the different VM instances configurations. The charts show as well that G4,

G5 and G6 result in less makespan than G1, G2 and G3; however, in terms of performance, G4, G5 and G6 cause higher costs as they use more number and/or expensive VM instances. For example, G2 configuration causes the longest makespan and the least cost for most tested workflows, which is expected due to using the expensive **c1.xlarge** VM instant type, which is very highly efficient, out of 5 VMs, whereas G6, which has 10 VMs, causes relatively low makespan but costs the highest cost. However, gaining the best makespan values does not necessary mean paying the highest costs or renting large number of VM instances, for example, G4 cause the lowest makespan values, but it does not cost the highest costs nor it uses the largest number of VMs.

7.2.4. Cost-makespan tradeoff evaluation

From the above, it is seen that the IWDC has a stable behavior irrespective of the chosen configurations but there is a tradeoff between the cost and the makespan which makes choosing the best configuration is a challenge that requires careful consideration. In order to analyze this challenge, the makespan against cost measurements are drawn in Fig. 52 to show the tradeoffs between the makespan and the cost for the different algorithms on the 6 different configurations [G1–G6].

Fig. 52(a)–(f) shows the results of the experiments of running the 6 algorithms to schedule 6 different groups using the 6 different groups of VMs defined above. The first impression of the figure is that IWDC shows clear enhancements in both the performance and the cost over the other algorithms; the enhancement here is even clearer than the enhancement in the homogenous case as the line joining the data points in this graph in most cases is close to the left-bottom corner, which indicates that IWDC give the best tradeoffs, i.e. when it is compared with the other algorithms; it gives the lowest makespan for the paid cost and in the same time it requires the lowest cost to achieve a certain makespan. This was expected, as mentioned before, because the variation in the VM specifications in each group gives the algorithm a better chance to show its efficiency in choosing the best execution paths for the workflows' tasks. It is also noticed, from the figure that the chosen VM specifications affect the performance and the cost significantly, as there is a large gap between the lowest cost (by running on G3) and the highest cost (by running on G6) and in another large gap between the lowest makespan (scored on G4) and the highest makespan (scored on G1). Another interesting result deduced from the figure is that running IWDC on the VMs of G4 (shown in red color) gives always the best makespan to a relatively low paid cost, in G4 there are 10 low-priced VMs instances with relatively low specifications. This shows an important feature of the cloud computing, as it indicates that there is no need to pay for VM instances with high specifications, as renting more low-specifications VMs in many cases can give relatively high performance.

7.2.5. The effect of the rented VM instance on the performance

In order to see the effect of the chosen VMs instances on cost and makespan time, we made a set of experiments in which we apply the IWDC algorithm to schedule the execution of a set of workflows on 5 homogeneous VM instances, where in each experiment we used VMs of a specific instance type to schedule the same workflows dataset. We used the 4 types of VM instance types presented earlier in Table 11; c1.xlarge, c4.large, m1.small, t1.micro. The results of these experiments are shown in Table 13 and drawn in Fig. 53. We notice that for Montage100, Epignomics100, Sipht, Inspiral100, CyberShake workflows, the t1.microresource type had the worst makespan but with lowest cost. This is not surprising given its relatively low capabilities. On the other hand, the

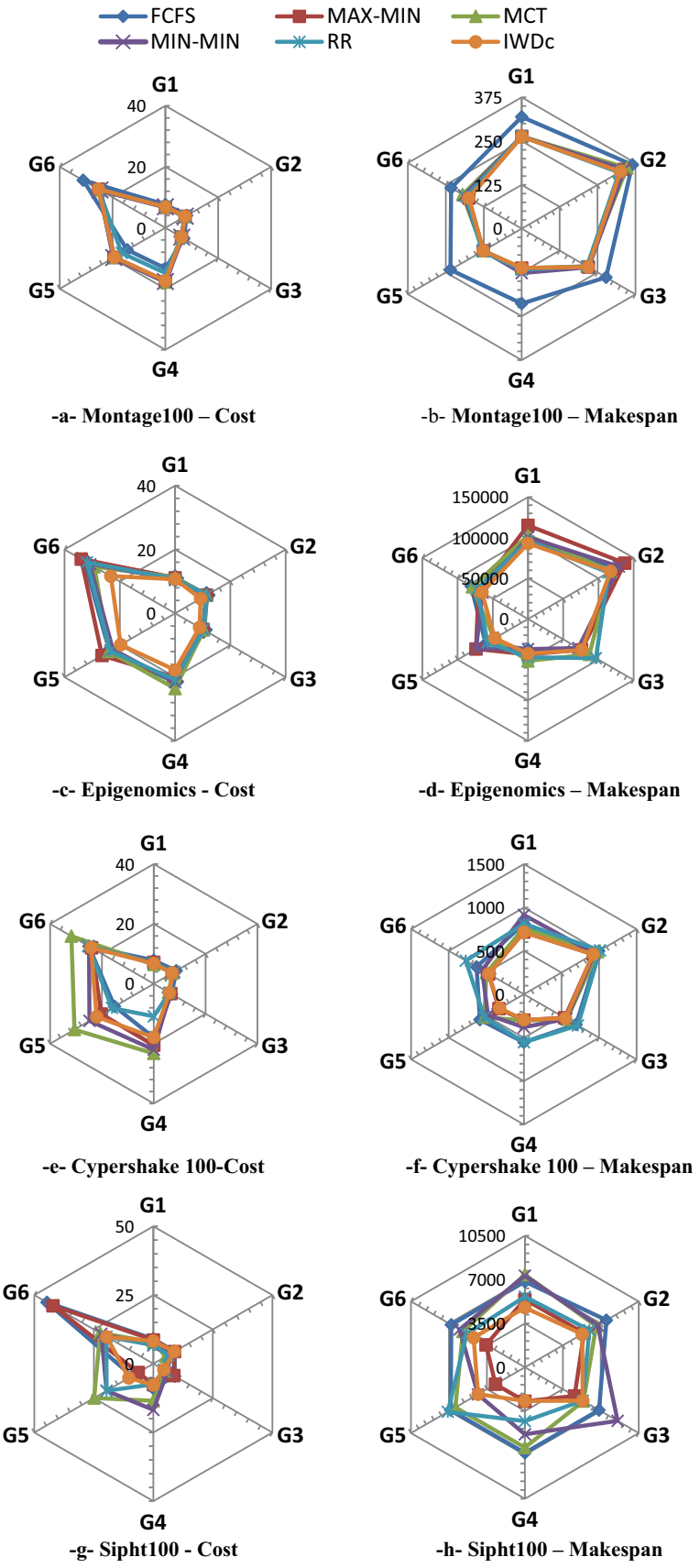


Fig. 51. The radial charts of the Makespan and cost values of the 6 different VM groups.

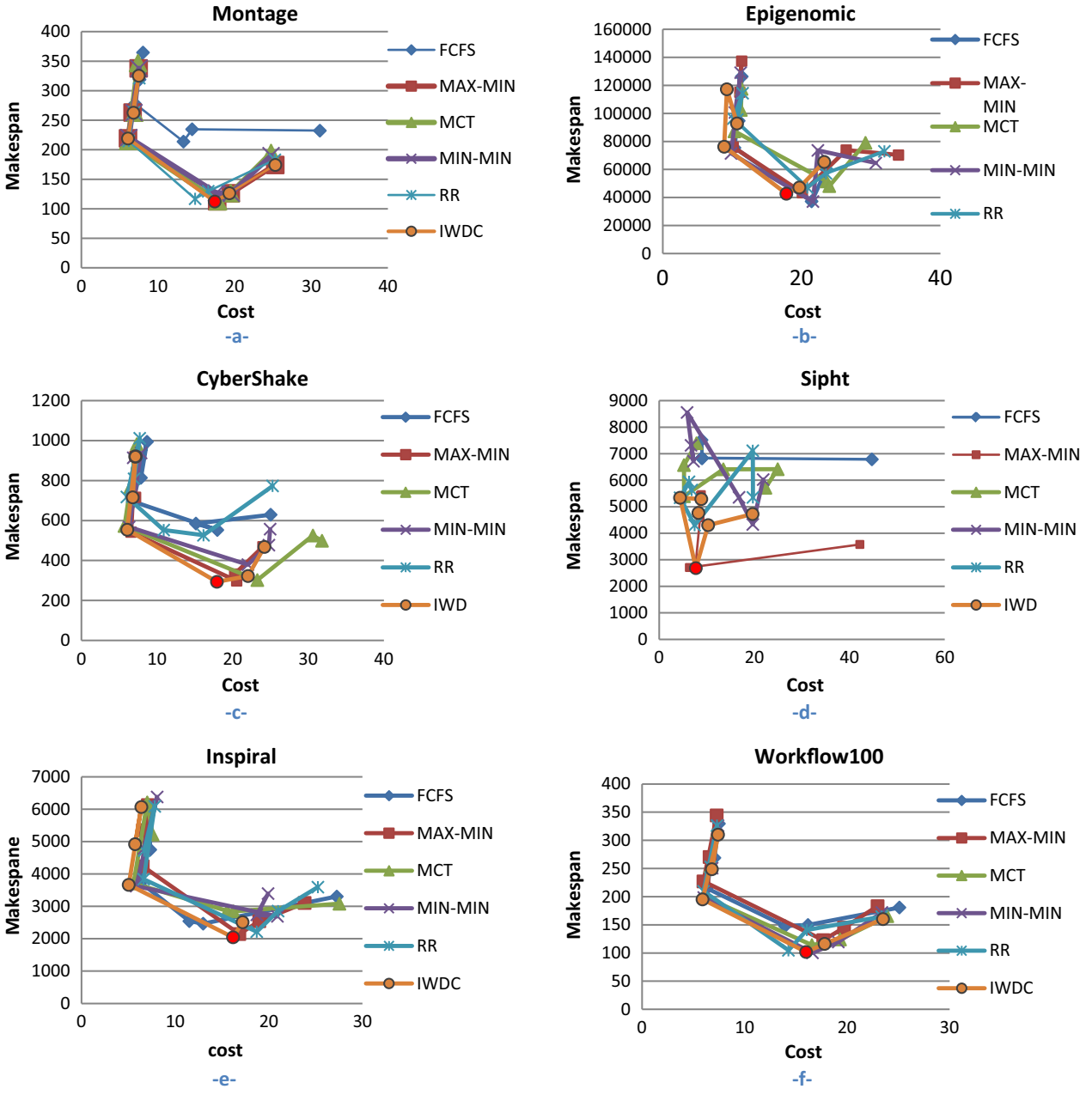


Fig. 52. Results of COST-Makespan tradeoff evaluation.

Table 13
Cost and Makespan values using various instant types.

	Montage		Epigenomics		CyberShake		Sipht		Inspiral	
	Cost	Makespan	Cost	Makespan	Cost	Makespan	Cost	Makespan	Cost	Makespan
c1.xlarge	52.52	102.7	52.52	34218.33	52.52	259.45	50.96	1892.6	52.52	1779.22
c4.large	10.1	128.29	14	42772.84	10.1	324.06	9.8	2365	10.1	2212.66
m1.small	4.44	256.45	4.44	91670.95	4.44	647.12	4.31	4731.4	4.44	4425.2
t1.micro	2.02	512.81	6.28	180607.94	2.02	1293.23	2.04	9462.7	2.02	8850.27

best EC2 performance was achieved on the c1.xlargefor all the workflows. This is likely due to the fact that c1.xlargehas twice as much memory as the next VM instance type. This extra memory helps the Linux kernel to reduce the amount of time the application spends waiting for I/O. However, the enhanced makespan values for using c1.xlarge instead of c4.large equal 25%, while they require additive paid cost between (275% and 420%), which is a very high cost compared with the gained speed. Hence, for non critical workflows in which the makespan minimization is required but with an acceptable cost, using c4.large is the best choice. On the other hand, the same figure shows that t1.micro can be the best choice for workflows that do not need high speed of execution, but it does have high change effect on the running cost. However, using m1.small can reduce the makespan time to half its value by just

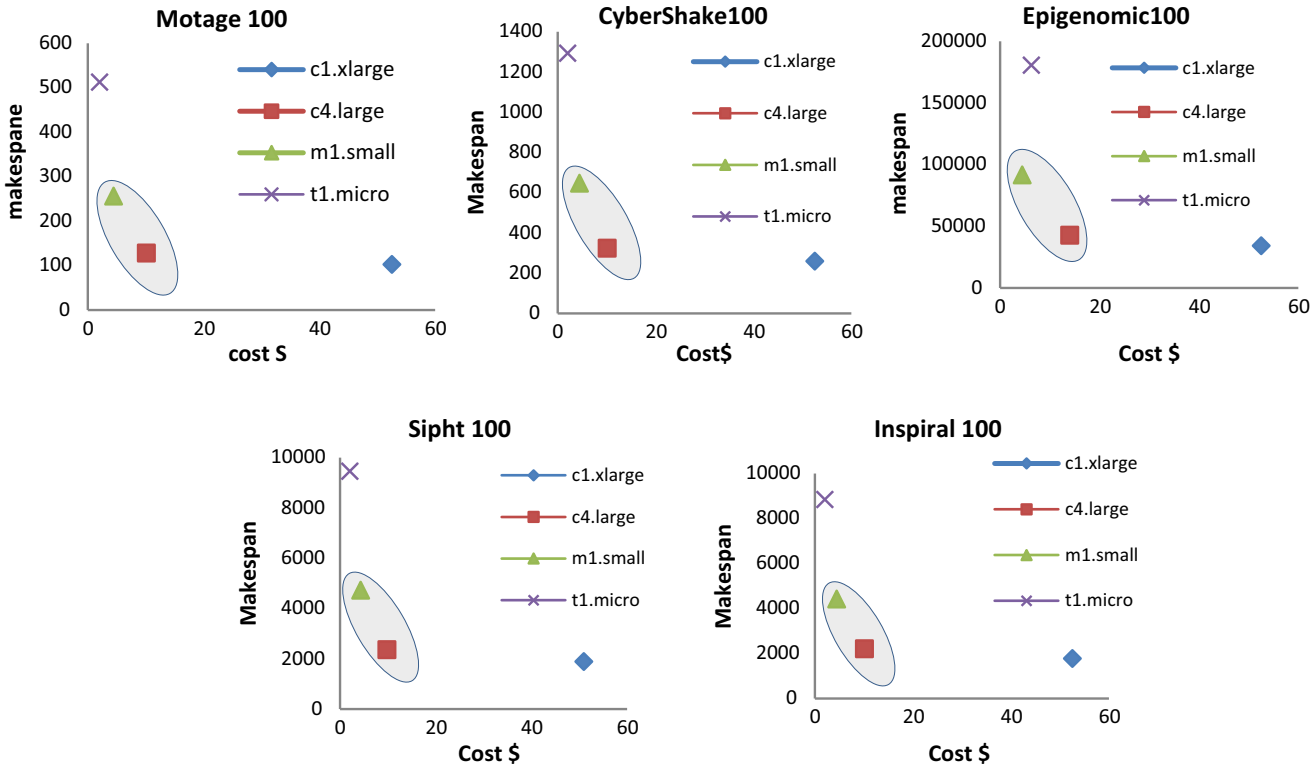


Fig. 53. Cost against Makespan using various instant types.

paying around the double price which is fair enough for people which want to balance between the speed and the cost. Hence, from these results, it can be seen that using m1.small and c4.large instance types can be the best choice for scheduling many workflows, where m1.small can be better for workflows that have the cost as a main concern over the makespan, while c4.large instance type is better for workflows that have the makespan as a main concern over the cost.

8. Conclusion

This research study focuses on developing effective workflow scheduling algorithm based on meta-heuristics. This research study presents IWDC algorithm for scheduling workflows on the cloud, which is an extension of the meta-heuristic IWD algorithm. The original IWD algorithm is adapted and extended and applied to solve workflow scheduling in this research. The performance of the IWDC algorithm was evaluated by embedding it in the workflowSim simulator and applying it on a set of workflows of different types and sizes and the results were compared with the results of other existing cloud scheduling algorithm; MIN-MIN, MAX-MIN, Round-Robin, FCFS and MCT. The experiments were conducted on different simulated cloud environments with different cost models. Results show that the proposed IWDC algorithm performance outperforms the other algorithms in most cases. Also, the costs of using IWDC are more economic than the other algorithms. In general, in comparison with other algorithms, IWDC shows stable behavior and reasonable costs with most workflows irrespective of the structure of the workflow. It is noticed also, that when applying the commercial Amazon EC2 cost models, the degree of cost saving of the IWDC over the other algorithms are less noticed than when some non-realistic cost models are applied, this is because Amazon EC2 ignores the cost of files transfer within the cloud to avoid the complexity of cloud management, while the other tested models take it in consideration. It is also noticed that IWDC outperforms the other algorithms more clearly in case of renting a heterogeneous set of VMs instead of homogeneous VMs, because the strength of IWDC algorithm is in its ability to assign the best VM to execute the tasks on a certain workflow's path. However, using homogeneous VMs makes no preference of any VM over the others and reduces the strength of the proposed algorithm over the other algorithms. IWDC showed its high performance and efficiency not only against heuristic algorithms, but also it provided better results over some meta-heuristic algorithms, e.g. PSO and C-PSO, although it is relatively simpler than them and requires less computation. It is also noticed that the efficiency of VM instance types affects the performance and the cost of scheduling the workflows, but this effect is not linearly proportional with the performance of the chosen VM, as very expensive VMs enhances the performance with a ratio that is much less than the ratio of the increase in the cost of renting them, while renting many moderate VMs can give a competitive performance with a much less cost. In the future, we plan to replace the IWD implementation used in this paper by an implementation of the improved IWD algorithm presented in [38] as it is expected to give better optimization. Also, we aim to make enhancements to the algorithm to optimize additional objectives, e.g. the energy usage of the resources.

References

[1] Pop F et al. Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments. In: 2013 27th international conference on advanced information networking and applications workshops (WAINA). IEEE; 2013.
[2] Kwok Y-K, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv (CSUR) 1999;31(4):406–71.
[3] Mao M, Humphrey M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis, ACM; 2011.

[4] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM 2008;51(1):107–13.
[5] Sfrent A, Pop F. Asymptotic scheduling for many task computing in big data platforms. Inf Sci 2015;319:71–91.
[6] Shah-Hosseini H. Problem solving by intelligent water drops. In: IEEE congress on evolutionary computation; 2007.
[7] Niu S, Ong S, Nee A. An improved intelligent water drops algorithm for achieving optimal job-shop scheduling solutions. Int J Prod Res 2012;50 (15):4192–205.
[8] Noferesti S, Shah-Hosseini H. A hybrid algorithm for solving steiner tree problem. Int J Comput Appl 2012;41(5):14–20.
[9] Agarwal K, Goyal M, Srivastava PR. Code coverage using intelligent water drop (IWD). Int J Bio-Inspired Comput 2012;4(6):392–402.
[10] Dadaneh BZ, Markid HY, Zakerolhosseini A. Graph coloring using Intelligent water drops algorithm. In: 2015 23rd Iranian conference on electrical engineering (ICEE), IEEE; 2015.
[11] Lua R, Yow KC. Mitigating ddos attacks with transparent and intelligent fast-flux swarm network. IEEE Network 2011;25(4):28–33.
[12] Shah-Hosseini H. Intelligent water drops algorithm: a new optimization method for solving the multiple knapsack problem. Int J Intell Comput Cybern 2008;1(2):193–212.
[13] Kamkar I, Akbarzadeh-T M-R, Yaghoobi M. Intelligent water drops a new optimization algorithm for solving the vehicle routing problem. In: 2010 IEEE international conference on systems man and cybernetics (SMC), IEEE; 2010.
[14] Duan H, Liu S, Lei X. Air robot path planning based on intelligent water drops optimization. In: IEEE international joint conference on neural networks, 2008. IJCNN 2008. (IEEE world congress on computational intelligence), IEEE; 2008.
[15] Shah-Hosseini H. Intelligent water drops algorithm for automatic multilevel thresholding of grey-level images using a modified Otsu's criterion. Int J Model Ident Control 2012;15(4):241–9.
[16] Hoang DC, Kumar R, Panda SK. Optimal data aggregation tree in wireless sensor networks based on intelligent water drops algorithm. IET Wireless Sens Syst 2012;2(3):282–92.
[17] Rayapudi SR. An intelligent water drop algorithm for solving economic load dispatch problem. Int J Electr Electron Eng 2011;5(2):43–9.
[18] Kayvanfar V, Zandieh M, Teymourian E. An intelligent water drop algorithm to identical parallel machine scheduling with controllable processing times: a just-in-time approach. Comput Appl Math 2015:1–26.
[19] Chen W-N, Zhang J. An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. IEEE Trans Syst Man Cybern Part C: Appl Rev 2009;39(1):29–43.
[20] Pandey S et al. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 2010 24th IEEE international conference on advanced information networking and applications (AINA), IEEE; 2010.
[21] Hu X-H et al. An IPSO algorithm for grid task scheduling based on satisfaction rate. In: International conference on intelligent human-machine systems and cybernetics, 2009. IHMSC'09, IEEE; 2009.
[22] Xu M et al. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In: 2009 IEEE International Symposium on parallel and distributed processing with applications, IEEE; 2009.
[23] Wu Z et al. A revised discrete particle swarm optimization for cloud workflow scheduling. In: 2010 international conference on computational intelligence and security (CIS), IEEE; 2010.
[24] Lin C, Lu S. Scheduling scientific workflows elastically for cloud computing. In: 2011 IEEE international conference on cloud computing (CLOUD), IEEE; 2011.
[25] Wu Z et al. A market-oriented hierarchical scheduling strategy in cloud workflow systems. J Supercomput 2013;63(1):256–93.
[26] Bilgaiyan S, Sagnika S, Das M. Workflow scheduling in cloud computing environment using cat swarm optimization. In: 2014 IEEE international advance computing conference (IACC), IEEE; 2014.
[27] Vasile M-A et al. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. Future Gener Comput Syst 2015;51:61–71.
[28] Berriman GB et al. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In: SPIE astronomical telescopes+ instrumentation. 2004: international society for optics and photonics.
[29] Juve G et al. Characterizing and profiling scientific workflows. Future Gener Comput Syst 2013;29(3):682–92.
[30] Laser Interferometer Gravitational Wave Observatory (LIGO). Available from: <https://pegasus.isi.edu/application-showcase/ligo/> [accessed on 16-5-2017].
[31] USC Epigenome Center. Available from: <http://epigenome.usc.edu> [last accessed on 15th May 2017].
[32] Graves R et al. CyberShake: a physics-based seismic hazard model for Southern California. Pure Appl Geophys 2011;168(3–4):367–81.
[33] SIPHT. <http://pegasus.isi.edu/applications/sipht.Last> [accessed on 15th May 2017].
[34] Deelman E et al. Pegasus, a workflow management system for science automation. Future Gener Comput Syst 2015;46:17–35.
[35] Amazon EC2. <http://www.aws.amazon.com/ec2/pricing/> [Last accessed on 15th May 2017].
[36] Chuang L, Tsai S, Yang C. Catfish particle swarm optimization. In: Proceedings of the IEEE swarm intelligence symposium; 2008. http://dx.doi.org/10.1109/SIS.2008.4668277.
[37] Nirmala SJ, Bhanu SMS. Catfish-PSO based scheduling of scientific workflows in IaaS cloud. Computing 2016;98(11):1091–109.
[38] Niu S, Ong S, Nee AY. An improved intelligent water drops algorithm for solving multi-objective job shop scheduling. Eng Appl Artif Intell 2013;26 (10):2431–42.