Cluster Computing
https://doi.org/10.1007/s10586-020-03079-1

Fair budget constrained workflow scheduling approach for heterogeneous clouds

Naela Rizvi[1] · Dharavath Ramesh[1]

Received: 4 February 2019 / Revised: 16 February 2020 / Accepted: 24 February 2020

Abstract
The phenomenal advancement of technology paved the way for the execution of complex scientific applications. The emergence of the cloud provides a distributed heterogeneous environment for the execution of large and complex workflows. Due to the dynamic and heterogeneous nature of the cloud, scheduling workflows become a challenging problem. Mapping and assignment of heterogeneous instances for each task while minimizing execution time and cost is a NP-complete problem. For efficient scheduling, it is required to consider various QoS parameters such as time, cost, security, and reliability. Among these, computation time and cost are the two notable parameters. In order to preserve the functionalities of these two parameters in heterogeneous cloud environments, in this paper, a fair budget-constrained workflow scheduling algorithm (FBCWS) is proposed. The novelty of the proposed algorithm is to minimize the makespan while satisfying budget constraints and a fair means of schedule for every task. FBCWS also provides a mechanism to save budget by adjusting the cost-time efficient factor of the minimization problem. The inclusion of a cost-time efficient factor in the algorithm provides flexibility to minimize the makespan or save budget. In order to validate the effectiveness of the proposed approach, several real scientific workflows are simulated, and experimental results are compared with other existing approaches, namely; Heterogeneous Budget Constrained Scheduling (HBCS), Minimizing Schedule Length using Budget Level (MSBL) and Pareto Optimal Scheduling Heuristic (POSH) algorithms. Experimental results prove that the proposed algorithm behaves outstandingly for compute-intensive workflows such as Epigenomic and Sipht. Also, FBCWS outperforms the existing HBCS in most of the cases. Moreover, FBCWS proves to be more time-efficient than POSH and more cost-efficient than MSBL. The effectiveness of the proposed algorithm is illustrated through the popular ANOVA test.

Keywords Workflow scheduling · DAG · Budget constraints · Makespan · Heterogeneous clouds

1 Introduction

Workflows have been widely used to model complex scientific and business applications. The evolution of these complex scientific applications with an urge of large scale computing and storage services demanded the design of a high-performance computing system. Therefore, the distributed and heterogeneous environments, like grids and clusters, have gained momentum in terms of computational instances to execute these complex scientific applications [1, 2]. Workflow applications consist of interdependent modules executed in a grid environment. However, with the emergence and the development of cloud infrastructure, users migrated their applications on the cloud to perform the operations in an integrated manner [3]. Cloud's 'pay per use' mechanism with the inclusion of the elasticity and scalability paradigm has invariably attracted many customers [4–6]. Users have to pay only for what they use.

Further, the cloud provides various services that can be categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [6]. IaaS provides hardware resources like memory, network,

✉ Dharavath Ramesh
drramesh@iitism.ac.in

Naela Rizvi
naela.17dr000330@cse.ism.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology (ISM), Dhanbad, Jharkhand 826004, India

computation, and storage in the form of virtual resources over the internet. PaaS provides the platform or the environment for the development and deployment of the applications. On the other hand, SaaS provides software/applications over the internet. This manuscript focuses on the IaaS clouds that offer the virtual resources of different characteristics on demand. Moreover, IaaS provides the flexibility of acquiring and releasing of the resources when required. As a result, this empowers the users to have more control over these resources and also dictates the development of the new scheduling strategy such that these resources are efficiently utilized with the desired optimal solutions.

Of late, many researchers pointed out the benefits of executing complex applications on an IaaS cloud [7, 8]. For the execution, the scientific applications are modeled as the workflows defined by a Direct Acyclic Graph (DAG). A DAG can be further decomposed into a collection of several dependent and parallel tasks. Scheduling of workflows includes mapping and assignment of virtual machines (VMs) to independent tasks. Execution of these workflows on the cloud environment is always constrained to some Quality of Services (QoS) parameters. Most of the applications have a deadline or budget constraints, which means execution must be performed within the given budget or given deadline constraints. The workflow scheduling in a heterogeneous environment becomes more challenging when the two QoS parameters (namely time and cost) are considered simultaneously. Therefore, a workflow scheduling problem falls within the category of a NP-Complete problem [9]. However, researchers have investigated various approaches to schedule workflows with minimal execution time and a reduced cost while satisfying the budget or deadline constraints.

Minimizing the execution time while satisfying budget constraints is not a trivial task. Although many notable works in this field have been done, they consider the homogenous environments only [10]. Arabnejad et al. [11] presented Heterogeneous Budget Constrained Scheduling (HBCS) algorithm for scheduling the budget-constrained workflows in a heterogeneous environment. However, the HBCS algorithm satisfies the budget constraint but generates an unfair schedule for the low priority tasks. Low priority tasks tend to select the cheapest processor, which may produce a schedule of longer makespan. Here, makespan is the time taken by the computing resources for the execution of all the tasks of the workflow. The proposed algorithm uses the same budget distribution method of HBCS and also analyses low priority tasks to avoid unfairness. On the other hand, in order to produce minimal makespan, Chen et al. [12] proposed MSBL algorithm. In this approach, the budget constraint of an application is converted to the budget level of each task; however, proper analysis of cost reduction has not been performed. FBCWS also reduces the cost of execution by setting lower time cost factor value for the defined minimization problem and ignoring the expensive VMs. Su et al. [15] introduce the POSH (Pareto Optimal Scheduling Heuristic) algorithm to generate a cost-effective schedule. In contrast, their algorithm ignores the minimization of makespan, which is one of the primal parameters of scheduling.

In order to address the mentioned issues, this paper considers the scheduling of budget-constrained scientific workflows in heterogeneous cloud environments. The objective of the proposed approach is to generate a fair scheduling strategy for workflows that satisfies the budget constraints as well as minimize the makespan of the schedule. The contribution of this paper is mentioned as follows.

- A fair budget constrained scheduling algorithm is presented for heterogeneous cloud environments.
- The makespan of the schedule has been minimized while satisfying budget constraints.
- FBCWS provides a fair means of scheduling for every task by categorizing tasks and considering VM assignment as a multi-criteria minimization problem.
- A mechanism to save cost has been introduced by adjusting low cost-time factor for a multi-criteria minimization problem.
- Extensive simulations have been performed to assert the better performance of FBCWS over existing algorithms, namely, HBCS, MSBL, and POSH.
- An ANOVA test has been performed to check the statistical significance of the proposed algorithm.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the problem definition including the system model. Section 4 analyses the existing model and the proposed model. Section 5 disseminates the performance analysis and experimental results. Section 6 illustrates the conclusion and future work.

2 Related work

The scheduling of workflows in a heterogeneous environment has gathered tremendous attention from the research community. A good number of methodologies have been proposed to provide suitable solutions for NP complete problems. Related studies for scheduling workflows can generally be classified as single objective workflow scheduling, multi-objective workflow scheduling, and QoS constrained scheduling. This section describes some significant works falling in these three categories.

2.1 Single objective workflow scheduling (SOWS)

A number of single objective workflow scheduling approaches have been presented in various forms. In SOWS, the algorithm minimizes only one objective, either execution time or cost. Two novel algorithms named Heterogeneous Earlier Finish Time (HEFT) and Critical-Path-on-a-processor (CPOP) algorithm for application workflow are proposed by Topcuouglu et al. [13]. HEFT selects the tasks based on an upward rank value. On the other hand, CPOP uses a summation of upward and downward rank values for selection. Both algorithms achieve faster execution time and good performance. Bittencourt et al. [14] proposed a cost optimization algorithm for scheduling the workflows in a hybrid cloud. This algorithm resolves the problem of what resources to be leased from a public cloud for aggregation in a private resource pool. Su et al. [15] utilize heuristic strategies to introduce two cost-efficient task scheduling strategies. The first one is based on Pareto dominance, while another algorithm tries to reduce the cost of non-critical tasks. A cost-saving heuristic algorithm considering both the computation cost as well as the communication cost of the workflow was proposed by Pandey et al. [16]. Li et al. [17] introduce a cost-efficient, coordinated scheduling mechanism for the hybrid workload. On the other hand, a low-cost rescheduling policy for executing workflows on the grid is introduced by Sakellariou et al. [18].

2.2 Multi-objective workflow scheduling (MOWS)

Fard et al. [19] introduced a multi-objective algorithm by considering factors like cost, reliability, energy consumption, and makespan. This algorithm applies the strategy of maximizing and minimizing the distance of a constrained vector. Moreover, this approach outperforms the existing bi-criteria heuristic and bi-criteria genetic algorithm. Zhu et al. [20] overcomes the difficulty of scheduling workflows in the cloud and introduces an evolutionary multi-objective optimization (EMO) for optimizing the cost and makespan. Pareto-based list scheduling called MOHEFT for commercial EC2 clouds was proposed by Durillo et al. [21]. Zhang et al. [22] exploited the concept of ordinal optimization for optimizing QoS parameters to generate suboptimal schedules for multitasking workflows in the cloud environment. Tan et al. [23] proposed a balanced policy algorithm considering time, cost, and trust parameter. Talukder et al. [24] introduced a multi-objective differential evolution algorithm for scheduling workflows on the grid.

2.3 QoS constrained workflow scheduling

Abrishami et al. [25] introduced two algorithms for workflow scheduling named; (i) IaaS Cloud Partial Critical Paths(IC-PCP) and (ii) IaaS Cloud Partial Critical Paths with Deadline Distribution (IC-PCPD2). IC-PCPD2 minimizes the total cost under given deadline constraints. Both the algorithms are suitable for executing large workflows, where IC-PCPD2 shows better performance than IC-PCP. Ghafouri et al. [26] introduced an algorithm named Constrained Budget-Decreased Time (CBDT). This algorithm combines the idea of heuristic backtracking, scheduling of critical and non-critical paths in a combined manner. A Budget Deadline Aware Scheduling (BDAS) algorithm was proposed by Arnebjad et al. [27] for e-science workflows while satisfying both the deadline and budget constraints. This work introduces a time–cost tradeoff for a heterogeneous environment. A simple budget and deadline constrained algorithm were developed by Sun et al. [28] for the execution of random and real workflows. This methodology outperforms other algorithms by achieving a higher planning success rate (PSR) value. Here, the PSR is the ratio between the number of successful schedules (schedule satisfying the given constraints) and the total number of simulation runs. Another variant of budget-constrained algorithm was proposed by Yu et al. [29] that utilizes a genetic algorithm for optimizing execution time while satisfying the budget criteria. However, this approach was further extended by the inclusion of deadline factor [30]. A methodology presented in [31] introduces a deadline-division-based algorithm (DET) for cost optimization. A cost-based scheduling algorithm that minimizes the cost while satisfying user deadline was proposed by Yu et al. [32]. A heuristic approaches considering both the budget and deadline constraints of workflows was proposed in [33]. Malawksi et al. [34] proposed algorithms for both task scheduling and resource provisioning by evaluating the performance through a broad range of deadlines and budget parameters. Poola et al. [35] proposed robust scheduling for workflows considering both deadline and budget constraints. Zheng et al. [36] proposed a feasible plan for the execution of workflows that considers time and cost constraints. Arabnejad et al. [37], Calheiros et al. [38], and Abrishami et al. [25] only considered deadline factor, while Yang et al. [39] and Rodriguez et al. [40] focuses on cost-constrained applications. Alkhanak et al. [41] provide an extensive review of the cost-aware scheduling technique.

3 Problem definition

3.1 Workflow execution model in cloud

As illustrated in Fig. 1, the scheduling model in the cloud environment comprises of three layers. (i) the task graph layer (ii) the resource layer and (iii) the cloud infrastructure layer. A task graph is composed of interdependent tasks or workflows. Interconnected virtual machines (VMs) form the resource layer and cluster of computers connected with network supports cloud infrastructure [10]. We use scientific workflows in the task graph and perform one to one mapping. Each task of a workflow is scheduled on different VMs for execution. A workflow is a collection of dependent tasks represented in the form of Directed Acyclic Graphs (DAG), which is a graph with no cycles. The task is represented by nodes, while dependencies between tasks are represented by the edges. Each dependency $E(t_i,t_j)$ represents a precedence constraint that indicates t_j (child node) can only be executed after t_i (parent node) completes its execution.

The target platform consists of a single cloud environment with various heterogeneous VMs of different characteristics. Let $V = \{VM_1, VM_2, VM_3 \ldots VM_n\}$ denotes the VM set. An application workflow runs on a set of VMs takes different execution time. A DAG is modeled by a set of tuples $G\ (T, E)$, where T is a set of nodes, and each $t_i \varepsilon T$ is an application task. E is a set of edges where each edge denotes communication or data transfer time between dependent tasks. The weight given to nodes and edges are the task computation and communication time. When the dependent tasks are placed on different VMs, then data transfer or communication time can be estimated by the size of output data file. This file is to be transferred divided by the average bandwidth; otherwise, data transfer time is zero when dependent tasks are placed on the same VM. In a given DAG, a task with no predecessor is an entry task, whereas a task with no successor is an exit task. We assume that the DAG has a single entry and single exit task. If there exist multiple entries or exist tasks, then a dummy entry or exit task having zero-weight dependencies is added to the DAG.

Figure 2 shows a sample of application workflow with ten tasks. Table 1 provides the execution time of ten tasks in three different VMs $\{VM_{1,}\ VM_{2,}\ VM_3\}$. As shown in Fig. 2, the weight of an edge determines the communication or data transfer time. For example, $C_{1,\ 2} = 18$ means the data transfer time between t_1 and t_2 is 18 if they are placed on different VMs.

3.2 Cost model and budget constraint

The cost model is based on a '*pay-per-use*' mechanism. Users have to pay for only that period of time they used VMs. Each VMs have a different base price because of their heterogeneous nature. The base price is the unit price for which VMs are charged for use at a particular time. Let p_vm_j is the base price of jth VM. Usually, the fastest VM is priced higher. However, this case is not always valid in heterogeneous environments. The proposed model considers that all the computational and storage services are placed on the same physical region and therefore, the average bandwidth between the shared VMs and storage services is assumed to be equal. Hence, the communication time is only depending on the data transfer time between the dependent tasks when these tasks are placed on different VMs. As most of the cloud providers, only charge for the amount of the resources being used. Therefore, we assume that the cost of a VM is determined only by the computation time for which the VM is being used. Furthermore, it is assumed that the internal data transfer is free, as most of the real cloud providers do not charge for the internal data transfer.

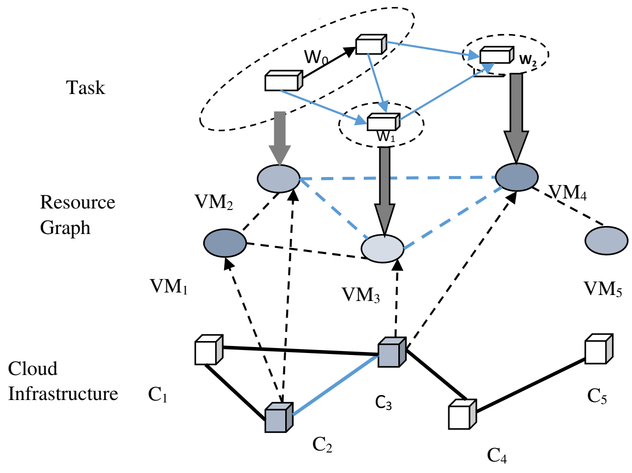


Fig. 1 Workflow execution model in cloud [10]

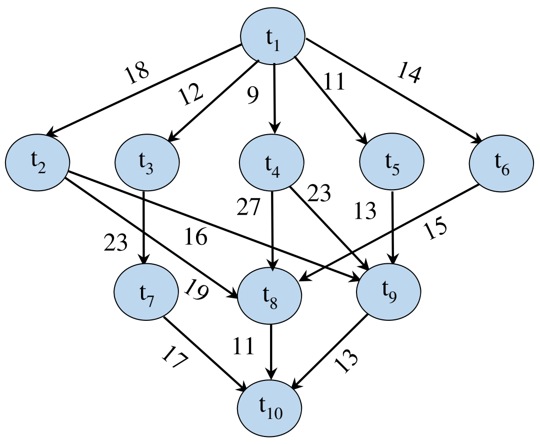


Fig. 2 A sample DAG of 10 task

Table 1 Execution time of tasks on different VMs

t_i	VM_1	VM_2	VM_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Accordingly, to normalize the diverse cost of heterogeneous VMs, we define the price of jth VM executing task t_i as;

$$Cost(t_i, vm_j) = p\_vm_j \times et_{i,j} \tag{1}$$

$$Cost(G) = \sum_{i=1}^{|T|} p\_vm_{f(i)} \times et_{i,f(i)} \tag{2}$$

where, $f\,(i)$ is the index of VM assigned to task t_i, $et_{i,f(i)}$ is the execution time of task t_i on f(i) VM and $p\_vm_{f(i)}$ is base price of $vm_{f(i)}$ [12]. Table 2 shows the different notations and symbols used in this paper.

The execution of every task on each VM provides the minimum cost as well as the maximum cost. The minimum cost of execution, i.e. $Cost_{min}(G)$ is the choice of least expensive VMs for execution wherein $Cost_{max}(G)$ is the summation of costly VMs for execution, respectively.

$$Cost_{\min}(G) = \sum_{i=1}^{|T|} Cost_{\min}(t_i) \tag{3}$$

$$Cost_{\max}(G) = \sum_{i=1}^{|T|} Cost_{\max}(t_i) \tag{4}$$

The cost budget must be greater than or equal to $Cost_{min}(G)$ and smaller than or equal to $Cost_{max}(G)$. Otherwise, the budget lying outside the defined range is not valid. Hence, our model assumes:

$$Cost_{\min}(G) \le Cost_{bg}(G) \le Cost_{\max}(G) \tag{5}$$

3.3 Problem formulation

The objective of the proposed model is to design an approach for assignment of tasks on appropriate VMs such that makespan of a DAG in a user-defined budget constraint is minimized. Makespan is the time at which the execution of the last task finishes. Makespan is determined only when the exit task finishes its execution. This is defined as follows:

$$makespan = AFT(t_{exit}) \tag{6}$$

where AFT is the Actual Finish time of exit task. In brief, the objective is to minimize the makespan under budget constraints. This is defined as;

$$Cost(G) = \sum_{i=1}^{|T|} Cost(t_i, vm_{f(i)}) \le Cost_{bg}(G) \tag{7}$$

4 Budget constrained and cost-efficient algorithms

This section comprises of two parts. Section 4.1 pertains to study of some existing algorithms namely HBCS [11], MSBL [12] and POSH [15]. HBCS and MSBL are budget constrained scheduling algorithms created with the aim of minimizing the makespan of budget-constrained applications, whereas POSH is the only cost-efficient algorithm. Section 4.2 introduces a fair budget-constrained workflow scheduling algorithm (FBCWS) for heterogeneous cloud environments. Our proposed FBCWS approach produces a fair scheduling strategy and also minimizes the makespan of running applications by satisfying the user-defined budget constraints. FBCWS also provides a mechanism to save budget by considering low-cost time factor value for the defined minimization problem.

4.1 Existing algorithmic approaches

Heterogeneous Budget Constrained Scheduling (HBCS): The objective of HBCS is to provide scheduling with minimum execution time under user-defined budget. Processors are selected according to the highest worthiness value, and tasks are arranged rank wise. The working of HBCS is as follows.

1. HBCS first obtains $Cost_{HEFT}(G)$ and $Cost_{min}(G)$ by invoking the HEFT algorithm.
2. DAGs are scheduled using HEFT if their cost of execution is lower than user-defined budget.
3. Task priorities are assigned using upward rank value. Tasks are selected in descending order of their priority.
4. For each task on every processor, HBCS determines the finish time, cost of execution, cost coefficient, and worthiness value. The worthiness value depends upon the available budget and the finish time of a processor.
5. The processor that has the highest worthiness value is selected for execution, and the desired schedule is obtained.

Table 2 Symbols and notations

Symbols and notations	Meaning
T	Set of tasks
W	Number of tasks in each level
E	Set of edges
V	Set of VMs
t_i	ith task
p_vm_j	Base price of *j*th VM
t_{exit}	Exit task
$et_{i,j}$	Execution time of t_i task on *j*th VM
$Cost(t_i, vm_j)$	Cost of executing t_i task on *j*th VM
$CT(t_x, t_i)$	Communication time between task t_x and task t_i
$Cost(G)$	Total cost of execution of DAG
$Cost_{min}(G)$	Minimum cost of execution
$Cost_{max}(G)$	Maximum cost of execution
$Cost_{bg}(G)$	User defined budget
$Cost_{HEFT}(G)$	Cost of execution using HEFT algorithm
v	Number of virtual machines
l_i	Level number
$ACT(t_i)$	Average computation time of task t_i on different VMs
$AST(t_i)$	Actual start time of task t_i
$pred(t_i)$	Predecessor task of t_i
$AFT(t_i)$	Actual finish time of t_i
β	Cost time factor
$Net_{i,j}$	Normalized execution time of task t_i on *j*th VM
$et_{max}(t_i)$	Maximum execution time of task t_i
$cost_{max}(t_i)$	Maximum cost of executing the task t_i
$avail(j)$	time at which *j*th VM is available for execution

To have a clear conception of HBCS, we illustrate the concept with a motivating example. We consider a DAG with 10 tasks, as illustrated in Fig. 2. We have assumed zero communication costs when tasks are on the same VM. The assumed base price of each VM is shown in Table 3. The minimum and maximum cost of executing DAG is computed using Eqs. (3) and (4), where $\text{Cost}_{\text{min}}(G) = 398$ and $Cost_{max}(G) = 935$ is obtained. The budget constraint of G as $Cost_{bg}(G) = 500$ is set to check the algorithm's efficiency. Table 4 illustrates the assignment of tasks using HBCS with makepan = 92 and $Cost(G) = 459$. This example verifies that HBCS produces the schedule for each task while satisfying budget constraints. But, HBCS proves to be unfair for low priority tasks by distributing more budget to high priority tasks.

Minimizing Schedule Length using Budget Level (MSBL): It introduces the concept of a budget level for each task of the application. The objective of MSBL is to transfer the budget constraints of an application to each task. The algorithm sorts the tasks according to rank values. The processor that has the minimum *EFT* (*Earliest*

Table 3 Assumed base price of VMs

p_vm_j	VM_1	VM_2	VM_3
	7	5	3

Table 4 Task assignment of the workflow using HBCS

Tasks (t_i)	$Cost_{bg}(t_i)$	$AST(t_i)$	$AFT(t_i)$	$f(t_i)$	$Cost(ti, f(t_i))$
t_1	129	0	9	3	27
t_3	159	21	34	2	65
t_4	134	34	42	2	40
t_2	148	27	40	1	91
t_5	87	9	19	3	30
t_6	84	19	28	3	27
t_9	105	56	68	2	60
t_7	57	57	64	1	49
t_8	76	69	74	1	35
t_{10}	76	85	92	2	35

Makepan = 92 and $Cost(G) = 459$

Finish Time) is selected for each task while satisfying individual budget constraints. The core details of MSBL are explained in the following manner.

1. MSBL first computes the minimum cost value of *DAG* and then converts the budget constraint of an application into tasks.
2. The possibility of finding a schedule that satisfies user-defined budget is verified.
3. The algorithm map the tasks on an appropriate processor. At every step, the task having a higher priority is selected, and its budget constraints are determined. The processor that has the minimum *EFT* and satisfying budget level constraint is selected. After the assignment to a processor, the spare budget is updated.
4. At last, Schedule length and cost of execution is computed.

Table 5 shows the assignment of tasks using MSBL having makespan = 87 and cost = 456. MSBL only focuses on minimizing the makespan while neglecting the cost of workflows.

Pareto Optimal Scheduling Heuristic (POSH): The POSH algorithm schedules tasks of an application in the heterogeneous processing environment. Tasks are assigned on cost-efficient VMs based on Pareto dominance. VMs selection is based on both the time and cost factors. The working of the POSH algorithm is divided into three phases:

1. Weighing Phase: This phase assigns the weight to the workflow. Weights assigned to nodes are execution time of tasks, and weights to edges are the communication time between VMs.
2. Prioritizing Phase: Tasks are sorted in descending order of their priority. The priority of each task is computed based on an upward priority value, which is the sum of the node weight and execution time of successors.
3. Mapping Phase: Assignment of task on cost-efficient VMs is based on Pareto dominance. A predefined objective function is used to choose VM to schedule a task at a low cost.

Tables 6 and 7 illustrates the scheduling of tasks on appropriate VMs using POSH algorithm. We have taken α = 0.5 and α = 0.2 to show the efficiency of the algorithm at different α values. POSH proves to be a cost-efficient algorithm but, at the same time, fails to minimize the makespan.

4.2 Proposed algorithm: fair budget constrained workflow scheduling (FBCWS)

In this section, we discuss the proposed algorithm FBCWS, which produces a fair scheduling strategy. FBCWS minimizes the makespan while satisfying the budget constraint of an application. This algorithm also provides the flexibility to save the budget at the same time, which is not possible in other budget-constrained algorithms. FBCWS works in two phases; (i) Task categorization and selection phase and (ii) VM selection phase. Tasks are selected priority wise whereas, VMs are selected according to task type to ensure fairness.

4.2.1 Task categorization and selection phase

To categorize tasks, FBCWS considers two types of tasks, namely, CPU intensive task and normal task. CPU intensive task requires more processing power and processing time than normal task. In order to determine the type of

Table 5 Task assignment of workflows using MSBL

Tasks (ti)	*Cost$_{bg}$(ti)*	*AST(ti)*	*AFT(t_i)*	*f(ti)*	*Cost(ti, f(t_i))*
t_1	40	0	9	3	27
t_3	74	9	28	3	57
t_4	67	18	26	2	40
t_2	89	28	46	3	54
t_5	75	26	39	2	65
t_6	49	46	55	3	27
t_9	94	62	74	2	60
t_7	75	51	58	1	49
t_8	65	55	69	3	42
t_{10}	79	80	87	2	35

Makespan = 87 and cost = 456

Table 6 Task assignment of workflows using POSH considering α = 0.5

Tasks (t_i)	*AST(t_i)*	*AFT(t_i)*	*f(t_i)*	*Cost(t_i, f(t_i))*
t_1	0	9	3	27
t_3	21	34	2	65
t_4	34	42	2	40
t_2	9	27	3	54
t_5	27	37	3	30
t_6	37	46	3	27
t_9	50	62	2	60
t_7	57	64	1	49
t_8	69	74	1	35
t_{10}	85	92	2	35

Makespan = 92 and cost = 422 at α = 0.5

Table 7 Task assignment of workflows using POSH α with = 0.2

Tasks (t_i)	$AST(t_i)$	$AFT(t_i)$	$f(t_i)$	$Cost(t_i, f(t_i))$
t_1	0	9	3	27
t_3	9	28	3	57
t_4	18	26	2	40
t_2	28	46	3	54
t_5	46	56	3	30
t_6	56	65	3	27
t_9	69	81	2	60
t_7	65	76	3	33
t_8	80	85	1	35
t_{10}	96	103	2	35

Makespan = 103 and cost = 398 at α = 0.2

tasks in a workflow, the average computation time of each task is determined in the following manner.

$$ACT(t_i) = \sum_{j=1}^{v} \frac{et_{i,j}}{v} \tag{8}$$

After ACT (t_i) is computed, the mean of average computation time $MACT$ (l_i) at each level of DAG is determined.

$$l(t_i) = \max_{t_x \in pred(t_i)} (l(t_x) + 1) \tag{9}$$

$$MACT(l_i) = \frac{1}{W} \sum_{i=1}^{W} ACT(i) \tag{10}$$

In order to categorize the tasks, workflows are divided level wise and each task is assigned to the respective level without intersection using Eq. 9. Generally, the level of a task in a workflow is determined by its distance from the start node having level number 1. For the exit task, the level number is always one level more than its longest parent. Task can be compute-intensive or normal task, which is decided by the mean of average computation time $MACT$ (l_i) at each level. The value of $MACT(l_i)$ at each level of the DAG distinguishes the task's type. Tasks in a level l_i having higher ACT than $MACT(l_i)$ are CPU intensive tasks. Most Time-Consuming Task List (MTCTL) has been created for CPU intensive tasks while normal tasks are placed in Less Time-Consuming Task List (LTCTL). After task categorization, task selection is done. Task selection is based on the priority of task, which is determined by its B-level value. B-level of task represents the length of the longest path from node t_i to exit node, including all the communication time. Task having higher B-level has higher priority and thus will be selected first. B-level of task is computed as;

$$B\text{-}level = ACT(t_i) + \max_{t_z \in set\ of\ immediate\ succ.(t_i)} \{CT(t_i, t_z) + Blevel(t_z)\} \tag{11}$$

4.2.2 VM selection phase

VM selection is based on budget constraints of each task as well as the type of the task. Setting the budget constraints on each task is determined by the following two quantities namely; Remaining Budget (*RB*) and Remaining cheapest budget (*RCB*). We define *RB* as budget left after scheduling the task and *RCB* as the cheapest cost VMs left for unscheduled tasks excluding current scheduled task [11]. *RCB* is updated every time for each task before scheduling and *RB* is repeatedly updated after VM selected for each task.

$$RCB = RCB - Cost_{\min}(t_i) \tag{12}$$

$$RB = RB - Cost(t_i, vm_j) \tag{13}$$

On the other hand, the user-defined budget is divided among each task to set the budget constraints at each task. Budget constraints on task t_i is defined by;

$$BC(t_i) = RB - RCB \tag{14}$$

where

$$Cost(t_i, vm_j) \leq BC(t_i) \tag{15}$$

Since the tasks in MTCTL require the fastest VMs for execution, VMs having higher processing capacity are selected for MTCTL tasks (Algorithm 1-line no (17–18)). As a consequence, execution time is reduced, leading to the very high cost of execution. In order to meet the budget constraint, the tasks in LTCTL must be scheduled on cost-efficient VMs. However, this step would be beneficial in compensating the higher cost. But, this will be unfair for LTCTL tasks to always schedule on slow VMs that produce longer makespan. Most of the algorithms ignore this aspect and become unfair to low priority tasks. Therefore, to avoid unfair treatment to LTCTL tasks, FBCWS first checks for the VMs, which are cost-efficient as well have faster execution speed (Algorithm 2; line no. 3). If no such VMs are found, then FBCWS considers VM assignment as a multi-criteria minimization problem (Algorithm 2; line no (7–8)). We state the minimization problem with the combination of execution time and cost as an objective function. FBCWS selects those VMs from remaining available VMs that minimize the given function.

$$Minimize : \beta \times Net_{i,j} + (1 - \beta) \times NCost(t_i, vm_j) \quad for\ all\ vm_j \in v \tag{16}$$

Subject to

$$Net_{i,j} = et_{i,j}/et_{\max}(t_i) \tag{17}$$

$$NCost(t_i, vm_j) = Cost(t_i, vm_j)/Cost_{\max}(t_i) \tag{18}$$

$$\beta \in [0, 1] \tag{19}$$

where β is the cost time factor. We take a normalized value of execution time and cost because time and cost may have a different scale, and thus it is not valid to perform linear formulation directly from actual time and cost. For obtaining fair scheduling, we consider higher β value. For the simulation, $\beta = 0.8$ has been set to achieve lower execution time for LTCTL tasks to achieve fairness. For saving the budget, a low β value is always preferred. Both of these cases are illustrated in Tables 10 and 11. After finding appropriate VMs, tasks are scheduled on selected VMs according to (Algorithm 1; line no. 25).

$$EST(t_{entry}, vm_j) = 0 \tag{20}$$

$$EST(t_i, vm_j) = \max\{\underset{j\in v}{avail(j)}, \max_{t_x \in pred(t_i)} \{AFT(t_x) + CT(t_x, t_i)\}\} \tag{21}$$

$$EFT(t_i, vm_j) = EST(t_i, vm_j) + et_{i,j} \tag{22}$$

Here, *EST* denotes the earliest start time of task t_i on assigned *j*th VM, and *EFT* is the earliest finish time of task t_i on assigned *j*th VM. *EST* of entry task is taken zero. *CT* (t_x,t_i) is zero if t_x and t_i will be on the same machine [13]. Whereas, *AST* and *AFT* represent the actual start time and finish time.

Algorithm 1*: Fair Budget Constrained Workflow Scheduling (FBCWS)*

Input: *DAG, execution time of all tasks on different VMs, communication time between VMs, User defined Budget*

Output: *Schedule S, makespan and cost of DAG*

```
1.  Determine the level of DAG and number of tasks in each level (using Eq. 9)
2.  Generate cost matrix by computing the cost of each task on different VMs using Eq. (1) and (2)
3.  Determine $Cost_{bg}(G)$ ,$Cost_{min}(G)$, $Cost_{max}(G)$ using Eq. (3),(4) and (5)
4.  Compute ACT of all tasks using Eq. (8)
5.  Determine MACT($l_i$) for each level of DAG defined in Eq. (10)
6.    If (ACT ($t_i$)>MACT($l_i$))
                Place task in MTCTL (Most Time Consuming Task List)
              else
                  Place it in LTCTL(Less Time Consuming Task List)
7.  end if
8.  Determine B Level of each task from Eq.  (11)
9.  Assign Priority of each task according to the descending order of B-level values.
10. Set RB=Budget and RCB= CheapestCost
11. while (there are tasks in dlist) do
12.         Select task ($t_i$) from queue having higher priority.
13.         Compute RCB using  Eq.(12)
14.          Assign  $BC(t_i)$ using Eq. (14)
15.       for  $( j = 1; j \le v; j++)$ do
16.               if $Cost(t_i, vm_j) \le BC(t_i)$ then
17.                     if Task ($t_i$) is from MTCTL
18.                          select fastest VM
                            else
19.                               Select   appropriate VM using  Algorithm 2
20.                       end if
21.                             end if
22.       end for
23. Update the Remaining Budget (RB) using Eq. (13)
24.         end while
25. Compute schedule using Eq. (20), Eq. (21) , Eq. (22) and Eq. (6)
26. Compute Cost(G) using Eq. (2)
27. Return makespan, cost
```

Algorithm 2: ***VM selection for tasks in LTCTL***

```
1.   Donot select the VM having highest cost of execution and lowest processing speed
2.   for (each remaining $VM \in v$) do
3.     Find $Cost_{min}(ti,vm_j)$ and $et_{min}(ti,vm_k)$ // where $v_j$ and $v_k$ are cheapest and fastest VM
4.       if ($vm_j$==$vm_k$)
5.         Select VM $vm_j$
6.           else
7.             compute $\beta \times et_{i,j} + (1-\beta) \times Cost(t_i, vm_j)$
8.               Select VM that minimizes the function for task $t_i$
9.       end if
10.  end for
```

Table 8 Value of $MACT(l_i)$ at each level of DAG

Level(l_i)	1	2	3	4
MACT(l_i)	13	13.6	12.55	14.67

4.3 An illustration

An illustrative example is provided to understand the working of FBCWS. A DAG of 10 tasks, as illustrated in Fig. 2 is considered. The cost matrix is computed from the given Tables 1 and 3.

First, the average computation time (*ACT*) of each task is calculated. Afet this, the Mean of ACT (*MACT* (l_i)) of each level of DAG is determined (shown in Table 8). The value of *MACT* (*li*) determines the two lists, namely; Most Time-Consuming Task List (MTCTL) and Less Time-Consuming Task List (LTCTL). According to the value of *MACT* (*li*), FBCWS categorizes t_1, t_2, t_3, t_9, t_{10} as compute-intensive tasks and placed them in MTCTL wherein t_4, t_5, t_6, t_7, t_8 are normal tasks and are placed in LTCTL. B-level of each task is determined to decide the priority of tasks for execution. Task having higher B-level value has higher priority and scheduled first. Table 9 shows the priority of tasks based on B-level. By taking highest priority task from the sorted task list, the appropriate VM has been selected for each task. Additionally, *EST* and *EFT* of every task on assigned VM is determined by using Eqs. (20) or (21) and (22). Table 10 shows the task assignment of workflow with $Cost_{bg}(G) = 500$ with β = 0.8. The makespan of an application obtained is 80, which is less than HBCS, MSBL, and POSH algorithm. The execution cost is recorded as 471 that satisfies the budget constraints. The complete schedule of the application is also shown in Fig. 3. At the same time, a case has been illustrated, where FBCWS provides a mechanism to save the budget. Table 11 shows the task assignment to VMs when we set β = 0.2. The cost of the execution gets reduced to 455, and makespan becomes 90. The schedule of application is also shown in Fig. 4. Here, FBCWS produces minimum makespan and low-cost schedule than HBCS and proves to be more cost-efficient than MSBL.

Table 10 Tasks assignment of a workflow when β = 0.8

Tasks (t_i)	$Cost_{bg}$ (t_i)	*AST* (t_i)	*AFT* (t_i)	*f* (t_i)	*Cost* (t_i, $f(t_i)$)
t_1	129	0	9	3	27
t_3	159	21	32	1	77
t_4	122	18	26	2	40
t_2	136	32	45	1	91
t_5	75	9	19	3	30
t_6	72	19	28	3	27
t_9	105	61	73	2	60
t_7	78	45	52	1	49
t_8	80	53	58	1	35
t_{10}	80	73	80	2	35

Makespan(G) = 80 and Cost(G) = 471

5 Performance evaluation and experimental analysis

5.1 Datasets and simulation setup

We have simulated FBCWS, HBCS, MSBL, and POSH algorithm with the benchmark dataset having 10–1000

Table 9 B-level of tasks

Tasks	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
B-level	108.00	77.00	80.00	80.00	69.00	63.33	42.66	35.67	44.33	14.67

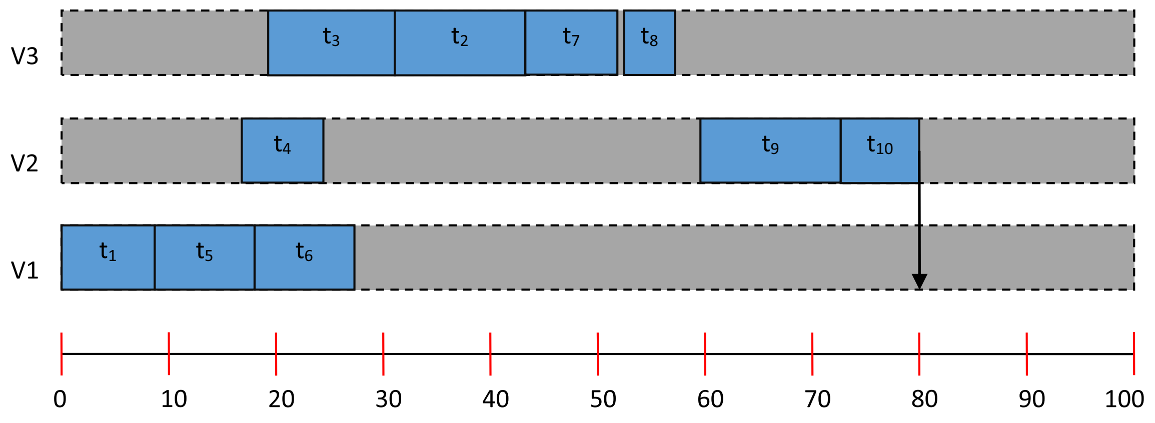


Fig. 3 Scheduling of sample workflow using FBCWS with β = 0.8

tasks. Five real scientific workflows of diverse characters such as Cybershake for earthquake hazard, LIGO (Inspiral) for analyzing data obtained from analyzing the gravitational waves from the coalescing of a binary system, Montage for astronomy, Sipht and Epigenomic for biology are analyzed. We categorize these workflows into compute-intensive and data-intensive workflows. Inspiral, Epigenomic, and Sipht are computed intensive workflows incurring high computation cost. On the other hand, Montage and Sipht belong to data-intensive workflows, in which data transfer cost is higher than computation cost. Each workflow has a different structure shown in Fig. 5. Further, structure and characterization of different workflows with a detailed definition is presented by Bharathi et al. [42].

For the simulation, we assumed a cloud environment that consists of a cloud provider offering 5, 10, 20, and 50 sets of VMs of different characteristics and capabilities. Each VM has a unit price based on the pricing similar to EC2 pricing. The average bandwidth between the computation services is assumed to be 20 Mbps. The simulations of the proposed algorithm were performed using CloudSim simulator running on Intel(R) Core (TM) i7-8550U CPU @ 1.80 GHz 1.99 GHz and 8 GB RAM on Windows 10.

Table 11 Task assignment of workflow when β = 0.2

Tasks (t_i)	$Cost_{bg}$ (t_i)	*AST* (t_i)	*AFT* (t_i)	f (t_i)	*Cost* $(t_i, f(t_i))$
t_1	129	0	9	3	27
t_3	159	21	32	1	77
t_4	122	18	26	2	40
t_2	136	32	45	1	91
t_5	75	9	19	3	30
t_6	72	19	28	3	27
t_9	105	61	73	2	60
t_7	78	55	66	3	33
t_8	80	53	58	1	35
t_{10}	80	83	90	2	35

Makespan(G) = 90 and Cost(G) = 455

5.2 Performance metrics

To evaluate and compare the proposed algorithm with other existing algorithms, we consider the normalized makespan (*NM*) defined by;

$$NM(G) = \frac{makespan_{FBCWS}}{makespan_{HEFT}} \tag{23}$$

where *NM* normalizes the proposed algorithm makespan by the lower bound value obtained from the makespan of HEFT. HEFT [13] has been used here since it produces minimum makespan for a DAG in a heterogeneous environment.

To compare the achieved cost of schedule, we consider Normalized Cost *NC* metric defined by;

$$NC(G) = \frac{Cost_{FBCWS}(G)}{Cost_{bg}(G)} \tag{24}$$

where *NC*(*G*) value greater than 1 means the budget constraint has not been satisfied. Therefore, *NC* lower than 1 is always preferable.

5.3 Experimental analysis

The performance of FBCWS is validated with HBCS, MSBL, and POSH algorithm over five types of different workflows. We define, β = 0.8 for FBCWS to ensure fairness to each task of LTCTL. For the POSH algorithm, we set α = 0.2 as the algorithm performs better for a lower α value. We consider the strict budget constraints of fixed size for evaluation. For better analysis, we group the

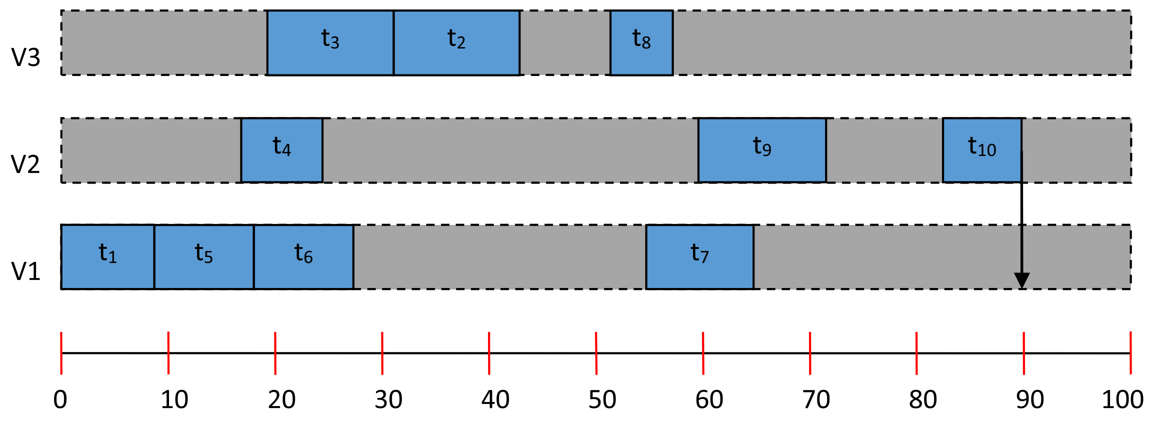
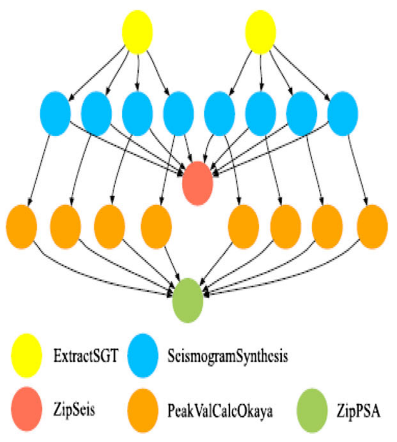
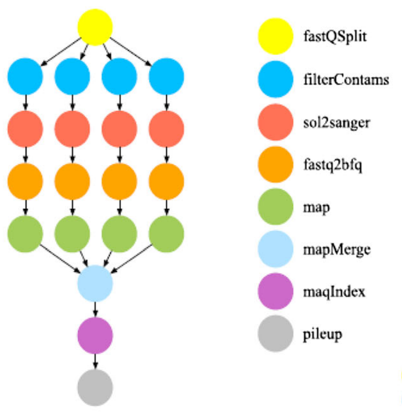


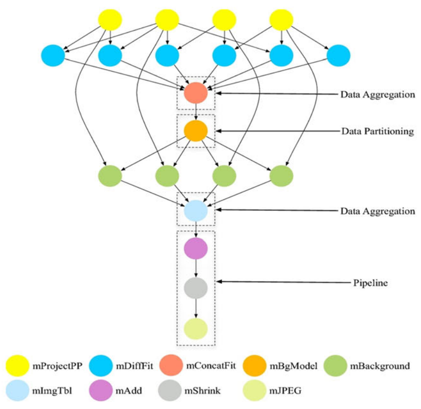
Fig. 4 Scheduling of sample workflow using FBCWS with β = 0.2



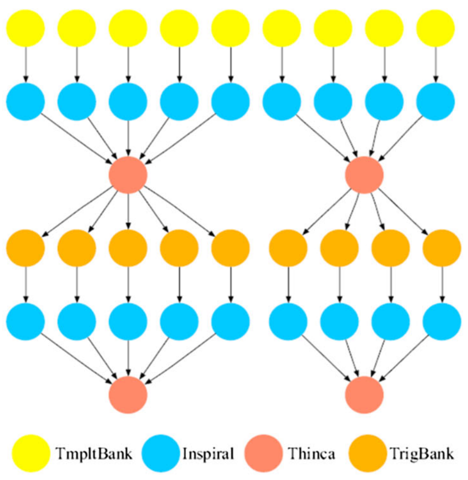
(a) Cybershake



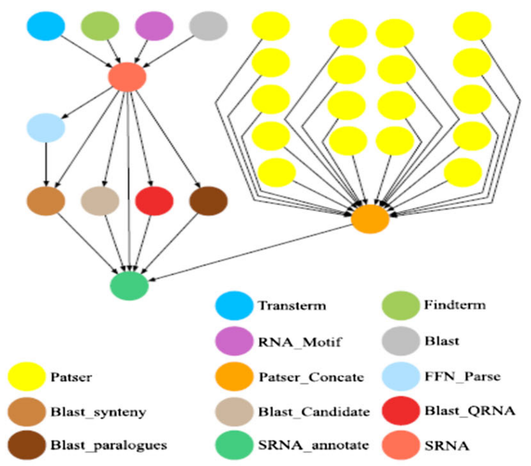
(b) Epigenomic



(c) Montage



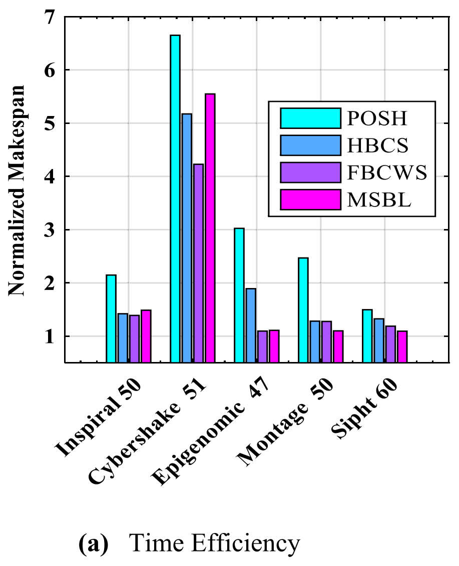
(d) LIGO (Inspiral)



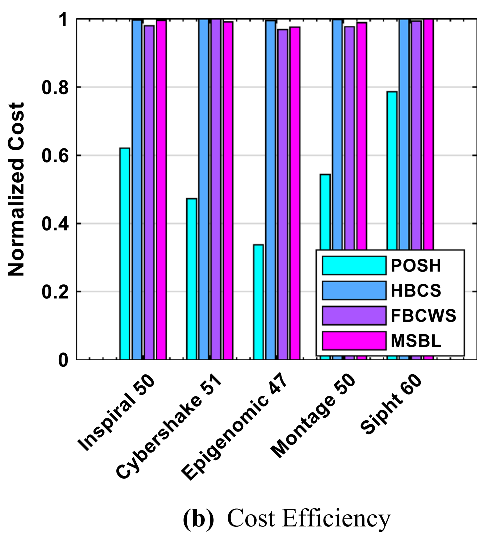
(e) Sipht

Fig. 5 Types of workflows where circles represent jobs of different types [42]. **a** Cybershake, **b** Epigenomic, **c** Montage, **d** LIGO (Inspiral), **e** Sipht

Fig. 6 Workflows with small range. **a** Time efficiency, **b** cost efficiency



(a) Time Efficiency



(b) Cost Efficiency

workflows based on the size of the tasks, as mentioned below.

(1) Small size workflows, consisting of 10–50 tasks nodes
(2) Medium size workflows, having 100–400 tasks nodes
(3) Large size workflows, consisting of 800–1000 tasks nodes

The algorithm having minimum *NM* (Normalized Makespan) and *NC* (Normalized cost) value is found more efficient. Analysis of different workflows for various algorithms is discussed in the below mentioned subsequent sections.

5.3.1 Performance analysis of small size workflows

Figure 6 shows the result achieved for small size workflows. Figure 6a shows the time efficiency, and Fig. 6b shows the cost efficiency of the considered algorithms. As shown in Fig. 6a, FBCWS has the best performance for Inspiral, Cybershake, and Epigenomic by achieving smaller normalized makespan value than the other algorithms. From Fig. 6a and b, it is observed that FBCWS utilizes the whole budget for Cybershake to achieve minimal makespan. MSBL proves to be more time-efficient than FBCWS for Montage and Sipht. However, FBCWS is more cost-efficient than MSBL. Further, POSH is less efficient in terms of time but is highly cost-efficient shown in Fig. 6b. FBCWS performs remarkably for the Epigenomic workflow by minimizing the makespan as well as the cost of execution.

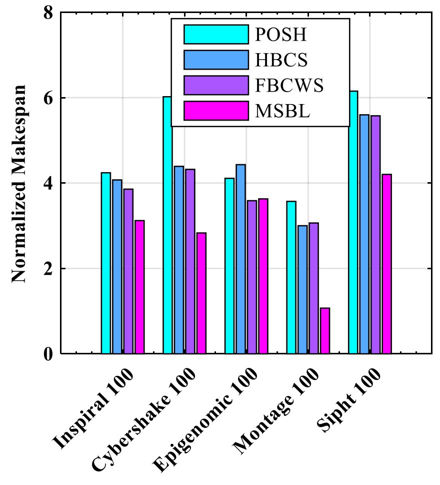
5.3.2 Performance analysis of medium size workflows

Results obtained for medium size workflows are shown in Fig. 7. Figure 7a and b shows the comparison result of the workflows of 100 tasks and Fig. 7c and d illustrates the results of workflows having 400 tasks. As depicted in Fig. 7a and c, it is found that FBCWS proves to be more time-efficient than POSH as well as for HBCS in most of the workflows. Moreover, FBCWS outperforms for Epigenomic by achieving minimum makespan at a reduced cost. From Fig. 7c and d, it has also been observed that, with the increase in workflow size, FBCWS becomes more time and cost-efficient for Sipht. Moreover, it has also been analyzed FBCWS lags behind MSBL, as MSBL produces schedule of minimum makespan most of the time. However, FBCWS is more cost efficient than MSBL.

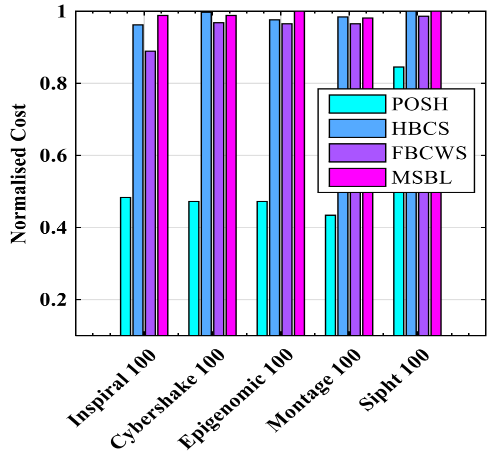
5.3.3 Performance analysis of large size workflows

As illustrated in Fig. 8, with the increase in the size of workflows, the performance of FBCWS is significantly improved. FBCWS achieve higher efficiencies for Epigenomic and Sipht. On the other hand, due to the high data transfer rate, FBCWS is not that efficient for Montage and Cybershake. As the proposed approach doesn't consider any mechanism to minimize data transfer time and, therefore, not perform well for the data-intensive workflows. In the case of Inspiral, FBCWS achieves lower *NM* and *NC* value than HBCS and thus proves to be more efficient than HBCS. However, FBCWS is more cost efficient than MSBL but lags behind MSBL, which is more time-efficient for Inspiral, Cybershake, and Montage. For all the types of workflows, POSH always produces schedule of higher makespan and hence, proves to be inefficient. However,

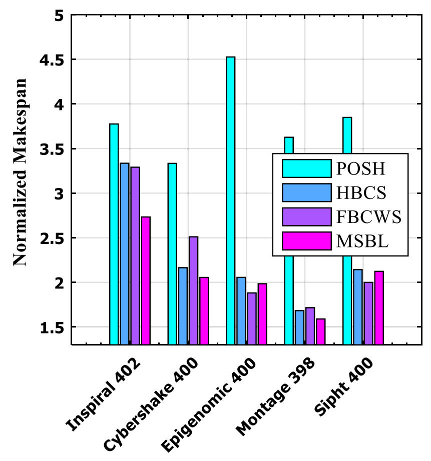
Fig. 7 Workflows of medium size. **a** Time efficiency, **b** cost efficiency, **c** time efficiency, **d** cost efficiency



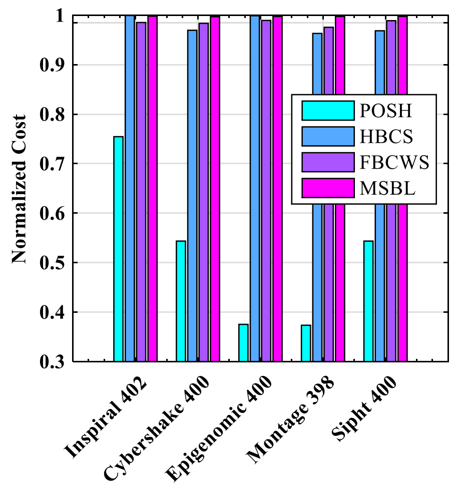
(a) Time Efficiency



(b) Cost Efficiency



(c) Time Efficiency



(d) Cost Efficiency

POSH is a cost-efficient algorithm. It always utilizes the minimum cost to produce schedule of longer makespan.

5.4 Analysis of variance (ANOVA) test

In this section, the statistical significance of the obtained result has been verified through the ANOVA test. ANOVA is a statistical tool to check whether the mean of the given groups are equal or not. It helps in determining whether the null hypothesis is rejected or not. We have performed ANOVA test on the makespan value obtained for all types of considered workflows. For validation, we consider 15 medium and large size workflows ranging from 400 to 1000 as a sample. In our case, we define the null hypothesis as;

$H_0 : \mu_{FBCWS} = \mu_{POSH} = \mu_{HBCS} = \mu_{MSBL}$ and alternative hypothesis as;

$H_1 : \mu_{FBCWS} \neq \mu_{POSH} \neq \mu_{HBCS} \neq \mu_{MSBL}$

In order to reject the null hypothesis, the value of F-static must be greater than F- critical. In addition to this, P-value must be less than the selected α-level (α = 0.05).

From Table 12, it is certain that the null hypothesis can be rejected because F-static is greater than F-critic, and P-value is also less than α-level. Thus, it is proved that the obtained experimental results are statistically significant.

6 Conclusion and future work

In this paper, we have proposed a novel algorithm named FBCWS for scientific workflow scheduling. The proposed algorithm minimizes the makespan while satisfying the budget constraints. FBCWS also provides a mechanism to save budget by setting a low value of the cost-time factor of

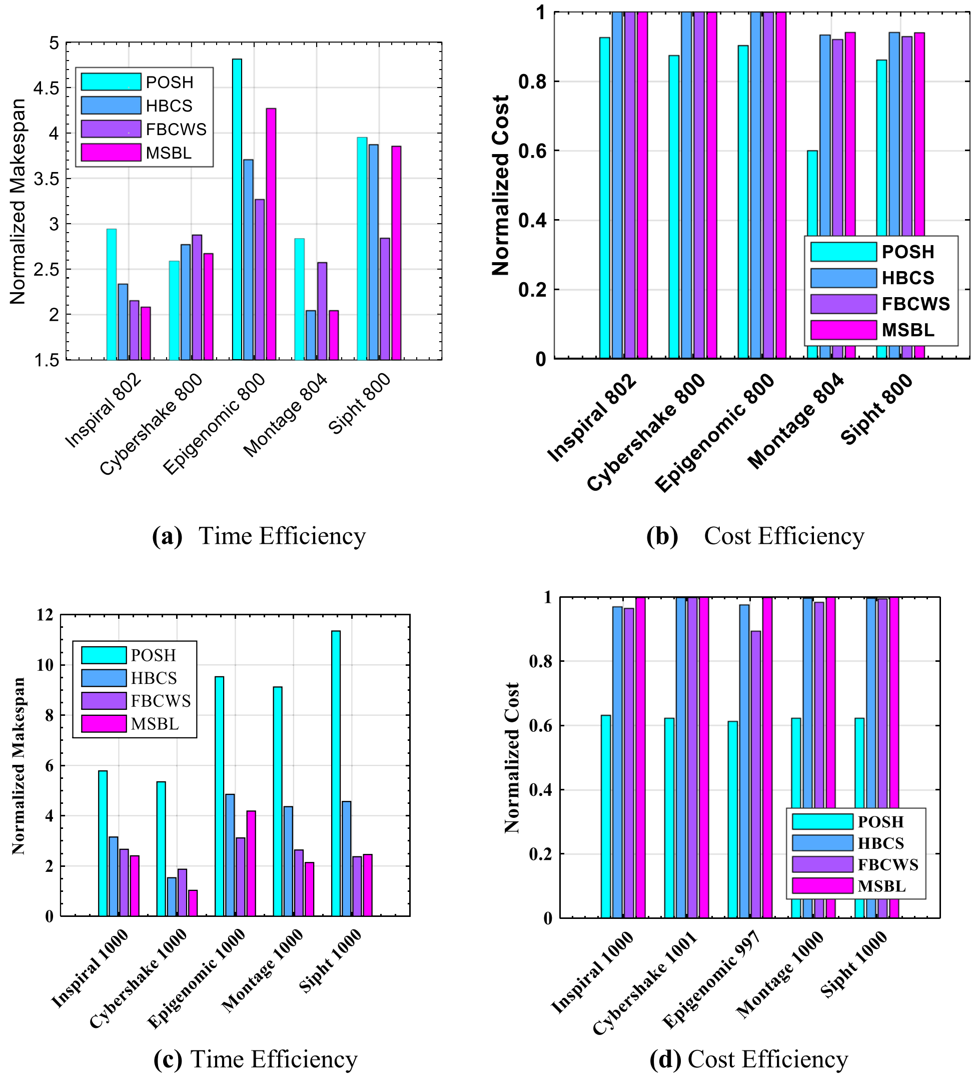


Fig. 8 Workflows of large size. **a** Time efficiency, **b** cost efficiency, **c** time efficiency, **d** cost efficiency

Table 12 The ANOVA test

Groups	Count	Sum	Average	Variance
POSH	15	1,246,834	83,122.27	5.92E + 09
HBCS	15	698,655	46,577	1.21E + 09
FBCWS (proposed)	15	550,946	36,729.73	3.88E + 08
MSBL	15	559,658	37,310.53	5.39E + 08

Source of variation	SS	df	MS	F	P-value	F crit
Between Groups	2.16E+10	3	7.21E+09	3.582847	0.019273	2.769430932
Within Groups	1.13E+11	56	2.01E+09			
Total	1.34E+11	59				

the defined minimization function. However, this approach has been shown for sample workflows only. FBCWS also proves to be a fair algorithm as it categorizes the task and tries to schedule LTCTL tasks, i.e., tasks of low priority to those VMs that have minimum execution time by adjusting high value for the cost-time factor. In FBCWS, tasks are analyzed based on their type, and accordingly, they are assigned to the capable VMs to ensure fairness while this fairness is not provided to the HBCS, MSBL, and POSH algorithm. Tasks of MTCTL are assigned to the VMs having minimum execution time and hence, try to minimize the execution time. For the *LTCTL or* low priority tasks, FBCWS doesn't select those expensive VMs which don't have minimum execution time. This step further improves the efficiency of FBCWS and proves to be efficient in minimizing the cost and time of execution. Further, FBCWS also provides flexibility for the selection of most time-efficient or cost-efficient VMs. FBCWS can also prove to be cost-efficient by adjusting low value for the cost-time factor of the minimization problem. No such advantages are provided by the HBCS and MSBL algorithm.

In order to show the efficiency of the proposed FBCWS, the experiments are conducted over the five different types of workflows. Along with this, simulation results are also compared with the results of existing HBCS, MSBL, and POSH algorithms. Based on the performance metrics, it is concluded that the FBCWS gives the best results for compute-intensive workflows like Epigenomic and Sipht. For Inspiral, FBCWS outperforms HBCS, whereas, for data-intensive workflows like Montage and Cybershake, FBCWS lags behind MSBL in terms of makespan. FBCWS is more time-efficient than POSH for all the sizes of workflows. Furthermore, to prove the significance of the obtained experimental result, ANOVA test has been performed. However, FBCWS doesn't give emphasis on data-intensive workflows. Therefore, in future work, the enhancement of the FBCWS algorithm can be done for better performance of data-intensive workflows. Along with this, FBCWS algorithm can be further explored for dynamic scheduling and a multi-cloud environment. Further, it is also intended to develop a strategy for loopy workflows as well as the design of a new workflow generator which consider the loopy workflows.

Acknowledgements This work is supported by the Indian Institute of Technology (ISM), Dhanbad, Govt. of India. The authors wish to express their gratitude and heartiest thanks to the Department of Computer Science & Engineering, Indian Institute of Technology (ISM), Dhanbad, India, for providing their continuous research support.

References

1. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow scheduling algorithms for grid computing. Metaheuristics for scheduling in distributed computing environments, pp. 173–214. Springer, Berlin (2008)
2. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. J. Grid Comput. **3**(3–4), 171–200 (2005)
3. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. Grid Computing Environments Workshop, 2008. GCE'08, pp. 1–10. IEEE, Piscataway (2008)
4. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
5. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010)
6. Weinhardt, C., Anandasivam, A., Blau, B., Borissov, N., Meinl, T., Michalk, W., Stößer, J.: Cloud computing: a classification, business models, and research directions. Bus. Inform. Syst. Eng. **1**(5), 391–399 (2009)
7. Juve, G., Deelman, E.: Scientific workflows in the cloud. Grids Clouds and Virtualization, pp. 71–91. Springer, London (2011)
8. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the use of cloud computing for scientific workflows. IEEE Fourth International Conference on eScience, 2008, eScience'08, pp. 640–645. IEEE, Piscataway (2008)
9. Lewis, H.R.: Review: Garey Michael R. and Johnson David S. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco1979, x+ 338 pp. J. Symbol. Logic. **48**(2), 498–500 (1983)
10. Wu, C., Lin, X., Yu, D., Xu, W., Li, L.: End-to-end delay minimization for scientific workflows inclouds under budget constraint. IEEE Trans. Cloud Comput. **1**, 1–1 (2015)
11. Arabnejad, H., Barbosa, J.G.: A budget constrained scheduling algorithm for workflow applications. J. Grid Comput. **12**(4), 665–679 (2014)
12. Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., Li, K.: Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Future Gener. Comput. Syst. **74**, 1–11 (2017)
13. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
14. Bittencourt, L.F., Madeira, E.R.M.: HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. J. Internet Serv. Appl. **2**(3), 207–227 (2011)
15. Su, S., Li, J., Huang, Q., Huang, X., Shuang, K., Wang, J.: Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. **39**(4–5), 177–188 (2013)
16. Pandey, S., Wu, L., Guru, S.M., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010, pp. 400–407. Piscataway, IEEE (2010)
17. Li, J., Su, S., Cheng, X., Song, M., Ma, L., Wang, J.: Cost-efficient coordinated scheduling for leasing cloud resources on hybrid workloads. Parallel Comput. **44**, 1–17 (2015)
18. Sakellariou, R., Zhao, H.: A low-cost rescheduling policy for efficient mapping of workflows on grid systems. Sci. Program. **12**(4), 253–262 (2004)
19. Fard, H.M., Prodan, R., Barrionuevo, J.J.D., Fahringer, T.: A multi-objective approach for workflow scheduling in heterogeneous environments. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012, pp. 300–309. IEEE, Piscataway (2012)
20. Zhu, Z., Zhang, G., Li, M., Liu, X.: Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans. Parallel Distrib. Syst. **27**(5), 1344–1357 (2016)
21. Durillo, J.J., Prodan, R.: Multi-objective workflow scheduling in Amazon EC2. Clust. Comput. **17**(2), 169–189 (2014)

22. Zhang, F., Cao, J., Li, K., Khan, S.U., Hwang, K.: Multi-objective scheduling of many tasks in cloud platforms. Future Gener. Comput. Syst. **37**, 309–320 (2014)
23. Tan, W., Sun, Y., Li, L.X., Lu, G., Wang, T.: A trust service-oriented scheduling model for workflow applications in cloud computing. IEEE Syst. J. **8**(3), 868–878 (2014)
24. Talukder, A.K.A., Kirley, M., Buyya, R.: Multiobjective differential evolution for scheduling workflow applications on global grids. Concurr. Comput. Pract. Exp. **21**(13), 1742–1756 (2009)
25. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
26. Ghafouri, R., Movaghar, A., Mohsenzadeh, M.: A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. Peer Peer Netw. Appl. (2018). https://doi.org/10.1007/s12083-018-0662-0
27. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans. Parallel Distrib. Syst. (2018). https://doi.org/10.1007/s10586-018-1751-9
28. Sun, T., Xiao, C., Xu, X.: A scheduling algorithm using sub-deadline for workflow applications under budget and deadline constrained. Clust. Comput. (2018). https://doi.org/10.1007/s10586-018-1751-9
29. Yu, J., Buyya, R.: A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. Workshop on Workflows in Support of Large-Scale Science, 2006. WORKS'06, pp. 1–10. IEEE, Piscataway (2006)
30. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**(3–4), 217–230 (2006)
31. Yuan, Y., Li, X., Wang, Q., Zhu, X.: Deadline division-based heuristic for cost optimization in workflow scheduling. Inf. Sci. **179**(15), 2562–2575 (2009)
32. Yu, J., Buyya, R., Them, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. First International Conference on e-Science and Grid Computing, 2005, IEEE, Piscataway (2005)
33. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
34. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Future Gener. Comput. Syst. **48**, 1–18 (2015)
35. Poola, D., Garg, S.K., Buyya, R., Yang, Y., Ramamohanarao, K.: Robust scheduling of scientific workflows with deadline and budget constraints in clouds. IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), 2014, pp. 858–865. Piscataway, IEEE (2014)
36. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control. J. Grid Comput. **11**(4), 633–651 (2013)
37. Arabnejad, V., Bubendorfer, K., Ng, B.: Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. Future Gener. Comput. Syst. **75**, 348–364 (2017)
38. Calheiros, R.N., Buyya, R.: Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans. Parallel Distrib. Syst. **25**(7), 1787–1796 (2014)
39. Yang, Y., Liu, K., Chen, J., Liu, X., Yuan, D., Jin, H.: An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows. IEEE Fourth International Conference on eScience, 2008, eScience'08, pp. 374–375. IEEE, Piscataway (2008)
40. Rodriguez, M.A., Buyya, R.: Budget-driven scheduling of scientific workflows in IaaS clouds with fine-grained billing periods. ACM Trans. Auton. Adapt. Syst. **12**(2), 5 (2017)
41. Alkhanak, E.N., Lee, S.P., Khan, S.U.R.: Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. Future Gener. Comput. Syst. **50**, 3–21 (2015)
42. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. Third Workshop on Workflows in Support of Large-Scale Science, 2008. WORKS 2008, pp. 1–10. IEEE, Piscataway (2008)

Naela Rizvi is currently working as a Junior Research Fellow at the Department of Computer Science and Engineering in Indian Institute of Technology (ISM) Dhanbad, Jharkhand, India. She received the degree of ME in Software Engineering from BIT Mesra, Ranchi. She received B.E in Computer Science from Radharaman Institute of Technology and Science, Bhopal. Her area of research includes Resource management and Task scheduling in cloud environment.

Dharavath Ramesh is a Professor (Asst.) in the Computer Science and Engineering Department at Indian Institute of Technology Dhanbad, Jharkhand, India. He obtained his Ph.D. from Indian Institute of Technology (Indian School of Mines), Dhanbad under the supervision of Professor Chiranjeev Kumar. His Master's degree in Computer Science and Engineering with a specialization as Software Engineering is from Jawaharlal Nehru Technological University, Hyderabad, India. His Bachelor of Engineering degree in Computer Science and Engineering is from KITS, Warangal, India. Dr. Ramesh's research interests include Blockchain & Distributed Computing, Distributed Databases, Cloud Databases, Modelling Big Data, Processing Big Data, Virtualization and Scheduling in Cloud Environment, Load balancing approaches, Deep learning strategies, Brain Computer Interaction, Community Detection in Social Networks, ML paradigms in Agriculture. He has published over a hundred technical papers in refereed journals and conference proceedings. He has served in various capacities for journals and conferences. He was Organizing Co-Chair for RAIT 2018, RAIT 2014, and advisory committee member of various International and National conferences/workshops/symposiums. He has served on the program committees of various conferences including BDA, ICDCIT, ADCONS, RAIT, and ICDE. He is a senior member of the IEEE and a professional member of ACM.