iCATS: Scheduling Big Data Workflows in the Cloud Using Cultural Algorithms

Seyed Ziae Mousavi Mojab
Department of Computer Science
Wayne State University
Detroit, Michigan
mousavi@wayne.edu

Mahdi Ebrahimi
Department of Math. and Computer Science
Lawrence Tech. University
Southfield, Michigan
mebrahimi@ltu.edu

Robert G. Reynolds
Department of Computer Science
Wayne State University
Detroit, Michigan
reynolds@cs.wayne.edu

Shiyong Lu
Department of Computer Science
Wayne State University
Detroit, Michigan
shiyong@wayne.edu

***Abstract*— Workflow scheduling has remained a critical functionality of modern data-centric workflow management systems. Cloud computing, which provides practically unlimited computing and storage resources, has enabled a new generation of data-centric workflows, called big data workflows. New big data workflow scheduling algorithms should optimally utilize the characteristics of cloud computing such as heterogeneous virtual machines, the elastic resource provisioning model, and the pay-as-you-go pricing model, as well as the time and monetary cost to transfer large amounts of data. In this paper, we consider one case of the general big data workflow scheduling problem where a deadline, δ, is given for a workflow, W, and the goal is to minimize the monetary cost of running W in the cloud while satisfying the given deadline, δ. To this end, we leverage the power of Evolutionary Algorithms (EA) in order to search for the best solution within a reasonable planning time. More specifically, we introduce an innovative fitness function that combines the time and monetary cost of a workflow in one metric. Based on the EA and the fitness function, we design a deadline-constrained big data workflow scheduling algorithm, called iCATS (Improved Cultural Algorithms-based Task Scheduling). Extensive experiments demonstrate the statistical advantages of iCATS over existing representative EA workflow scheduling algorithms, including random (Rand), Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Cultural Algorithms (CA).**

Keywords- big data workflows; Cultural Algorithms; workflow scheduling; cloud computing; evolutionary algorithms.

I. INTRODUCTION

Workflow scheduling is a key component of workflow management systems [1, 2, 3, 4]. The challenges of big data in terms of volume, variety, and velocity, and the potential unleashed by cloud computing [5] in the provisioning of practically unlimited computing and storage resources brought about active research on a new generation of data-centric workflows, called *big data workflows* [6, 7]. Currently, three lines of research are being pursued for big data workflow scheduling in the cloud. The first is the category of deadline-constrained workflow scheduling [4, 8]. The second is that of budget-constrained workflow scheduling [6, 9]. And the third line of research considers both the budget and deadline constraints [11, 12], and attempts to optimize both based on some tradeoffs between the two [1, 3]. In this research, the focus is on the design and implementation of a workflow scheduling algorithm for deadline-constrained big data workflow applications. Several challenges must be addressed in order to solve a deadline-constrained big data workflow scheduling problem:

First, given two schedules *sch1* and *sch2* for a workflow *W*, what performance metric should be used to compare the quality of the two schedules?

Second, cloud computing uses an elastic resource provisioning model, so the workflow scheduling algorithm not only needs to decide the mapping and scheduling of individual workflow tasks, but also make decisions on cloud virtual machine provisioning and deprovisioning.

Third, cloud computing uses the pay-as-you-go pricing model. For cloud computing, we need to consider monetary cost in addition to other Quality of Service (QoS) requirements such as makespan to compute cost.

Finally, one should consider the data transfer time, which can be significant for big data workflows.

In this paper, we leverage the power of EA [13] to search for the best solution within a reasonable time. Furthermore, we introduce an innovative fitness function that combines the temporal and monetary costs of a workflow schedule.

Using EA, and innovative strategies in knowledge source (KS) exploration and exploitation, we design a deadline-constrained big data workflow scheduling algorithm, called iCATS (Improved Cultural Algorithms-based Task Scheduling). Extensive experiments demonstrate the advantages of iCATS over existing representative workflow scheduling algorithms, including Random, GA [12], PSO [14], and CA [15].

The rest of the paper is organized as follows. First, in Section II, we provide some background information on workflow scheduling and Cultural Algorithms. Section III presents related work. In Section IV, we define and formalize our cloud-based workflow system model. In Section V, we describe our workflow scheduling algorithm, iCATS, in detail. Then, in Section VI, the experimental results are shown and discussed. Finally, the conclusions and future work are presented in Section VII.

II. BACKGROUND

A. Workflow Scheduling

The big data workflow scheduling problem is related to the Flow-Shop scheduling, which is a special case of the Job-Shop scheduling problem [16]. The Job-Shop scheduling problem aims to minimize the makespan by assigning *n* independent jobs of varying processing times to *m* machines of varying processing power. The Flow-Shop scheduling

978-1-7281-0059-3/19/$31.00 ©2019 IEEE
DOI 10.1109/BigDataService.2019.00020

problem, in addition, imposes an order on the execution of the *n* jobs. In contrast, the big data workflow scheduling problem needs to consider not only the precedence order between jobs, but also the data transfer between them, which might be large in volume. In addition, we need to consider the characteristics of cloud computing, especially the elastic virtual machine provisioning model, in which the pool of computing virtual machines is dynamic. This elastic nature of cloud computing introduces a new opportunity but also a challenge since it requires consideration of user-driven QoS constraints, such as budget, deadline, reliability, and security.

In general, a big data workflow scheduling algorithm needs to make four kinds of decisions:

1) *Virtual machine provisioning and deprovisioning*: how many VMs do we need and what types? When do we need to provision and deprovision these VMs?
2) *Task mapping:* on which VM should a workflow task *T* be running?
3) *Task scheduling:* after a task *T* is mapped to a *VM*, the system needs to decide when to run *T* on *VM*.
4) *Data transfer scheduling:* the system needs to decide when a dataset *D* needs to move from one VM to another.

B. Cultural Algorithms

One of the evolutionary computational systems that simulates social evolution is the Cultural Algorithms (CA) proposed by Reynolds [13, 15]. Due to its nature, culture can be viewed as a complex adaptive system, in which different heterogeneous agents are working together and interacting with the environment. This interaction of intelligent agents can result in higher-level behaviors that can be applied to the solution of problems at different ranges of temporal and spatial complexity.

CA can be conceptualized as a knowledge intensive evolutionary search process. While a heuristic can be viewed as a shortcut to a solution, a meta-heuristic like GA searches through a space of heuristics to find an appropriate one. A Cultural Algorithm is a hyper-heuristic that employs problem related knowledge to search through a space of meta-heuristics to find an optimal social configuration.

In traditional methods of Evolutionary Computation there was no implicit or explicit mechanism for storing and transmitting the knowledge from one generation to another. As a competitive advantage, CA provides an explicit mechanism for selecting, storing, and evolving the knowledge during a typical search. CA, as a dual-inheritance system, has two major components: the Population Space and the Belief Space [13] that are able to evolve in parallel. In addition to those two components, there is a communication protocol that allows the Belief Space and the Population Space to interact with each other and exchange their knowledge (Figure 1).

In each generation, individuals in the Population Space are first evaluated with an objective function *obj()*. An acceptance function *accept()* is then used to determine which individuals will be allowed to update the Belief Space. Experiences of those chosen individuals are then added to the contents of the Belief Space via function *update()*. The Belief Space can be viewed as a network of knowledge sources. Thus, an update to one knowledge can be propagated to other

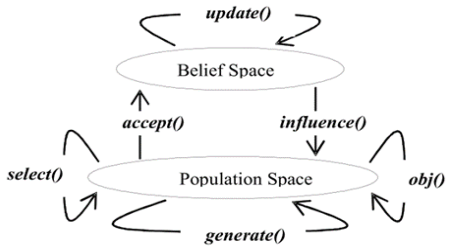

Figure 1: Cultural Algorithms [13].

knowledge sources in the network. For the workflow scheduling application described here the Belief Space is comprised of three knowledge sources as follows:

- **Normative knowledge:** it is used to store the highest and lowest values for different numeric attributes.
- **Situational knowledge:** it contains the best exemplar (Elite) found in each generation of the population.
- **Domain knowledge:** it consists of fitness values of the individuals and gets updated in every iteration.

Knowledge from the Belief Space can influence the selection of individuals for the next generation of the population, analogous to the evolution of human culture, through the *influence()* function. This supports the idea of dual inheritance in that the Population Space and the Belief Space are updated at each step based upon feedback from each other. The influence of the knowledge sources updates the knowledge of an individual in the social fabric or network. The CA repeats this accept-update and influence process for each generation until the termination condition is met.

III. RELATED WORK

Several evolutionary approaches have been proposed to solve workflow scheduling problem in the cloud [12, 14]. One category of approaches is based on particle swarm optimization (PSO) [14, 17]. They can be single- or multi-objective optimization tools with the goal of minimizing the total of workflow execution cost. Genetic Algorithms (GA) have also been proposed to address the problem of workflow scheduling in cloud computing [12, 18]. In these approaches, chromosomes represent the encoding of the cloud computing virtual machines and workflow task assignments. Knowledge produced during search resides in the chromosome structures. If a top performer is modified, it may mean that the knowledge is no longer carried in the population. Individuals carry the burden of accumulated knowledge in their chromosomal structures.

CA represents a knowledge intensive approach such that the knowledge collected by a population is subsequently transferred into the Belief Space. Thus, while a top performer is modified, information about its performance still resides in the corresponding Belief Space. Traditionally in a CA, the KSs compete with each other to generate improved individuals in the population in the spirit of "survival of the fittest". The KS that generates more fit individuals in the population is more likely to be used.

What differentiates iCATS from the traditional approach is that it allows KSs in the Belief Space to cooperatively produce new individuals as well. This set of individuals is

called the Comprehensive Elite since it takes elite members from Situated Knowledge and combines them with knowledge from other sources such as normative knowledge to generate a combined generalized solution. This approach is applied and compared to the existing approaches above.

Another category of workflow scheduling algorithms is the cluster-based. For cluster-based workflow scheduling algorithms [10, 11], the goal is to minimize the total data movement between clusters by possibly assigning similar workflow tasks into the same cluster. Workflow tasks can be clustered based on either their execution times or the data movement sizes. For list-based workflow scheduling algorithms [19], workflow tasks are ranked and sorted based on their execution times and data dependencies. Then, the ranked tasks are assigned to the cloud virtual machines for execution.

In our previous work [8, 9], we proposed cluster-based scheduling algorithms in a heterogeneous cloud computing environment. We have used single-objective workflow scheduling optimization and considered the deadline and budget separately as the QoS constraints.

IV. System Model

To execute a big data workflow in the cloud, we need to model the cloud and big data workflow first. A cloud computing environment is modeled as follows:

Definition 4.1 (Cloud Computing Environment *C*): A cloud computing environment *C* is a 7-tuple *C(VMT, VMI, Type, VMC, VPrice, DTR, DPrice)*, where:

- *VMT* is a set of virtual machine types, each individual virtual machine type is denoted by VMT_j.
- *VMI* is a set of virtual machine instances, each virtual machine instance is denoted by VMI_i.
- *Type: VMI → VMT* is the virtual machine type function. $Type(VMI_i)$ returns the virtual machine type of instance VMI_i.
- *VMC: VMT → R^+* is the virtual machine capacity function. $VMC(VMT_j)$ returns the computation speed of virtual machine type VMT_j in terms of MIPS (Million Instructions Per Second). R^+ is the set of all real positive numbers.
- *VPrice: VMT → R^+* is the virtual machine cost function. $VPrice(VMT_j)$ returns the monetary cost for using a virtual machine of type VMT_j in terms of US dollars per hour.
- *DTR: VMI × VMI → R^+* is the data transfer rate function. $DTR(VMI_1, VMI_2)$ returns the network bandwidth between VMI_1 and VMI_2 in terms of MBytes per second.
- *DPrice: VMI × VMI → R^+* is the data transfer cost function. $DPrice(VMI_1, VMI_2)$ returns the monetary cost of transferring data from VMI_1 to VMI_2 in terms of US dollars per MB.

Definition 4.2 (Big Data Workflow *W*): A big data workflow is modeled as a 4-tuple *W = (T, D, TSize, DSize)*, where:

- *T* is a set of tasks in the workflow *W*. Each individual task is denoted by T_k.
- $D \subseteq T \times T$ is a set of data dependency edges among tasks. $D_{k1, k2}$ denotes the data dependency from T_{k1} to T_{k2}.
- *TSize: T → R^+* is the task size function. $TSize(T_k)$ returns the size of task T_k in terms of million instructions to measure the workload of the code for task T_k.
- *DSize: D → R^+* is the data size function. $DSize(D_{k1,k2})$ returns the size of the data product $D_{k1,k2}$ that is produced by T_{k1} and consumed by T_{k2} in terms of MBytes.

In order to execute a big data workflow *W* in a cloud computing environment *C,* each workflow task should be mapped to a cloud virtual machine instance. We define the task mapping function, *M*, as follows:

Definition 4.3 (Task Mapping *M*): Suppose there are *I* virtual machine instances and *K* workflow tasks, a task mapping is represented by a *K*-element vector *M* such that *M(k)* indicates the virtual machine instance to which T_k is mapped. Figure 2 shows a workflow with five tasks, T_1 - T_5, mapped to three cloud virtual machine types, VMI_1, VMI_2, and VMI_3. Here, the task mapping is *M* = <1, 2, 1, 2, 3> means tasks T_1, and T_3 are mapped to virtual machine instance VMI_1 *(M(1)= M(3) = 1)*, tasks T_2 and T_4 are mapped to virtual machine instance VMI_2 *(M(2) = M(4) = 2)*, and task T_5 is mapped to virtual machine instance VMI_3 *(M(5)= 3)*.

We define a big data workflow graph as a weighted DAG that includes a set of tasks and their data dependencies. The weights of the tasks and data edges are based on the average task computation and average data transfer time, respectively. A big data workflow graph can be defined formally as:

Definition 4.4 (Big Data Workflow Graph *G*): Given a workflow *W* in a cloud computing environment *C* and task mapping *M*, a big data workflow graph *G* is modeled as a weighted directed acyclic graph with a 7-tuple *G(T, D, M, TCT, DTT, TCC, DTC)*, where:

- The vertices of the graph represent a set of tasks *T*.
- The edges of the graph represent a set of data dependencies *D*.
- *M* is a task mapping in the cloud.
- *TCT: T×M → R^+* is the task computation time function. $TCT(T_k, M)$ returns the computation time of T_k on virtual machine *M(k)*. It is defined as:

$$TCT(T_k, M) = \frac{TSize(T_k)}{VMC(Type(M(k)))}$$

- *DTT: D×M → R^+* is the data transfer time function. It returns the data transfer time of $D_{k1,k2}$ from virtual

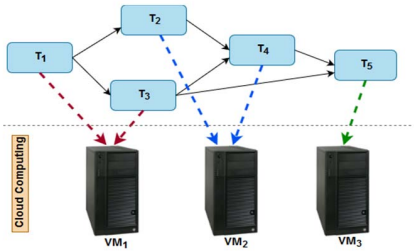

Figure 2: An example of workflow task mapping in the cloud.

machine $M(k_1)$ to virtual machine $M(k_2)$. It is defined as:

$$DTT(D_{k1,k2}, M) = \begin{cases} 0, & if\ M(k_1) = M(k_2) \\ \dfrac{DSize(D_{k1,k2})}{DTR(M(k_1), M(k_2))}, & if\ M(k_1) \neq M(k_2) \end{cases}$$

- *TCC:* $T \times M \rightarrow R^+$ is the task computation cost function. $TCC(T_k, M)$ returns computation monetary cost of T_k on virtual machine $M(k)$. It is defined as: $TCC(T_k, M) = TCT(T_k, M) \times VPrice(Type(M(k)))$
- *DTC:* $D \times M \rightarrow R^+$ is the data transfer cost function; It returns the data transfer monetary cost of $D_{k1,k2}$ from virtual machine $M(k_1)$ to virtual machine $M(k_2)$. It is defined as:

$$DTC(D_{k1,k2}, M) = DSize(D_{k1,k2}) \times DPrice(M(k_1), M(k_2))$$

Once each workflow task is assigned to some cloud virtual machine instances then, the workflow scheduler will specify the actual execution start time and finish time of each workflow task. We denote T_{entry} as the first task and T_{exit} as the end task of a workflow. The actual start time is defined as follows:

Definition 4.5 (Actual Start Time *AST*): Given a workflow graph G and task mapping M the actual start time of T_k on a virtual machine $M(k)$, denoted by $AST(T_k, M)$, is the actual start time when all predecessors of task T_k have completed their executions and all input data have arrived at virtual machine $M(k)$. It is officially defined as:

$$AST(T_k, M) = \begin{cases} 0, & if\ T_k = T_{entry} \\ max\{LDAP(T_k, M(k)), getAvailTime(M(k))\}, & if\ T_k \neq T_{entry} \end{cases}$$

where:

1. $LDAP(T_k, M(k)) = \max_{T_{k'} \in T_{k's\ parents}} \{AST(T_{k'}, M) + TCT(T_{k'}, M) + DTT(D_{k',k}, M)\}$
2. $getAvailTime(M(k)) = \begin{cases} ProvisionTime(M(k)), & \text{if } T_k \text{ is the first task in } M(k) \\ AFT(T_{k'}, M), & \text{Otherwise} \end{cases}$

where $T_{k'}$ is the immediate preceding task of T_k in $M(k)$.

3. $TCT(T_{entry}, M) = 0$ and $DTT(D_{entry,k}, M) = 0$.

Definition 4.6 (Actual Finish Time *AFT*): Given a workflow graph G and task mapping M, the actual finish time of T_k, denoted by $AFT(T_k, M)$, is defined as:

$$AFT(T_k, M) = AST(T_k, M) + TCT(T_k, M)$$

Workflow makespan is the total time needed to execute the whole workflow starting from the beginning task, T_{entry} through the end task, T_{exit}. Our goal is to come up with an optimal workflow schedule such that the workflow execution cost is minimized while the workflow makespan meets the given deadline. We define the makespan and cost as follows:

Definition 4.7 (Workflow Makespan *WMS*): Given a workflow graph G in a cloud computing environment C and task mapping M, the total execution time of the workflow denoted as WMS, is defined as:

$$WMS(G, M) = AFT(T_{exit}, M).$$

Definition 4.8 (Workflow Execution Cost *WCO*): Given a workflow graph G in a cloud computing environment C and task mapping M, the total execution cost of the workflow, denoted as WCO, is defined as:

$$WCO(G, M) = \sum_{T_k \in T} TCC(T_k, M) + \sum_{D_{k1,k2} \in D} DTC(D_{k1,k2}, M)$$

iCATS is an evolutionary algorithm that operates on a set of scheduling solutions in a population. In order to evaluate each generated solution, one must define a quality or fitness function. We formally define our fitness function as follows:

Definition 4.9 (Fitness Score): Suppose WMS is the makespan, WCO is the cost, and δ is the provided deadline. The fitness score is defined as:

$$Fitness(G, M, \delta) = \begin{cases} 0.5 + \left[0.5 * \frac{maxCost - WCO(G,M)}{maxCost}\right], & if\ WMS(G, M) \leq \delta \\ 0.5 * \left(\frac{\delta}{WMS(G,M)}\right), & if\ WMS(G, M) > \delta \end{cases}$$

where, $maxCost$ is the workflow execution cost once we map all the workflow tasks into the most expensive (the fastest) cloud virtual machine. A fitness score between 0 - 1 is produced for each generated scheduling solution. If the makespan of a scheduling solution is greater than the deadline δ, ($WMS > \delta$), then its score will be between [0 - 0.5]. And, if a solution meets the deadline ($WMS \leq \delta$) then its score will be between [0.5 - 1]. In this case, the closer the fitness score to 1, the better the scheduling solution. Therefore, we define our scheduling problem as follows:

Definition 4.10 (Workflow Cost Minimization): Given a workflow W in a cloud computing environment C, task mapping M, and deadline δ, the deadline-constrained scheduling problem is formalized to search for the optimal task mapping, M_{opt} as:

$$M_{opt} = \underset{M}{\text{argmax}}\ Fitness(G, M, \delta)$$

In the next section, we present a cultural algorithms based method called iCATS to search for the optimal task mapping M_{opt} in the space of all possible task mappings.

V. THE ICATS ALGORITHM

In this paper, we propose a new big data workflow scheduling under deadline constraints using an improved version of the Cultural Algorithms (iCATS). The improved version of Cultural Algorithms employs a novel mechanism of synthesizing a globally optimal solution called Comprehensive Elite. The Comprehensive Elite is unique to our approach in that it allows for multiple knowledge sources to collaborate in the generation of a new population of individuals. The experimental results will show the competitive advantage of our approach over competitors.

Algorithm 1 presents the iCATS algorithm. Workflow W, deadline δ, virtual machines specifications VM, and iCATS

Algorithm 1: iCATS

Input: Workflow *W*, Deadline δ, Virtual machines specifications *VM*, iCATS configurations *CC*
Output: The optimal workflow scheduling solution *sch*, and the corresponding makespan, and cost

Begin
1. Situational_K ← {}; Normative_K ← {};
2. BLF ← {Situational_K, Normative_K}; // Create the Belief Space
3. List_BestSolutions ← {Situational_K};
4. Randomly generate the initial population *Pop*, and calculate the fitness score of each individual solution.
5. **While** termination condition not reached **do**
6. Pop_Elites ← rank solutions based on their fitness score, and choose the top performers
7. BLF ← Update_BLF(BLF, Pop_Elites); //Alg. 2
8. CompElite ← makeCompElite(BLF, Pop_Elites, CC.nElites); //Alg. 3
9. List_BestSolutions ← BLF.Situational_K
10. dx = 0; // step size
11. **For** i = 1 to CC.Pop_size **do**
12. **For** j = 1 to W.num_Tasks **do**
13. dx = round(BLF.Normative_K.Size[j] × rnorm(1));
14. **If** (Pop[i].VM[j] < BLF.Situational_K.VM[j]) **then**
15. dx = abs(dx);
16. **ElseIf** (Pop[i].VM[j]>BLF.Situational_K.VM[j]) **then**
17. dx = -1 × abs(dx);
18. **End If**
19. Pop[i].VM[j] ← Pop[i].VM[j] + dx;
20. **End For**
21. Pop[i].Makespan ← Makespan(Pop[i]);
22. Pop[i].Cost ← Cost(Pop[i]);
23. **End For**
24. Apply crossover and mutation to randomly selected individuals.
25. Pop ← Pop ∪CompElite
26. Calculate the fitness score of each new individual solution
27. **End While**
28. M ← BLF.Situational_K; // add the best solution found to M
29. **For** Each pair M_k in M **do**
30. Insert (T_k, VM_j, AST(T_k, VM), AFT(T_k, VM) into schedule *sch*
31. Insert (Provision, VM_j of VM(T_k) type, ProvisionT(VM_j)) into schedule *sch*
32. **End For**
33. **For** Each active VM_j **do**
34. DeprovisionT(VM_j) = The AFT of the last task assigned to VM_j
35. Insert (Deprovision, VM_j, DeprovisionT(VM_j)) into schedule *sch*
36. **End For**
37. Return *sch*, makespan and execution cost of *sch*
End // End of algorithm

configurations *CC* are the four required inputs for iCATS. iCATS returns a near-optimal scheduling solution with the least execution cost as a set of records (<task, virtual machine number, actual start time, actual finish time>) for all the workflow tasks. In addition, the solution has the records of both provisioning and deprovisioning of the virtual machines. After parsing the workflow specification and generating the weighted DAG, the maxCost is calculated. iCATS creates the Belief Space configuration and produces the initial population from a uniform random distribution. Then, it assesses the performance of each individual solution using the fitness function (Definition 4.9). In the next step, iCATS selects the top performing individuals and updates the Belief Space and makes the Comprehensive Elite.

Next, iCATS generates a new population by using the influence function, based on the knowledge available in the Belief Space (lines 10-23). The influence function, in fact, causes individuals to evolve within a range found in the normative knowledge and to move toward the best exemplar found in the situational knowledge. iCATS then computes the makespan, and cost for each solution of the newly generated population. In order to increase the diversity of scheduling solutions, iCATS applies both crossover and mutation operators to some randomly selected solutions (line 24). In line 25, the Comprehensive Elite is added to the population. Then the fitness score of each new individual solution is calculated in line 26. In lines 29-32, iCATS generates the scheduling solution *sch,* based on the task mapping information *M,* and cloud virtual machine configuration. iCATS inserts the deprovisioning records into the scheduling solution as well (lines 33-36). These records represent what and when the idle virtual machines should be released. Finally, iCATS returns the best workflow scheduling solution as well as the corresponding makespan and cost (line 37).

iCATS calls *Update_BLF()* function to update the Belief Space. *Update_BLF()* is presented in Algorithm 2. In a loop (lines 1-15), both the Situational and Normative knowledge are determined based on the population of elites. In lines 2-4, if the fitness score of the solution is better than the Situational fitness score then the Situational knowledge is replaced with the best solution found in the Elites. In the next step, the VM_{min} and VM_{max} for each task (range of virtual machines used for each task) are specified. In addition, the corresponding fitness score as the lower bound and the upper bound are specified (lines 5-14). In line 16, the range of virtual machines used for each task in the Elites are computed and finally the updated Belief Space is returned (line 17).

iCATS also calls *makeCompElite()* to create the comprehensive Elite using different sources of knowledge available in the Belief Space. In line 1, an empty solution is created. In line 2, the list of Elites are sorted based on their

Algorithm 2: Update_BLF

Input: Belief Space *BLF*, Population of Elites *Pop_Elites*
Output: Updated Belief Space *BLF*

Begin
1. **For** i = 1 to |Pop_Elites| **do**
2. **If** (Pop_Elites[i].FScore > Situational_K.FScore) **then**
3. Situational_K ← Pop_Elites[i];
4. **End If**
5. **For** j = 1 to Pop_Elites.nTask **do**
6. **If** (Pop_Elites[i].VM[j] < BLF.Normative_K.Min[j])
7. BLF.Normative_K.Min[j] = Pop_Elites[i].VM[j];
8. BLF.Normative_K.LowerB[j] = Pop_Elites[i].FScore;
9. **End If**
10. **If** (Pop_Elites[i].VM[j] > BLF.Normative_K.Max[j]) **then**
11. BLF.Normative_K.Max[j] = Pop_Elites[i].VM[j];
12. BLF.Normative_K.UpperB[j] = Pop_Elites[i].FScore;
13. **End If**
14. **End For**
15. **End For**
16. BLF.Normative_K.Size = BLF.Normative_K.Max - BLF.Normative_K.Min;
17. Return BLF;
End

Algorithm 3: makeCompElite

Input: Belief Space *BLF*, Population of Elites *Pop_Elites*, Number of Elites to choose *CC.nElites*
Output: Comprehensive Elite *CompElite*
Begin
1. CompElite ← [];
2. Rank the Elites decreasingly based on their fitness scores
3. topElites ← Select *CC.nElites* number of top-ranked Elites
4. totalFscore ← sum(topElites.Fscores);
5. freqMatrix ← calcFrequency(topElites); //Calculate VMs' frequency per column.
6. **For** i = 1 to nTask **do**
7. VM_Score ← 0;
8. **For** j = 1 to CC.nElites **do**
// calculate the normalized relative frequency
9. VM_freq ← find_VM_Freq(freqMatrix, topElites[j].VM[i]);
10. Relative_freq ← VM_freq/CC.nElites;
11. Normalized_VM ← (Relative_freq × topElites[j].Fscore) / totalFscore;
12. **If** (Normalized_VM >= VM_Score) **then**
13. Best_VM ← topElites[j].VM[i]; // Record the VM number
14. VM_Score ← Normalized_VM; // Update VM_Score
15. **End If**
16. **End For**
17. CompElite.VM[i] ← Best_VM;
18. **End For**
19. Return CompElite;

fitness scores. Then, a number of top-ranked Elites, *nElites*, are selected (line 3). And the sum of their fitness scores is calculated in line 4. In line 5, given the list of top-ranked Elites, the frequency of each VM, is computed per column. In line 6-18, for each task, the normalized relative frequency for each VM is calculated. In this loop a VM with the highest normalized frequency is selected and assigned to the corresponding task in the comprehensive Elite. At the end, in line 19, the Comprehensive Elite is returned.

VI. Experimental Results

In this section, we present and discuss the experimental results and compare iCATS with the other competitive evolutionary workflow scheduling approaches.

A. Performance Evaluation

We compare the performance of iCATS with Random, GA, PSO, and CA approaches on selected workflow scheduling problems. In order to evaluate and compare the performance of these approaches, we use four synthetic workflow applications based on real scientific workflows named: Montage, CyberShake, Epigenomics, and LIGO Inspiral (Figure 3) [20]. These workflow applications are developed for various scientific domains such as bioinformatics and earthquake data sets. We select different sizes of these workflow applications. We assume each task can be executed on every cloud virtual machine. Each type of cloud virtual machine consists of an hourly cost for virtual machine utilization and the execution time is based on the complexity level of the analytics workload.

In Tables 1 and 2, we list the information of workflow tasks and virtual machines used in our experiments. Table 2 includes empirical data of ten cloud virtual machine types with their computation capacities and the associated costs adopted from Amazon EC2. In this list, the first virtual machine type, VM_1, is the slowest and cheapest option and

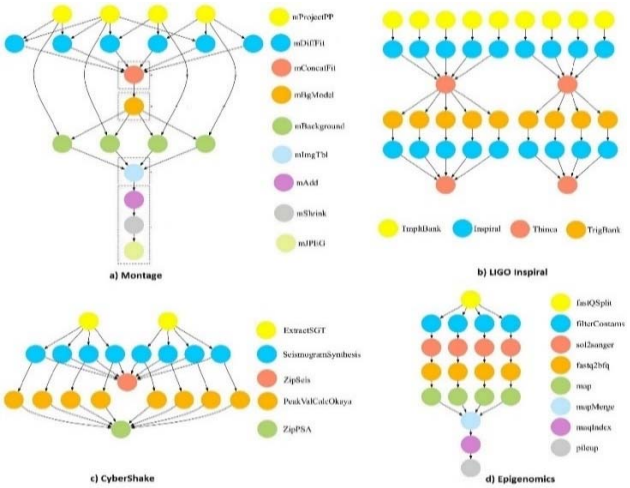

Figure 3: The structure of scientific workflows [20].

TABLE 1: DEFAULT SETTINGS USED IN THE EXPERIMENTS.

# of workflow Tasks	25-5000
# of instructions in a Task	1000 - 10000
Data size (MB)	0.1 - 100
# of virtual machines	10
Population size	100 - 500
Maximum generation	70 - 100
Crossover probability	0.8 - 0.9
Mutation probability	0.1 - 0.2
Maximum iteration	200
Number of Elites	0.2 × Pop

TABLE 2: CLOUD VIRTUAL MACHINE TYPES USED IN THE EXPRIMENTS.

VM Type	Million Instructions per second (MIPS)	Data Transfer (MB/S)	Cost per Hour ($)
VM_1	50	10	0.095
VM_2	200	25	0.38
VM_3	300	35	0.57
VM_4	450	50	0.76
VM_5	600	65	0.95
VM_6	700	80	1.14
VM_7	750	95	1.26
VM_8	850	110	1.57
VM_9	1000	125	2.16
VM_{10}	1200	140	2.48

virtual machine of type VM_{10} is the fastest and the most expensive option in terms of cost.

We demonstrate the performance of Random, GA, PSO, CA, and iCATS approaches in terms of the workflow execution cost and makespan defined in section IV. In our experiments, we assume the data transfer rates among all virtual machine types are fixed.

B. Results and Analysis

iCATS was evaluated against the other four approaches, Random, GA, PSO, and CA, using four distinctive workflow applications. The comparisons were compared with different levels of workflow complexity and with different provided deadlines. We did the experiments by varying the complexity of the workflows in term of the number of tasks and presented the results of cost parameter.

We compared the results of all five approaches and have demonstrated the relative cost minimizations by varying the number of workflow tasks from 25 to 5000. In Figure 4, we show the workflow execution cost in terms of dollar by varying the number of workflow tasks and fixing the number of virtual machines to 10. In these experiments, sufficient deadlines were provided to execute each workflow as there are some cases that the provided deadlines are not enough to complete the workflow. The workflow execution costs are increased by increasing the number of workflow tasks over all four strategies.

When compared to other workflows, CyberShake workflow has a relatively simpler structure and consequently the results of the different scheduling algorithms appear to be very similar. It can be seen that iCATS algorithm outperforms Random, GA, PSO, and CA approaches. This results in greater improvement margin with a greater number of workflow tasks. In the next step, we compared the cloud virtual machine utilization of the five approaches (Figure 5). We calculated the workflow makespans by varying the numbers of workflow tasks for all the four workflow applications. From the figure, we can observe iCATS exhibited statistically better performance when compared to the other approaches across all four categories of workflow applications and all sizes of each. iCATS has efficiently utilized the max provided deadline to minimize the workflow total execution cost. In addition, as the size of each of the four workflow applications was increased, the iCATS system was observed to be less affected by the problem increases in all cases across all problems. This is a particularly good trait for a scheduler in the cloud since environments and problem sizes can change dramatically over time.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a new evolutionary workflow scheduling algorithm by adopting an improved version of Cultural Algorithms by allowing different knowledge sources to collaborate on a new solution. The goal was to minimize the workflow execution cost while meeting the specified deadlines. We compared our strategy with Random, GA, PSO, and CA approaches.

The results of the comparison illustrate the performance advantages of iCATS approach. Our main contributions are 1- adopting CA to design the novel workflow scheduling solution; 2- improving CA performance by adding a comprehensive solution (Comprehensive Elite) to the population; and 3- defining a new fitness function to consider both workflow makespan and execution cost in one formula. In the future, we plan to improve the performance of our strategy by adopting a novel rule-based Cultural Algorithms with improved heuristic to find the optimal scheduling solutions. In addition, we will compare iCATS with more existing scheduling algorithms. Also, there will be situations when a multi-objective problem needs to be addressed. The extension of Cultural Algorithms to multi-objective problems will be considered as well. It will be of interesting to see how the iCATS algorithm is able to accommodate scaling up for those types of problems.

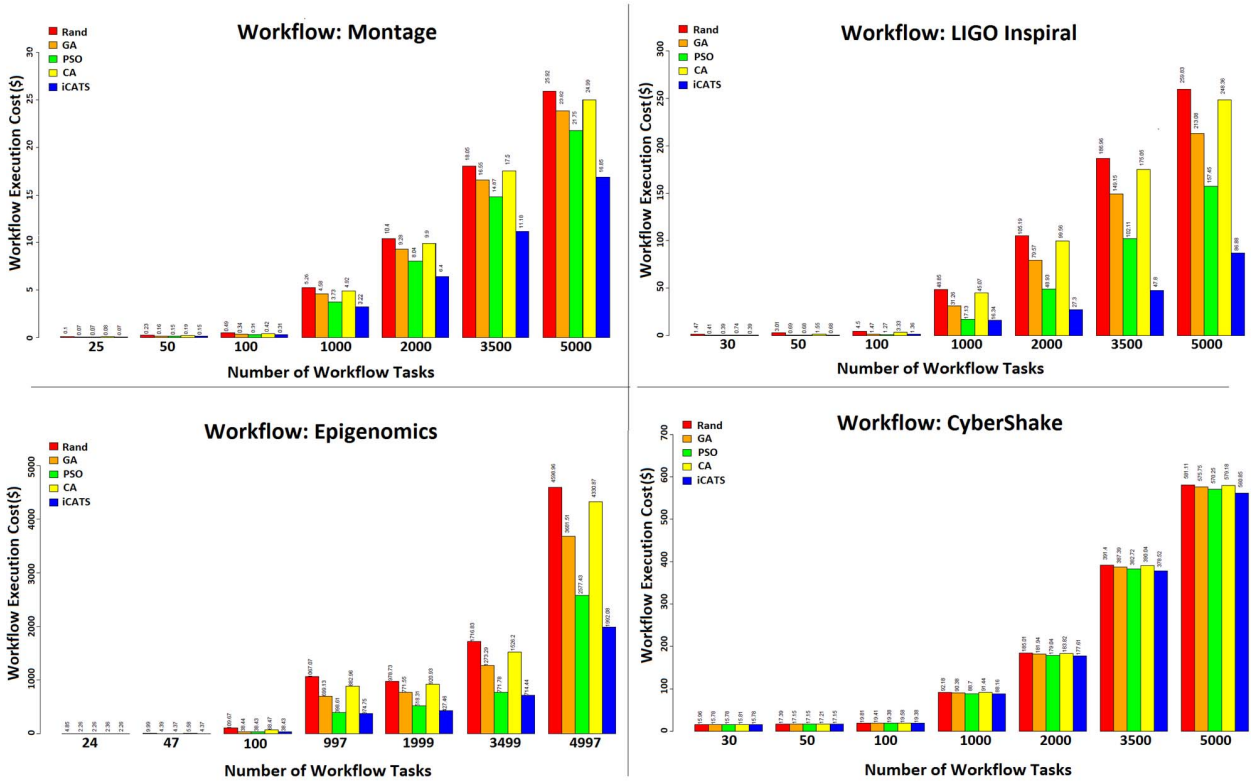

Figure 4: Comparisons of Workflow Execution Cost.

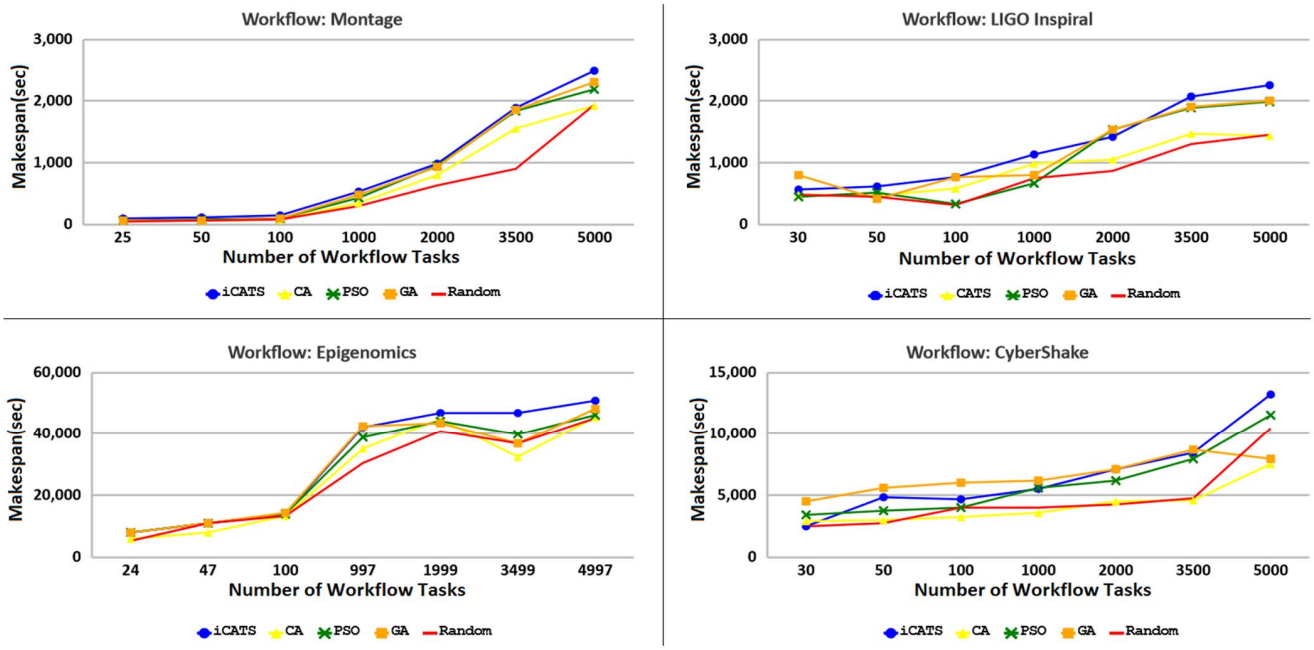

Figure 5: Cloud virtual machine utilization.

ACKNOWLEDGMENT

This work is supported by National Science Foundation, under grant NSF ACI-1738929 and 1747095.

REFERENCES

[1] C. Lin, and S. Lu, "SCPOR: An elastic workflow scheduling algorithm for services computing", pp. 1-8, SOCA 2011.

[2] J. Liu, E. Pacitti, P. Valduriez, D. Oliveira, and M. Mattoso, "Multi-objective scheduling of Scientific Workflows in multisite clouds", Future Generation Comp. Syst. 63, pp. 76-95, 2016.

[3] H. Arabnejad, and J. G. Barbosa, "Multi-QoS constrained and Profit-aware scheduling approach for concurrent workflows on heterogeneous systems", Future Generation Comp. Syst. 68, pp. 211-221, 2017.

[4] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds", Future Generation Comp. Syst. 29(1), pp. 158-169, 2013.

[5] F. Ian, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared", In Grid Computing Environments Workshop, 2008. GCE'08, pp. 1-10, IEEE, 2008.

[6] C. Q. Wu, and H. Cao, "Optimizing the Performance of Big Data Workflows in Multi-cloud Environments Under Budget Constraint", SCC, pp. 138-145, 2016.

[7] F. Llwaah, J. Cala, N. Thomas, "Simulation of Runtime Performance of Big Data Workflows on the Cloud", EPEW, pp.141-155, 2016.

[8] M. Ebrahimi, A.Mohan, and S. Lu, "Scheduling Big Data Workflows in the Cloud under Deadline Constraints", in Proc. of the IEEE International Conference on Big Data Computing Service and Applications (BigDataService 2018), pp. 33-40, 2018.

[9] A. Mohan, M. Ebrahimi, S. Lu, A. Kotov, "Scheduling Big Data Workflows in the Cloud under Budget Constraints", in Proc. of the IEEE Scalable Cloud Data Management Workshop, in conjunction with IEEE Conference on Big Data, pp. 2775-2784, 2016.

[10] K. Bochenina, N. Butakov, A. Dukhanov, and D. Nasonov, "A clustering-based approach to static scheduling of multiple workflows with soft deadlines in heterogeneous distributed systems", in Proc. Of the Procedia Computer Science 51, pp. 2827-2831, 2015.

[11] A. Deldari, M. Naghibzadeh, S. Abrishami, and A. Rezaeian, "A Clustering Approach to Scientific Workflow Scheduling on the Cloud with Deadline and Cost Constraints", in Proc. of the Amirkabir International Journal of Modeling, Identification, Simulation & Control 46.1, pp. 19-29, 2014.

[12] J. Yu, and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms", in Proc. of the Scientific Programming, v.14 n.3,4, pp. 217-230, 2006.

[13] R. G. Reynolds, and B. Peng, "Cultural algorithms: Modeling of how cultures learn to solve problems", Proc. 16th Int. Conf. Tools Artif. Intell. (ICTAI) pp. 166-172, 2004.

[14] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization based heuristic for scheduling workflow applications in cloud computing environments", In Advanced information networking and applications (AINA), 24th IEEE international conference on, IEEE, pp. 400–407, 2010.

[15] M. Z. Ali, R. G. Reynolds, "Cultural algorithms—A tabu search approach for the optimization of engineering design problems", Soft Comput. vol. 18 no. 8 pp. 1631-1644, 2013.

[16] S. Mirshekarian, D. Sormaz, "Correlation of Job-Shop Scheduling Problem Features with Scheduling Efficiency", Expert Systems with Applications, Volume 62, pp. 131-147, 2016.

[17] M. A. Rodriguez, and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds", in IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 222-235, April-June 2014.

[18] J. Huang, "The workflow task scheduling algorithm based on the GA model in the cloud computing environment", JSW 9, pp. 873–880, 2014.

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

[20] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and Profiling Scientific Workflows", Future Generation Computer Systems, vol. 29, no. 3, pp. 682-692, 2013.