A Multiple-Objective Workflow Scheduling Framework for Cloud Data Analytics

Orachun Udomkasemsub
Computer Engineering Department
King Mongkut's University of Technology Thonburi (KMUTT)
Bangkok, Thailand
orachun_chun@hotmail.com

Li Xiaorong
Institute of High Performance Computing (IHPC)
Agency for Science, Technology and Research (A*STAR), Singapore
lixr@ihpc.a-star.edu.sg

Tiranee Achalakul
Computer Engineering Department
King Mongkut's University of Technology Thonburi (KMUTT)
Bangkok, Thailand
tiranee@cpe.kmutt.ac.th

Abstract— **One of the most important characteristics of a cloud system is elasticity in resources provisioning. Cloud fabric often composes of massive and heterogeneous types of resources allowing the sciences and engineering applications in many domains to collaboratively utilize the infrastructure. As the cloud systems are designed for a large number of users, a large volume of data, and various types of applications, efficient task management is needed for cloud data analytics. One of the popular methods used in task management is to represent a set of tasks with a workflow diagram, which can capture task decomposition, communication between subtasks, and cost of computation and communication. In this paper, we proposed a workflow scheduling framework that can efficiently schedule series workflows with multiple objectives onto a cloud system. Our designed framework uses a meta-heuristics method, called Artificial Bee Colony (ABC), to create an optimized scheduling plan. The framework allows multiple constraints and objectives to be set. Conflicts among objectives can also be resolved using Pareto-based technique. A series of experiments are then conducted to investigate the performance in comparison to the algorithms often used in cloud scheduling. Results show that our proposed method is able to reduce 57% cost and 50% scheduling time within a similar makespan of HEFT/LOSS for a typical scientific workflow like Chimera-2.**

Keyword: workflow scheduling, multiple-objective optimization, cloud computing, Artificial Bee Colony

I. INTRODUCTION

Cloud computing is an emerging technology that combines a large amount of computer resources into a virtual place so as to provide an on-demand computing facility to users. The cloud system will provide its computing resource to users according to the user requests, in which the amount and capacity of compute resource are highly configurable. A cloud-based system allows pay-as-you-go for billing and users can access their purchased resource from anywhere at any time. Because of its elasticity, cloud computing is suitable for the execution of complicated computational tasks and scientific simulations, which may require a spike of computational resources, e.g., computing nodes and storages.

In addition, cloud systems are suitable to serve data intensive applications and simulations. These applications often compose of various complex computational tasks. Examples include earthquake science, gravitational-wave physics, bioinformatics, and climate change analytics. Data intensive applications involve a large number of input, output, and/or intermediate data. The tasks of these applications may also have dependencies among each other. In order to handle such a complex data flow, workflow programming is introduced.

Workflow [1] is a model that describes the sequence of the tasks by using directed acyclic graphs (DAG). It consists of nodes that are linked according to their flow dependencies: G = (V, E), which G represents the graph, V represents a vertex or a node that is an operation or a task, and E represents the edge that shows the relationship between two nodes. Due to the characteristic of a graph, a parent node is always executed before its child nodes since the data from the parent node might be passed down. Once the workflow is constructed, the overall sequence of the application is presented. Moreover, it is convenient to add a new node or edit the sequence of the tasks in the workflow. The groups of tasks that are frequently used can be created as the workflow templates. The workflow is executed using the given parameters or input data. The result is obtained as the output at the last step. A workflow can be submitted to a cloud system for execution.

In order to optimize the performance of the resource provisioning process in a cloud system, a workflow scheduling framework is needed. The framework can allocate the tasks within the workflow into appropriate resources according to certain scheduling strategies. In general, the execution time of a workflow varies depending on the nature of tasks. Normally, the bigger the workflow, the longer execution time it may take. Workflows for scientific simulations are typically big and can be compute-intensive, data-intensive, and/or communication-intensive. In other words, the tasks in the applications are concerned with the complicated computations, amount of data involved, as well as the communication between the tasks. The users may also work towards different goals, such as, the shortest possible execution time, the most inexpensive execution cost, or the optimized throughput. Moreover, some users may need to create a schedule plan that satisfies more than one goal. Due to the aforementioned variety of users' requirements, the adoption of the brute force search techniques to optimize the schedule plan is not practical.

In this paper, the design of a multiple-objective scheduling framework and a meta-heuristic workflow scheduling algorithm based on the Artificial Bee Colony (ABC) are introduced. The proposed framework aims to deal with complex workflows for data analytics by handling various workflow structures, short term and long term scheduling, dependency mapping on a heterogeneous computing environment, and most importantly multiple objectives that may contradict one another. Techniques in the classes of heuristics and meta-heuristics are surveyed

978-1-4673-1921-8/12/$31.00 ©2012 IEEE

and presented in section 2. Section 3 provides background study on our method of choice. The design of a scheduling framework is described in section 4. Section 5 presents experiments and results that compare and contrast our framework and the popular methods used currently. Finally, the concluding remarks are offered in Section 6.

II. RELATED WORK

A large variety of algorithms have been proposed to solve the scheduling problems [2]. In this section, we investigate existing algorithms based on heuristic and meta-heuristic methods. These categories of algorithms are powerful and can handle both the long-term and the short-term scheduling with a large number of massive workflows in a reasonable amount of time.

A. Heuristic Methods

Heuristic methods are generally used to find a solution of a problem without exhaustively searching the entire solution space. The methods depend on information learning of a specific problem domain. Each heuristic-based approach is often designed to fit only a particular type of problem. Each method uses a set of application-specific rules to increase the probability of getting a good solution. The algorithms can generally generate a good solution in a small computational time. In task scheduling problem domain, common heuristic-based approaches can be divided into three categories: Individual Task Scheduling, List Scheduling, and Cluster-Based and Duplication-based Scheduling.

The most simple one is *individual task scheduling* [3] but it may suit only those workflows which have simple structures such as a pipeline. It continually maps each individual ready task into a resource that can provide the earliest finish time for that task without any consideration of other tasks and their dependencies. The *list scheduling* algorithms are good for workflows which have many tasks competing for limited number of resources. It assigns a priority to each task and creates a schedule based on the assigned priorities. There are two modes of list scheduling: batch mode [4] and dependency mode [5]. The batch mode approach sets tasks' priority based on their execution time whereas dependency mode does the same but based on the weight (communication and execution of tasks) of their critical path. The benefit of using these list scheduling algorithms is that the overall execution time is usually minimized. However, the algorithms cannot deal with a resource competition problem where workflows contain many parallel tasks with equal critical path length. The other drawback is the need to repeatedly calculate the priorities, which increases time of scheduling for a large workflow.

The *cluster-based and duplication-based scheduling* algorithms [7] focus on minimizing the makespan. The algorithms try to reduce the communication time between interdependent tasks by clustering communication intensive tasks and mapping them into the same resource. Some tasks are also duplicated to avoid data transmission between resources.

Even though, these aforementioned methods are effective, they can satisfy only a single scheduling objective. For example, they can give a schedule that has the smallest makespan but the execution cost can be high. In order to modify the algorithms to cope with multiple objectives, the required time to compute the scheduling plan may increases tremendously. Moreover, it is rather difficult to design a generally scheduling framework for various types of workflows based on these heuristics algorithms. We then explore the use of Meta-Heuristics approaches which are used widely in various optimization problems currently.

B. Meta-Heuristic Methods

In contrast to heuristic-based method, meta-heuristic methods make no assumptions on the problem domain and can deal with a massive search space. The methods find a near-optimal solution by improving a candidate solution according to a given quality measurement. A near optimal schedule can thus be generated for all workflow structures with a single general framework. There are several algorithms presented in the current literature.

Simulated Annealing (SA) [8] and Genetic Algorithm (GA) [9] are the classic meta-heuristic methods. They both use the same assumption that better solutions can be derived from good solutions. The algorithms randomly modify a current good solution to create a new solution in each iteration. In SA, the annealing process is adopted to create a new solution. In GA, a new solution is created by using genetic operations: selection, crossover, and mutation. The new solutions will be accepted based on their quality of workflow schedule.

Swarm Intelligence (SI) is another group of meta-heuristic methods designed based on an intelligent group of simple self-organized agents working to reach the global goal. In SI, multiple solutions may be created and/or updated in each iteration. Particle Swarm Optimization (PSO) [10] adjusts the solutions based on each individual solution and the global best solution, similar to a migratory behavior of fish schools and bird flocks. In contrast, Ant Colony Optimization (ACO) [11] and Artificial Bee Colony (ABC) [15] do not directly involve the global best solution in the search method. Based on ant and bee foraging behavior, the methods make a collective decision within a subgroup of agents.

One of the problems in the meta-heuristic methods is the ability to avoid getting stuck with suboptimal solutions. In GA, mutation process is used to solve the problem. ACO allows the pheromone to evaporate in order to force the search of a new path. ABC, on the other hand, use scout bee agents to randomly generate a new solution when a solution can no longer be improved for some iterations.

These meta-heuristic algorithms have been compared in many literatures [12][13]. The results from the previous works [14] showed that ABC-based algorithm outperforms other algorithms in finding optimal solutions for job shop scheduling problem domains, which is similar to the workflow scheduling problem defined in this paper. In other words, ABC can converge quickly and the optimum schedule can be found with a fewer number of iterations required. ABC also has a lower algorithm complexity when compared to many meta-heuristic algorithms. Even though PSO and ABC have a similar search process and

require comparable computing time, ABC has a better chance for solution improvement within each iteration. In ABC, employed and onlooker bees allow the search process to converge quickly. If some solutions become trapped in local optima, scout bees will also find a new solution outside the local space. Moreover, ABC can produce an optimal solution by considering the entire workflow rather than some parts of the workflow. It can thus be used with various types of workflow structures.

In this paper, we proposed our design of a multiple-objective workflow scheduling framework based on ABC for cloud data analytics. Next section provides a brief description of the ABC algorithm.

III. ARTIFICIAL BEE COLONY ALGORITHM

Artificial bee colony algorithm (ABC) [15] is one member of swarm-based algorithms for optimization problem defined by Dervis Karaboka in 2005 that simulate the foraging behavior of honeybee colonies. The model consists of three types of bee: employed bee, onlooker bee, and scout bee. The food source position represents the solution to the problem and the nectar amount of food source represents the quality of associated solution. ABC repeats iterations after iterations until the pre-defined stopping criteria are met. ABC can be described in three phases as explained below.

Firstly, *in the initialization phase*, the amount of food sources or solution vectors X, where X_{ij} is a solution of the i^{th} employed bee in the j^{th} dimension, will be randomly initialized using (1). U_j and L_j are the upper bound and lower bound of X_{ij} respectively, and then assigned to each employed bee. Next, *in the employed phase*, each employed bee will search for new food source based on the neighbor of the current associated food source in the memory. The new food source can be determined by using (2). R_j is a random solution in the j^{th} dimension. Φ is a random number from -1 to 1. After the new food source position is determined, the better one, which has a higher fitness value, is kept. After that, *in the onlooker bee phase*, each onlooker bee will select a food source from one employed bee based on the fitness. The food source with higher fitness will have more chance to be selected by onlooker bees. The probability of each food source to be selected can be calculated using (3) where $fit(X_i)$ is the fitness value of the solution X_i and n is the number of bees.

$$X_{ij} = L_j + rand(0,1) * (U_j - L_j) \quad (1)$$

$$V_{ij} = X_{ij} + \Phi (X_{ij} - R_j) \quad (2)$$

$$P_i = fit(X_i) / \sum_{1}^{n} fit(X_i) \quad (3)$$

Once the food sources are selected by every onlooker bee, the better neighbor food source positions are determined as occurred in employed bee phase. In the scout bee phase, if the fitness of food sources from employed bee is not improved for some limit, these food sources will be abandoned. The employed bees that have these abandoned food sources will become scout bees and find other food sources randomly to replace the abandoned food sources in order to avoid getting local optima.

ABC has several strengths. First, it has more chances to get better solutions. ABC algorithm has been benchmarking against Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) in [12]. The results showed that ABC can give lower values from the benchmark functions compared to the others in every dimension. Moreover, good optimization algorithm must have ability of exploitation and ability of exploration, and ABC has these abilities. ABC exploits the food sources by letting the onlooker bees to select solutions based on their fitness, and then improving them. In addition, ABC explores the search space by having scout bees to find new solution randomly. These are the reasons why ABC is among the best optimization algorithms.

IV. ADOPTING ABC IN A MULTIPLE-OBJECTIVE WORKFLOW SCHEDULING PROBLEM

We design an ABC-based algorithm for multiple-objective workflow scheduling & optimization and conduct experiments to study its efficiency and effectiveness by comparison. The goal of the experimental framework is to optimize the makespan and cost of execution of the application within a budget constraint in the scheduling process. The main challenge in adopting ABC is to define fitness functions and values, which are described in section IV.A. Section IV.B then describes the framework of ABC adoption for multiple-objective workflow scheduling.

A. Definition of the Fitness Functions and Values

The objectives of our study are to find the global optimization of the makespan that satisfies the cost constraint, i.e. we try to find a schedule that minimizes the makespan value as well as minimizes the execution cost.

The first objective value is the total cost of a schedule, which can be calculated using (4). For each resource, *R*, its cost consists of the execution cost (EXEC) and the communication cost (COMM) as in (5). The execution cost of a resource is the total execution time (EXET) of every task which is scheduled to be run on this resource multiply by the unit cost of using this resource ($EXEC_R^0$). The execution cost of each resource can be calculated using (6). Equation (7) shows the calculation of the communication cost based on an amount of data transfer during the execution. From the equation, DAT_T^{in} and DAT_T^{out} represents an amount of data which is required to be transferred in and out of the originally assigned resource of task T. The total communication of resource R is the product between the cost of the unit transfer ($COMM_R^0$) and the sum of the inbound and outbound data transfer for all tasks scheduled on R.

$$TotalCost = \sum_{R} (COST_R) \quad (4)$$

$$COST_R = EXEC_R + COMM_R \quad (5)$$

$$EXEC_R = EXEC_R^0 \sum_{T \to R} EXET_{T,R} \quad (6)$$

$$COMM_R = COMM_R^0 \times \left(\sum_{T \to r \neq R} DAT_T^{in} + \sum_{T \to r \neq R} DAT_T^{out} \right) \quad (7)$$

Next, the second objective value is the makespan of a workflow (the finish time of the last task), which can be computed as illustrated in Algorithm 1.

The algorithm iterates through all tasks. For each task, its start time and finish time are computed. If the task has parents, the maximum of ended time plus communication time from all parents are calculated as a ready time of the task. The start time is the maximum between parents' ended time and ended time of the resource allocated for the task. The end time of the task is the start time plus the execution time of the task on the scheduled resource. Then, the end time of the resource is updated. After iteration ends, the end time of the last task is returned as the schedule makespan.

In our work, we cannot compare the quality of any two solutions for both objectives at once. The best solution in term of makespan may be of low quality when execution cost is considered. Therefore, Pareto analysis concept is applied to balance the solution quality according to both objectives. All solutions evaluated to be non-dominated and deliver a total benefit relatively close to the optimal one in all dimensions will be stored in a Pareto-optimal set. The best solution can then be selected from this set.

The two objectives can then be combined for multiple-criteria decision making within our ABC framework, objectives must be prioritized. Then, the weights are used to merge fitness values from multiple objectives into one equation. Weights are used as the coefficient of each fitness value as shown in (8). fit(X) is an integrated fitness value from all objectives. W_i is a weight for objective i, and $fit_i(X)$ is a fitness value from objective i. However, fitness values from each objective may have different ranges of values. Thus, these fitness values must be normalized into the same scale. Equation (9) shows a method used for fitness value normalization where fit^{min} and fit^{max} are the minimum and maximum possible fitness value respectively. Finally, a pair of solutions will be compared using this aggregated fitness function and the best solution can be selected.

$$fit(X) = \sum_i W_i fit_i(X) \tag{8}$$

$$fit_i(X) = \left(fit_i(X) - fit_i^{min} \right) / \left(fit_i^{max} - fit_i^{min} \right) \tag{9}$$

In the next section, the ABC algorithm is described in the context of a workflow scheduling problem.

B. Algorithm Mapping

In our work, a solution representation is defined using a vector of integer. The length of the vector is equal to the number of tasks in the input workflow. The index represents task ID and the value represents resource ID. For example, Fig. 1 shows the schedule of a workflow containing n tasks. The first, second, and third tasks will be run on the fourth, fifth, and first resource respectively.

A solution vector can be generated by randomly selecting a set of IDs of available resources as shown in (10). From the equation, X is a solution and X[t] is a resource used to run task t. R_i is an available resource and $U[R_1,R_2,\ldots R_n]$ is a function that select a resource ID randomly. Moreover, solution may be randomly adjusted to improve the solution quality. The algorithm randomly picks some tasks and may re-assign those tasks to other resources as described in (11). In the equation, t is a pre-defined probability used to adjust the solution. U[0,1) is a uniform random number from 0:inclusive to 1:exclusive.

$$X[t] = U[R_1, R_2, \ldots, R_n] \tag{10}$$

$$X[t] = \begin{cases} U[R_1, R_2, \ldots, R_n] & \text{if } U[0,1) > t \\ X[t] & \text{Otherwise} \end{cases} \tag{11}$$

1	*2*	*3*	*...*	*n*
4	5	1	…	9

Figure 1. Example of a vector used to represent a solution.

Based on the aforementioned solution representation, ABC is adopted to optimize the workflow scheduling as illustrated in Algorithm 2. The steps of operation can be divided into 4 phases: initial phase, employed phase, onlooker bee phase, and scout bee phase.

At *the initial phase*, the initial parameters such as the number of bee agents are set. The algorithm then generates a number of random solutions, which is equal to the specified number of employed bees. Every solution in this phase is generated by randomly assigning tasks to available resources as (10). The solution that cannot satisfy the budget constraint will be assigned with negative infinity as its fitness value. The optimal set of solutions is updated when a new solution is found. After that, each solution is adjusted by re-assigning some tasks to new resources based on neighboring food sources using (11) in *the employed phase*. In other words, an employed bee updates its old food source by randomly taken food source information (some task-resource assignments) from its neighbors. All solutions' fitness values are then re-calculated and the old food source in the employed bee's memory will be replaced by the new candidate food source, if the new task assignment can dominate the old one. In this case, the new solution can dominate the old one either if the new one has lower cost and lower or equal makespan, or if the old one has lower or equal cost and lower makespan.

In *the onlooker bee phase*, the onlooker bees select, from all solutions produced by employed bees, a set of solutions based on probability value calculated using (3) and adjust them as employed bees do. The best solution is then identified by comparing the fitness values of the selected candidate solutions. If the new best solution identified by the onlooker bees is not improved for a specified time period, the solution will be abandoned in *the scout bee phase*. The scout bee will then replace abandoned solutions with newly generated solutions obtained in a similar fashion as described in the initial phase or with the best so far solution from the optimal set.

The algorithm repeats for a specified number of iterations and the best solution from the optimal set representing a task schedule, is returned.

Algorithm 1 Makespan Calculation

```
FOR each resource r DO
  last ended time of r <-- 0
END FOR
FOR each task t in schedule ordered by task number DO
  current resource r <-- resource that run task t
  IF task t has parent tasks THEN
    task ready time trt <-- 0
    FOR each parent tasks p of task t DO
      IF t = r THEN
        trt = MAX(trt, ended time of p)
      ELSE
        comm_time = output size of p /
          communication speed;
        trt = MAX(trt, ended time of p + comm_time)
      END IF
    END FOR
  END IF
  start time <-- MAX(trt, last ended time of r)
  ended time of t <-- start time + execution time of t
on r
  last ended time of r <-- ended time of t
END FOR
RETURN ended time of last task
```

Algorithm 2 ABC-based Multiple-Objective Workflow Scheduling

```
frontier <-- empty list
//Initial phase
FOR i=1 to number of bees DO
  solution sols[i] <-- new random solution
  calculate fitness value of sols[i]
  evaluate sols[i]
END FOR
FOR loop = 0 to number of iterations DO
  //Employed bee phase
  FOR i=1 to number of bees DO
    sols[i] <-- adjust sols[i]
    calculate fitness value of sols[i]
    evaluate sols[i]
  END FOR
  //Onlooker bee phase
  FOR EACH onlooker bee ob DO
    solution s <-- select solution sols[i] from sols
      based on fitness
    sols[i] <-- adjust s
    evaluate sols[i]
  END FOR
  //Scout bee phase
  FOR i=1 to number of bees DO
    IF sols[i] not improved for a specified times THEN
      sols[i] <-- random(new random solution,
        best solution in frontier)
      evaluate sols[i]
    END IF
  END FOR
END WHILE
RETURN best solution in frontier
```

V. EXPERIMENTS

This section presents our experimental method and results. The goal of the experiments is to compare the scheduling plans created using the ABC framework described in section VI and the algorithms used in many grid schedulers (HEFT and HEFT/LOSS).

A. Experimental Method

The algorithm presented in the previous section was implemented in Java and executed with a sample case of workflow and resources. To evaluate the performance of our proposed algorithm, we study two different workflows. One is called Chimera-2 containing 123 tasks, whose structure is illustrated in the Fig. 2. Chimera-2 was selected because it represents a big and complex real-world workflow application with a large number of tasks and dependencies [5]. Besides Chimera-2, a random generated workflow structure with 87 tasks (RandomWF) was also used. The amount of data dependencies are the random generated between 10 to 1,000 data unit. We showed that our proposed scheduling framework is cappable for any arbitary workflows.

In the experiments, we randomly generated three classes of resources: four of good quality resources, six medium quality, and ten poor quality resources. The better resources have higher performance and require shorter time to execute a task. However, they require higher cost of execution (budget).

In an initial phase of the algorithm framework, we initialized the parameters as shown in the Table I.

We run ABC algorithm with two different settings. In ABC #1, the weight of cost and makespan were set to 0.0 and 1.0, respectively. In this case, the proposed ABC algorithm only optimizes the makespan with a constraint that the solution's cost must not exceed the given budget. Hence, it is a single objective optimization with the constraint.

In ABC #2, the weight of cost and makespan were set to 0.5 in order to optimize based on two objectives with the same priority. With the two objectives, the solution in the ABC #2 is expected to obtain a solution with a low cost and a short makespan.

In addition to ABC framework, HEFT and HEFT/LOSS (HEFT with LOSS extension) were also implemented in Java to compare with ABC #1. All three algorithms were implemented under Ubuntu 10.04 with Xeon E5500 CPUs and 2 GB ram. The algorithms' effectiveness and efficiency were then measured by the solution quality (low cost and short makespan) and the execution time to obtain a schedule plan, respectively. The cost of the resulting schedules must not exceed the specified budget. Each experiment was repeated 200 times.

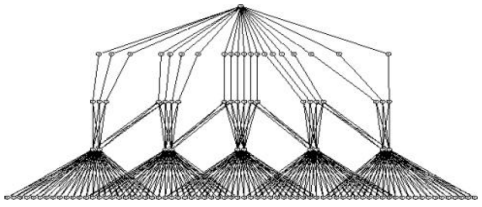

Figure 2. Chimera-2 workflow structure [5].

B. Results and Discussions

We gathered the values of solutions' makespan, cost and scheduling time (execution time) by running each algorithm in the experimental settings. Fig. 3 to 5 show the results.

From the results, HEFT/LOSS gives both lower cost and lower makespan than HEFT, while HEFT takes much less scheduling/execution time, i.e. the duration for a scheduling algorithm to find the final solution, than HEFT/LOSS. Although HEFT was fast, it produced a very high makespan and its cost did not meet the budget constraint. Hence, HEFT is impractical for use in this case of workflow scheduling and will not be considered further in results comparison.

Comparing with HEFT/LOSS, ABC #1 (ABC in the first experiment with 0 weight of cost and 1 weight of makespan) was able to achieve 17% lower cost and 9% lower makespan for Chimera-2. For RandomWF, ABC #1 gave a schedule almost as good as HEFT/LOSS with only 2.49% higher makespan and 0.28% higher cost. However, the scheduling time of ABC#1 was almost twice as fast as HEFT/LOSS for both workflows. Solutions from both algorithms met the budget constraint in all cases. This result concludes that with a single objective optimization, ABC is more efficient than HEFT/LOSS for a typical scientific workflow structure.

From the results of ABC #2 (ABC in the second experiment with 0.5 weight of cost and 0.5 weight of makespan), we observed that the makespan in the solution of ABC #2 is 35% and 28% higher than that of HEFT/LOSS for Chimera-2 and RandomWF, respectively. However, the cost of ABC #2 is 57% and 4.49% lower than that of HEFT/LOSS for Chimera-2 and RandomWF. As these two objectives, makespan and cost, are conflicted with each other in the workflow scheduling, our proposed algorithm resolves the conflict by allowing the makespan to be higher in order to lower the cost. These results present two main advantages of our proposed framework over HEFT/LOSS; First, the algorithm is significantly faster making it more practical to be embedded into cloud schedulers. Second, with comparable results, our framework is more flexible. The priority of objectives can be given by setting the weight values and additional objectives can be added in optimization rules.

As ABC contained random elements when searching for the best solution, it generates slightly different solutions in each run. We conducted the experiments 200 times, and evaluated the performance variation using standard deviation. Table II shows that both ABC #1 and ABC #2 are able to maintain relatively low standard deviations of makespan and cost. As both HEFT and HEFT/LOSS do not include randomness in each run, the standard deviation of makespan and cost from HEFT and HEFT/LOSS were zero.

Fig. 6 to 8 shows the plots of makespan and cost of ABC #1 and ABC #2 with the increasing of the number of iterations for both Chimera-2 and RandomWF.

The makespan of both ABC #1 and ABC #2 for Chimera-2 converged faster than that of RandomWF. This result shows that the algorithm is more suitable to the structured workflow than the unstructured one. When the second objective (cost) is added into the picture, ABC #2 performed better overall for both types of workflows. In other words, ABC #2 is better in balancing the quality of solutions between two objectives. Moreover, the plot of both Chimera-2 and RamdonWF follow the same trend, which shows that our proposed algorithm provides multiple-obejective shceduling solution for any arbitary workflows.

TABLE I. PARAMETER SETTINGS OF THE EXPERIMENTS

Parameters	Values	
	ABC #1	**ABC #2**
Number of bees	50	
Number of iterations for stopping criteria	500	
Number of iterations for unimproved solution (before the solution is abandoned)	20	
Weight of cost	**0**	**0.5**
Weight of makespan	**1**	**0.5**
Budget constraint	1,300,000 (Chimera-2)	
	18,000 (RandomWF)	
Communication speed	10.0 data unit/time unit	

TABLE II. STANDARD DEVIATION OVER AVERAGE OF MAKESPAN, COST OF THE SCHEDULE, AND SCHEDULING TIME FROM EACH ALGORITHM

SD/AVE		Chimera-2	Random WF
Makespan	ABC #1	**0.03**	**0.02**
	ABC #2	**0.03**	**0.04**
	HEFT	0.00	0.00
	HEFT/LOSS	0.00	0.00
Cost	ABC #1	**0.00**	**0.06**
	ABC #2	**0.00**	**0.03**
	HEFT	0.00	0.00
	HEFT/LOSS	0.00	0.00
Scheduling Time (ms)	ABC #1	**0.07**	**0.02**
	ABC #2	**0.07**	**0.02**
	HEFT	**0.13**	**0.14**
	HEFT/LOSS	**0.07**	**0.02**

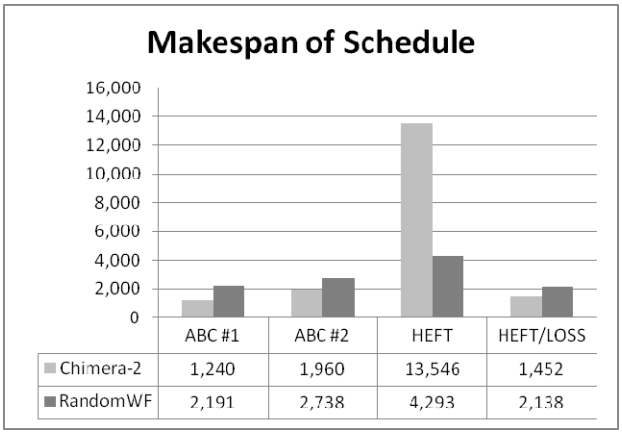

Figure 3. Makespan of running scheduling algorithms ABC#1, ABC#2, HEFT and HEFT/LOSS for Chimera-2 and RandomWF.

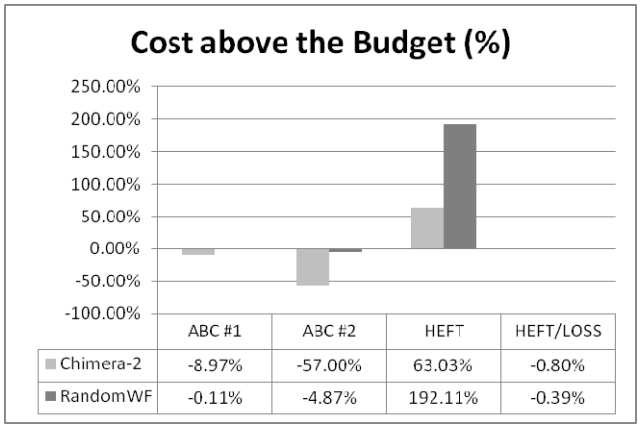

Figure 4. Cost of running scheduling algorithms ABC#1, ABC#2, HEFT and HEFT/LOSS for Chimera-2 and RandomWF.

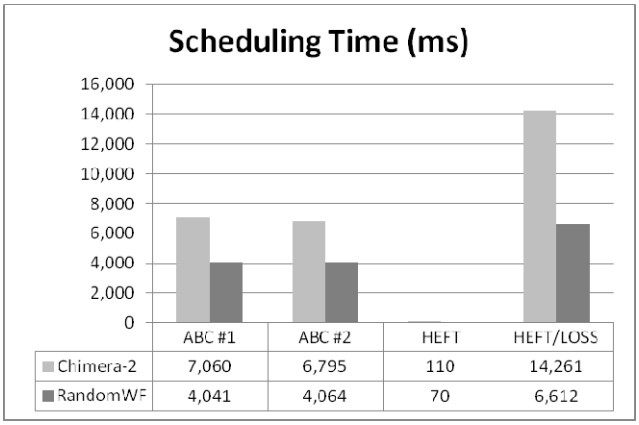

Figure 5. Time in milliseconds taken by running scheduling algorithms ABC#1, ABC#2, HEFT and HEFT/LOSS for Chimera-2 and RandomWF.

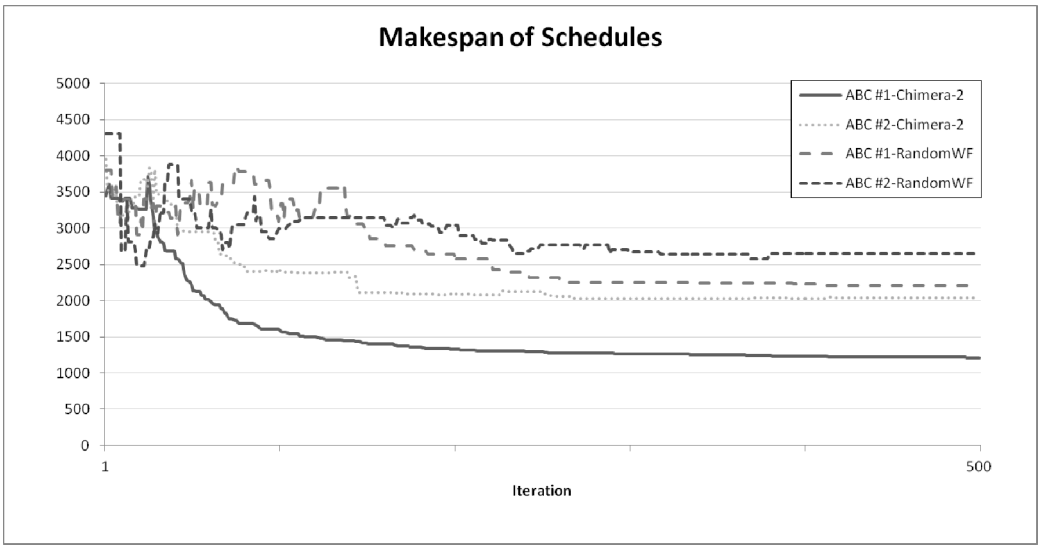

Figure 6. Makespan of schedules from ABC #1 and ABC #2 for the Chimera-2 workflow and RandomWF.

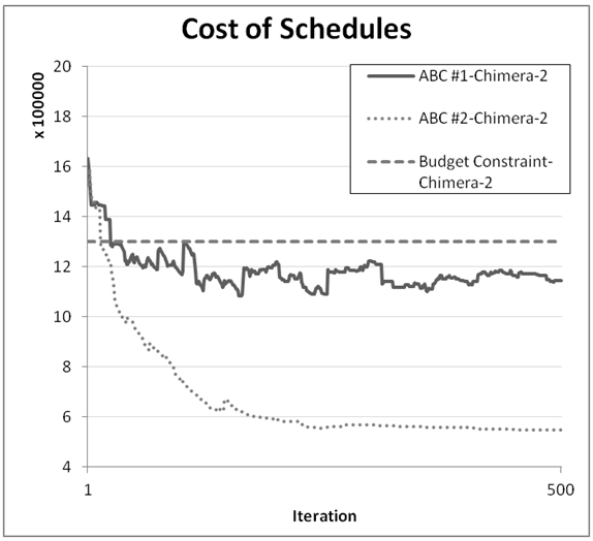

Figure 7. Cost of schedules from ABC #1 and ABC #2 for the Chimera-2 workflow.

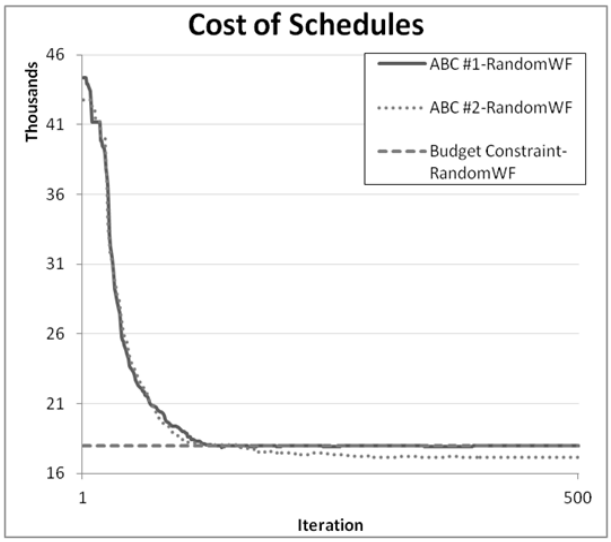

Figure 8. Cost of schedules from ABC #1 and ABC #2 for random generated workflow.

From these experimental results, we can conclude that our proposed algorithm outperforms HEFT/LOSS in both the single objective optimization with the constraint, and multiple objectives optimization problem for structured workflow. It can also produce the solution within a reasonably short computation time.

VI. CONCLUSION

In this paper, an algorithm for workflow scheduling optimization considering multiple objectives based on Artificial Bee Colony has been proposed. The framework allows multiple objectives and constraints to be set in order to optimize the performance of data analytics workflow scheduling in cloud environments. The objective conflicts are resolved using Pareto analysis. The experiments were conducted in various settings and the performance was studied in terms of both solution quality and computational speed. The results were then discussed in comparison to the current algorithms used in several existing schedulers e.g., HEFT and HEFT/LOSS. From the experimental results, we can conclude that our algorithm not only produces a better schedule based on multiple objectives, but also requires a significantly less time for scheduling. With some further adjustment, we believe that the algorithm framework will be practical and has a potential to be adopted in real-world schedulers in the future.

REFERENCES

[1] Reijers, and Hajo A., "Design and Control of Workflow Processes: Business Process Management for the Service Industry," New York: Springer, 2003.

[2] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing, Metaheuristics for Scheduling in Distributed Computing Environments," F. Xhafa and A. Abraham (eds), ISBN: 978-3-540-69260-7, Springer, Berlin, Germany, 2008.

[3] Marek Wieczorek, Radu Prodan and Thomas Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," SIGMOD Record, Vol. 34, No. 3, pp. 56-62, 2005.

[4] M. Maheswaran, S. Ali, H.J.Siegel, D. Hensgen, and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems," in Proc. 8th Heterogeneous Computing Workshop (HCW'99), Apr. 1999.

[5] Luiz F. Bittencourt, Rizos Sakellariou, Edmundo R. M. Madeira, "DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm," in Proc. 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pp.27-34, 2010.

[6] J. Blythe et al., "Task scheduling strategies for workflow-based applications in grids," in Proc. Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), vol. 2, pp.759-767, 2005.

[7] Samantha Ranaweera, Dharma P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," in Proc. 14th International Parallel and Distributed Processing Symposium (IPDPS'00), pp. 445, 2000.

[8] A. YarKhan and J. J. Dongarra, "Experiments with Scheduling Using Simulated Annealing in a Grid Environment," Grid 2002, November 2002.

[9] J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms," Scientific Programming Journal, 14(3-4): 217 - 230, IOS Press, Amsterdam, The Netherlands, 2006.

[10] Gerhard Venter , Gerhard Venter, "Particle Swarm Optimization," AIAA Journal, 2002.

[11] Marco Dorigo, Christian Blum, and Mauro Birattari, "Ant colony optimization and swarm intelligence," in Proc. 6th international conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008.

[12] Dervis Karaboga and Bahriye Akay, "A comparative study of Artificial Bee Colony algorithm," Erciyes University, The Department of Computer Engineering, Melikgazi, 38039 Kayseri, Turkey, 2009.

[13] D. Karaboga and B. Basturk. "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," Journal of Global Optimization, 39, 2007, 459-471.

[14] Anan Banharnsakun, Booncharoen Sirinaovakul, and Tiranee Achalakul, "Job Shop Scheduling with the Best-so-far ABC," Engineering Applications of Artificial Intelligence, 2011.

[15] Dervis Karaboga and Bahriye Basturk, "Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems," Springer Berlin / Heidelberg, 2007.

[16] S. Binato et al., "A GRASP for job shop scheduling". In Ribeiro and Hansen, eds, Essays and surveys on metaheuristics, pp59-79, Kluwer Academic Publishers, 2001.

[17] T. A. Feo and M. G. C. Resende, "Greedy Randomized Adaptive Search Procedures", Journal of Global Optimization, 6:109-133, 1995.

[18] L. Young, S. McGough, S. Newhouse, and J. Darlington. "Scheduling Architecture and Algorithms within the ICENI Grid Middleware". In UK e-Science All Hands Meeting, IOP Publishing Ltd, Bristol, UK, Nottingham, UK, Sep. 2003; 5-12.

[19] Christian Blum and Daniel Merkle, "Swarm intelligence: introduction and applications," Springer, Barcelona, Odense, April 2008.

[20] William Voorsluys, James Broberg, and Rajkumar Buyya, "Cloud Computing: Principles and Paradigms, Introduction to Cloud Computing", John Wiley & Sons, Inc. 2011.

[21] R.T. Marler and J.S. Arora, "Survey of multi-objective optimization methods for engineering, Structural and Multidisciplinary Optimization", Vol. 26, No. 6, pp. 369-395, 2004.

[22] A. Mandal et al., "Scheduling strategies for mapping application workflows onto the grid," in Proc.14th IEEE International Symposium on High Performance Distributed Computing, 2005, (HPDC'05), pp.125-134, 2005.