Chapter 3

Cost and Time Based Task Scheduling Algorithms

3.1 Introduction

In recent years, the escalating demand for efficient resource utilization in cloud environments has propelled the exploration of advanced task scheduling algorithms. The effectiveness of these algorithms is paramount in optimizing both cost and time dimensions, crucial factors influencing the overall performance of cloud systems. Among the plethora of scheduling algorithms, the Jaya Optimization Algorithm has emerged as a noteworthy contender due to its ability to swiftly converge to optimal solutions without the need for complex parameter tuning.

This chapter delves into the realm of Cost and Time-Based Task Scheduling Algorithms, with a specific focus on leveraging the capabilities of the Jaya Optimization Algorithm. We navigate through the intricacies of workflow scheduling, introducing the Jaya Optimization Algorithm and contrasting its performance against various nature-inspired algorithms, including Particle Swarm Optimization, Genetic Algorithm, Ant Colony Optimization, Honey Bee, and Cat Swarm Optimization. The evaluation criteria revolve around execution cost and makespan, and benchmark functions such as Montage, CyberShake, Inspiral, and Sipht are employed to gauge the algorithms' effectiveness.

Building upon this foundation, we explore the challenges of workflow scheduling in cloud environments, considering dependencies and conflicting Quality of Service (QoS) objectives. Subsequently, an innovative Jaya-based algorithm is introduced, aiming to minimize both makespan and execution cost. This approach allows users to assign weights based on their priorities, offering a flexible and user-centric solution to the intricate

task scheduling problem. Comparative analyses against other algorithms underscore the superior performance of the Jaya algorithm concerning both cost and time metrics.

3.2 Task Scheduling Using Jaya Algorithm in Cloud

In this section, the Jaya Optimization Algorithm is implemented for workflow scheduling, and a comparative analysis is conducted with four nature-inspired algorithms, namely Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Ant Colony Optimization (ACO), Honey Bee, and Cat Swarm Optimization (CSO). The fitness function is kept consistent across all algorithms, and the evaluations are performed using the CloudSim simulation toolkit. Previous research has extensively explored the application of PSO, GA, ACO, Honey Bee, and CSO using diverse criteria. The comparative results focus on the execution cost and makespan of the algorithms, considering both independent sets of tasks and sets of tasks following a workflow schedule. Benchmark workflows, including Montage, CyberShake, Inspiral, and Sipht, are employed for the workflow scheduling assessments. The findings indicate that Jaya outperforms other algorithms by delivering comparable results in a significantly reduced time frame, showcasing its rapid convergence capabilities[28].

3.2.1 Problem Statement

The optimisation problem at hand is to determine an optimal mapping that minimizes the total execution cost of a workflow schedule, taking into consideration both communication and execution costs. Additionally, it is desirable to compute this optimal mapping quickly. To represent the workflow schedule, we employ a dependency matrix denoted as $M_{i,j} = \{a, b\}$, where a signifies the level of a task in the workflow, and b indicates the number of tasks present in that level. Virtual Machines (VMs) or Personal Computers (PCs) are considered as computing resources, each with its unique capability. The tasks T are to be allocated to these VMs. We have developed a parser to generate a dependency matrix, enabling the evaluation of various costs associated with the workflow schedule.

Fitness Function

A fitness function serves as a criterion optimized to achieve the desired outcome. For the purpose of comparison, the same fitness function is employed across all algorithms, ensuring a standardized basis for evaluation. The proposed algorithm assigns processors to tasks, and the fitness function calculates the execution cost of the mapping. It iteratively minimizes the execution cost (fitness function) by adjusting the mapping. In each subsequent iteration, a mapping with reduced execution cost is generated. The fitness function incorporates parameters such as communication cost, execution cost of each task on each

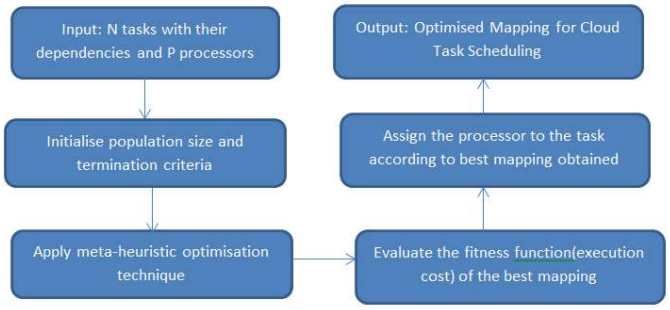

Fig. 3.1 Methodology used in cloud scheduling heuristic algorithm

processor, and the file size of tasks to be communicated. The fitness function, as defined in [60], is utilized. Let M represent a mapping assigned in a specific iteration. P denotes the set of processors, and T is the set of tasks. $C_{exe}(M_p)$ signifies the execution cost of running tasks on the allocated processors according to M. u_{tp} represents the running cost of task t on processor p. $C_{ac}(M_p)$ denotes the access cost of communicating data between processors to which tasks t_1 and t_2 are assigned. $comm_{M(t1),M(t2)}$ represents the communication cost between the processors where tasks are mapped, and $data_{t1t2}$ is the file size to be communicated. $C_{total}(M_p)$ is the sum of access cost and execution cost. The fitness function is defined as the minimization of the maximum cost (Cost(M)) for each mapping along with the time taken to reach that mapping (T_{run}).

$$C_{exe}(M_p) = \sum_{t} u_{tp} \qquad \forall M(t) = p \tag{3.1}$$

$$C_{ac}(M_p) = \sum_{t1 \in T} \sum_{t2 \in T} comm_{M(t1),M(t2)} data_{t1t2} \qquad \forall M(t1) = p \quad M(t2) \neq p \tag{3.2}$$

$$C_{total}(M_p) = C_{exe}(M_p) + C_{ac}(M_p) \tag{3.3}$$

$$FitnessFunction(f) : Minimize(Cost(M)) + Minimise(T_{run}) \tag{3.4}$$

3.2.2 Proposed Methodology

The model shown in Fig. 3.1 shows the approach in which the meta-heuristic optimization techniques are implemented. This section introduces a scheduling algorithm designed to achieve an optimized mapping for a workflow schedule. The process consists of two main parts: a) the scheduling heuristic outlined in Algorithm 1 and b) Jaya for task scheduling mapping illustrated in Algorithm 2.

Cloud Scheduling Heuristic

The execution cost of running tasks on different processors is calculated using the Task to Processor (TP) matrix. The overall execution cost is determined by summing the execution costs of individual tasks based on the obtained mapping. Additionally, communication cost is computed for data transfer between processors using the Processor to Processor (PP) matrix. The assumed knowledge of input and output data sizes aids in this calculation. The dependency matrix is generated according to a predefined parser, which also assigns costs to TP and PP matrices. The running time of the algorithm is considered to accurately evaluate the makespan. These processes are integral to the accurate calculation of costs and the generation of the dependency matrix, ensuring the comprehensive evaluation of the scheduling algorithm.

Algorithm 1 Cloud Scheduling Heuristic

1: Calculate the execution cost of running each task on each processor (TP Matrix).
2: Calculate the communication cost between each processor (PP Matrix).
3: **while** tasks are waiting to be allotted **do**
4: Apply Algorithm 2 on the set of tasks that are waiting to be allotted.
5: Assign task t_i to processor p_j according to the mapping obtained in step 3.
6: **if** any unscheduled task is left **then**
7: Go to step 3.

Jaya Algorithm

Jaya is a parameterless algorithm designed to iteratively move towards the best solution and away from the worst solution [77]. The term "Jaya" originates from Sanskrit, meaning "Victory," as the algorithm consistently strives for victory in all circumstances. The process involves utilizing n candidate solutions, identifying the best and worst among them, and adjusting the solutions based on the Fitness Function. The Fitness Function evaluates the value of a candidate, which is then modified according to Equation 3.5. Jaya Algorithm stands out for its lack of algorithm-specific parameters. Consequently, the worst candidate continuously improves with each iteration. Given the imperative need for efficient task scheduling in the Cloud, the rapid convergence of the Jaya Algorithm makes it a suitable candidate for application in Cloud scheduling scenarios.

Analysis of Jaya Algorithm: Suppose the population size is P, the number of variables in each candidate solution is V which is Number of Cloudlets in our algorithm i.e. constant. So, Complexity of Jaya is $O(P \times V)$ i.e. $O(P)$.

$$X'_{ij} = X_{ij} + |rand[0,1](best_j - X_{ij}) - rand[0,1](worst_j - X_{ij})| \tag{3.5}$$

Algorithm 2 Jaya

Input: $max\_iterations, P, V$
Output: X_{ij}

procedure JAYA
  for $i = 1; i \leq P; i++$ **do**
    for $j = 1; i \leq V; i++$ **do**
      Randomly initialise X_{ij}
  Calculate $best$ and $worst$ from X_{ij}.
  while Termination not reached **do**
    Calculate $X'_{ij} = X_{ij} + |rand[0,1](best_j - X_{ij}) - rand[0,1](worst_j - X_{ij})$
    Calculate f for X_{ij}.
    if X'_{ij} better than X_{ij} **then**
      Update X_{ij}.
    Update $best$ and $worst$.
  Return X_{ij}

where,
X_{ij} the value of i^{th} candidate in j^{th} iteration
$best_j$ the best candidate
$worst_j$ the worst candidate value in j^{th} iteration.

3.2.3 Implementation and Results

This section presents the results and the experimental setup for Jaya Algorithm execution and implementation. The algorithms taken for comparison are PSO[60], GA[89], CSO[11], ACO[70] and Honey Bee. The criteria for comparison has been kept same i.e. Fitness Function, so all these algorithms are modified accordingly.

Experimental Setup

The experimentation of the proposed algorithm is performed in a personal computer DOS based system with 8GB RAM. All the algorithms are implemented in CloudSim for both independent set of tasks and workflow scheduling also. For workflow scheduling, a dependency matrix is introduced. Then the input will also consist of dependency matrix which shows the inter-dependency of the tasks. The algorithm is then executed level wise for each set of ready tasks at each level. The performance is evaluated on the basis of convergence, cost and makespan. A new parser has been introduced to parse different workflows according to number of tasks.

Results

This section represents results which are used for comparison. The various parameters used for comparison are iteration number at which algorithm converges, total time taken to reach that mapping and execution cost obtained for most optimised mapping.

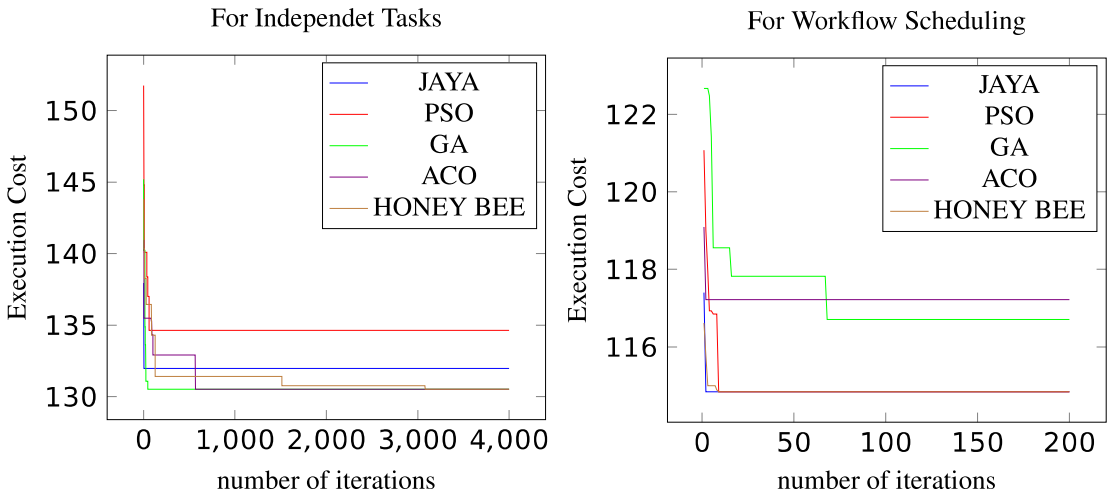

Fig. 3.2 Convergence of algorithm

Makespan

Table 3.1 shows the running time of all the algorithms for 10,000 iterations. From the table, we can see that in the case of workflow scheduling, ACO, Honey Bee, take more time to complete 10,000 iterations. Jaya takes less time as compared to ACO and Honey Bee to complete 10,000 iterations but more time compared to PSO, GA and CSO. This makespan is for 10,000 iterations but Jaya converges very quickly as compared to other algorithms. Also, the makespan is taken for benchmark workflows for 25, 30, 50, 60 and 100 tasks for montage, CyberShake, Inspiral and Sipht. So, the optimised cost obtained is reached faster in our algorithm than other algorithms compared. Figure 3.2 shows the convergence of all the algorithms for independent and workflow scheduling for a set of 10 tasks. So, the time to reach the optimised mapping is less.

Iteration number at which Convergence occurs

All the algorithms are run for 10,000 iterations in cloud. The iteration number at which the algorithm converges, or we can say which gives the best-optimized solution is plotted on Fig 3.3. From the bar graph we can see that for independent set of tasks Honey Bee takes 3081 iterations to produce an optimized mapping which is far way more than Jaya which only takes 3 iterations to produce an optimized mapping. Similar results can be seen for

Table 3.1 Running time of the algorithm

MAKESPAN	PSO	GA	ACO	HONEY	CSO	Jaya
10 Independent	104	206	2724	2259	150	**418**
10 Workflow	302	1790	3397	6395	291	**1321**
Montage 25	558	2040	40032	34262	700	**5698**
Montage 50	1198	3939	296452	142011	1201	**21378**
Montage 100	1605	4676	232887	377775	1861	**51772**
CyberShake 30	473	1717	67510	38952	538	**5887**
CyberShake 50	742	2554	286381	107059	775	**14852**
CyberShake 100	1300	4700	1922583	383653	1441	**51772**
Inspiral 30	645	2479	66607	53986	1.12	**7904**
Inspiral 50	1119	2555	265690	107059	1104	**20085**
Inspiral 100	1929	7453	2132288	611889	2059	**84091**
Sipht 30	401	1428	67510	32538	479	**4694**
Sipht 60	719	2283	509958	132846	811	**17112**
Sipht 100	1325	4261	2499593	385628	1826	**49797**

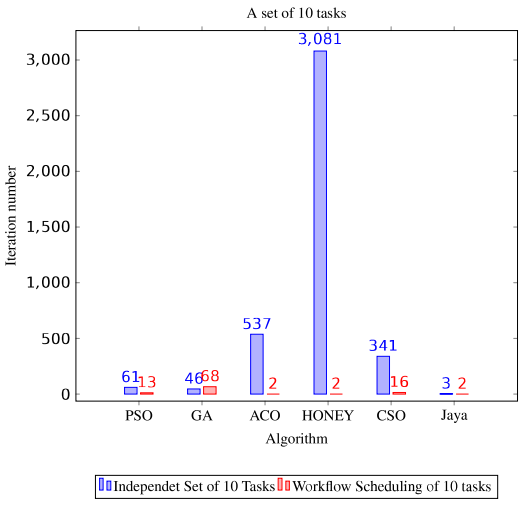

Fig. 3.3 Iteration number at which algorithm converges for a set of 10 tasks

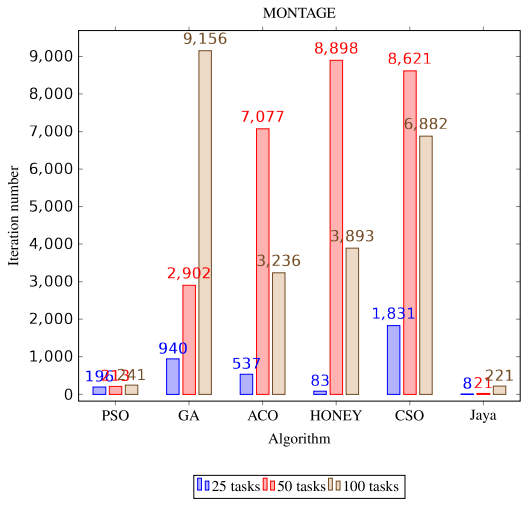

Fig. 3.4 Iteration number at which algorithm converges for Montage benchmark workflow

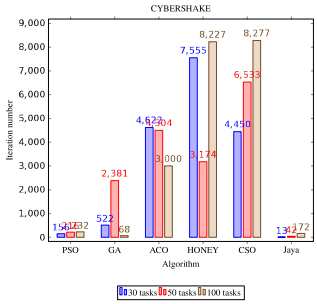

Fig. 3.5 Iteration number at which algorithm converges for CyberShake benchmark workflow

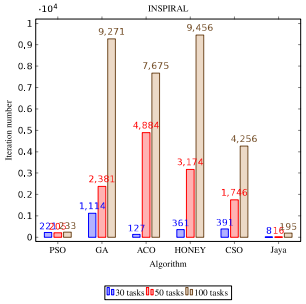

Fig. 3.6 Iteration number at which algorithm converges for LIGO benchmark workflow

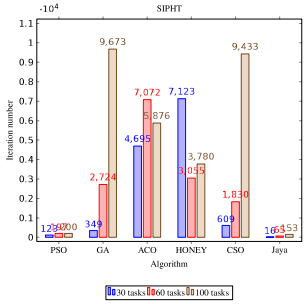

Fig. 3.7 Iteration number at which algorithm converges for SIPHT benchmark workflow

various benchmark workflows i.e. Jaya takes less iterations to converge as compared to others as shown in Fig. 3.4, 3.5, 3.6 and 3.7.

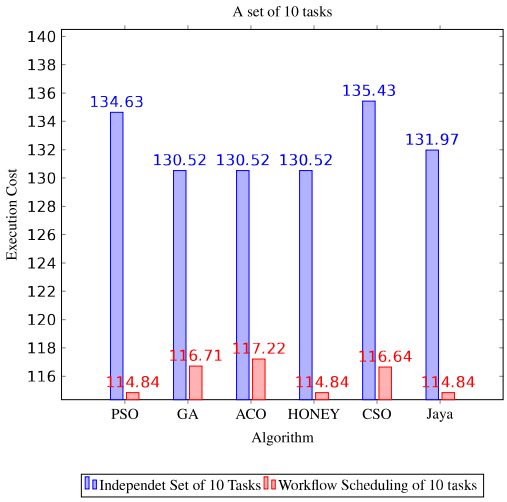

Fig. 3.8 Execution cost for a set of 10 tasks(hypothetical)

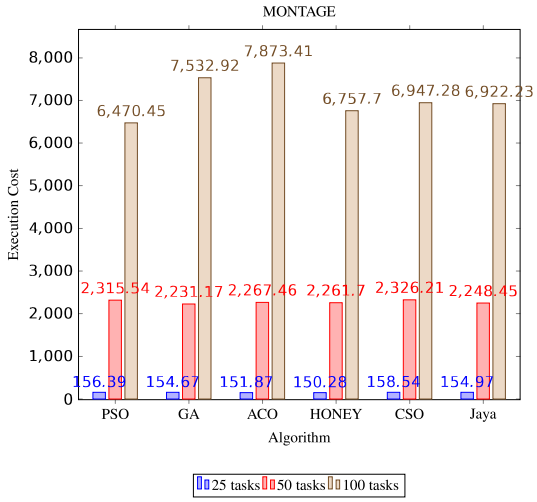

Fig. 3.9 Execution cost for Montage benchmark workflow

Cost

The execution cost, representing the running cost of tasks on processors and the communication cost between processors, was evaluated using the fitness function. The graph in Fig. 3.8 illustrates the minimum execution cost achieved by each algorithm in the cloud. The Jaya algorithm consistently produced lower execution costs compared to other algorithms.

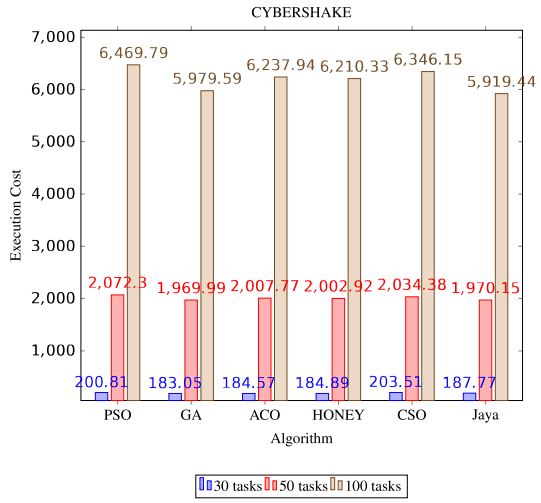

Fig. 3.10 Execution cost for Cyber-Shake benchmark workflow

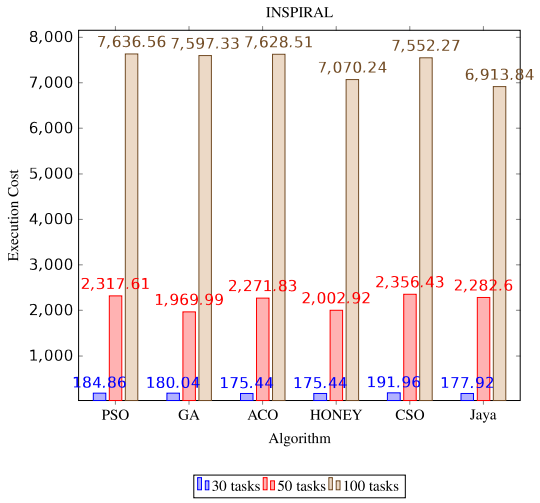

Fig. 3.11 Execution cost for LIGO benchmark workflow

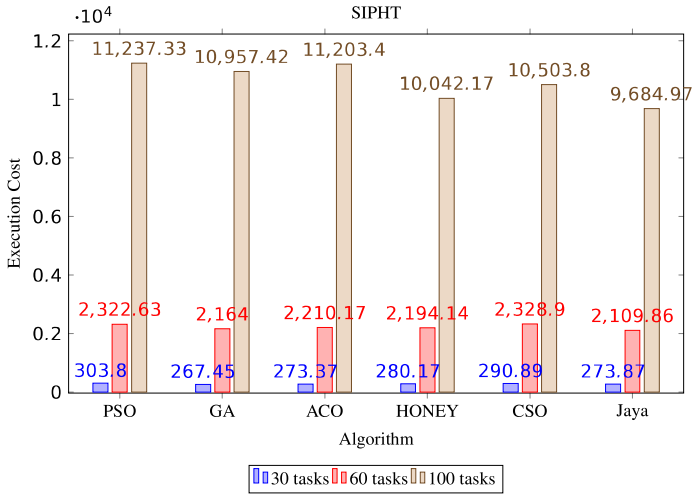

Fig. 3.12 Execution cost for SIPHT benchmark workflow

Results for benchmark workflows with varying numbers of tasks (25, 30, 50, 60, and 100) are shown in Fig. 3.9, 3.10, 3.11, and 3.12.

The convergence speed of the proposed approach is depicted in Fig. 3.2, showing that the Jaya algorithm converges faster than other algorithms. Additionally, Fig. 3.8, 3.9, 3.10, 3.11, and 3.12 confirm that the proposed technique ensures minimal execution costs for all benchmark problems. The number of iterations required to reach the optimal mapping is minimal, as demonstrated by Fig. 3.3, 3.4, 3.5, 3.6, and 3.7.

The efficiency of the proposed approach lies in its parameterless nature, overcoming the need for adjusting parameters present in other algorithms. Leveraging the best and worst solutions for optimization, the Jaya algorithm exhibits faster convergence, establishing it as a promising and efficient approach for job scheduling in cloud computing environments.

3.3 User Defined Weight Based Budget and Deadline Constrained Workflow Scheduling in Cloud

This section extends the workflow scheduling algorithm based on Jaya, aiming to minimize both cost and time according to user-defined weights. The algorithm's performance is evaluated through comparisons with Min-Min, Max-Min, Round Robin, HEFT (Heterogeneous Earliest Finish Time First), FCFS (First Come First Serve), MCT (Minimum Completion Time), and DHEFT (Dynamic Heterogeneous Earliest Finish Time First) using the WorkflowSim simulation tool. The assessments are conducted considering both cost and time metrics. The chapter concludes by illustrating a trade-off between the specified objectives. The results demonstrate that Jaya outperforms other algorithms consistently across multiple iterations when considering both cost and time.

3.3.1 Problem Statement

Our challenge revolves around the concept of Efficient Resource Provisioning, where factors such as communication cost, resource capacity, and available bandwidth significantly impact the overall execution time and cost. The key objective is to optimize the schedule while adhering to Service Level Agreement (SLA) constraints. In our modeling approach, tasks are represented as cloudlets, jobs, or tasks within the WorkflowSim framework, while resources are depicted as Virtual Machine units. Our primary focus is to assign the most suitable Virtual Machine for each cloudlet, optimizing execution time and cost. Additionally, considerations like load balancing and dynamic workload must be factored in to achieve a solution that is close to optimal. The characteristics of a typical cloud, such as the pay-as-you-go model and an hourly-pricing structure, can also be incorporated

to mimic real-time cloud workflow scheduling. The overarching goal is to develop a multi-objective workflow scheduling optimization model.

3.3.2 Proposed Methodology

Base Algorithm

The Jaya algorithm, introduced by R. V. Rao in 2016[62], has garnered attention due to its ability to optimize functions in a few iterations without the need for parameter tuning. Leveraging this characteristic, a novel approach is proposed to apply the Jaya algorithm to Weight-Based Workflow Scheduling in the Cloud. Notably, Jaya has demonstrated superior performance compared to GA, PSO, and other algorithms in various studies. For instance, the Self-Adaptive Multi-Population-based Jaya (SAMP-Jaya) algorithm, designed to solve constrained and unconstrained numerical and engineering optimization problems, outperformed GA, PSO, Cuckoo Search Algorithm, and others[63].

In the context of Weight-Based Workflow Scheduling, each candidate solution is represented as a 1N matrix, where N is the number of cloudlets in a specific workflow level. With M candidate solutions, a matrix of size MN is generated, and computations are performed. The mapping of each cloudlet to a particular VM is updated iteratively using Equation 3.6.

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}(X_{j,best,i} - |X_{j,k,i}|) - r_{2,j,i}(X_{j,worst,i} - |X_{j,k,i}|) \tag{3.6}$$

where, $X'_{j,k,i}$ updated value of the j_{th} variable for the k_{th} candidate during the i_{th} iteration. $r_{1,j,i}$ and $r_{2,j,i}$ are two random numbers for j_{th} variable in i_{th} iteration in range [0,1]. $X_{j,best,i}$ and $X_{j,worst,i}$ are values of j_{th} variable for best and worst candidate solution during i_{th} iteration.
Each candidate solution is associated with a fitness function depending on the objective to be optimized. It is computed at each iteration for every candidate and is a measure of the candidate to survive or not. The Algorithm 2 is used here also.

Cost Optimal Jaya Algorithm

In this algorithm, the objective is to schedule all cloudlets to virtual machines in a manner that minimizes the total execution cost within the user-specified budget. The fitness function, which gauges the quality of a solution, remains the same as discussed in Section 3.3.2, with the key difference being the incorporation of cost as a parameter in the evaluation.

The equation for calculating the fitness function value for a candidate T is provided below:

$$F_{cost,T} = \left(\sum_{c \epsilon T} (unit\_execution\_cost(j) * estimated\_execution\_time(c) + cost\_per\_bandwidth * amount\_of\_data\_transfer(c)) \right) \leq BUDGET \tag{3.7}$$

where c refers to a cloudlet that is assigned to a VM j.

Time Optimal Jaya Algorithm

In this algorithm, the goal is to schedule all cloudlets to virtual machines in a manner that minimizes the total execution time. Unlike the previous version where cost was the parameter for evaluating the fitness function, in this case, time is the key parameter. The equation for calculating the fitness function value for a candidate T is as follows:

$$F_{time,T} = \left(\sum_{c \epsilon T} \frac{cloudlet\_length(c)}{mips(j)} \right) \leq DEADLINE \tag{3.8}$$

where c refers to a cloudlet that is assigned to a VM j.

Cost and Deadline Based Jaya Algorithm

The execution time is primarily influenced by the computational power of resources and communication bandwidth. Greater resource computational power and higher bandwidth availability contribute to a reduction in execution time. Virtual Machines (VMs) can be characterized by parameters such as MIPS rating, number of processing elements, unit cost per execution time, and more. Optimal scheduling on a more powerful VM leads to reduced execution time. However, the associated costs, including storage, bandwidth, and unit execution cost, tend to be higher for resources with greater computational power.

This creates a trade-off between cost and execution time, necessitating a strategy to balance and achieve both objectives concurrently. To address this conflict and optimize for both cost and execution time, we adopt a basic yet effective technique of weighted sums. By assigning weights to the cost and time fitness functions based on the priority of each goal, we aim to strike a balance between the two conflicting objectives.

We calculate fitness function for this case as follows :

$$F_{tradeoff} = Weight_d * F_{Time} + Weight_c * F_{Cost} \tag{3.9}$$

where,

F_{Time} and F_{Cost} has been defined in Equations 3.7 and 3.8.

$Weight_d$ is the deadline weight and $Weight_c$ is the cost weight.

Similar steps are followed as in case of cost and deadline based model with change in fitness function.

Here, $Weight_d$ and $Weight_c$ are user defined and they can set the priority according to their needs. So, if they need it to be Cost Effective, they can make $Weight_c = 100$ and $Weight_d = 0$ and vice-versa for Time Effective. The weights can be anything which the user like, making it **Weight Based Workflow Scheduling Algorithm**.

3.3.3 Implementation and Results

This section presents us with the experimental setup and simulation tool used, alongwith, the results obtained on comparing it with various other algorithms.

Experimental Setup

The performance evaluation involves various algorithms executed on a personal computer with Windows 10, 8GB RAM, and a 64-bit processor. The implementation is done in Java using Eclipse as an Integrated Development Environment (IDE), coupled with the simulation tool WorkflowSim. Different configurations of Virtual Machines (VMs) are utilized, ranging from 5 to 20, depending on the number of cloudlets (tasks) in the scientific workflows. The choice of 5, 10, and 20 VMs is made to accommodate varying cloudlet sizes, with a maximum of 1000 tasks in scientific workflows. The configuration of VMs introduces heterogeneity by randomly generating MIPS values. Initial Candidate Solutions are also randomly generated, and the best and worst solutions are selected based on the fitness function's criteria. The simulation tool employed, WorkflowSim, is an extension of CloudSim, a Java-based simulation tool that emulates a cloud environment. CloudSim offers features such as DataCenter, VMs, Tasks, Scheduler, Broker, enabling the creation and testing of allocation policies in a simulated cloud environment.

WorkflowSim extends CloudSim by incorporating components like Workflow Mapper, Clustering Engine, Workflow Engine, and Workflow Scheduler. This extension enables the implementation and evaluation of workflow scheduling in a cloud environment. The simulation aims to provide insights into the performance of various scheduling algorithms in terms of execution time, cost, and other relevant metrics.

Results

This section presents the results which involves total cost of the schedule, total time taken by the workflow for execution and a trade-off between the two.

Cost Based Jaya for Workflow Scheduling

Jaya algorithm has been used to schedule the workflows with cost as the single objective for fitness function.It is observed that it outperforms basic scheduling techniques which are FCFS, MINMIN, MAXMIN, RoundRobin, MCT, HEFT and DHEFT algorithm. The results have been shown in Fig. 3.13, 3.14, 3.15 and 3.16. The Table 3.2 shows the value

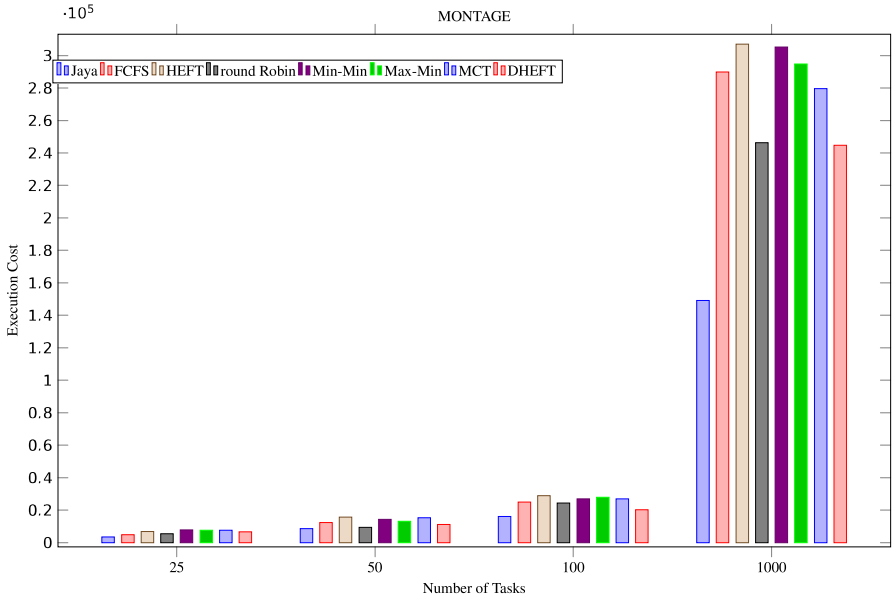

Fig. 3.13 Execution cost for various algorithms for Montage workflow

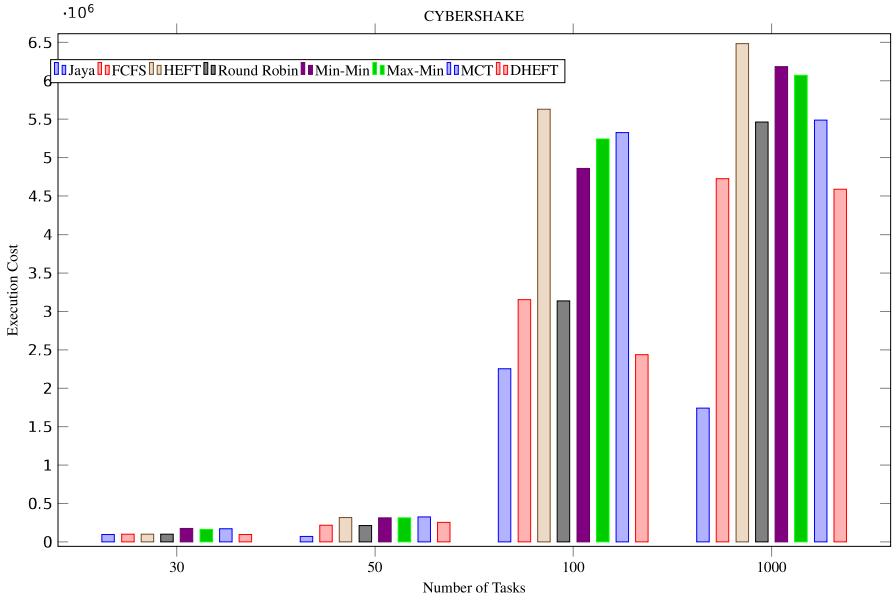

Fig. 3.14 Execution cost for various algorithms for CyberShake workflow

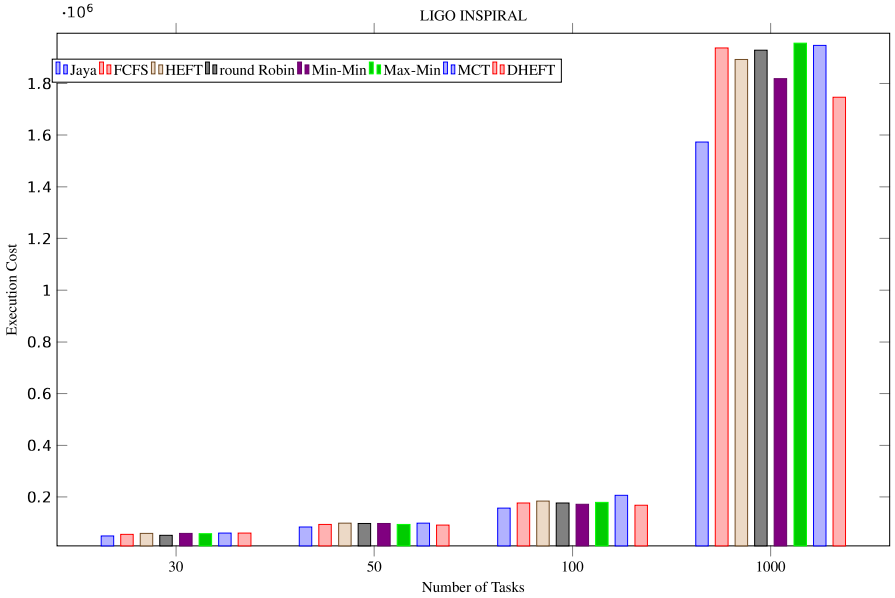

Fig. 3.15 Execution cost for various algorithms for LIGO workflow

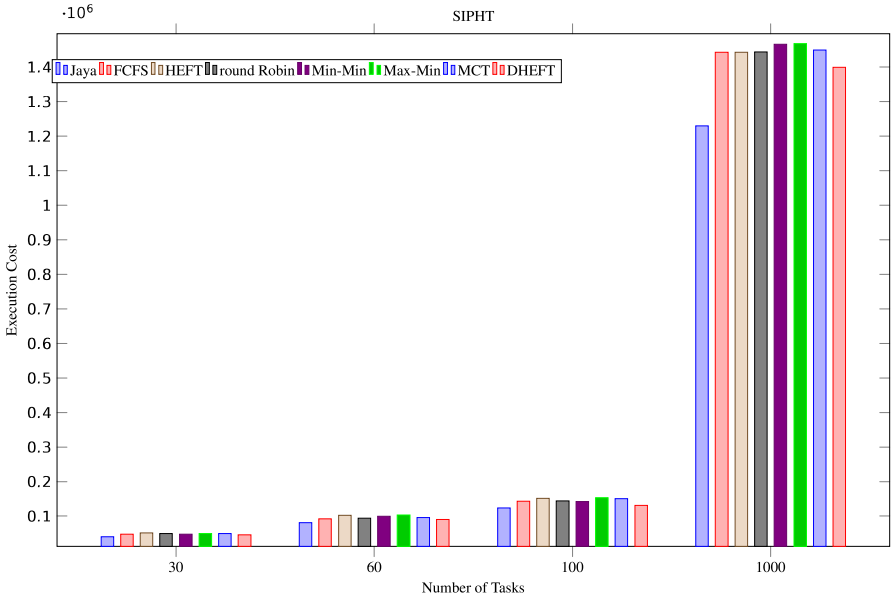

Fig. 3.16 Execution cost for SIPHT workflow

of the cost when executed for 100 tasks in WorkflowSim and the average of the same has been taken to avoid any biasing caused by randomisation of initialisation.

TEST NUMBER	JAYA	FCFS	HEFT	RR	MIN-MIN	MAX-MIN	MCT	DHEFT
1	**19274**	23959	30170	20640	26466	27178	25282	20456
2	**14634**	26501	27618	25025	26397	26415	27235	21326
3	**18280**	27268	31866	26420	28375	28791	25023	19854
4	**16650**	21707	24594	23906	28395	28132	26640	21478
5	**13528**	24196	26405	27736	25861	27598	28672	18250
6	**17241**	23265	28177	21164	26240	28966	25931	17458
7	**13730**	23963	30388	27109	25430	27287	26582	20478
8	**17646**	28508	28668	29337	25792	29678	28383	25474
9	**14298**	24033	29821	25019	28604	29695	25566	20152
10	**15465**	25886	31170	17071	28044	25706	30730	16452
Average	**16074.6**	24928.6	28882.8	24342.7	26960.4	27944.6	27004.4	20137.8

Table 3.2 Comparison of different approaches with Jaya for cost optimization for Montage workflow with 100 tasks

Deadline Based Jaya for Workflow Scheduling

Jaya algorithm has been used to schedule the workflows with time as the single objective of fitness function. It is observed that it outperforms basic scheduling techniques like FCFS, MINMIN, MAXMIN, RoundRobin, HEFT and DHEFT algorithm. The results are plotted in Fig. 3.17, 3.18, 3.19 and 3.20. The Table 3.3 shows the value of the makespan when executed for 100 tasks in WorkflowSim.

TEST NUMBER	JAYA	FCFS	HEFT	RR	MIN-MIN	MAX-MIN	MCT	DHEFT
1	**25.89**	40.8	31.31	33.63	29.51	30.29	27.13	30.81
2	**26**	43.81	31.46	43.43	27.45	27.46	29.28	32.45
3	**32.77**	40.9	30.01	41.09	27.26	28.36	29.50	35.7
4	**24.83**	37.07	31.41	40.1	30.44	29.81	29.40	34.87
5	**24.77**	32.98	30.24	36.32	28.01	29.28	29.76	42.58
6	**30.31**	29.45	28.58	30.3	29.82	29.27	28.09	26.5
7	**29.32**	40.5	31.68	28.32	29.73	27.56	30.64	27.85
8	**24.18**	34.79	33.40	35.34	28.78	28.68	26.05	24.58
9	**27.76**	29.59	29.08	40.68	28.69	28.84	27.52	26.75
10	**25.25**	37.38	30.09	33.27	29.75	29.94	32.56	28.85
Average	**27.108**	36.727	30.726	36.248	28.944	29.028	33.943	31.94

Table 3.3 Comparison of different approaches with Jaya for makespan(sec) for Montage workflow of 100 tasks

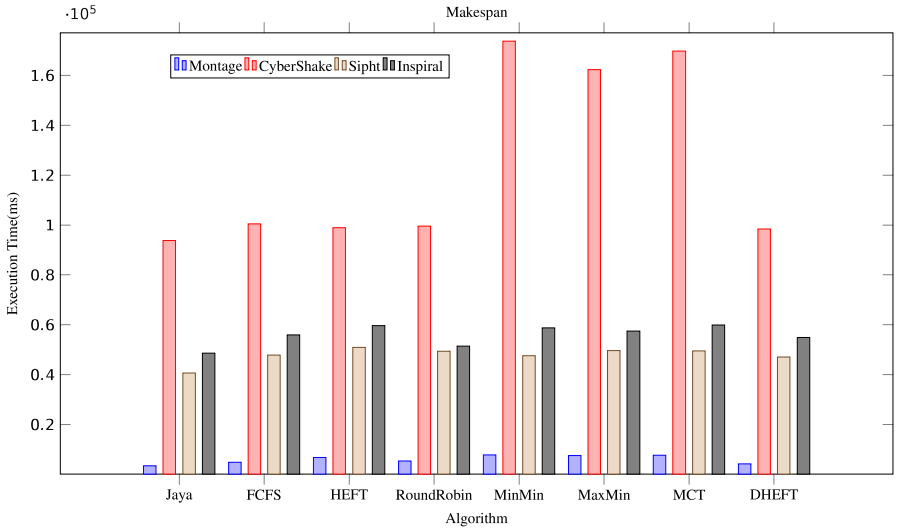

Fig. 3.17 Makespan for various workflow with 25/30 tasks

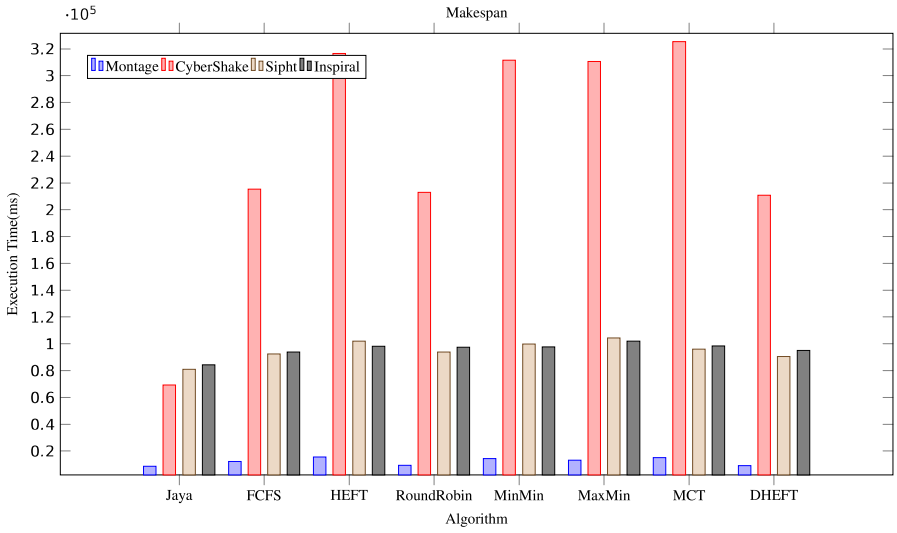

Fig. 3.18 Makespan for various workflows with 50/60 tasks

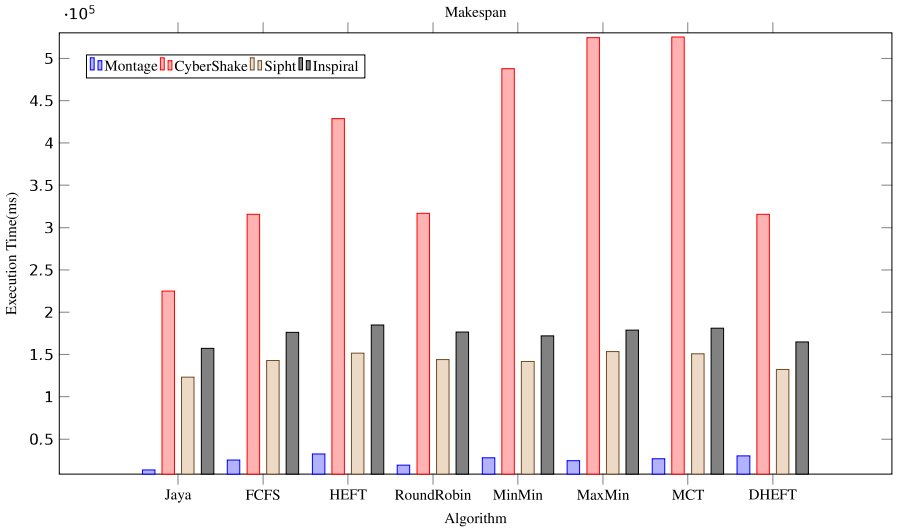

Fig. 3.19 Makespan for various workflows with 100 tasks

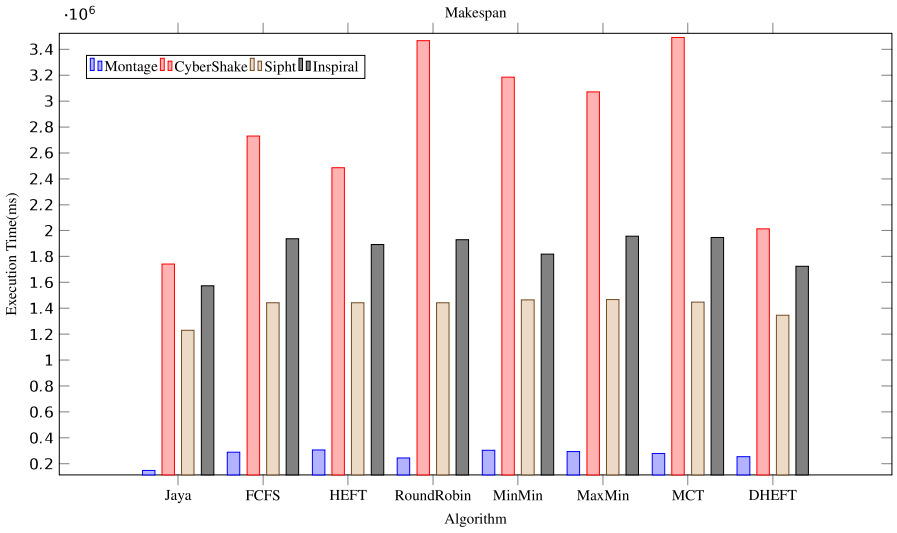

Fig. 3.20 Makespan for various workflow with 1000 tasks

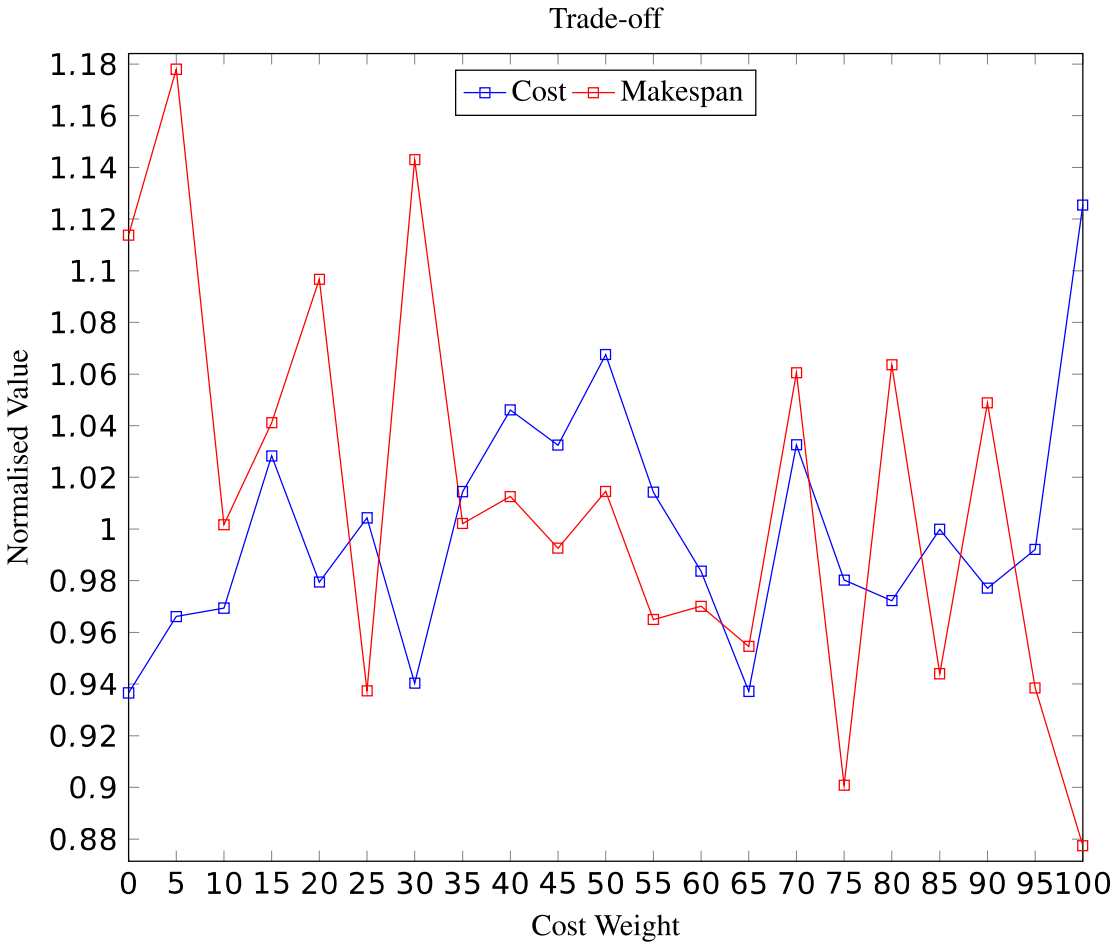

Fig. 3.21 Trade off between execution cost and time

COST-WEIGHT	Averaged Cost	Averaged Makespan	TIME-WEIGHT
0	16950.5	27.06662	100
5	14942	43.99196	95
10	14716.1	49.16281	90
15	15059.7	39.56244	85
20	14643.3	56.88574	80
25	14763.5	42.22553	75
30	15552	49.71274	70
35	14115.2	44.74581	65
40	14815.1	45.47305	60
45	15276.4	45.23362	55
50	16078.6	47.5567	50
55	15550.2	46.52782	45
60	15755.4	47.46463	40
65	15279.2	39.94374	35
70	14162.1	53.57653	30
75	15126.2	43.94003	25
80	14751.6	58.10891	20
85	15487.5	48.80431	15
90	14600.3	46.95364	10
95	14551.2	55.21875	5
100	14105.8	52.20954	0

Table 3.4 Variations in average cost and average makespan due to variations in weights

Deadline and Cost Based Jaya for Workflow Scheduling

The Jaya algorithm is applied to schedule workflows considering both deadline and cost as multiple objectives of fitness function, with different weights assigned to prioritize these objectives. A tradeoff analysis is conducted, anticipating that higher priorities would result in more significant achievements for the respective objectives. However, due to averaging values over random iterations, the results may not be explicitly clear, but the tradeoff is observable. Fig. 3.21 illustrates the normalized values of these results, presented as averages. The weights assigned to deadline and cost are represented by the x and $100-x$ values, respectively, showcasing the tradeoff between them in a specific iteration. As depicted in Table 3.4, the values exhibit minimal variation even with changes in the assigned weights. This demonstrates the robust performance of the Jaya algorithm under different weight configurations, consistently outperforming other implemented algorithms. The graph may exhibit variations due to the randomness introduced by Jaya in updating the solution, leading to occasional fluctuations in the output compared to the previous weight configuration.

3.4 Summary

In this chapter, we delved into the intricate domain of task scheduling in cloud environments, with a specific focus on the Jaya Optimization Algorithm. In Section 3.2, the algorithm was introduced and compared with various nature-inspired algorithms such as Particle Swarm Optimization, Genetic Algorithm, Ant Colony Optimization, Honey Bee, and Cat Swarm Optimization. The evaluation criteria centered around execution cost and makespan, employing benchmark functions like Montage, CyberShake, Inspiral, and Sipht. Building upon this foundation, Section 3.3 extended the exploration by addressing the challenges of workflow scheduling in the cloud according to user defined weights. The focus shifted to dependencies and conflicting Quality of Service (QoS) objectives. An improved Jaya-based algorithm was proposed, aiming to minimize both makespan and execution cost. This approach allowed users to assign weights based on their priorities, offering a flexible and user-centric solution to the intricate task scheduling problem. Comparative analyses against other algorithms underscored the superior performance of the Jaya algorithm concerning both cost and time metrics. Together, this chapter provided a comprehensive exploration of the Jaya Optimization Algorithm's effectiveness in addressing the complexities of task scheduling in cloud environments. The algorithm demonstrated its prowess in achieving optimal solutions swiftly, without the need for intricate parameter tuning. The comparisons with other algorithms highlighted the Jaya algorithm's superior

performance in optimizing both cost and time dimensions, contributing valuable insights to the field of cloud computing.