Genetic Algorithm Based Bi-Objective Task Scheduling in Hybrid Cloud Platform

Leena V. A., Ajeena Beegom A. S., and Rajasree M. S., *Member, IACSIT*

***Abstract*—Hybrid cloud is a type of the general cloud computing platform, that is composed of both public and private cloud. Scheduling plays a key role in the efficient use of hybrid cloud resources. In this paper, focus is on a scheduling algorithm for hybrid cloud that tries to optimize both execution time and cost. Execution time and cost are conflicting objectives, i.e. when one is made better, the other becomes worse off. Multiobjective evolutionary algorithm is used to find the optimal schedule. The widely used scheduling implementations seen in hybrid cloud try to optimize either execution time or cost, but not both simultaneously. The proposed algorithm is compared with the more widely used scheduling optimization techniques and seen to have much better performance.**

***Index Terms*—Evolutionary optimization algorithms, hybrid cloud, pareto-optimality, scheduling.**

I. Introduction

Cloud computing is a paradigm that refers to buying computer resources which could be infrastructure, software, or services; and paying based on usage levels. Cloud service providers (CSP) are equipped with large data centers. Users request CSP for resources, and the CSP provides them from their pool of resources. The user has to pay the CSP based on the usage level. This is profitable for both users and CSPs. Users save the cost of design, setup and maintenance of data centers, while CSPs give access to resources to large number of users, and thereby make profit. Users can also scale up (i.e. increase resource usage) very easily, based on increase in requirements. They need not equip themselves with resource levels needed at peak times [1].

Virtualization is the back bone of cloud. For each user request, a virtual machine is instantiated with the requested resources in the server. The user application will run on this virtual machine. Virtualization provides isolation and integrity for applications of different users running on a single cloud [1].

Hybrid cloud is a type of cloud computing infrastructure that is composed of both public and private cloud. When resources in private cloud are not sufficient, resources from public cloud are taken on lease, to complete the execution of an application. Hence in hybrid cloud, the benefits of both public cloud and private cloud can be utilized by an application. Hybrid cloud has the advantages of optimal resource provisioning, scalability and pay-as-you-use benefits.

Scheduling is the process of mapping the jobs submitted, to the resources available. A job is usually split into multiple tasks. Hence scheduling can be redefined as mapping of tasks to a selected group of resources. Scheduling plays an important role in a cloud. Scheduling algorithms can be designed to meet different application requirements such as minimizing makespan, minimizing cost, fairness, reducing energy consumption, minimizing response time and making the system deadline aware. Based on the application requirement, algorithms optimize any one of the above requirements.

The different task scheduling algorithms, commonly used in cloud to optimize any one of the aforementioned requirements are opportunistic load balancing (OLB) [2], minimum execution time (MET) [3], minimum completion time (MCT), Min-min [4], Max-min heuristic [5], [6], and heterogeneous earliest finish time (HFET). Even metaheuristic search techniques [7]-[10] are widely employed for scheduling.

But hybrid cloud is a system with a different set of needs and requirements. In a hybrid cloud both the requirements, cost and execution time are of paramount importance. For applications in a hybrid cloud, cost is an issue; as usage of public cloud incurs cost. But also user would want to get the result as fast as possible. Hence in such cases, both cost and execution time has to be optimized. The aforementioned algorithms for scheduling in cloud optimize only a single requirement. Hence it is not apt for usage in hybrid cloud.

In this work, the problem of developing a hybrid cloud scheduler that maximizes performance (or minimizes makespan) and minimizes cost is addressed. This is done by determining whether tasks have to be scheduled to either private cloud (internal cloud) or to the public cloud (external cloud). Cost can be minimized by sending more jobs to the private cloud. Execution time can be minimized by sending more jobs to public cloud. Hence both cost and execution time are conflicting objectives, as when one is made better; other tends to be worse off. The idea is to choose the best trade-off point between application completion time and cost. Heuristic algorithms can be used to achieve this objective.

The rest of the paper is organized as follows. Section II describes the background of this work. Section III gives the related work and Section IV gives the problem formulation. Section V describes the proposed system. Sections VI and VII give the implementation and result analysis respectively.

Manuscript received August15, 2014; revised November 12, 2014.

Leena V. A. and Ajeena Beegom A. S. are with the College of Engineering, Trivandrum, Kerala, India (e-mail: leena.jaleel@gmail.com, ajeena@cet.ac.in).

Rajasree M. S. is with IIITM-K, Kerala, India (e-mail: rajasree.ms@iiitmk.ac.in).

DOI: 10.7763/IJCTE.2016.V8.1012

II. Background

BoT (Bag of Tasks) applications are composed of a large number of independent tasks. These tasks can be parallely executed in independent machines. The massive parallelism of BoT applications frequently overwhelms the capacity of private clouds. This capacity limitation can be overcome with the use of resources from the public cloud. The problems associated with hybrid cloud are twofold. The first problem is concerned with the number of objectives to be optimized and the second one is related to the finding of a feasible schedule.

In hybrid cloud, tasks have to be scheduled to either external cloud (EC) or internal cloud(IC), based on the scheduling objective being optimized. Most of the research work in hybrid cloud optimizes a single scheduling objective, typically makespan. Some studies consider more than one criterion like cost, execution time or QoS while developing scheduling techniques. Although multiple criteria have been considered, their aim is to optimize a single objective while all other criteria have been made constraints. For example when the objective is to minimize execution time and cost, it will be changed to minimizing cost while meeting application deadline or minimizing execution time while meeting user's budget. This is done since, the complexity of hybrid cloud scheduling problem increases as the number of objectives to be optimized increases i.e. why mostly when there are two objectives to be optimized, one is converted into a constraint.

Research into simultaneously optimizing objectives, that may be conflicting with each other e.g., cost and execution time, has not been done till date. But, this is quite an important issue as a tradeoff between performance and cost can be attained and used. So this work aims in providing a feasible solution that can be used for optimizing such objectives simultaneously in hybrid cloud.

For a given set of tasks, the problem of finding a feasible schedule is in fact, a search problem. The structure of the search space is basically a tree. The root of the tree is the empty schedule. An intermediate vertex is a partial schedule and a leaf is a complete schedule. All leaves will correspond to feasible schedules. The goal of the scheduling algorithm is to search for a leaf that meets our optimization constraints [11].

When the above scheduling scenario is considered with respect to a cloud computing environment, the number of resources and tasks in consideration is very large. An optimal algorithm, in the worst case, may make an exhaustive search that is computationally intractable. In order to make the algorithm computationally tractable even in the worst case, a heuristic approach is better.

III. Related Work

Hybrid cloud is a recent innovation of the cloud computing infrastructure. So specific problems of hybrid cloud have not been dealt much by the research community. One of the first papers on hybrid cloud is by Mattess *et al*. [12]. It gives details about Aneka, which is a middleware platform for deploying and managing the execution of applications on different clouds. The scheduling policies rely on a dynamic pool of external resources hired from commercial providers in order to meet peak demand requirements. To save hiring costs, the hired resources are released when they are no longer required.

Calheiros *et al*. [13] presents architecture for the dynamic provisioning and scheduling of cloud resources, considering the characteristics of the whole organization workload. They also propose a novel approach for billing users for the utilization of public cloud resources. The paper describes a system with an external queue, deadline queue and a regular queue. The jobs are chosen from each queue based on the available resources.

The above two papers describe the basic hybrid cloud architecture. The specific scheduling methodologies with respect to hybrid cloud came in later papers. Kailasam *et al*. [14] has proposed a scheduling methodology for scenarios, where ordered throughput is needed. In their work, the scheduler will determine the optimal number of resources that must be provisioned on the external cloud. It will also decide when and where and which jobs have to be burst to maximize ordered throughput. Their algorithm schedules computation intensive jobs to external cloud, as resources are plentiful there. This can only be used in specialized situations, where order of the output has to be preserved.

Zuo *et al.* [15] has proposed a method to solve deadline con- strained task scheduling (DCTS) problem, with the objective of maximizing the profit. Particle Swarm Optimization (PSO) based scheduling approach is proposed to solve this problem. However, standard PSO easily traps into local optima. To overcome the disadvantage of standard PSO, a self-adaptive learning PSO based scheduling approach is proposed here.

The above papers describe the scheduling of BoT(Bag of Tasks) applications. Workflow applications are different from BoT in their characteristics. In a workflow, there is dependency between different tasks. Therefore a workflow scheduler results in a system with different specifications.

Senna *et al.* [16] proposed a strategy to schedule workflows in hybrid cloud, in order to minimize costs and meet deadlines. The initial schedule considers only the private resources to check if they already satisfy the deadline. If the deadline is not satisfied, the algorithm select tasks considering start time as well as priority. The selected tasks will be rescheduled considering the public cloud as well. The algorithm repeats these steps, until the deadline is met or a certain number of iterations are reached.

Thanavanich *et al.* [17] has proposed two energy-aware scheduling heuristics, that take into consideration not only makespan conservation, but also energy reduction. In this, they have improved upon the already existing HEFT (Heteroge- neous Earliest Finish Time) and the CPOP (Critical Path on a Processor) algorithms.

Bittencourt *et al.* [18] has proposed HCOC algorithm. The user stipulates a deadline D for the workflow, and the proposed algorithm tries to optimize the monetary execution costs, while maintaining the execution time lower than D.

IV. Problem Formulation

A. Cloud Computing Model

In our model, the cloud computing system consists of

private and public cloud, hosted in datacenters. These datacenters contain large number of virtual machines (VMs). Each VM has a specific amount of processing power, communication bandwidth, RAM, and storage associated with it. Based on the capability of each VM, tasks are allocated to each VM. The execution time of each task depends on the capability of the VM to which it is allocated. If VM1 has more processing power than VM2, then the same task will be executed in lesser time by VM1 than VM2. There is a cost associated with the usage of VMs. In the private cloud, as it is owned by the organization itself, the cost of usage is taken to be zero. But for the public cloud, there is a usage cost, and this cost is directly proportional with the capability of VM taken on lease.

B. System Model

Binary Integer Programming (BIP) is used for mathematically formulating the problem. Consider that there are C cloud providers, I instance type and N tasks for an application. Here, P_{im} is the price for usage of the m^{th} instance of the i^{th} cloud. X_{jim} is a state variable. X_{jim}=1, if the j^{th} task is assigned to the m^{th} instance of the i^{th} cloud, else 0. r_{jim} is the running time of j^{th} task in the m^{th} instance of the i^{th} cloud. The objective functions, cost (CT) and execution time (ET) are defined as:

$$\text{CT}=\sum_{j=1}^{N}\sum_{i=1}^{C}\sum_{m=1}^{I} P_{im}.X_{jim}.r_{jim} \quad (1)$$

$$ET=\sum_{j=1}^{N}\sum_{i=1}^{C}\sum_{m=1}^{I} X_{jim}.r_{jim} \quad (2)$$

Our aim is to minimize functions CT and ET simultaneously subject to the constraints:

$$\sum_{j=1}^{N}\sum_{m=1}^{I} mem_m.X_{jim} \le mem_c, \vee i \in [1,C] \quad (3)$$

$$\sum_{j=1}^{N}\sum_{m=1}^{I} cpu_m.X_{jim} \le cpu_c, \vee i \in [1,C] \quad (4)$$

where mem_m is the memory available at each instance and, mem_c is the memory available at a particular cloud, cpu_m is the *cpu* resource of each instance and, cpu_c is the *cpu* resource available at particular cloud.

V. Proposed System

A. Bi-Objective Optimization

There are some classical methods for optimization of multiple objectives. First one is the method of objective weighting. In this all the objective functions are combined to get a single objective function, $Z=\sum_{i=1}^{N} w_i.f_i(x)$ where the weights w_i are fractional numbers ($0 \le w_i \le 1$). Second one is the method of distance functions, in which a decision vector (e_i) is used to get the single objective function Z, $Z=[\sum_{i=1}^{N}| f_i(x)-e_i |]^{\frac{1}{r}}, 1 \le r \le \infty$ The third method is min-max formulation, which attempts to minimize the relative deviations from individual optimum. It is represented as, $minimize\ F(x)=\max[Z_j(x)],\ j=1,..N$ [19].

In the aforementioned methods, multiple objectives are combined into a single objective. Single objective optimization results in a single solution. While in these cases, it would be better if the choice is given to the decision maker as it cannot be said that only a single solution is correct. Also these methods require prior information about the optimum before optimization can be done. To overcome these disadvantages, an evolutionary multi-objective optimization algorithm, namely Nondominated Sorting Genetic Algorithm II (NSGAII), where both objectives are considered on an individual basis for reaching the final solution, has been used.

Evolutionary optimization algorithms start with a population of solutions. Usually the initial population consists of randomly generated solutions. If some knowledge about the characteristics preferred in the initial solution is known, then it can be used to make the algorithm converge faster.

In this work, the two objective functions, given in equations (1) and (2) have to be optimized. Here, a variation of GA is used. In GA, on the initial population, selection, crossover and mutation operations are performed. For selection, a fitness function is used. Solutions with higher value for this fitness function will be selected, to be part of the mating population.

During crossover, two solutions from the mating pool are taken randomly and the information in the parent solutions will be exchanged. Crossover is done with a crossover probability. It denotes the proportion of the population taking part in crossover. The new solutions obtained after crossover will be a part of the new population along with the solutions that were not selected for crossover.

After crossover, mutation is performed. Mutation involves making small changes to the solution. It is performed with a mutation probability. For example, if the mutation probability is 1/m, then only one out of m solutions will be changed. The mutated as well as unchanged solutions will become a part of the modified population. All the above operations will be performed until a termination criterion is met. The termination criterion is usually based on the number of iterations to be performed.

GA can ideally be used only for single objective optimization. NSGAII is a variation of GA that uses a concept called domination to optimize multiple objectives. The domination between two solutions is defined as follows. A solution x_1 is said to dominate the other solution x_2, if the solution x_1 is no worse than x_2 in all objectives and the solution x_1 is strictly better than x_2 in at least one objective [19].

Fig. 1 is an example that shows how a nondominated set is identified. It depicts two functions f_1 and f_2, to be optimized. This means that simultaneously the value of f_1 has to be maximized while the value of f_2 has to be minimized. Each point represents the values of f_1 and f_2 at that point. Take each pair of points and see whether one point dominates the other.

Get the set of all points not dominated by any of the other points. This forms the nondominated set. Consider the 1st point, it gets dominated by the 3rd point. Hence 1st point will not be part of the nondominated set and so on. The nondominated points in Fig. 1 are 3, 5, and 6 [20].

Multiple solutions are present in the nondominated set. Since none of the solutions is better than any other in the solution set, one of them can be chosen by considering the need of the user and the characteristics of the application. This means that problem specific knowledge can be used to select the final result. This result is used for scheduling in the cloud.

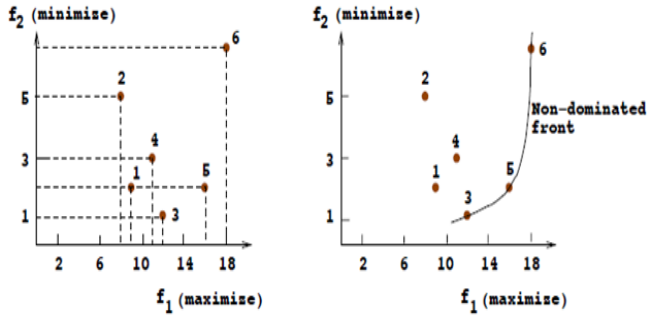

Fig. 1. Multiobjective optimization example.

As given in Algorithm1, the proposed system creates a random initial population, P. In the t^{th} generation, the child population, Q_t is created. The parent population and the child population is combined to get R_t. The population is sorted into fronts based on the domination principle using the algorithm Nondomination-sort(). It assigns a rank to each individual in the population, corresponding to its domination level. The individuals in the first front are assigned rank 1 (the highest level of domination) and the ones in the second front are assigned rank 2 and so on.

Algorithm 1: Proposed System

for Each Task Set do
$P1$←Create-initialpopulation()
for Each Individual in $P1$ do
Evaluate ET and CT
end for
$Q1$←Create-childpopulation($P1$)
t←1
while t←Max-no-of-generations do
Rt=Pt U Qt
Front[] ← Nondomination-sort(Rt)
while |Pt+1 | + |Front[i]| ≤ N do
crowding-distance(Front[i])
Pt+1=Pt+1 U Front[i]
Increment i
end while
Sort Front[i] on crowding distance
Pt+1=Pt+1 U Front[i][1 : (N–|Pt+1|)]
Qt+1=Create-childpopulation(Pt+1)
t=t+1
end while
Pareto-set← Front[1]
Avg←ET/|Pareto–set|
schedule←Most cost effective of Pareto-set elements
with ET<Avg
Add schedule to the scheduling queue.
end for

A new population, P_{t+1} is created by adding the individuals in each front. When a situation is reached such that all the elements in a front cannot be added, the individuals in that front are sorted in descending order based on its crowding distance. Then the individuals with higher crowding distance are added to P_{t+1}. To get the crowding distance for the respective front, crowding-distance function is called. The next step is to create a child population. For this, selection of parents for crossover is on the basis of crowding distance and rank. The ones with lesser rank are selected. If both individuals under consideration have the same rank then the one with higher crowding distance is selected. Crossover and mutation is done on the selected population to get the modified population.

The whole process is repeated a number of times. At the end of it, the individuals in the first front forms the pareto optimal set.

For selecting an individual from the pareto optimal set, the average of the execution time for the set is taken. All the individuals having the value of execution time below the average is selected and from among those individuals the most cost effective one is chosen.

Algorithm 2: Nondomination-sort(P)

for each p ε P do
S_p=φ
n_p=0
for each q ε P do
if p<q then
S_p=S_p ∪ q
else if q<p then
n_p=n_p + 1
end if
end for
if n_p=0 then
p_{rank}=1
F_1=F_1 ∪ p
end if
end for
i=1
while F_i≠φ do
Q=φ
for each p ε F_i do
for each q ε S_p do
n_q=n_q − 1
if n_q= 0 then
q_{rank}=i + 1
Q=Q ∪ q
end if
end for
end for
i=i + 1
F_i=Q
end while

In the algorithm of Nondomination-sort(), there are two important state variables associated with each individual, p

1) S_p, a set of individuals that p dominates.
2) n_p, domination count - the number of individuals that dominate p.

At first, F1 (first nondomination front) is found. For this, each individual is compared with every other individual in the population, to see if it dominates (domination is denoted by the symbol <) the other individuals. Also values of the state variables associated with each individual are updated. For example, consider individual p, the values for S_p and n_p,

in comparison with all other individuals in the population is found. If n_p is equal to zero after all comparisons it means that none of the individuals dominate *p*.

All individuals with domination count equal to zero, is put into the first front and assigned rank 1. To get the next front, the same process can be repeated after discarding the members of the first front. Instead, to reduce the number of iterations another method is used. For each individual p in the previous front, consider the elements in its S_p set (the individuals being dominated by p) and reduce its domination count by one. This is done since all elements of previous fronts are discarded to identify the next front. The individuals with domination count equal to zero, is put into the next front The above procedure is repeated to get all fronts [21].

Crowding distance of the i^{th} individual, denoted by $I[i]_{dt}$ indicates how near an individual is to its neighbours. Individuals with large crowding distance give a diverse population.

Algorithm 3: crowding-distance(I)

```
l=|I|
for Each i do
I [i]dt=0
for Each objective m do
I=sort(I, m)
I [1]dt=I [1]dt =∞
for i=2 to l-1 do
Update I [i]dt according to (5)
end for
end for
end for
```

In crowding-distance() algorithm, for each objective, the elements in the front are sorted based on objective values. Boundary points are assigned an infinite distance value. Intermediate points are assigned values according to:

$$I[i]_{dt} = (I[i+1].m - I[i-1].m) / (f_m^{\max} - f_m^{\min}) \quad (5)$$

where $I[i].m$ refers to the m^{th} objective function value under Consideration and $f_m^{\max}$, $f_m^{\min}$ are the maximum and minimum values of the m^{th} objective function.

Algorithm 4: Create – childpopulation(Pop)

```
while Elements in New Pop<N do
for i=1 to 2 do
indiv1, indiv2←Random individuals from Pop
if rank [indiv1]< rank[indiv2] then
parent [i]←indiv1
else if rank[indiv1]> rank[indiv2] then
parent [i]←indiv2
else if rank[indiv1]=rank[indiv2] then
if I[indiv1]dt >I[indiv2]dt then
parent [i]←indiv1
else if I[indiv1]dt< I[indiv2]dt then
parent [i]← indiv2
end if
end if
end for
   children ← crossover(parent[1],parent[2])
NewPop ← mutation(children)
end while
```

B. Scheduling

A set of pareto-optimal points is generated through the above mentioned multiobjective optimization algorithm. The values of both objective functions (CT and ET) for these points are also obtained. The selection of a single point is guided by application specific knowledge. The schedule corresponding to that point, gives the optimal mapping between tasks and machines. According to this schedule, task assignment to processors is done. This schedule determines which of the tasks has to be assigned to the public cloud and which all has to be assigned to private cloud. In the same way all sets of tasks of an application is executed in hybrid cloud.

VI. Implementation

CloudSim is a toolkit (library) for simulation of cloud computing scenarios. It provides basic classes for describing data centers, virtual machines, applications, users, computational resources, and policies for the management of diverse parts of the system. These components are put together to evaluate multiobjective optimization of scheduling in hybrid cloud.

The experiments use 6 VMs. Two VMs are in the public cloud and incur usage charges. Four VMs are part of the private cloud and does not incur any charge. The VMs in the private cloud have 200, 250, 300 and 350 MIPS processing capability, while the ones in the public cloud have 750 and 1000 MIPS processing capability.

The application is split into task sets. Each task set corresponds to sets of tasks that can be executed in parallel. In this work, the task set is comprised of six tasks each. Corresponding to each task set, a task schedule is generated, by using multiobjective evolutionary optimization algorithm NSGAII. The task schedules generated for all task sets are combined to get the application schedule.

First, an initial population is randomly created. The population will consist of potential solutions. Each potential solution is a schedule of the task set under consideration. Permutation encoding is used for representing the scheduling problem. A schedule (1,5,0,3,2,4) means that the 1^{st} task is assigned to the 1^{st} machine and 1^{st} task is assigned to the 5^{th} machine and so on.

In the above algorithm, to accommodate the multi-objective nature of the problem, the selection procedure uses nondominated sorting as opposed to the normal selection techniques used in GA. The rest of the procedure is similar to GA. Two fitness values for each of the potential solutions are calculated based on the precomputed execution time and cost. Based on these fitness values, nondominated sorting and ranking is done.

Individuals are selected based on rank and one point crossover of the selected individuals is performed. This procedure is repeated until the termination criteria are met. The pareto- optimal schedule for the task set is obtained as the result. The above procedure is repeated for all task sets in the given application.

The parameters that are used in the algorithm are given in Table I. The pareto-optimal points corresponding to feasible task schedules are obtained as the result. From this a single point is selected, by using application specific knowledge.

Corresponding to this point, a task schedule is obtained. The task schedule maps tasks to processors. The task schedules generated for all task sets are combined to get the application schedule Based on the schedule application is executed in CloudSim and makespan and cost for executing the application in the cloud is estimated and analyzed.

TABLE I: PARAMETERS USED IN THE ALGORITHM

Parameters	Value
Population Size	100
Number of Generations	100
Crossover Rate	0.95
Mutation Rate	0.2

VII. RESULTS AND DISCUSSION

This section presents the results obtained from our study. The pareto-optimal solution obtained, is analyzed based on the cost and execution time for different sets of tasks.

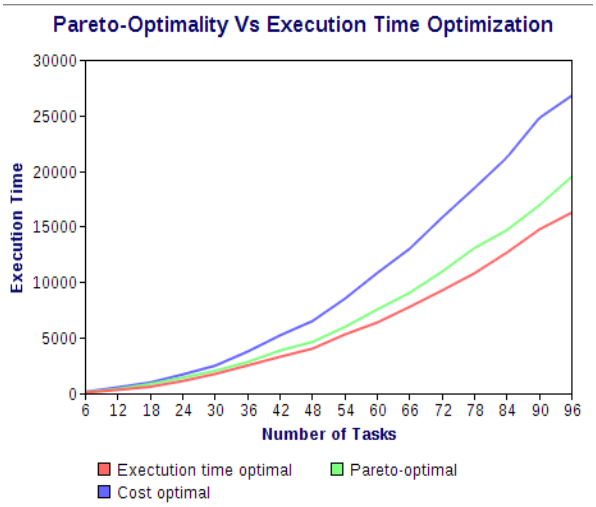

Fig. 2. Variation in execution time.

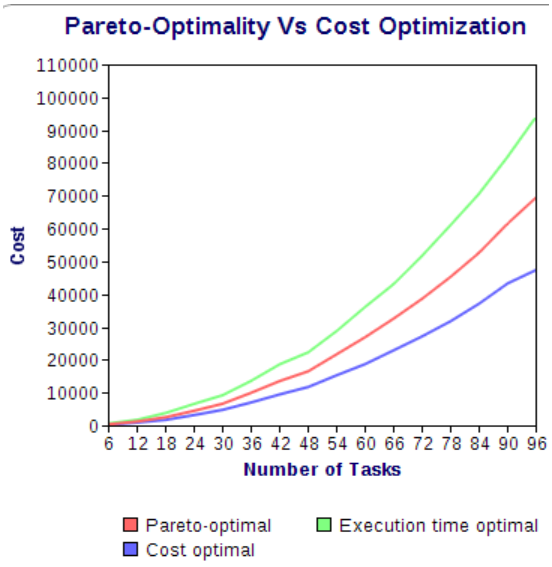

Fig. 3. Variation in cost.

The execution time of the schedule obtained after bi-objective optimization is compared with the execution time of the schedule obtained after cost optimization as seen in Fig. 2. In the same way, pareto-optimal schedule is compared with schedule obtained after optimization of execution time in Fig. 3. This is done by taking the total execution time and total cost in both single objective optimization cases and comparing with the total execution time and total cost obtained from the pareto optimal solution. It is seen that this gives a middle ground between both execution time optimization and cost optimization and can be used for scheduling in the cloud.

Table II gives the average values of CT and ET for 16 applications, when different scheduling algorithms are used. On an average, the execution time of an application is 29% less if a pareto-optimal algorithm is used for scheduling instead of a cost optimal scheduling algorithm. Also on an average, the cost of executing an application is 26 % less if a pareto-optimal algorithm is used for scheduling instead of an execution time optimal scheduling algorithm.

In our experiment, the total execution time of the pareto-optimal schedule is very near to that of the minimum possible execution time (which is equivalent to single objective optimization of execution time). And also pareto-optimal schedule has much lesser usage charges when compared to the schedule for single objective optimization of execution time. So its overall efficiency is higher.

TABLE II: AVERAGE ET AND CT VALUES FOR DIFFERENT ALGORITHMS

Algorithm	Execution Time	Cost
Execution Time Optimal	6053	33961
Pareto Optimal	7112	25223
Cost Optimal	10047	17638

VIII. CONCLUSION

An application on arrival is split into multiple tasks. Using the information from these tasks, scheduling of tasks on different machines is done. Our primary contribution is the proposal and implementation of an algorithm for the simultaneous optimization of execution time and cost of scheduling, in hybrid cloud. NSGAII algorithm is used for the same. The implementation results show the feasibility of such a scheduling strategy. Bi-objective optimization of scheduling, gives lesser cost of usage than optimization based on execution time. It also gives an execution time very close to the execution time obtained in the optimization of execution time alone. This gives a better choice to users. Now users do not have to choose between the two extremes of optimizing either on the basis of execution time or optimizing on the basis of cost. It gives a tradeoff point between application execution time and cost.

REFERENCES

[1] V. A Leena, B. A. S. Ajeena, and M. S. Rajasree, "Inter-cloud scheduling technique using power of two choices," in *Proc. IEEE International Conference on Computational Intelligence and Computing Research(ICCIC)*, December 2013.

[2] S. Nagadevi, K. Satyapriya, and D. Malathy, "Economic cloud schedulers for optimized task scheduling," *International Journal of Advanced Engineering Technology*, vol. IV, issue I, Jan.-March 2013.

[3] H. D. Kim and J. S. Kim, "An On-line scheduling algorithm for grid computing systems," in *Proc. Second International Workshop on Grid and Cooperative Computing*, 2003.

[4] F. Wang, N. Helian, and G. Akanmu, "User-priority guide Min-Min scheduling algorithm for load balncing in cloud computing," in *Proc. National Conference on Parallel Computing Technologies*, 2013.
[5] U. Bhoi and P. N. Ramanuj, "Enhanced max-min task scheduling algorithm in cloud computing," *International Journal of Application or Innovation in Engineering and Management*, vol. 2, issue 4, April 2013.
[6] G. Ming and H. Li, "An improved algorithm based on max-min for cloud task scheduling in recent advances in computer science and information engineerin," *Lecture Notes in Electrical Engineering*, Springer, vol. 125, pp. 217-223, 2012.
[7] S. Bitam, "Bees life algorithm for job scheduling in cloud computing," in *Proc. ICCIT*, 2012.
[8] P. kumar and A. Verma, "Independent task scheduling in cloud computing by improved genetic algorithm," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, issue 5, May 2012.
[9] X.-Q. Song, L. Gao, J.-P. Wang, "Job scheduling based on ant colony optimization in cloud computing," in *Proc. International Conference Computer Science and Service System (CSSS)*, June 2011, pp. 3309-3312.
[10] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, Nitin, and R. Rastogi, "Load balancing of nodes in cloud using ant colony optimization," in *Proc. 14th International Conference on Modelling and Simulation*, 2012.
[11] W. Zhao and K. Ramamritham, "Simple and integrated heuristic algorithms for scheduling tasks with time and resource constraints," *Journal of Systems and Software*, vol. 7, issue 3, pp. 195-205, September 1987.
[12] M. Mattess, C. Vecchiola, S. K. Garg, and R. Buyya, "Cloud bursting: Managing peak loads by leasing public cloud services," *Cloud Computing: Methodology, Systems, and Applications*, October 2011.
[13] R. N. Calheiros and R. Buyya, "Cost-effective provisioning and scheduling of deadline constrained applications in hybrid clouds," *Web Information Systems Engineering - WISE*, 2012, pp. 171-184.
[14] S. Kailasam, N. Gnanasambandam, J. Dharanipragada, and N. Sharma, "Optimizing ordered throughput using autonomic cloud bursting schedulers," *IEEE Transactions on Software Engineering*, 2012.
[15] X. Zuo, G. Zhang, and W. Tan, "Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid Iaas Cloud," *IEEE Transactions on Automation Science and Engineering*, 2013.
[16] L. F. Bittencourt, C. R. Senna, and E. R. M. Madeira, "Scheduling service workflows for cost optimization in hybrid clouds," in *Proc. International Conference on Network and Service Management (CNSM)*, 2010.
[17] T. Thanavanich and P. Uthayopas, "Efficient energy aware task scheduling for parallel workflow tasks on hybrids cloud environment," in *Proc. International Computer Science and Engineering Conference (ICSEC)*, 2013.
[18] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, Springer, vol. 2, issue 3, pp. 207-227, December 2011.
[19] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Journal of Evolutionary Computation*, vol. 2, no. 3, 1994.
[20] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction*, Indian Institute of Technology Kanpur, February 2011.
[21] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, April 2002.

Leena V. A. was born in Kerala, India in 1989. She is a postgraduate student in the Computer Science Department of College of Engineering, Trivandrum since 2012. She received the bachelor degree in computer science and engineering from Mahathma Gandhi University in 2011. Her major fields of interest include cloud computing, optimization techniques, neural networks, fuzzy systems, and evolutionary algorithms.

Ajeena Beegom A. S. is an associate professor in College of Engineering, Trivandrum, Kerala, India. She received the bachelor degree in computer science and engineering from University of Kerala in 1995 and master degree from National Institute of Technology, Tiruchirappalli, India in 2005. Her active research topics include cloud computing and data mining.

Rajasree M. S. is the director of Indian Institute of Information Technology and Management-Kerala (IIITM-K), India. She received the bachelor degree in computer science and engineering from National Institute of Technology, Calicut. She completed her master and PhD degrees from IIT-Madras. She is a professor in computer science and engineering and has worked in Government Engineering Colleges as a lecturer, professor and principal. Her research interests include distributed and object oriented software engineering, cloud computing and pattern recognition. She has a good number of publications in international conferences and journals and has also served in the programme/technical committees of a large number of national/international conferences.