International Journal of Research in Engineering, Science and Management
Volume-2, Issue-1, January-2019
www.ijresm.com | ISSN (Online): 2581-5792

Cloud Workflow Scheduling Using Deadline based Cost Efficient Approach

Snehal Nemichandji Kankariya
PG Student, Department of Computer Engineering, SSVPS B. S. Deore College of Engineering, Dhule, India

***Abstract*: Recently, executing workflow applications is become common in the cloud because this causes workflow application to use resources on demand. This is also advantage over traditional workflow scheduling algorithms that only aims on optimizing the execution time. It includes method to minimize execution cost of a workflow in clouds under a deadline constraint and uses a metaheuristic algorithm L-ACO and a simple heuristic ProLiS. ProLiS distributes the deadline to each task using upward rank mechanism, and uses a two-step list scheduling methodology: rank tasks and sequentially allocates each task a service which meets the sub-deadline and minimizes the cost. L-ACO uses ant colony optimization for performing deadline-constrained cost optimization and uses the same deadline distribution and service selection methods as ProLiS to build solutions The MMAS framework is utilized for the pheromone updating in L-ACO. Moreover, in order to guide the search towards a near-optimal solution meeting the deadline, the deadline constraint is relaxed and this relaxation is gradually diminished until is removed completely.**

***Keywords*: Ant colony optimization, Deadline, Cloud Computing, Workflows, Workflow Scheduling.**

1. Introduction

Cloud computing is one of the popular and latest leading technology. Cloud computing is a large scale distributed computing paradigm in which a pool of abstracted, virtualized, dynamically scalable and services are delivered on demand to external customers over the internet. Here cloud consists of set virtual machine which include both computational and storage facility. Cloud computing provides three important services. They are infrastructure as a service, platform as a service and software as a service. These services are available in pay per use on demand model. Scheduling is one of the most famous activity in cloud computing environment to increase the efficiency of work and performance of task. Task scheduling is valuable concept which is greatly affects the behavior of the performance of tasks. Workflow application can be useful in many areas such as astronomy, bioinformatics, and physics, for the development of Scientific Application. These workflows contain hundred or thousand number of tasks which can be represented by Directed Acyclic Graph, in which node represents the Task and edge represents the relationship between tasks. Cloud computing has public model in which resources can be hire by paying charge for it, called as pay-per-use system. In this model resources are dynamically scalable according to the need of application. So, the resources for executing the workflow can be provisioned on demand and also its number can be increased until it contains enough budgets to support it [1]. Cloud computing provides Infrastructure as a Service model, in which user obtains the virtual machines as resource and deploy their workflow applications on it. Cloud makes this a suitable platform to execute deadline-constrained scientific workflows. One of the constraints for workflow execution is the budget allotted for it, which becomes limitation for hiring the number of resources. This is because the cloud providers apply charges for resource utilization for time interval. Our work aims at scheduling the workflow, such that the execution is completed before the deadline and within the budget constraint.

2. Related work

To schedule the scientific workflows in Software as a Service Cloud S. Abrishami et al proposed a new algorithm based on the partial critical path in 2012. This algorithm tries to minimize the workflow execution cost by meeting its deadline. This algorithm first schedules the critical path of the workflow and then finds the partial critical path to each task on critical path [5]. In 2013, Salid Abrishami, Mahmoud Naghibzadeh and Dick H. J. E pema proposed Deadline-Constrained Workflow Scheduling Algorithm for Infrastructure Service. In this paper, execution time is minimized by maintaining the user defined deadline. The author implements two different algorithms based on PCP. First is Cloud Partial Critical Path and second with Deadline Distribution [3]. The disadvantage of this algorithm is that they didn't consider the data transfer time during provisioning and scheduling.

The author Rajkumar buyya et al proposed a combine resource provisioning and scheduling for scientific workflow execution. They used Meta heuristic optimization algorithm, Particle Swarm Optimization to minimize the execution cost. This algorithm performs better for smaller sized workflow [4]. Wu et al. propose a heuristic algorithm called PCP-B2 [7] and the budget distribution is implemented via a binary search method. Another idea for this problem is to first construct a schedule that has good performance on one considered objective, and then keep swapping tasks between resources to improve as much as possible for the other objective. The LOSS approach starts with a schedule with relatively short execution time, and repeats reassigning tasks to cheaper resources until the overall cost is lower than the given budget.

Sakellariou, R., Zhao proposed a basic model for workflow applications that modeled as directed acyclic graph (DAGs) and that allow to schedule the nodes of DAG onto resources in a way that satisfies a budget constraint and is optimized for overall time. Thus, the aim is to find the schedule that gives the shortest makespan for a given DAG and a given set of resources without exceeding the budget available [7]. In order to reduce the limitations of previous algorithms, proposed algorithm uses probabilistic upward rank mechanism and pheromone trail such that the tasks are complete its execution before deadline.

3. System model

A workflow application can be represented by a directed acyclic graph , DAG = (V, E), where, V is a set of n tasks $\{t_1, t_2,\ldots t_n\}$, and E is a set of precedence dependencies. Each task represents an indivisible individual application with a certain amount of computation workload w_i. A precedence dependency ei,j=(ti, tj) indicates that task tj can start executing only after task ti finishes. Furthermore, if there is data transmission data i,j attached onto ei,j, then tj can start only after the data from ti has been received. The source and the destination of a dependency ei,j is called the parent task and the child task, respectively. To generalize the workflow with one entry and one exit, two dummy tasks t_{entry} and t_{exit} with zero execution time are added to the beginning and the end of the workflow, respectively [1].

When task ti is allocated to service s_l, the execution time can be calculated via:

$$ET_{i\,,l} = w_i / p(s_l)$$

Moreover, if s_l is leased by the user from LST_l (lease start time) to LFT_l (lease finish time), the required cost can be calculated via:

$$EC_l = (LFT_l - LST_l) /TI\times c\ (s_l)$$

The data transfer time of a dependency $e_{i,j}$, $TT_{i,j}$, depends on the amount of data to be transferred and when both tasks t_i and t_j are executed on the same service, $TT_{i,j}$ becomes zero. Since assumption is that all services of the provider are in the same physical region, so the average bandwidth (bw) between the computation services is roughly equal and the internal data transfer is free [1].

$$\begin{array}{ll} TT_{i,j}= Data\ i,j/\ bw & \text{if } ser(t_i)!=ser(t_j) \text{ and} \\ TT_{i,j}= 0 & \text{otherwise.} \end{array}$$

A. System architecture

Fig. 1 represents the system architecture of the system. The user will submit the workflow, which has to be executed in the cloud. User has to provide the XML file representing Directed Acyclic Graph (DAG) structure of workflow. The DAX [9] files contains list of tasks, dependencies between tasks, their computation time and size of the input and output files generated by the tasks. It also contains information about task as task id, its runtime and name of the task. This uses a metaheuristic algorithm L-ACO as well as a simple heuristic ProLiS. ProLiS distributes the deadline to each task, proportionally to a novel definition of probabilistic upward rank, and follows a two-step list scheduling methodology: rank tasks and sequentially allocates each task a service which meets the sub-deadline and minimizes the cost.

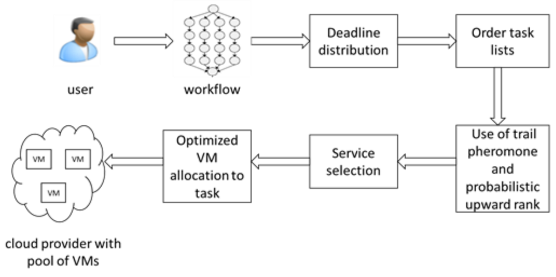
user
workflow
Deadline distribution
Order task lists
Use of trail pheromone and probabilistic upward rank
Service selection
Optimized VM allocation to task
VM
VM
VM
cloud provider with pool of VMs

Fig. 1. System architecture

B. Probabilistic list scheduling algorithm

It is simple heuristic algorithm. It consists of following three steps:

1) Deadline distribution

This is based on probabilistic upward rank mechanism which is calculated via:

$$pr_i = MAX\ \{\ pr_j + y_j * Data_{i,j}/\ bw\ \}_{tj\in\ ti\text{'s children}} + w_i / p(s^*)$$

Here, γ_j is a Boolean variable denoting whether transmission time to t_j is considered in the calculation of pr_i. Specifically, following equation can be obtained [1].

$$\begin{array}{ll} y_j = 0 & \text{if } 1 - \theta^{ccrj} < rand() \\ y_j = 0 & \text{otherwise.} \end{array}$$

Here, ccr_j is the computation to communication ratio of t_j, rand() is a function returning a random number in [0, 1), and θ is a parameter larger than 1. Therefore, the less ccr_j is, the larger the probability that γ_j returns 0 is, and vice versa.
Deadline distribution based on probabilistic upward rank is implemented via following equation, where pr_i is used instead of r_i.

$$psd_i = D *\{\ pr_{entry} - pr_i + w_i/p(s^*)\}/\ pr_{entry}$$

2) Task ordering

This is also based on probabilistic upward rank mechanism since it is aware of the fact that data transmission time can be zero in contrast to its previous work like upward rank, static level.

3) Service selection

The first criterion for service selection is to select a service which meets its sub-deadline and minimizes the cost increment of adding t_i . This increment is not directly calculated as the cost of running t_i on s_l, but is calculated as the execution cost of s_l after adding t_i minus that before adding t_i [1].

However, in the case when no service can meet the sub deadline, the criterion to select a service from R is instead to minimize the finish time of the task. But in case, if the selected service is not of the fastest type, then it is tried to set its type to a faster level and update the finish time of each task deployed on it. Because of this the probability for the solution to meet the overall deadline is increased.

4) L-ACO algorithm

Ant Colony Optimization (ACO) is based on the capability of real ants to find the shortest path between their nest and a food source. It is metaheuristic algorithm. It uses the ACO metaheuristic to modify the task ordering step in ProLiS to solve the workflow scheduling problem [1]. It has also three steps as deadline distribution, task ordering and service selection. It uses same methods as used in ProLis for deadline distribution and service selection. But in case of task ordering it uses probabilistic upward rank as well as pheromone trail mechanism where pheromone trail $\tau_{i,j}$ is defined as the desirability of selecting task t_j just after task t_i. $\tau_{i,j}$ is initialized to a uniform value and then continuously updated in the loop of ACO algorithm.

In L-ACO an ant colony with a size of colSize is created and the pheromone trail is initialized. Then, each ant builds a solution for the problem, the local best solution is stored to lbSol by comparing all the built solutions, and afterwards the pheromone trail is updated. The same procedure is repeated until the iteration number k increases to maxNo and finally the global best solution is returned as output. There must be guarantee that the solution obtained through the service selection step does not violate precedence constraints between tasks, the ordered task list should be a topological ordering of the workflow graph. Specifically, a topological sort of a DAG is a linear ordering of all its vertices such that if DAG contains an edge (u, v), then u appears before v in the ordering. To preserve the precedence dependencies, ant must undergo the Kahn's algorithm to generate a task ordering based on the pheromone trail and heuristic information. Here there is use of an improved independent optimization method to handle constraint optimization problem where deadline is relaxed initially in some iterations and after that it no longer relaxed.

The deadline constraint D at the kth iteration of L-ACO is relaxed to $D_\varepsilon(k)$ and ε comparison > ε between two solutions (f_1, φ_1) and (f_2, φ_2) is introduced to compare solutions [6].

$$D_\varepsilon(k) = D + \mathrm{MAX}\{0, M_{base} - D\} * (1-k/k_T)^{cp} \quad \text{if } 0<=k< k_T$$
$$D_\varepsilon(k) = D \quad \text{if } k> k_T$$
$$(f_1, \varphi_1) > \varepsilon\, (f_2, \varphi_2) = f_1 < f_2 \quad \text{if } \varphi 1, \varphi_2 < D\varepsilon\,(k)$$
$$(f_1, \varphi_1) > \varepsilon\, (f_2, \varphi_2) = f_1 < f_2 \quad \text{if } \varphi 1 = \varphi_2$$
$$(f_1, \varphi_1) > \varepsilon\, (f_2, \varphi_2) = \varphi_1 < \varphi_2 \quad \text{Otherwise}$$

Here M_{base} is the makespan value when allocating all tasks to the slowest service, k is the current iteration number of L-ACO, k_T is the iteration number where relaxation is terminated, and cp is the parameter to control the curve of $D_\varepsilon(k)$. The relaxed deadline $D_\varepsilon(k)$ decreases gradually with the increase of k until reaching k_T. After that, the deadline is no longer relaxed in order to obtain solutions with no constraint violation.

Finally, updation of pheromone trail is done by using MAX-MIN Ant System (MMAS) mechanism where the pheromone values on pheromone trails are bound between an upper and lower limit (τ_{min} and τ_{max}) in order to avoid search stagnation and enhance exploration. At the beginning of L-ACO, pheromones of all edges are initialized to τ_{max}. In each iteration, the ant which builds the best solution, i.e., which ranks highest via ε comparison, deposits a certain amount of pheromone on each visited edge [11].

Meanwhile, a percentage of existing pheromones in all edges evaporate. Thus, at the end of the kth iteration of the algorithm, the pheromone is updated according to the following formula:

$$\Gamma_{i,j}\,(k+1) = (1-\rho) * \Gamma_{i,j}(k) + \Delta\, \Gamma_{i,j}(k)$$

Where, ρ is the pheromone evaporation coefficient, and $\Delta\tau i,j(k)$ is the amount of pheromone deposited by the ant that builds the best solution (denoted as $s_{best}(k)$), defined as:

$$\Gamma_{i,j}(k) = 1/\, f_{best}^{(k)} \quad \text{if } e_{i,j} \in s_{best}(k)$$
$$\Gamma_{i,j}(k) = 0 \quad \text{Otherwise[1].}$$

Table 1
Virtual Machine Types

Type	Speed	Cost
0	1.0	0.12
1	1.5	0.195
2	2.0	0.28
3	2.5	0.375
4	3.0	0.48
5	3.5	0.595
6	4.0	0.72
7	4.5	0.855
8	5.0	1.0

4. Simulation of cloudsim

CloudSim [8] is a new open source toolkit developed using java that generalized, and advanced simulation framework allows simulation of Cloud computing and application services. CloudSim is a simulation tool for creating cloud computing environment and used as the simulator in solving the workflow scheduling problem. CloudSim allows us to create a data center with a set of hosts and number of virtual machines as resources. Each task of a workflow can be assigned to appropriate virtual machine once it's all parent tasks get executed.

A. Simulation description

The result analysis was conducted on Dell PC with 2.0 GHz Intel i5 CPU and 4 GB of memory running windows 7 and CloudSim. Cloudsim is used to construct nine virtual machines in single data center. The XML files of four workflows are given as Input to algorithm. These workflows contain the number of tasks, and these tasks are provided for scheduling. It uses single datacenter and following nine types of virtual machines as shown in table 1. The simulation result shows that, the algorithm improves the performance of cloud system. Λ is set to small values to have tight deadlines. To evaluate the ability of each approach to obtain a valid solution that meets deadline constraints, λ is varied from 0.005 to 0.05 with a step of 0.005. Below tables show average success ratio of each approach to obtain a valid solution under different conditions.

Λ is also set to larger values so that to have more proper comparison of all four algorithms. The main observation is that the required normalized cost decreases with rising λ. But this varies from workflow to workflow. The performance of PSO [Particle Swarm Optimization] is specific to the workflow application and the value of λ. For example, PSO fails to obtain 100% success ratio when deadline is too tight that is its success ratio is 0.9 when λ is 0.005 in case of cybershake. In case of Ligo, when λ is 0.005 its success ratio is 0.In case of montage, success ratio is 0 when λ is 0.005 and 0.01.And ICPCP, LACO, ProLis obtains 100% success ratio when λ € [0.005, 0.05]. Also when λ is very small, not all approaches can obtain valid solutions and it is meaningless to compare normalized costs when the deadline is not met. Hence, with λ varying from 0.005 to 0.05, Tables 11-14 shows normalized cost of each approach only when the corresponding success ratio is 100%. In most cases ProLiS outperforms PSO and ICPCP, though it may not obtain valid solutions when the deadline is very tight. L-ACO performs the best of all and it achieves a success ratio of 100% for all λ and all workflows.

Fig. 2 to Fig. 5 shows line charts of success ratio versus deadline vector with deadline factor varying from 0.005 to 0.05 run on Cybershake, Ligo, montage, genome of size of 500.And also 20 instances of each size is used.

Fig. 6 to Fig. 9 shows line charts of normalized cost versus deadline vector with deadline factor varying from 0.005 to 0.05 run on Cybershake, Ligo, montage, genome of size of 500.And also 20 instances of each size is used. It shows that normalized cost of LACO is least among all four approaches. This shows that it performs best among all four algorithms. The required normalized cost decreases with rising λ.

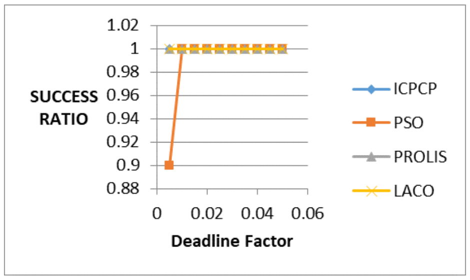

Fig. 2. Cybershake

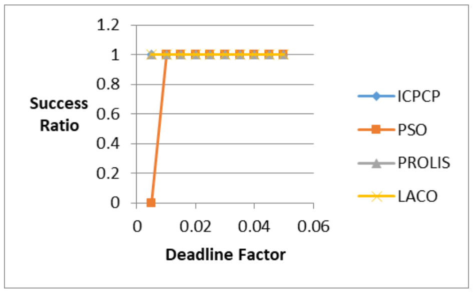

Fig. 3. Ligo

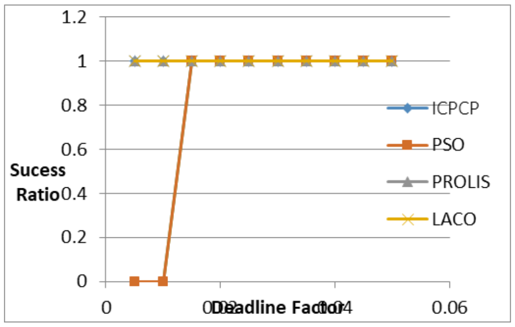

Fig. 4. Montage

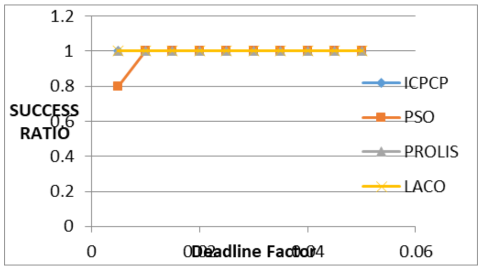

Fig. 5. Genome

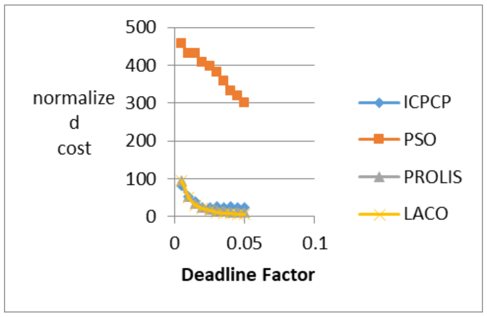

Fig. 6. Cybershake

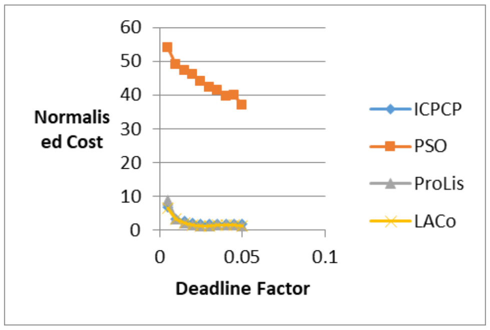

Fig. 7. Ligo

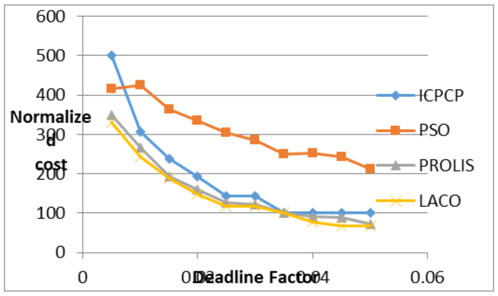

Fig. 8. Montage

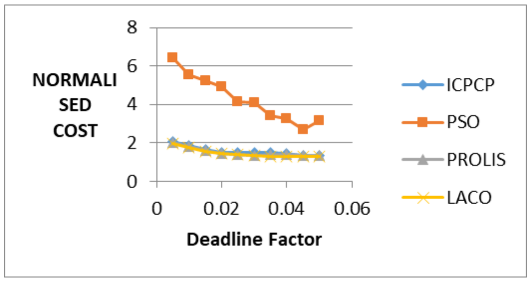

Fig. 9. Genome

It has also been proved that if iteration number increases there is no change in success ratio. Cost of LACO decreases. Cost of all other algorithms changes irregularly, approximately increase and decrease alternatively. If iteration increases further from 50 to 55 cost of LACO decreases and cost of all other algorithms decreases irregularly. It has been proved that When bandwidth is increased, data transfer time decreased and hence execution time decreases that's why cost of PSO, ProLiS, LACO decreases but in case of ICPCP cost decreases when deadline factor is in range [0.03,0.5]. Success ratio of PSO improved.

B. Benchmarks

Also there are two benchmarks which are used to calculate deadline of workflow D.

- *Cheap Schedule* - In this benchmark, only one instance of the cheapest service is used, and the workflow is scheduled by the simple and effective greedy algorithm HEFT [2]. Its makes pan and required execution cost are denoted as M_C and C_C, respectively;
- *Fast Schedule* - In this benchmark, only the fastest type of services is used and the workflow is scheduled by HEFT [2] as well. Its makes pan and required execution cost are denoted as M_F and C_F, respectively.

Also there is introduction of the deadline factor λ ($\lambda \in [0, 1]$) to represent the looseness degree of deadlines and the deadline of a workflow is determined based on λ as shown in:

$$D = M_F + (M_C - M_F) \times \lambda.$$

5. Conclusion

Metaheuristic algorithm L-ACO and a simple heuristic ProLiS aims to minimize the execution cost of workflow application under a user-defined deadline constraint. The success rate of L-ACO is the highest and it obtains the solutions meeting required constraints achieving lower cost. The performance of the simple heuristic ProLiS is very competitive. Further it is also proved that normalized cost can be decreased and success ratio can be improved by can be improved by increasing number of iterations and bandwidth.

References

[1] F. I. Quanwang Wu, "Deadline-constrained Cost Optimization Approaches for Workflow Scheduling in Clouds," *IEEE Transactions on Parallel and Distributed Systems,* vol. 2, no. 1, pp. 1-12, 3 august 2017.
[2] S. H. a. M.-y. Haluk Topcuoglu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274., 2002.,* vol. 13, no. 3, pp. 260-274, 2002.
[3] M. N. a. D. H. E. S. Abrishami, "Deadline-constrained work- flow scheduling algorithms for Infrastructure as a Service Clouds," *Future Genera- tion Computer Systems,* vol. 29, no. 1, pp. 158-169, 2013.
[4] R. B. M. A. Rodriguez, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud.*
[5] Mahmoud Naghibzadeh, Dick H.J. Epema Saeid Abrishami, "Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths," *IEEE,* vol. 8, p. 23, August 2012.
[6] S. S. T. Takahama, "Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation," *IEEE Congress on Evolutionary Computation,* pp. 1-9, 2010.
[7] R. Sakellariou *et al.*, "Scheduling Workflows with Budget Constraints," *Integrated Research in GRID Computing: Core GRID Integration Workshop*, pp. 189-202, Boston, MA: Springer US, 2007.
[8] CLOUDS. [Online]. http://www.cloudbus.org/cloudsim/
[9] The XML files that describe the workflow applications are available via https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[10] M. Malawski, "Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," *IEEE,* vol. 1, no. 2, pp. 1-11, 2014.
[11] A. H. H. H. T. Stützle, "MAX–MIN Ant System," *Future Generation Computer Systems,* vol. 16, no. 8, pp. 889-914, 2000.