Ant Colony Optimisation Algorithm with Different Variation Methods to Solve Grid Workflow Scheduling Problem

[I]Pankaj D.Khambre, [II]Aarti Deshpande, [III]Deeksha Singh, [IV]Sajju Gautam
[I,II]Dept. of Comp. Engg. GHRCEM, Pune, India
[III,IV]Dept. of Info Technology, BVDUCOE, Pune, India

Abstract
Ant Colony Optimization (ACO) is a relatively new computational intelligence paradigm inspired by the behaviour of natural ants (Dorigo & Stutzle, 2004). Ants often find the shortest path between a food source and the nest of the colony without using visual information. In order to exchange information about which path should be followed, ants communicate with each other by means of a chemical substance called pheromone. As ants move, a amount of pheromone is dropped on the ground, creating a pheromone trail. The more ants follow a given trail, the more attractive that trail becomes to be followed by other ants. This process involves a loop of positive feedback, in which the probability that an ant chooses a path is proportional to the number of ants that have already passed by that path .Hence, individual ants, following very simple rules, interact to produce an intelligent conduct at the higher level of the ant colony.

Index Terms
Variations, Grid Computing, Antnet, Fault Tolerance in Grid, Cost Driven.

I. Introduction

In recent years, many research works have been devoted to ant colony optimization (ACO) techniques in different areas. It is a relatively novel meta-heuristic technique and has been successfully used in many applications especially problems in combinatorial optimization. ACO algorithm configures the conduct of real ant colonies in manifesting the shortest path between food sources and nests. Ants can communicate with one another through chemicals called pheromones in their immediate environment. The ants release pheromone on the ground while walking from their nest to food and then go back to the nest. The ants move according to the amount of pheromones, the richer the pheromone trail on a path is, the more likely it would be followed by other ants. So a shorter path has a higher amount of pheromone in probability, ants will be liable to choose a shorter path. Through this mechanism, ants will eventually find the shortest path. Artificial ants imitate the behaviour of real ants, but can solve much more complicated problem than real ants can. ACO has been widely applied to solving various combinatorial optimization problems such as Travelling Salesman Problem (TSP), Job-shop Scheduling Problem (JSP), Vehicle Routing Problem (VRP), Quadratic Assignment Problem (QAP), etc. Although ACO has a powerful ability to discover the solutions to combinational optimization problems, it has the problems of stagnation and premature convergence and the convergence speed of ACO is very slow. Those problems will be more obvious when the problem size increases. Therefore, several extensions and improvements versions of the original ACO algorithm were introduced over the years. Various adaptations: dynamic control of solution construction , mergence of local search, a strategy is to partition artificial ants into two groups: scout ants and common ants and new pheromone updating strategies , using candidate lists strategies are studied to improve the quality of the final solution and lead to speedup of the algorithm. All these studies have contributed to the improvement of the ACO to some extents, but they have little obvious effect on increasing the convergence speed and obtaining the global optimal solution.
In the proposed system, the main modifications introduced by ACO are the following. First, to avoid search stagnation and ACO is more effective if ants are initially placed on different cities. Second, information entropy is introduced which is adjust the algorithm's parameters. Additionally, the best performing ACO algorithms for the TSP improve the solutions generated by the ants using local search algorithms. The experiment results show that the algorithm proposed in this study can substantially increase the convergence speed of the ACO.
In this paper, an improved ant colony optimization algorithm is developed for comparing the different methods applied . This algorithm is used to produce near-optimal solutions to the TSP. The paper is organized as follows: Section 2 describes ACS algorithm for the scheduling problem. Section 3 describes variations of ACS algorithm. Section 4 explains the architecture of workflow management in grids. In Section 5, the application of ACO such as antnets is explained. Section 6 makes the conclusion.

II. ACS Algorithm For The Scheduling Problem

The elementary idea of ACO is to simulate the foraging behaviour of ant colonies. When a group of ants sets out from the nest to search for the food source, they use a special kind of chemical to communicate with each other. The chemical is referred to as pheromone. Once the ants discover a path to food, they deposit pheromone on the path. By sensing pheromone on the ground, an ant can follow the trails of the other ants to the food source. As this process continues, most of the ants tend to choose the shortest path as there have been a huge amount of pheromones accumulated on this path. This collective pheromone-depositing and pheromone-following behaviour of ants becomes the inspiring source of ACO. In this paper, we apply the ACS algorithm [31], which is one of the best ACO algorithms so far [32], to tackle the workflow scheduling problem in grid applications. Informally, the algorithm can be viewed as the interplay of the following procedures.

1. Initialization of algorithm

All pheromone values and parameters are initialized at the beginning of the algorithm.

2. Initialization of ants

A group of M artificial ants are used in the algorithm. In each iteration, each ant randomly selects a constructive direction and builds a sequence of tasks.

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{i,j}^{\alpha}(t).\eta_{i,j}^{\beta}(t)}{\sum_{s\in allowed\ k}\tau_{i,s}^{\alpha}(t).\eta_{i,j}^{\beta}(t)} & j \in allowed_k \\ 0 & Otherwise \end{cases}$$

Where $\tau_{i,j}^{\alpha}$ is the pheromone concentration of edge

e(i,j), $\eta_{i,j}^{\beta}$ is the heuristic information to move an

ant from node i to node j . The $\eta_{i,j}^{\beta}$ is formulated as follows

$$\eta_{i,j}^{\beta} = \xi_{bandwith\ \ h} + \frac{(1-f_1.packet\_loss_{ij})}{f_d.delay_{ij}+f_c.cos(t)_{ij}}$$

3. Solution construction

M ants set out to build *M* solutions to the problem based on pheromone and heuristic values using the selection rule of the ACS algorithm.

4. Local pheromone updating

Soon after an ant maps a service instance sji to task *Ti* , the corresponding pheromone value is updated by a local pheromone updating rule.

$$\tau(i,j) = (1-\rho_l).\tau(i,j) + \rho_l.\Delta\tau(i,j)$$

Where ρ_l $(0< \rho_l <1)$ is the local pheromone decay parameter.
$\Delta\tau(i,j)$ = log(bandwidth$_{ij,}$ bandwidth_max) + 1/delay$_{ij}$ +1/cost$_{i,j}$

5. Global pheromone updating

After all ants have completed their solutions at the end of each iteration, pheromone values corresponding to the best-so-far solution are updated by a global pheromone updating rule. After all ants have finished once path selection, only the globally best path which objective function value is smallest is selected to update global pheromone. Thus, we may reserve the current optimal path. The pheromone level is updated by using the global updating rule of formula

$$\tau(i,j) == (1-\rho_l).\tau(i,j) + \rho_l.\Delta\tau(i,j)$$

$$\Delta\tau(i,j) = \begin{cases} \dfrac{Q}{f}(i,j)\epsilon\ p \\ O(i,j) \notin p \end{cases}$$

Where ρ_g $(0 < \rho_g < 1)$ *is the global pheromone volatile factor,* *Q is constant which is used to control the amount of pheromone value deposited by the globally function of path constructed by the globally best ant.*

6. Terminal test

If the test is passed, the algorithm will be ended. Otherwise, go to step 2) to begin a new iteration.

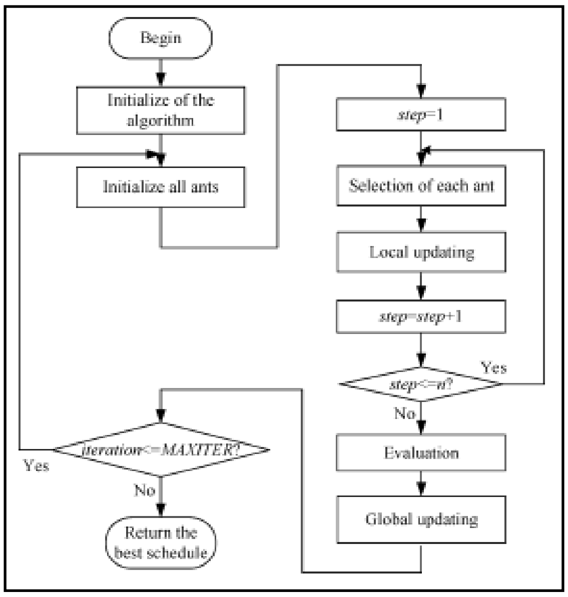

Fig. 1: The flowchart of the algorithm. These procedures are described in detail shortly.

III. Variations

In this paper three variations of the ACO algorithms that is ACO1,ACO2 and ACO3 has been suggested and applied over different network models as discussed below:

a) ACO1: In the first method, no ants(packets)[4] are allowed to visit a node that is already visited by it in its journey that is no ants(packets)[are allowed to make a loop in its path. While selecting the next node, it is checking that the node has already been visited by it or not. If the node is an already visited one, then just discard the node and check for the other available nodes. If a packet reaches in a state such that it has no other way except to select an already visited node, then stop the packet and mark the packet as unsuccessful.

Algorithm:

```
Procedure ACO1(source, destination)
{
assign initial positions to a set of ants(packets) at the
given source;
while(source != destination)
 {
call find next node(source);
set new source = selected node;
source = new source;
}
update the pheromone table using the paths selected by the
successful packets;
send a next set of packets(ants) guided by the information left by
all the previous visitor ants(packets);
}
Procedure find next node(source)
 {
check the routing table entries corresponding to the source if a
link exists with the source
 {
if (the node is already visited)
 {
cancel the node;
else if (the node not already visited)
```

```
 {
mark the node as eligible of being selection
}
}
using the pheromone information, select a node
from the list of eligible nodes;
return the selected node;
}
```

b) ACO2: In the second method, the packets are allowed to make loops in their paths and visit an already visited node provided that it will not visit only the last node visited by it. However, this method may give rise to a situation where an ant(packet)may fall in an infinite loop. To avoid this situation after a certain interval of time if an ant(packet)has not reached the destination, the packet is marked as unsuccessful and its journey is forcefully stopped. In this method, after the completion of the journey of the successful packets, the paths of the packets are checked and made free of any of the loops that may have formed during the course of its journey, and then the pheromone deposit of the nodes of the loop free paths are updated.

Algorithm:

```
Procedure ACO2(source, destination)
{
assign initial positions to a set of ants(packets) at the
given source;
while(source != destination && no_of_hops <=max hops)
{
call find next node(source);
set new source = selected node;

source = new source;
}
Remove all the cycles that may have formed in the path of the
packets(ants);
Update the pheromone table with the cycle free paths selected by
the successful packets;
Send a next set of ants(packets) followed by the information left
by all the previous visited ants(packets);
}
Procedure find next node(source)
 {
check the routing table entries corresponding to the source if (a
link exists with the source)
{
if (the node is the last visited node)
{
cancel the node;
else if ( not the last visited node),)
{
mark the node as eligible of being selection
}
}
using the pheromone information, select a node
from the list of eligible nodes;
return the selected node
}
```

c) ACO3: When the network model is large enough, the restriction of not visiting the last visited node only can be little modified with the restriction that the ants(packets)will not visit the last n number of nodes already visited by it. This modification is done to reduce the chances of forming loops in the path of the journey and hence reducing the overhead of removing the cycles from the path of the successful packets after the completion of its journey. Here also the journey of the ants(packets)are suspended after a certain interval of time and the packets failed to reach the destination are marked as unsuccessful. In this method , a Tabu list is maintained to keep the list of the last n number of nodes visited by a packet so that the last n number of nodes are not selected while selecting the next node for going to the destination. Here experiments are carried out for large networks by varying the size of this Tabu list to find the optimum size of the Tabu list with the given number of nodes in the network model.

Algorithm:

```
Procedure ACO3(source, destination)
{
assign initial positions to a set of ants(packets) at the
given source;
while(source != destination && no_of_hops <=max hops)
{
call find next node(source);
set new source = selected node;
source = new source;
}
Remove all the cycles that may have formed in the path of the
packets(ants);
Update the pheromone table with the cycle free paths selected by
the successful packets;
Send a next set of ants(packets) followed by the information left
by all the previous visited ants(packets);
}
Procedure find next node(source)
{
check the routing table entries corresponding to the source if (a
link exists with the source)
{
if (the node is the last visited node)
{
cancel the node;
else if (the node is among the last n visited nodes)
{
mark the node as eligible of being selection
}
}
using the pheromone information, select a node
from the list of eligible nodes;
return the selected node.
```

IV. Grid Computing

Grid technology is defined as the technology that enables resource virtualization, on demand provisioning and resource sharing between organizations.

Grid computing appears to be a promising trend for three reasons:

1. Its ability to make more cost-effective use of a given amount of computer resources.
2. A way to solve problems that cannot be approached without an enormous amount of computing power.
3. It suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as collaboration towards a common objective.

Architecture For Workflow Management i n Grids

Many computation-intensive applications in science and business can be described as workflows. According to [34], we can image the process of work flow applications as a four-level model (Fig. 1). A user may first submit a workflow application in abstract language in the *abstract specification level*. Then, the grid system has to select and configure the application components of the application to form an *abstract workflow*. The abstract workflow specifies the execution order of tasks. As already mentioned, under OSGA [3], tasks in a workflow are implemented by Web service instances provided by GSPs. The implementation of a task may be supported by various service instances provided by different GSPs with different QoS parameters, but only a single service instance is chosen for the execution of the task by the scheduler. Therefore, the third level is on a mission to map the tasks in the abstract workflow to service instances so that a *concrete workflow* is generated. For example, in Fig. 1, every task *Ti* in the abstract workflow is corresponding to a set of service instances

Si $=\{s_i^1, s_i^2, . . . , s_i^{m_i}\}$. $s_i^1, s_i^2, . . . , s_i^{m_i}$ are marked by circles in rectangle *Si* and *mi* is the total number of service instances for *Ti* . The service instances marked by gray circles are finally selected to map to corresponding tasks to form a concrete workflow in this example. At last, in the *implementation level*, different GSPs may use different structures, which are transparent to users, to execute the same task with different QoS parameters. In this four-level model, mapping the abstract workflow into the concrete workflow with QoS considerations is the most important step in grid applications [34]. In general, this task is completed by a workflow management system (WfMS). According to the market-driven structure of global computational girds [20], [25], the architecture of WfMS can be illustrated by Fig. 2.We can view the workflow management cycle in WfMS as the interplay of the following nine steps.

Step 1) The service instances provided by GSPs are registered to grid market directory (GMD). In a grid market, GMD [35] is used to manage the information, including the type, the provider, and the QoS parameters, of all service instances.
Once an organization tends to promote services to the grid market, it first registers itself to GMD as a GSP. Then the information of new service instances is also registered.

Step 2) Users define and submit the abstract workflow toWfMS. QoS requirements of the workflow application are also submitted in this stage.

Step 3) As soon as WfMS accepts a workflow application, it asks GMD for an information list of the corresponding service instances. Typically, it collects the type, provider, access directory, and QoS parameters of each related service instance.

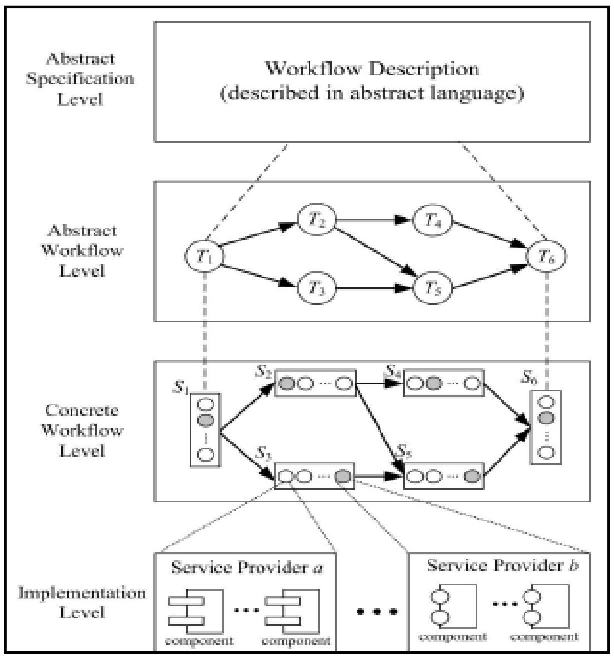

Fig. 2: Process of executing a workflow application in grids under OGSA: a four-level model.

Step 4) WfMS enquires of each related GSP whether the service instances will be available. Only available services instances will be adopted.

Step 5) WfMS executes a scheduling algorithm to map the abstract workflow into a concrete workflow. Every task in the abstract workflow is mapped to an available service instance by the scheduler. The goal of scheduling is to achieve the users' QoS requirements and preferences.

In Fig. 2, we can see that the scheduler is composed of two modules. The scheduling module maintains a scheduling algorithm to generate optimal schedules, while the QoS parameter recorder module records the run-time performance (such as reliability and availability of each service instance and
GSP). These historical records will be used to predict and adjust the QoS parameters of service instances for future scheduling tasks.

Step 6) WfMS accesses the related GSPs to make reservations for all the service instances according to the solution schedule generated by

Step 7) The concrete workflow is executed according to the solution schedule.

Step 8) The contract between users and GSPs may be violated at times due to many reasons such as the failure of GSPs' resources. In this case, a service instance may be delayed or even become unavailable, which makes the following service instances fail to follow the schedule. Therefore, a rescheduling mechanism is required for violation management.

Step 9) After the completion of each service instances, the actual QoS is fed back to the QoS parameter recorder module so that historical QoS statistics can be updated. After reaching their destination, ants retrace their path and update nodes

routing information according to the quality of the path. Routing information is statistical estimates of the time-to-go to the destination maintained in pheromone
arrays.

V. Antnet

In this paper we perform a survey on modified AntNet routing algorithm using Multiple Ant- Colony Optimization. Multiple ant colonies with different pheromone updating mechanism have different searching traits. By leveraging this feature, much of work is done by designing a set of adaptive rules to facilitate the collaboration between these colonies. This approach can balance the diversity and convergence of solutions generated by different ant colonies and also overcome the problem of Stagnation.

AntNet is a result of the application of ACO on the problem of Internet routing. They communicate indirectly by information they leave behind in the routers on their path. Over time, this information leads to optimal routing paths between the routers in the network. while exploring the network ants update the probabilistic routing tables and construct a statistical model of the nodes local traffic.

The algorithm uses two types of ants namely,forward ants and backward ants to collect network statistics and to update the routing table.

In each node there are two types of queues, low priority and high priority. The data packets and the forward ants use low priority queues, whereas the backward ants use the high priority queues.

1. Forward Ants who gather information about the state of the network, and

2. Backward Ants who use the collected information to adapt the routing tables of routers on their path. The task of the Forward Ants is collecting information about the state of the network. The Forward Ant is transformed into a Backward Ant. This Backward Ant will follow exactly the same path as the Forward Ant but in the opposite direction. Stagnation occurs when a network reaches its convergence(or equilibrium state); an optimal path is chosen by all ants and this recursively increases an ant's preference for selected path.

VI. Fault Tolerance In Grid Using Ant Colony Optimization

By day to day developing the grid systems, it is necessary to apply new methods for allocating the resources to achieving the high performance in heterogeneous computing environment. This paper aims to seeking for a new approach by which one can allocate the tasks using a modified version of Ant Colony Optimization algorithm such that this algorithm cannot be involved in local minimum. Tasks will be entered to the system by Directed Acyclic Graph (DAG). This method tried to allocate the tasks to the processors in short time such that the tolerability of system may be considerably increased against faults.

Any allocation stage comprised from two different phases and for any phase there has been provided new heuristic function.

VII . Proposed Method

In the algorithm provided in this paper, after preparing and making the graph together with preparing the initial parameters, the movement of ants will be started in different iterations. Any transferring movements of ants have been comprised from 2 stages.

1.Choosing the next task that must be executed.

2.Finding the best processor for the task chosen in the previous stage.

After any ant allocated all tasks existing in the graph to the processors, there will be conducted a local pheromone updating operation along the route. Then, at the end of any repetition where all ants passed whole route, once the global pheromone update will be executed on the best route found among whole routes passed by ants. At the end of all iterations, finally the optimal route will be selected according to the best evaluation function as well as reaching more fault toleration.

1. Different states for a task

A task in the proposed alternative may have 5 different states: not scheduled, schedulable,
scheduled, running, finished.

2. Choosing the task after the current one

With probability function of (4) any ant will be transferred from task_ to task__ and selects it.

$$P(t_i,t_j) = \frac{[\tau(t_i,t_j)]^{\alpha_1}.[\eta(t_i,t_j)]^{\beta_1}}{\sum_{q\in N_i^k}[(t_i,t_j)]^{\alpha_1}.[\eta(t_i,t_j)]^{\beta_1}}$$

$$* Uniform\,[0\ldots1]$$

$N_i^k$is the set of tasks that is from the current taskt_i in the ant K in the schedulable state. This means that all their parents have been put in the finished state. In this function, $\tau(t_i,t_j)$ is considered equal with the rate of pheromone present between two taskst_i and t_j and consider the $\eta(t_i,t_j)$ equal with the size of Blevel(tj). In any DAG, Bottom-level (Blevel) of node $t_j$equals with the weight of longest route of node t_j to exit node. This weight includes computation time of node as well as communication time on the edges of graph[17]. The reason for choosing Blevel (t_j) as heuristic function is that the more is this value, it means this node is located in an improper state based on schedule and must be chosen earlier. The value of Blevel (t_j) is calculated retrospectively and in time O(t+e) where, t is the number of tasks and e is the number of graph edges. As indicated in equation (1), the size of probable function after calculation is multiplied in a uniform distribution between zero and one. This ensures that for choosing the next tasks in comparison with usual ant colony algorithm, we may not get involved in the local minimum.

Because the size considered for β1 depends on the number of iterations, consider β1=In (Iteration
Number).

3. Choosing the processor for selected task

After choosing t_j, ant must find the best existing processor based on following probable function for scheduling to this task. The name of this processor is p_j.

$$P(t_i,t_j) = \frac{[\tau(t_i,t_j)]^{\alpha_2}.[\eta(t_i,t_j)]^{\beta_2}}{\sum_{q\in N_i^k}[(t_i,t_j)]^{\alpha_2}.[\eta(t_i,t_j)]^{\beta_2}} * Uniform\,[0\ldots1]$$

In this function, $\tau(t_i,p_j)$is the size of pheromone between task

t_j and processor p_j. It must be
mentioned that initial pheromone between task-task as well as between task-processor can be
considered differently. For heuristic function between taskt_j and processorp_j we have:

$$\eta(t_i, p_j) = \frac{1}{FT(t_j, p_j) - \text{sent}\,T(t_j)}$$

Where sentT $(t_j)$is equal with the time when immediately after being selected the taskt_j will be
sent to the processor for schedule. The value of FT(t_j, p_j)is when the running of task tj has been finished on processor $p_j$and will be calculated according to equation (7).
FT(t_j, p_j)=Start Time(p_j) + ETC(t_j, p_j)

Because, the time when there is schedulable taskt_j,might be after the time when the processor $p_j$has been released (ready time), therefore the value of Start Time(pj) will be calculated by following equation.

Start Time (p_j)= *Max*(Ready Time(p_j), Sent T(t_j)) +*Max*(*Parents Comm Cost* (t_j), t_j)
Where the value of Start Time (p_j) will be finally summed by the maximum value of communication time between parent nodes t_j and the node itself by which the time for transferring the data among the tasks can also be applied in the calculation. The same as equation(4), the value of probable function task-processor can also be multiplied in the uniform distribution.

VIII. Cost Driven

Many researchers believe that economic principles will influence the grid computing paradigm to become an open market of distributed services, sold at different prices, with different performance and QoS [1]. This new paradigm is known as *utility grid*, versus the traditional community grid in which services are provided free of charge with best effort service. Although there are many papers that address the problem of scheduling in traditional grids, there are only a few works on this problem in utility grids. The multi objective nature of the scheduling problem in utility grids makes it difficult to solve, specially in the case of complex jobs like workflows. This has led most researchers to use time consuming meta-heuristic approaches, instead of fast heuristic methods. In this paper we propose a new heuristic algorithm for scheduling workflows in utility grids, and we evaluate its performance on some well-known scientific workflows in the grid context.
The main difference between community grids and utility grids is QoS: while community grids follow the best-effort method in providing services, utility grids guarantee the required QoS of users via Service Level Agreements (SLAs) [2]. An SLA is a contract between the provider of resources
and the consumer of those resources describing the qualities and the guarantees of the service provisioning. Consumers can negotiate with providers on required QoS and the price to reach an SLA. The price has a key role in this contract: it encourages providers to advertise their services to the market, and encourages consumers to define their required qualities more realistically. Obviously, traditional resource management systems for community grids are not directly suitable for utility grids, and therefore, new methods have been proposed and implemented in recent years [3].propose a new QoS-based workflow scheduling algorithm, called the *Partial Critical Paths* (PCP) algorithm. The objective function of the PCP algorithm is to create a schedule that minimizes the total execution cost of a workflow, while satisfying a user-defined deadline for the total execution time. First, the PCP algorithm tries to schedule the (overall) critical path of the workflow such that it completes before the user's deadline and the execution cost is minimized. Then it finds the *partial critical path* to each scheduled task on the critical path and executes the same procedure in a recursive manner .Critical Path heuristics are widely used in workflow scheduling. The *critical path* of a workflow is the longest execution path between the entry and exit tasks of the workflow. Most of these heuristics try to schedule *critical tasks* (*node*), i.e., the tasks belonging to the critical path, first by assigning them to the resources that process them earliest, in order to minimize the execution time of the entire workflow. Our proposed algorithm is based on a similar heuristic, to schedule the critical nodes first, yet not to minimize the execution time, but to minimize the price of executing the critical path before the user-specified deadline. After scheduling all critical nodes ,each of them has a start time that is a deadline for its parent nodes, i.e., its (direct) predecessors in the workflow. So then we can carry out the same procedure by considering each critical node in turn as an exit node with its start time as a deadline, and creating a *partial critical path* that ends in the critical node and that leads back to an already scheduled node. In our *Partial Critical Path* (PCP) algorithm, this procedure continues recursively until all tasks are scheduled successfully. In the following sections, we elaborate on the details of the PCP algorithm.

A. Basic Definitions

In our PCP scheduling algorithm, we want to find the critical path of the whole workflow, and partial critical paths. In order to find these, we need some (idealized, approximate) notion of the start time of each workflow task before we actually schedule the task. This means that we have two notions of the start times of tasks, the earliest start time computed before scheduling the workflow, and the actual start time computed by our scheduling algorithm. For each unscheduled task *ti* we define its Earliest Start Time *EST* (*ti*) as the earliest time *ti* can start its computation, regardless of the actual service that will process the task(that will be determined during scheduling). Clearly, it is not possible to compute *EST* (*ti*) exactly, because a grid is
Algorithm 1 The PCP Scheduling Algorithm

```
1: procedure SCHEDULEWORKFLOW(G(T,E), deadline)
2: request available services for each task in G from GMD
3: query available time slots for each service from related GSPs

4: add $t_{entry}$, $t_{exit}$ and their corresponding edges to G
5: compute $MET(ti)$ for each task according to formula 1
6: compute $MTT(ei,j)$ for each edge according to formula 2
7: compute $EST(ti)$ for each task in G according to formula 3

8: mark $t_{entry}$ and $t_{exit}$ as scheduled

9: set $AST(t_{entry}) \leftarrow 0, AST(t_{exit}) \leftarrow deadline$
10: if (Schedule Parents($t_{exit}$) is successfull) then
11: make advance reservations for all tasks in G according to Schedule
12: else
13: return (failure)
14: end if
```

15: end procedure
a heterogeneous environment and the computation time of tasks varies from service to service.

Furthermore, the data transmission time is also dependent on the selected services and the bandwidth between their providers. Thus, we have to approximate the execution and data transmission time for each unscheduled task. Among the possible approximation options, e.g., the average, the median, or the minimum, we select the minimum execution and data transmission time. We define the Minimum Execution Time *MET* (*ti*) and the Minimum Transmission Time *MTT* (*ei,j*) as follows:

$$MET(t_i) = \min_{s \in S_i} ET(t_i, s)$$

$$MTT(e_{i,j}) = \min_{s \in S_i, r \in S_j} TT(e_{i,j}, s, r)$$

Having these definitions, we can compute the Earliest Start Time for each unscheduled task, *EST* (*ti*), as follows:

$EST(t_{entry}) = 0$

$$EST(t_i) = \max_{t_p \in parents\ t_i} EST(t_p) + MET(t_p) + MTT(e_{p,i})$$

For each scheduled task we define the Selected Service *SS* (*ti*) as the service selected for processing *ti* during scheduling, and the Actual Start Time *AST* (*ti*) as the actual start time of *ti* on that service. These attributes will be determined during scheduling.

B. The PCP Scheduling Algorithm

Algorithm 1 shows the pseudo-code of the overall PCP algorithm for scheduling a workflow. In line 4, two dummy nodes t_{entry} and t_{exit} have been added to the task graph, even if the task graph already has only one entry or exit node. This
is necessary for our algorithm, but we won't actually schedule these two tasks. After computing the required parameters in lines 5 - 7, dummy nodes t_{entry} and t_{exit} are marked as scheduled in line 8, and then the Actual Start Time of t_{entry} is set to the user's deadline. This enforces the parents of t_{exit} ,i.e., the actual exit nodes of the workflow, to be finished before the deadline. Finally, in line 10 the function Schedule Parents is called for t_{exit} . In order to show how the algorithm works, we will trace its operation on the sample graph shown in Figure
1 (the numbers simply indicate node numbers, not execution times). In this figure, t_{entry} and t_{exit} have been shown with *S* and *E*, respectively. In this stage, Schedule Workflow calls Schedule Parents for node *E*

IX. FUTURE WORK

1: procedure
SCHEDULEWORKFLOW (*G*(*T,E*)*, deadline*)
2: request available services for each task in G from GMD
3: query available time slots for each service from related GSPs

4: add t_{entry} , t_{exit} and their corresponding edges to G
5: compute *Cost constraints:*
Given a schedule *K*(*K*1, . . . , *Kn*), the total cost of *K* (*K.cost*) must not be larger than a user defined variable *budget*, i.e.,
K.Cost = $\sum_{i=1}^{n} s_i^{k_i} .c \leq Budget$.
6: compute *Heuristic C: Cost Greedy (CG):* The CG heuristic biases the artificial ants to select the service instances with lower cost. Suppose that an ant's heuristic type is the CG heuristic, then the heuristic value of mapping *sji* to *Ti* (denoted as CG*ij*) is set to

$$\eta ij = CGij = \frac{\max\_cost_i - s_i^j.t + 1}{\max\_cost_i - \min_{cost_i} + 1}$$

where *min costi* = min1≤*j*≤*mi{sji .t}* and *max costi* = max1≤*j*≤*mi{sji .t}*.A service instance with lower cost will be associated with a higher heuristic value and *ηij* ∈ (0*,* 1].

7: compute *Cost* for each task in G

8: mark t_{entry} *and* t_{exit} as scheduled

9: set *AST*(t_{entry})← 0,*AST*(t_{exit}) ← *deadline*
10: if (ScheduleParents(*texit*) is successfull) then
11: make advance reservations for all tasks in G according to Schedule
12: else
13: return (failure)
14: end if
15: end procedure

X. Multistage Graphs:

Applying the Suggested Deadline Heuristic of ACO algorithm Involves computing EST and BEST for a particular task
Multistage Graphs would be useful in finding the shortest route between source and destination which is also a good tradeoff in terms of cost associated with that route.
Solution can be simulated using both FORWARD COSTS and BACKWARD COSTS.

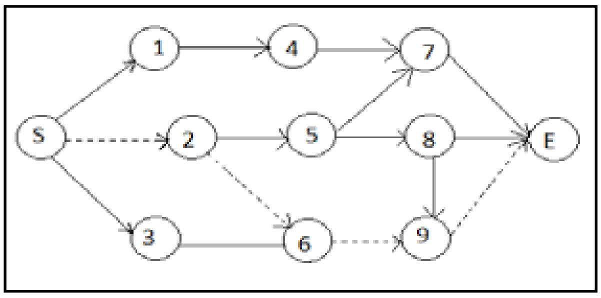

Fig. 3: A sample workflow

XI. Conclusions

An ACS algorithm for a large-scale workflow scheduling problem in computational grids has been proposed. The scheduling algorithm is designed for workflow applications in market driven or economy-driven grids under the service-oriented architecture. In the algorithm, different QoS parameters are considered, including reliability, time, and cost. Users are allowed to define QoS constraints to guarantee the quality of the schedule. Moreover, the optimizing objective of the algorithm is based on the user-defined QoS preferences. Three variants of ACO for Network Routing is proposed and implemented on various standard Network Models. The results show that the type of the variations of the ACO that should be applied on the network, obviously depends on the structure, size and the application. According to the problem of resource allocation in the grid, obtained results indicate that proposed method is a proper tool for solving this problem. We also applied the communication time among tasks on the processors on the edges of graphs. In this problem, we applied a changed version of ant colony optimization for solving the problem. Any ant, when resource allocation to the related processor, has been composed from two different phases for selecting the next task as well as finding the best processor for that. This paper is a study to overcome the problem of Stagnation and congestion by using Multiple Ant-Colony Optimization. In the improved version, of ACO, Multiple Ant-Colony Optimization can find more than one optimal outgoing interfaces are identified as compared to only one path, which are supposed to provide higher throughput and will be able to explore new and better paths even if the network topologies gets changed very frequently.

References

[1] *Ant Colony Algorithm for Job Scheduling in Grid Computing June 12, 2012 deb jyoti paul P.Mathiyalagan et al. / (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 02, 2010, 132-139*

[2] *Modified Ant Colony Algorithm for Grid Scheduling Mr. P.Mathiyalagan 1, S.Suriya2, Dr.S.N.Sivanandam3*
1. Lecturer , Department of Computer Science and Engineering, PSG College of Technology, Coimbatore.
2. Post Graduate Student Department of Computer Science and Engineering, PSG College of Technology, Coimbatore.
3. Professor and Head Department of Computer Science and Engineering,PSG College of Technology, Coimbatore

[3] *International Journal of Grid Computing & Applications (IJGCA) Vol.2, No.1, March 2011 DOI: 10.5121/ ijgca.2011.2102 14 Fault Tolerance In Grid Using Ant Colony Optimization And Directed Acyclic Graph VahidModiri1Morteza Analoui2 and Sam Jabbehdari3 Cost-driven Scheduling of Grid Workflows Using Partial Critical Paths Saeid Abrishami, Mahmoud Naghibzadeh Ferdowsi University of Mashhad Mashhad, Iran {s-abrishami, Dick Epema Delft University of Technology Delft, The Netherlands d.h.j.*

[4] *International Journal of Innovative Computing, Information and Control ICIC International c 2009 ISSN 1349-4198 Volume 5, Number 12, December 2009 pp. 1–ISII08–247 ENHANCED ARTIFICIAL BEE COLONY OPTIMIZATION Pei-Wei TSai1, Jeng-Shyang Pan1, Bin-Yih Liao1, and Shu-Chuan Chu2*

[5] *Department of Electronic Engineering, National Kaohsiung University of Applied Sciences 415 Chien Kung Road, Kaohsiung City 80778, Taiwan pwtsai@bit.kuas.edu.tw, {jspan; byliao}@cc.kuas.edu.tw 2Department of Information Management, Cheng Shiu University 840 Cheng Cing Road, Kaohsiung County 83347, Taiwan, R.O.C. Received December 2008; revised August 2009*

[6] *T. VetriSelvan, P. Chitra, P. Venkatesh, "Parallel Implementation of Task Scheduling using ant colony Optimization", International Journal of Recent Trends in Engineering, Vol. 1, No. 1, pp. 339- 343, 2009*

[7] *J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," J. Grid Comput., vol. 6, pp. 255–276, 2008*