ISSN:2231-0711
www.ijcset.net

Improved Ant Colony Optimization for Grid Scheduling

D.Maruthanayagam
MCA Department, Gnanamani Collge of Technology, Namakkal, Tamilnadu, India

Dr. R.Uma Rani
Department of Computer Science, Sri Sarada College for Women, Salem,Tamilnadu, India

ABSTRACT:
This paper focuses on applying one of the rapidly growing non-deterministic optimization algorithms, the ant colony algorithm in Grid computing. It is growing rapidly in the distributed heterogeneous systems for utilizing and sharing large-scale resources to solve complex scientific problems. Scheduling is the most recent topic used to achieve high performance in grid environments with several conflicting objectives. Within this paper, methods have been developed and applied for scheduling techniques in grid computing. It aims to find a suitable allocation of resources for each job with the comparison of ACO and proposed ACO. This paper, proposes an improved ant colony scheduling algorithm combined with the concept of RASA. The first step for this to select a set of computers and a network connection (switching, routers, Ethernet, Myrinet Etc.,) for an application. A task algorithm from RASA first estimates the completion time of the tasks on each of the available grid resources and then applies the Max-min and Min-min algorithms. Allocation of resources to a large number of jobs in a grid computing environment presents more difficulty than in network computational environments. This proposed algorithm is evaluated using the simulated execution times for a grid environment. Before starting the grid scheduling, the expected execution time for each task on each machine must be estimated and represented by an Expected Time calculation. The proposed scheduler allocates adopt the system environment freely at runtime. This resource optimally and adaptively in the scalable, dynamic and distribute controlled environment. Conclude of this propose depending upon the performance of the grid systems.

Key words: ***Grid Computing, Job Scheduling, Heuristic Algorithm, opportunistic Load Balancing, genetic Algorithms, Simulated Annealing algorithms and Max-Min Ant system.***

I. INTRODUCTION

In the past few years nature-inspired techniques have been widely used for various optimization problems in design, planning, scheduling, communication, etc. One field, which is receiving increasing interest from several researchers, is the scheduling problem in Grid Computing Environment. A variety of heuristic algorithms have been designed to schedule the jobs in computational Grid. The most commonly used algorithms are OLB, MET, MCT, Min-Min and Max-Min. Reduction of makespan (measure the throughput of the grid system) is the prime aim of grid scheduling. The ACO becomes very popular *heuristic* algorithm to apply for solving grid scheduling problems. Ant colony algorithm are increasingly being used in various real-world applications such as the travelling salesman problem (TSP), the quadratic assignment problem (QAP), the Job Shop Scheduling Problem (JSP), telecommunication routing and load balancing, etc. and it has been shown that they perform well compared to other non-deterministic algorithms such as genetic algorithms(GA), simulated annealing(SA), etc.

Scheduling is a significant research area in Grid Computing. Many Scheduling algorithms have been designed and applied in computational grid. However, those algorithms are not always able to produce efficient scheduling in heterogeneity computing resources. Often, skilled personnel who understand the requirements, the simulation model representations and solve complex computing scheduling problems efficiently and physical production issues fill this technical gap. Scheduling algorithm is a technique of increase the throughput, Quality of Service (QoS) and reduces the cost during the job scheduling process for dynamically allocate the efficient processors. It needs to be done under multiple objectives and constraints.

Grid Computing enables aggregation and sharing of geographically distributed computational, data and other resources as single, unified resource for solving large scale compute and data intensive computing application. Management of these resources is an important infrastructure in the Grid computing environment. It becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access and cost models, and have dynamically varying loads and availability.

The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like Grid where jobs need to be executed by computing resources, and the requirement of these resources is difficult to predict. Due to highly heterogeneous and complex computing environments, the

chances of faults increases [1], and therefore number of resources available to any given application highly fluctuates over time which reduces the performance and the efficiency. Therefore it is necessary to design a mechanism for scheduling to improve the efficiency in such infrastructure. This work focuses on scheduling in a Grid environment. Grid application performance is critical in Grid computing environment. So to achieve high performance there is a need to understand the factors that can affect the performance of an application due to Scheduling, which is one of most important factors that influence the overall performance of application.

The significance of this research lies in the potential of the developed ant colony optimization algorithms for job scheduling in Grid Computing and the associated cost and time. Further, despite the recent advancements in the field of scheduling algorithms**,** the following issues have not been addressed by other scheduling algorithms

- Opportunistic Load Balancing **(OLB)** keeps almost all machines busy all possible time. Yet the solution is not optimal.
- Minimum Execution Time **(MET)** creates an imbalance condition among the machines. Allocating all the smallest tasks to the same fastest resources. Hence this solution is static.
- Minimum Completion Time **(MCT)** takes calculation of minimum completion time for a job is longer.
- The drawback of **Min-Min** is that, too many jobs are assigned to a single grid node. This leads to overloading among jobs.
- When compare to **Max-Min**, Min-Min is the best one because implementation of Max-Min is difficult.

Most of the algorithms have some drawbacks compare than ant colony optimization. All of the algorithms use a classical multi-objective technique: weighted sum approach or single-objective function to obtain the optimum solution. Since there is more than one optimum solution for a multi-objective problem, previous algorithms are not capable of generating these conflicting optimum solutions (or trade-off solutions) in a single run. If all the trade-off solutions need to be found, these algorithms must be run several times with different parameter values. Normally these experiments are time consuming and if they need to be run several times, they are even more time consuming. Moreover, ant colony optimization along with Resource Aware Scheduling Algorithm (RASA) is another important part of this research area and none of the algorithms are able to provide reduce the makespan of jobs as well as to find out the optimal resource in Grid Computing. In this paper, the above issues are addressed. In addition to these issues, the paper suggests some modifications to ant colony optimization and a new Enhanced Ant Colony System based on RASA in Grid Scheduling for heterogeneous computing environment.

The proposed algorithm can be improved using some form of operating systems, hardware, and software, different storage capacities, CPU speeds, network connectivity's and technologies needs. In this method we first find the problem resources and those total execution times equal to the makespan of the solution, and attempt to move or swap set of jobs from the problem processor to another resource that has the minimum and maximize of makespan as compared with all other resources.

As RASA consist of the max-min and min-min algorithms and both have no time consuming instructions. ACO and RASA algorithms incorporate in which intend to optimize workflow execution times on grids have been presented here. The comparison of these algorithms in computing time, applications and resources scenarios has also been detailed. In dynamic grid environments this information that can be retrieved from a many servers includes operating system, processor type and speed, the number of available CPUs and software availability as well as their installation locations. The distributed monitoring system is designed to track and forecast resource conditions. The n tasks can obviously intercommunicate. A general model should take into consideration that the communication phase can happen at any time with I/O phases. To overcome these difficulties our new algorithm is proposed.

II. MATERIALS & METHODS

A. Local vs. Global

The local Scheduling discipline determines how the processes resident on a single CPU are allocated and executed; a global Scheduling policy uses information about the system to allocate processes to multiple processors to optimize a system wide performance objective. Grid Scheduling falls into the Global Scheduling [4].

B. Static vs. Dynamic

The next level in the hierarchy (under the Global Scheduling) is a choice between static and dynamic Scheduling. This choice indicates the time at which the Scheduling decisions are made. In case of static Scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled. By contrast, in the case of dynamic Scheduling, the basic idea is to perform task allocation on the fly as the application executes.

C. Optimal vs. Suboptimal

In the case that all information regarding the state of resources and the jobs is known, an optimal assignment could be made based on some criterion function, such as minimum makespan and maximum resource utilization. But due to difficulty in Grid scenarios to make reasonable assumptions which are usually required to prove the optimality of an algorithm, current research tries to find suboptimal solutions, which can be further divided into the following two general categories.

D. Approximate vs. Heuristic

The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. The factors, which govern their decision, are:

- Availability of a function to evaluate a solution.
- The time required evaluating a solution.
- The ability to judge the value of an optimal solution according to some metric.
- Availability of a mechanism for intelligently pruning the solution space.

Heuristic represents the class of algorithms**,** which make the most realistic assumptions about a priori knowledge concerning process and system loading characteristics. It represents the solutions to the Scheduling problem, which cannot give optimal answers and require the most reasonable amount of cost and other system resources to perform their function. The evaluation of this kind of solution is usually based on experiments in the real world or on simulation**.** Heuristic algorithms are more adaptive to the Grid scenarios [5].

E. Cooperative vs. Non-cooperative

If a distributed Scheduling algorithm is adopted, the next issue that should be considered is whether the nodes involved in job Scheduling are working cooperatively or non-cooperatively. In the non-cooperative case, individual schedulers act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of *system e.g.* application-level schedulers**.** In the cooperative case, each Grid Scheduler has the responsibility to carry out its own portion of the Scheduling task, but all schedulers are working toward a common system-wide goal.

F. Distributed vs. Centralized

In dynamic Scheduling scenarios, the responsibility for making global Scheduling decisions may lie with one centralized Scheduler, or be shared by multiple distributed schedulers. The centralized strategy has the advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck [8].

G. Heuristic Scheduling Algorithms

Many algorithms were designed for the scheduling of Meta tasks in computational grids is reviewed in this section. One of the easiest techniques in grid scheduling is Opportunistic Load Balancing (OLB). It workflow tasks in Grid environments are difficult because resource availability often changes during workflow execution. Opportunistic Load Balancing attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources. A typical distributed system will have a number of interconnected resources who can work independently or in cooperation with each other. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed.

Heuristic Task Scheduling Algorithm in Grid computing environment based upon the predictive execution time of tasks. It obtains a scheduling strategy by employing mean load as heuristic information and then selects both the maximum-load and the minimum-load machines. We reassign tasks between two machines to raise the load of the machine with lower-load and reduce that of the machine with higher-load under the mean load heuristic [6].

Ant Colony was used to solve many problems such as traveling salesman problem, vehicle routing problem, graph coloring problem, etc. using ACO in Grid processor scheduling problem leads to finding an optimal or near optimal solution after reasonable amount of time. A new heuristic function is introduced to lead ants to select best processor for executing each process. Also a new fitness function is presented to evaluate the fitness of solutions founded by each iteration's ants. The pheromone updating rule is defined so that prompt new iteration ants to follow the best solutions found in previous iterations.

H. Opportunistic Load Balancing (OLB)

Load balancing for a huge no of system is important problems which have to be solved in order to enable the efficient use of parallel computer systems. This problem can be compared to problems arising in natural work distribution processes like that of scheduling all activities (tasks) needed to construct a large building. The essential objective of a load balancing consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system. The algorithm adopts an observational approach and exploits the idea of scheduling a job to a site that will probably run it faster. The opportunistic algorithm takes into account the dynamic characteristics of Grid environments without the need to probe the remote sites. We compared the performance of the opportunistic algorithm with different scheduling algorithms in a context of a workflow execution running in a real Grid environment. The Opportunistic algorithm benefits from the dynamic aspects of the Grid environment. If a site happens to perform poorly, then the number of jobs assigned to this site decreases. Similarly, if a site process jobs quickly, then more jobs are scheduled to that site**.**

I. Minimum Execution Time (MET)

The first available machine is assigned a job with the smallest execution time. It neither considers the ready time nor the current load of the machine and also the availability of the resources at that instant of time is not taken into account. The resources in grid system have different computing power. Allocating all the smallest tasks to the same fastest resource redundantly creates an imbalance condition among machines. Hence this solution is static. Since the number of resources is much less than

the number of tasks, the tasks need to be scheduled on the resources in a certain order. Many of the batch mode algorithms intend to provide a strategy to order and map these parallel tasks on the resources, in order to complete the execution of these many processor tasks at earliest time. They can also be applied to optimize the execution time of a workflow application which consists of lot of independent parallel tasks with a limited number of resources [3].

J. Minimum Completion Time (MCT)

It uses the ready time of the machine to calculate the job's completion time (ready time of the machine + execution time of the job). It calculates the completion time of current job in the earliest available machines. From the list, the job with smallest completion time is selected and is assigned to that machine. This means the assigned job may have a higher execution time than any other job. This algorithm calculates the completion time of current unfinished job in only one earliest available node**.** But, the same job may be completed in lesser time in some other machine which is available at that time.

K. Min-Min

It starts with a set of unmapped tasks. The minimum completion time of each job in the unmapped set is calculated. This algorithm selects the task that has the overall minimum completion time and assigns it to the corresponding machine. Then the mapped task is removed from the unmapped set [4]. The above process is repeated until all the tasks are mapped. When compared with MCT, Min-Min considers all the unmapped tasks during their mapping decision. The smaller makespan can be obtained when more tasks are assigned to machines that complete them the earliest and also execute them the fastest.

L. Max-Min

First it starts with a set of unmapped tasks. The minimum completion time of each job in the unmapped set is found. This algorithm selects the task that has the overall maximum completion time from the minimum completion time value and assigns it to the corresponding machine. The mapped task is removed from the unmapped set. The above process is repeated until all the tasks are mapped. On comparison with MCT, Max-Min considers all unmapped tasks during their mapping decision. The Max-Min may produce a balanced load across the machine. When compare to Max-Min, Min-Min is the best one.

M. Ant Colony Optimization (ACO)

Ant colony optimization (ACO) was first introduced by Marco Dorigo as his Ph.D. thesis and was used to solve the TSP problem [2]. ACO was inspired by ant's behavior in finding the shortest path between their nests to food source. Many varieties of ants deposit a chemical pheromone trail as they move about their environment, and they are also able to detect and follow pheromone trails that they may encounter. With time, as the amount of pheromone in the shortest path between the nest and food source increases, the number of ants attracted to the shortest path also increases. This cycle continues until most of the ants choose the shortest path. As this work is a cooperative one and none of the ants could find the shortest path separately, Max-Min Ant System is based on the basic ACO algorithm but considers low and upper bound values and limits the pheromone range to be between these values. Defining those values, lets MMAS avoid ants to converge too soon in some ranges. In ACO one ant participate in each iteration search and also there is no pheromone evaporation rule. Hence the ant algorithm is suited for usage in Grid computing task scheduling.

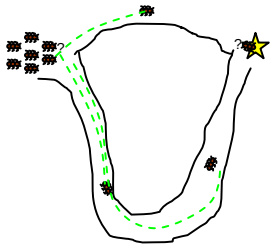

Figure-1: Ants try to move from one place to another.

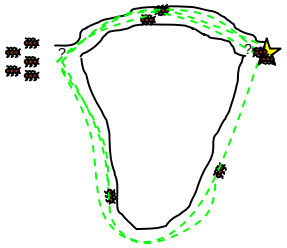

Figure-2: Ants are reinforcing the trail with its own pheromone.

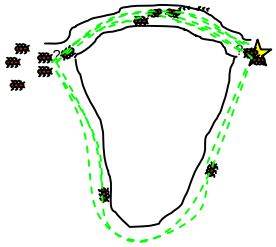

Figure-3: Ants choose the shortest path.

The above figures 1, 2, and 3 are shown the ant's behavior. In the grid environment, the algorithm can carry out a new task scheduling by experience, depending on the result in the previous task scheduling. In the grid computing environment, this type of scheduling is very much helpful. Hence this paper proposes the ant algorithm for task scheduling in Grid Computing.

N. Resource Aware Scheduling Algorithm (RASA)

The algorithm builds a matrix *C* where *Cij* represents the completion time of the task *Ti* on the resource *Rj*. If the number of available resources is odd, the min-min strategy is applied to assign the first task, otherwise the max-min strategy is applied. The remaining tasks are assigned to their appropriate resources by one of the two strategies, alternatively. For instance, if the first task is assigned to a resource by the min-min strategy, the next task will be assigned by the max-min strategy. In the next

round the task assignment begins with a strategy different from the last round. For instance if the first round begins with the max-min strategy, the second round will begin with the min-min strategy. Jobs can be farmed out to idle servers or even idle processors. Many of these resources sit idle especially during off business hours. Policies can be in places that allow jobs to only go to servers that are lightly loaded or have the appropriate amount of memory/processors characteristics for the particular application. In this experimental results show that if the number of available resources is odd it is preferred to apply the min-min strategy the first in the first round otherwise it is better to apply the max-min strategy the first.

Alternative exchange of the min-min and max-min strategies results in consecutive execution of a small and a large task on different resources and hereby, the waiting time of the small tasks in Max-Min algorithm and the waiting time of the large tasks in min-min algorithm are ignored. As RASA consist of the max-min and min-min algorithms and have no time consuming instruction, the time complexity of RASA is $O(mn^2)$ where m is the number of resources and n is the number of tasks (similar to Max-Min and Min-Min algorithms) [1].

III. PROPOSED WORK

The grid scheduler finds out the better resource for a particular job and submits that job to the selected systems. The grid scheduler does not have control over the resources and also on the submitted jobs. Any machine in grid can execute any job, but the execution time differs. The resources are dynamic in nature. As compared with the expected execution time, the actual time may vary when running the jobs in the allocated resources. So, the job placement has been determined according to the scheduling intension and then data move operations have been initiated for necessary task to transfer relevant machines. Processors are claimed after all job components have been placed. In between the job placement time and job claiming time the processors could be allocated to some other job and if this happens the job component can be re-placed on another task.

The time between job placement time and job claiming time is decreased by a fixed amount after every claiming failure. A job can fail for various reasons, e.g., badly configured or faulty nodes, hardware, and software errors. During this scheduling failed, job and counts the number of failures of the supposedly faulty node. When a job fails a previously set number of times then the job is removed and not rescheduled [6]. If the error count of a node exceeds a fixed number then that node is not considered by the co-allocator anymore. The states at the bottom depict the *happy flow*, i.e., the states a job goes through if nothing fails. Different errors occur at various states of a job. Depending on the kind of error, this system will chooses to end the job altogether or to retry the job.

The resubmit the job immediately done too quickly from new task of a machine, due to failure cannot claim its network. We also wait for job to finish so it can properly execute its clean up phase in which it removes the temporarily created works. When a job request with an incomplete or incorrect network specification is submitted the job will naturally, not be resubmitted and will exit immediately. Once all components are placed the claiming phase starts. In contrast to other jobs this is done once for the whole job, i.e., the components do not get claimed independently.

The claiming is done as a job submission request and can fail for many different reasons, e.g., misspelled or non-existent executable name, input jobs not present, local resource manager unavailable, etc. Some of these errors could be caused by the system itself and could be a local phenomenon. In this case the job can be retried. When a new component is successfully submitted, it is merged into the job component list of the malleable job. The first step of resource discovery in job scheduling is to determine the set of resources that the user submitting the job has access to, in this regard, computing over the grid is no different from remotely submitting a job to a single task: without authorization to run on a resource the job will not run. At the end of this step the user will have a list of machines or resources to which he or she has access. The main difference that grid computing lends to this resources that are not authorized for use. Problem is sheer numbers [7]. It is now easier to get access to more resources, although equally difficult to keep track of them. Also, with current stage implementations, a user can often find out the status of many more machines than what he or she resources that are not authorized for use.

When a user is performing scheduling at the Grid level, the most common solution to this problem is to simply have a list of account names, machines, and passwords written down somewhere and kept secure. While the information is generally available when needed, this method has problems with fault tolerance and scalability for few stages, to proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources. The set of possible job requirements can be very broad and will vary significantly between jobs. It may include static details (the operating system or hardware for which a binary of the code is available, or the specific architecture for which the code is best suited) as well as dynamic details (for example, a minimum RAM requirement, connectivity needed, time space needed). Some schedulers are at least allowing for better coarse-grained information about the applications fulfills.

The grid scheduler's aim is to allocate the jobs to the available nodes. The best match must be found from the list of available jobs to the list of available resources. The selection is based on the prediction of the computing power of the resource. The ant based algorithm is evaluated using the simulated execution times for a grid environment. Before starting the grid scheduling, the

expected execution time for each task on each machine must be estimated and represented by an ET matrix. Each row of ET matrix consists of the estimated execution time for a job on each resource and every column of the ET matrix is the estimated execution time for a particular resource of list of all jobs in job pool.

Here the algorithm, r_j denotes the expected time which resource R_j will become ready to execute a task after finishing the execution of all tasks assigned to it. First, the C_{ij} entries are computed using the ET_{ij} (the estimated execution time of task T_i on resource R_j) and r_j values. For each task T_i, the resource that gives the earliest expected completion time is determined by scanning the i_{th} row of the C matrix (composed of the C_{ij} values). The task T_k that has the minimum earliest expected completion time is determined and then assigned to the corresponding resource from ACO algorithm.

$$C_{ij}=ET_j+r_j \quad \rightarrow (1)$$

Specification of the resources is according to resources speed (MIPS) and bandwidth (Mbps), specification of the tasks depends on instructions and data (MIPS) completion time of the tasks on each of the resources .Tasks/Resources R1, R2 and R3 four tasks *T1*, *T2*, *T3* and *T4* are in the meta-task *Mv* and the grid manager is supposed to schedule all the tasks within *Mv* on three resources *R1, R2* and *R3*. Table 1 is shown the specification of the resources and tasks.

1. procedure ACO
2. begin
3. Initialize the pheromone
4. While stopping criterion not satisfied do
5. Position each ant in a starting node
6. Repeat
7. for each ant do
8. Chose next node by applying the state transition rate
9. end for
10. until every ant has build a solution
11. Update the pheromone
12. end while
13. end

Figure-4: Pseudo code for Existing Ant colony Algorithm.

TABLE 1: SPECIFICATION OF THE RESOURCES AND TASKS.

Tasks	R1 (ready time MIPS)	R2 (ready time MIPS)	R3 (ready time MIPS)	R1 (Executed time MIPS)	R2 (Executed time MIPS)	R3 (Executed time MIPS)
T1	0.44	0.66	0.88	10.88	12.44	14.66
T2	10.88	12.48	14.68	42.66	60.22	62.66
T3	42.68	60.64	64.22	68.66	78.44	74.44
T4	68.68	82.22	92.42	98.44	94.44	102.22

Job scheduling system is the most important part of grid resource management system [9]. The scheduler receives the job request, and chooses appropriate resource to run that job. In this paper, the formulation of job scheduling is based on the expected time to compute (ETC) matrix. Meta-task is defined as a collection of independent task (i.e. task doesn't require any communication with other tasks). Tasks derive mapping statically. For static mapping, the number of tasks, t and the number of machines, m is known a priori. ETC (i, j) represents the estimated execution time for task t_i on machine m_j. The expected completion time of the task t_i on machine m_j is CT (t_i, m_j) = ready (i) + ETC(t_i, m_j) ready (i) is the machine availability time, i.e. the time at which machine m_j completes any previously assigned tasks [10]. The new algorithm is proposed and compare with existing algorithm also presented here.

It is start from a mechanism for defining the grid nodes as well as the input data sources and output data locations load balancing scheme to improve the scaling efficiency of the parallel computation and activity of each node in the grid. To collection of partial result sets from the nodes in the grid and then back to a centralized location. In this method, we achieve the optional additional analysis from the collected results.

The result of the algorithm will have four values (task, machine, starting time, executed completion time). Then the new value of free(j) is the starting time plus ET_{ij}. A heuristic function is used to find out the best resource.

$$\eta_{ij} = 1 / \text{free}(j) \quad \rightarrow (2)$$

Using the formula 3 the highest priority machine is found which is free earlier. Here four ants are used. Each ant starts from random resource and task (they select ET_{ij} randomly j_{th} resource and i_{th} job). All the ants maintain a separate list. Whenever they select next task and resource, they are added into the list. At each iteration, the ants calculate the new pheromone level of the elements of the solutions is changed by applying following updating rule

$$T_{ij} = 1 / Et_{ij} \quad \rightarrow (3)$$

The scheduling algorithm is executed periodically. At the time of execution, it finds out the list of available resources (processors) in the grid environment, form the ET matrix and start scheduling. When all the scheduled jobs are dispatched to the corresponding resources, the scheduler starts scheduling over the unscheduled task matrix ET. This is guaranteed that the machines will be fully loaded at maximum time. The P_{ij}'s value has been modified to include the ET_{ij} is modified to the following equation

$$P_{ij} = T_{ij}\,\eta_{ij}(1/ET_{ij}) / \sum T_{ij}\,\eta_{ij}(1/ET_{ij}) \rightarrow (4)$$

Furthermore, instead of adding ET_{ij}, execution time of the ith job by the jth machine (predicted), in the calculation of probability

$$P_{ij} = T_{ij}\,\eta_{ij} / \sum T_{ij}\,\eta_{ij} \quad \rightarrow (5)$$

---

for each tasks Ti and resources Rj allocations
Compute approximate Cij=Ej+rj to ant's resource allocate end for
do until all tasks in Mv are mapped
for each tasksTi and Rj
if the number of resources is even then
find the resource free times
for each task in Mv find the earliest completion
time and the resource that obtains it
find the task Tk with the
minimum earliest completion time
find the task Tk with the
maximum earliest completion time
assign task Tk to the resource Rl that gives
the better completion time from min and max
Choose place p randomly from set the resources
Suitable for event e, according to probabilities
end for
for each no of resource & tasks (ants)
best of C and Citeration best with Tmin and Tmax
end for
end for
end while

Figure-5: Proposed Algorithm.

This algorithm [10] can be improved using some form of operating systems, hardware, and software, different storage capacities, CPU speeds, network connectivity's and technologies needs. In this method we first find the problem resources and those total execution times equal to the makespan of the solution, and attempt to move or swap set of jobs from the problem processor to another resource that has the minimum and maximize of makespan as compared with all other resources [11]. After applying the above local optimum technique, find out the problem resource reduce time again, swap or move some of the jobs from the resource for relevant jobs. The search is performed on each problem processor and continues until there is no further improvement in the fitness value of the solution.

Using the *Mv* are mapped model, the scheduling problems are number of independent jobs to be allocated to the available grid resources. Because of no preemptive scheduling, each job has to be processed completely in a single machine [12]. Number of machines is available to participate in the allocation of tasks. The workload of each job the computing capacity of each resources (in MIPS) , m- represents the ready time of the machine after completing the previously assigned jobs of minimum earliest completion time find the task *Tk* with the maximum earliest completion time, where the executed machines represents the n-number of jobs and m-represents the number of machines.[13].

IV. RESULTS & DISCUSSION

The proposed algorithm target on grids if the number of available resources is odd, the min-min strategy is applied to assign the first task, otherwise the max-min strategy is applied. The remaining tasks are assigned to their appropriate resources by one of the two strategies, alternatively. For instance, if the first task is assigned to a resource by the min-min strategy, the next task will be assigned by the max-min strategy. Alternative exchange of the min-min and max-min strategies results in consecutive execution of a small and a large task on different resources and hereby, the waiting time of the small tasks in max-min algorithm and the waiting time of the large tasks in min-min algorithm are ignored. As RASA consist of the max-min and min-min algorithms and both have no time consuming instructions. ACO and RASA algorithms incorporate in which intend to optimize workflow execution times on grids have been presented here. The comparison of these algorithms in computing time, applications and resources scenarios has also been detailed. In dynamic grid environments this information that can be retrieved from a many servers includes operating system, processor type and speed, the number of available CPUs and software availability as well as their installation locations.

The distributed monitoring system is designed to track and forecast resource conditions. The *n* tasks can obviously intercommunicate. A general model should take into consideration that the communication phase can happen at any time with I/O phases. To overcome these difficulties our new algorithm is proposed.

In this method four ants are used. The number of ants used is less than or equal to the number of tasks. From all the possible scheduling lists find the one having minimum makespan and uses the corresponding scheduling list. Here three kinds of ET matrices are formed, first one consists of currently scheduled jobs and the next consists of jobs which have arrived but not scheduled. The scheduling algorithm is executed periodically. At the time of execution, it finds out the list of available resources (processors) in the grid environment, form the ET matrix and start scheduling. When all the scheduled jobs are dispatched to the corresponding resources, the scheduler starts scheduling over the unscheduled task matrix ET. This guarantees that the machines are fully loaded at maximum time.

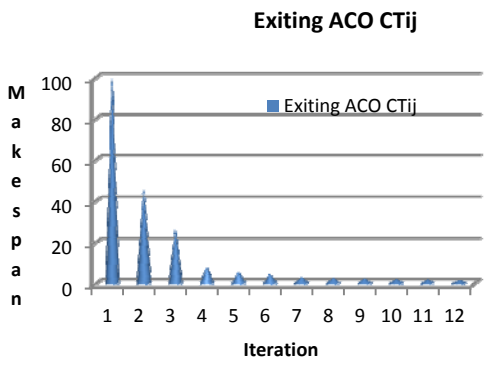

Figure-6: The Completion time of makespan for Exiting ACO Algorithm.

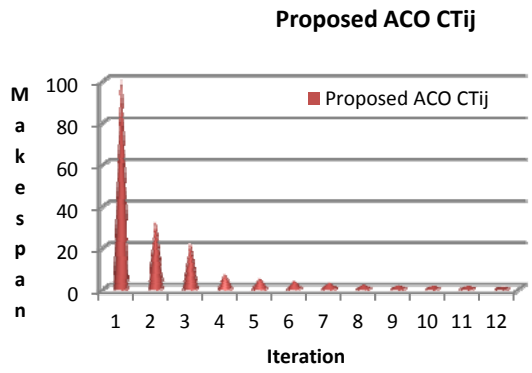

Figure-7: The Completion time of makespan for proposed Algorithm

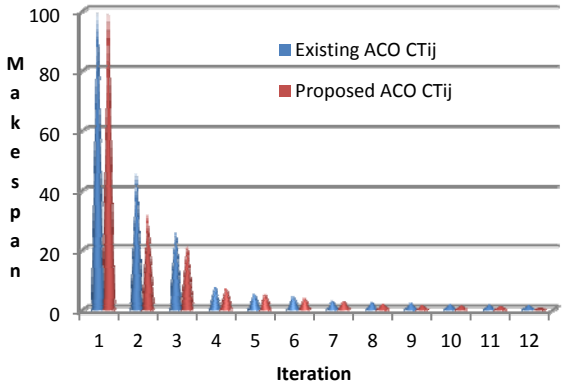

Figure-8: Compare the Completion time of makespans for Existing ACO &Proposed Algorithms.

These executions minimize the overall completion time of the tasks by finding the most suitable resources to be allocated to the tasks. It should be noticed that minimizing the overall completion time of the tasks does not necessarily result in the minimization of execution time of each individual task. The completion time of makespan for both ACO and proposed algorithms are illustrated in fig-6 and fig-7 respectively. Task is assigned to a resource by the min-min strategy; the next task will be assigned by the max-min strategy. In the next round the task assignment begins with a strategy different from the last round. For instance if the first round begins with the max-min strategy, the second round will begin with the min-min strategy. Jobs can be farmed out to idle servers or even idle processors. Many of these resources sit idle especially during off business hours. Fig-8 is shown the compare the completion times of makespan of ACO as well as proposed algorithm. Policies can be in places that allow jobs to only go to servers that are lightly loaded or have the appropriate amount of memory/processors characteristics for the particular application.

V. CONCLUSION

This paper investigates chosen job had been allocated to the best selected ant of each iteration. This process is repeated until all jobs have been scheduled and a complete solution has been built. Each ant in the colony builds a solution, in this manner in each iteration the searching of proper resource allocation on each processing jobs. This algorithm can find an optimal processor and network for each machine to allocate a job that minimizes the tardiness time of a job when the job is scheduled in the system. The proposed scheduling algorithm is designed to achieve high throughput computing in a grid environment. Min-min and Max-min algorithms are applicable in small scale distributed systems. When the numbers of the large tasks are more than the number of the tasks in a meta-task, the Min-min algorithm cannot schedule tasks, appropriately and the makespan of the system gets relatively large. It will be unlike the Min-min algorithm, the Max-min algorithm attempts to achieve load balancing with in resources by scheduling the large tasks prior to the small ones. However, within a computational grid environment high throughput is of great enhancement of resource allocation according to (CPU, network and operating system) system existing scheduling algorithms in large scale distributed system's cost of the communication and many other cases open problem in this area here we concentrate throughout mechanism of entire system needs .

REFERENCES

[1] *Saeed Parsa and Reza Entezari-Maleki* "RASA: A New Task Scheduling Algorithm in Grid Environment" World Applied Sciences Journal 7 (Special Issue of Computer & IT): 152-160, 2009,ISSN 1818-4952.

[2] K. Kousalya and P. Balasubramanie,"To Improve Ant Algorithm's Grid Scheduling Using Local Search". (HTTP://WWW.IJCC.US), VOL. 7, NO. 4, DECEMBER 2009.

[3] Fangpeng Dong and Selim G. Akl, "Scheduling Algorithms for Grid Computing:State of the Art and Open Problems", School of Computing, Queen's UniversityKingston, Ontario, Technical Report No. 2006-504, January 2006.

[4]. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task Mapping and Scheduling in Heterogeneous Computing Environments Genetic-Algorithm-Based Approach," Journal of Parallel and Distributed Computing, vol. 47, pp. 8-22, 1997.

[12] Szajda, D., Lawson, B., and Owen, J. Hardening "Functions for Large-Scale DistributedComputations". *IEEE Symposium on Security and Privacy*, 2003, pp. 216-224.Vanderbei, R. *Linear Programming: Foundations and Extensions*, Second Edition.Norwell: Kluwer, 2001, pp.136-141. <http://www.princeton.edu/~rvdb/LPbook>Accessed 17 March 2006.

[5] J.Brevik, D.Nurmi, and R.Wolski, "Automatic Methods for Predicting Machine Availability in Desktop Grid and Peerto- Peer Systems", In Proceedings of CCGRID'04, pp. 190- 199, 2004.

[6] "A Heuristic Algorithm for Task Scheduling Based on Mean Load"1 Lina Ni1,2, Jinquan Zhang1,2, Chungang Yan1, Changjun Jiang1 1 Department of Computer Science, Tongji University, Shanghai, 2000-92, China.

[7] P. Cremonesi and C. Gennaro. "Integrated performance models for SPMD applications and MIMD architectures". *IEEE Trans. on Parallel and Distributed Systems*, 13(12):1320–1332, 2002.

[8] E. Tsiakkouri et al., "Scheduling Workflows with Budget Constraints", In the CoreGRID Workshop on Integrated research in Grid Computing, S. Gorlatch and M. Danelutto (Eds.), Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, Nov. 28-30, 2005, pages347-357.
[9]. J. D. Ullman, "NP-complete Scheduling Problems," Journal of Computer and System Sciences, vol. 10, pp. 384-393, 1975.
[10] D.Maruthanayagam and Dr.R.Uma Rani, "Enhanced Ant Colony System based on RASA algorithm in Grid Scheduling", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (4) , 2011, 1659-1674,ISSN: 0975-9646.
[11]. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task Mapping and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," Journal of Parallel and Distributed Computing, vol. 47, pp. 8-22, 1997.
[13] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante. "Models of parallel applications with large computation and I/O requirements". *IEEE Trans. on Software Engineering*, 28:286–307, March 2002

Authors Profile

D.Maruthanayagam received his M.Phil, Degree from Bharathidasan University, Trichy in the year 2005. He has received his M.C.A Degree from Madras University, Chennai in the year 2000. He is working as Associate Professor in Master of Computer Applications Department, Gnanamani College of Technology, Pachal, Namakkal, Tamilnadu, India. His areas of interest include Computer Networks, Grid Computing and Mobile Computing.

Dr.R.Uma Rani received her Ph.D., Degree from Periyar University, Salem in the year 2006. She is a rank holder in M.C.A., from NIT, Trichy. She has published around 40 papers in reputed journals and national and international conferences. She has received the best paper award from VIT, Vellore, Tamil Nadu in an international conference. She has done one MRP funded by UGC. She has acted as resource person in various national and international conferences. She is currently guiding 5 Ph.D., scholars. She has guided 20 M.Phil, scholars and currently guiding 4 M.Phil, Scholars. Her areas of interest include information security, data mining, fuzzy logic and mobile computing.