ENHANCED ANT COLONY SYSTEM BASED ON RASA ALGORITHM IN GRID SCHEDULING

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

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

ABSTRACT:

This paper, proposes an enhanced ant colony scheduling algorithm combined with the concept of RASA. The Grid Environment has various operating systems, hardware, and software, different storage capacities, CPU speeds, network connectivities and technologies. Deployment is a very important phase as it bridges the gap between the user sytem (the resourses) and the grid (the resources).The first step for this to select a set of computers and a network connections (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 is more difficulty than in network computational environments. Resources to jobs will be allotted by resource discovery and filtering automatically which is composed of the selection of resources, idea specific scheduling and job submission. This algorithm is evaluated using the simulated execution times for a grid environment.

Key words: ***Grid Computing, Job Scheduling, Heuristic Algorithm, Load Balancing, scheduling algorithm, simulation, ant algorithm.***

I. INTRODUCTION

Grid environment is a distributed environment including different processors with various capabilities. One of the most important issues in resources is the problem of job scheduling. In most of the works accomplished in this field the purpose is finding an appropriate scheduling which minimizes the total tardiness time. Schedulers are to be selected in order to minimize the mean waiting time of processes in queues and also the mean length of queues [1]. The efficient scheduling of jobs on Grid systems is clearly critical because long wait time or queue's long length leads to grate waste of computational resources and also leads to finalization of deadline of some processes [2]. There are relatively a large number of task scheduling algorithms to minimize the total completion time of the tasks in distributed systems [3]. Actually, these algorithms try to 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.

Two well known examples of such algorithms are Min-min and Max-min. These two algorithms estimate the execution and completion times of each of the tasks on each of the grid resources [4]. The Min-Min-Average algorithm (MMA) for scheduling transaction-intensive grid workflows involving considerable communication overheads. First, we establish a fundamental design which can provide nearest neighbours, i.e., the nodes which have the highest network transmission speed with the specific node, for joint scheduling planning, and use real-time information to track the change of network transmission speed so that the scheduling can always be adapted to the current network situation automatically. Due to this adaptation, the communication time can be decreased significantly. It also can make the scheduling algorithm adapt to the change of network transmission speed dynamically [5]. The Max-Min-Average algorithm (MMA) for scheduling found during iteration or during the run of the algorithm, after each iteration only one single job adds with instruction and speed computes. This job may be the one which found the best solution in the current iteration or the one which found the best solution from the beginning of the trial. To avoid stagnation of the search, the range of possible best resourse's (instruction and speeds) on each solution component is limited to an interval [min, max] [6].

The algorithm, **RASA (Resource Aware Scheduling Algorithm),** applies the Max-min and Min-min strategies alternatively to assign tasks to the resources. RASA firstly estimates the completion time of the tasks on each of the available grid resources and then applies the Max-min and Min-min algorithms. The chosen job is then 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. Once all the ants have built a solution the pheromone trail update procedure is performed.

It was observed in the test runs that the ants often take some time to start building good solutions because it takes a few iterations before the pheromone trail is populated with good job-processor pairings. After that, ant systems where algorithmically enunciated for optimization in problems like the salesman traveller and others. Ants are social beings with high structured colonies based on very simple individual behavior. Ants smell pheromone and when choosing their way, they tend in probability to the paths marked with stronger pheromone concentrations. When the time pass the pheromone concentration decrease. Repeating same behavior they compose optimized trails that are dynamically defining and they use to find food sources and their nest.

II. MATERIALS & METHODS

Various algorithms have been designed to schedule the jobs in computational gird. The most commonly used algorithms are MET, MCT, Min-Min, Max-Min and ACO.

A. 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 [7].

B. 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.

C. 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 [9]. 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.

D. 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.

E. ACO

Ant colony optimization (ACO) was first introduced by Marco Dorigo as his Ph.D. thesis and was used to solve the TSP problem [10]. 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.

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.

F. RASA (Resource Aware Scheduling Algorithm) in GRID

The algorithm builds a matrix C where C_{ij} represents the completion time of the task T_i on the resource R_j. 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) [4].

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 relevent 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 [8]. 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 problem is sheer numbers [9]. 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 has

accounts on. As the number of resources grows, it 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 [16].

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 simply does not make sense to examine those resource and every column of the ET matrix is the estimated execution time for a particular resource of list of all jobs in the job pool.

Here the algorithm, r_j denotes the expected time which resource Rj 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 *ith* row of the *C* matrix (composed of the C_{ij} values). The task *Tk* that has the minimum earliest *expected* completion time is determined and then assigned to the corresponding resource from ACO algorithm.

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

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.

TABLE 1: SPECIFICATION OF THE RESOURCES AND TASKS.

Tasks	Instructions & data (MIPS) with ready time			Instructions & data (MIPS) with excuted time		
	R1	R2	R3	R1	R2	R3
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

Execution of ACO System (Old)

No of job & resources: 4 3
Jobs: T1
Jobs: T2
Jobs: T3
Jobs: T4
Resourse: R1
Resourse: R2
Resourse: R3

Workload and timing allotments

T1 and R1 ready time: 0.44

T1 and R1 expected time: 10.88

T1 and R2 ready time: 0.66

T1 and R2 expected time: 12.44

T1 and R3 ready time: 0.88

T1 and R3 expected time: 14.66

T2 and R1 ready time: 10.88

T2 and R1 expected time: 42.66

T2 and R2 ready time: 12.48

T2 and R2 expected time: 60.22

T2 and R3 ready time: 14.68

T2 and R3 expected time: 62.66

T3 and R1 ready time: 42.68

T3 and R1 expected time: 68.66

T3 and R2 ready time: 60.64

T3 and R2 expected time: 78.44

T3 and R3 ready time: 64.22

T3 and R3 expected time: 74.44

T4 and R1 ready time: 68.68

T4 and R1 expected time: 98.44

T4 and R2 ready time: 82.22

T4 and R2 expected time: 94.44

T4 and R3 ready time: 92.42

T4 and R3 expected time: 102.22

n ij values :	Task running compute	
0.01	[0][0]	1.00
0.01	[0][1]	0.45
0.01	[0][2]	0.26
0.01	[1][0]	0.09
0.01	[1][1]	0.06
0.01	[1][2]	0.05
0.01	[2][0]	0.05
0.01	[2][1]	0.04
0.01	[2][2]	0.04
0.01	[3][0]	0.03
0.01	[3][1]	0.03
0.01	[3][2]	0.02
Trail level values:	**Probability Makespan time with ETij**	
0.09	[0][0]	100.00
0.08	[0][1]	41.98
0.07	[0][2]	21.53
0.02	[1][0]	2.88
0.02	[1][1]	1.35
0.02	[1][2]	1.12
0.01	[2][0]	1.07
0.01	[2][1]	0.77
0.01	[2][2]	0.77
0.01	[3][0]	0.51
0.01	[3][1]	0.52
0.01	[3][2]	0.40
CTij values:	**Makespan time with CTij**	
11.76	[0][0]	100.00
13.32	[0][1]	45.51
15.54	[0][2]	26.15
57.34	[1][0]	7.62
74.90	[1][1]	5.23
77.34	[1][2]	4.40
132.88	[2][0]	2.89
142.66	[2][1]	2.49
138.66	[2][2]	2.27
190.86	[3][0]	1.88
186.86	[3][1]	1.79
194.64	[3][2]	1.53

Job scheduling system is the most important part of grid resource management system [11]. 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 ti on machine mj. The

expected completion time of the task ti on machine mj is ct (ti, mj) = ready (i) + ETC(ti, mj) ready (i) is the machine availability time, i.e. the time at which machine mj completes any previously assigned tasks [12]. 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 ETij. A heuristic function is used to find out the best resource

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

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 ETij randomly jth resource and ith 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

$$T_{ij} = 1 / \text{Et}_{ij} \rightarrow (3)$$

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 *Pij's* value has been modified to include the ETij is modified to the following equation

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

Further more, instead of adding ETij, execution time of the *i*th job by the *j*th machine (predicted), in the calculation of probability

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

The proposed algorithm

---

for each tasks T_i *and* resources R_j *allocations*

Compute approximate $C_{ij}=E_j+r_j$ *to ant's resource allocate end for*

do until all tasks in M_v are mapped

for each tasksT_i *and* R_j

if the number of resources is even then

find the resource free times

for each task in M_v find the earliest completion

time and the resource that obtains it

find the task T_k with the
minimum earliest completion time
find the task T_k with the

maximum earliest completion time

assign task T_k to the resource R_l 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

Execution of proposed System

No of job & resources: 4 3
Jobs: T1
Jobs: T2
Jobs: T3
Jobs: T4
Resourse: R1
Resourse: R2
Resourse: R3

Workload and timing allotments

T1 and R1 ready time: 0.44

T1 and R1 expected time: 10.88

T1 and R2 ready time: 0.66

T1 and R2 expected time: 12.44

T1 and R3 ready time: 0.88

T1 and R3 expected time: 14.66

T2 and R1 ready time: 10.88

T2 and R1 expected time: 42.66

T2 and R2 ready time: 12.48

T2 and R2 expected time: 60.22

T2 and R3 ready time: 14.68

T2 and R3 expected time: 62.66

T3 and R1 ready time: 42.68

T3 and R1 expected time: 68.66

T3 and R2 ready time: 60.64

T3 and R2 expected time: 78.44

T3 and R3 ready time: 64.22

T3 and R3 expected time: 74.44

T4 and R1 ready time: 68.68

T4 and R1 expected time: 98.44

T4 and R2 ready time: 82.22

T4 and R2 expected time: 94.44

T4 and R3 ready time: 92.42

T4 and R3 expected time: 102.22

---

n ij values :

0.01
0.01
0.01
0.01
0.01
0.01
0.01
0.01
0.01
0.01
0.01

Trail level values:

0.09
0.08
0.07
0.02
0.02
0.02
0.01
0.01
0.01
0.01
0.01
0.01

CTij values:

11.76
13.32
15.54
57.34
74.90
77.34
132.88
142.66
138.66
190.86
186.86
194.64

Min- Max Completion time:
Min Completion time: 11.760000
Max Completion time: 1.07374176.000000

Task running compute

---

[0][0]	1.00
[0][1]	0.30
[0][2]	0.21
[1][0]	0.15
[1][1]	0.10
[1][2]	0.02
[2][0]	0.02
[2][1]	0.02
[2][2]	0.01
[3][0]	0.01
[3][1]	0.01
[3][2]	0.01

Probability Makespan time with ETij

---

[0][0]	100.00
[0][1]	31.98
[0][2]	11.53
[1][0]	1.66
[1][1]	0.25
[1][2]	0.12
[2][0]	0.03
[2][1]	0.43
[2][2]	0.33
[3][0]	0.31
[3][1]	0.26
[3][2]	0.18

Min- Max ET time:
Min ET Time: 0.513769
Max ET time: 1.0737417600.000000

Makespan time with CTij

---

[0][0]	100.00
[0][1]	32.00
[0][2]	21.56
[1][0]	7.02
[1][1]	5.00
[1][2]	3.85
[2][0]	2.84
[2][1]	2.00
[2][2]	1.52
[3][0]	1.34
[3][1]	1.22
[3][2]	0.42

This algorithm can be improved using some form of operating systems, hardware, and software, different storage capacities, CPU speeds, network connectivities 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.[13]. 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 relevent jobs.The search is performed on each problem processor and continues until there is no further improvement in the fitness value of the solution. The following diagram (Fig 1) shows machine execution time for the introduced mechanism.

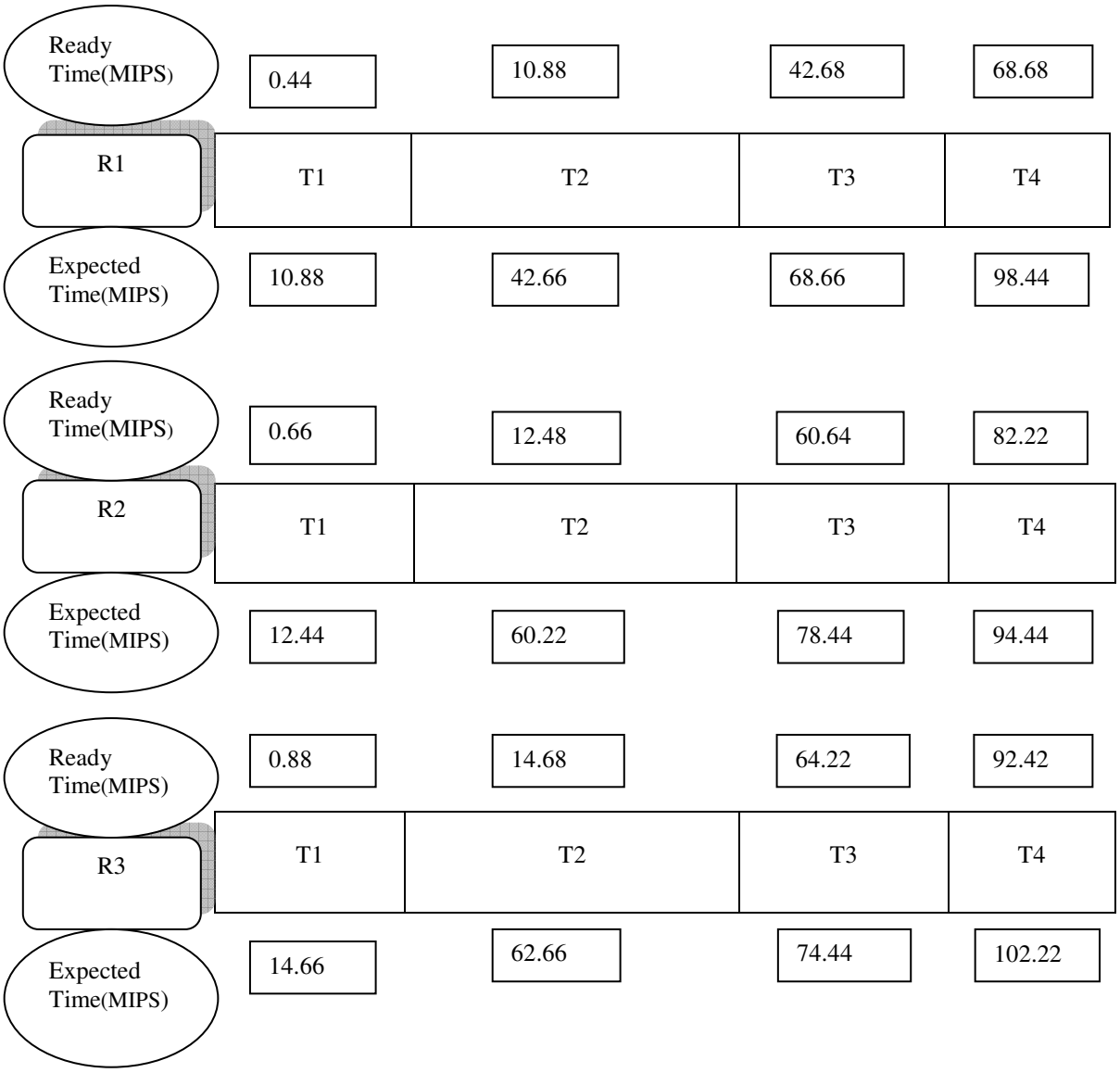

Fig 1: 3x4 (resources and tasks) processing times for both ACO & proposed Algorithm.

Using the *Mv* are mapped model, the scheduling problems are number of independent jobs to be allocated to the available grid resources [14]. Because of no preemptive scheduling, each job has to be processed completely in a single machine. 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.[15].

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 diffculties 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 guarantes that the machines are fully loaded at maximum time. The processing times of both ACO and proposed algorithms are shown in fig 2.

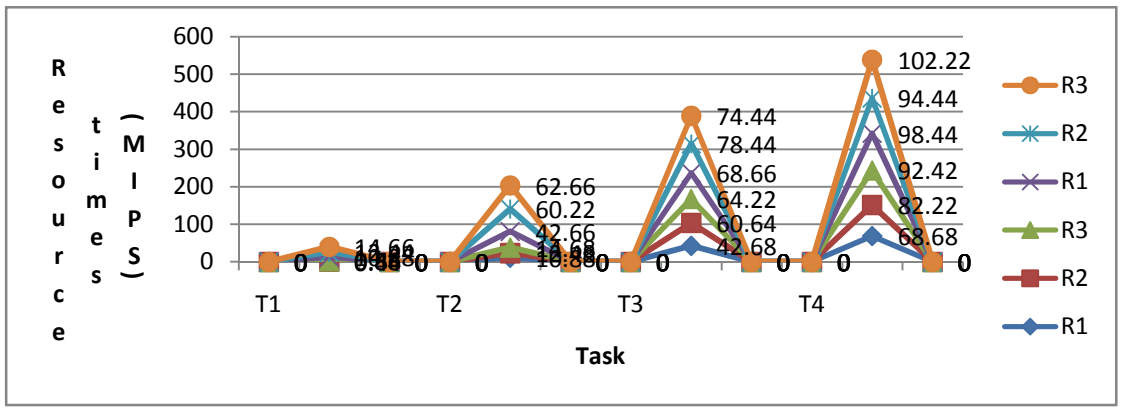

Fig 2: The processing times for both ACO & proposed Algorithm

These exeution 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 3 and Fig 4 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 5 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.

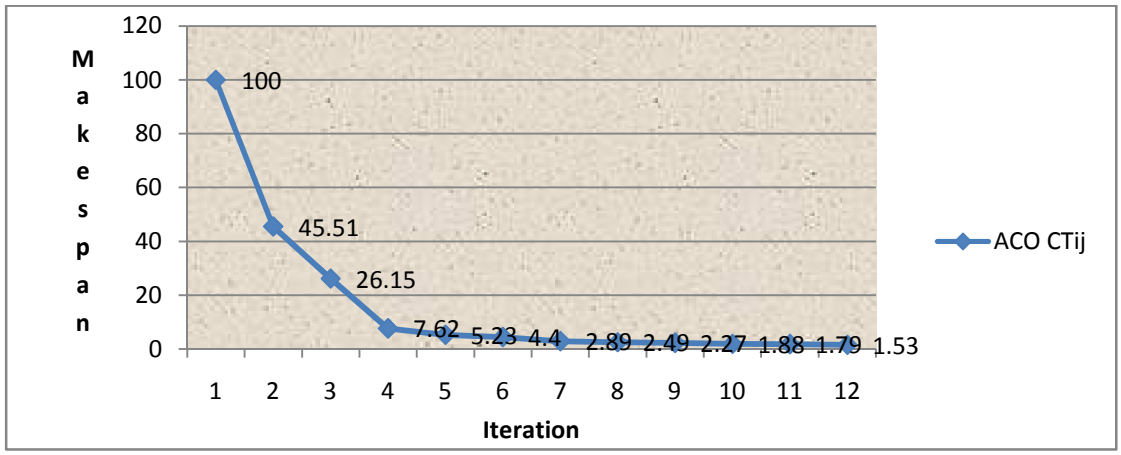

Fig 3: The Completion time of makespan for ACO.

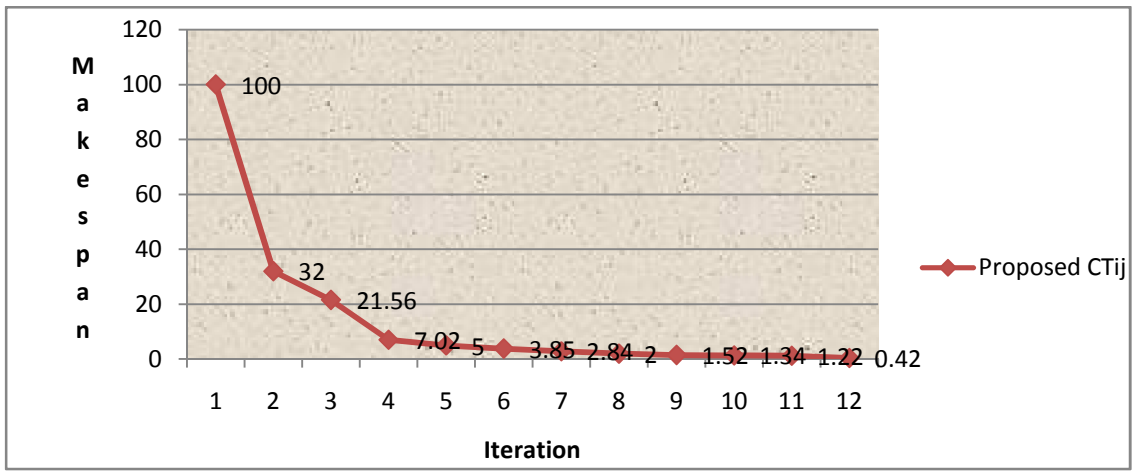

Fig 4: The Completion time of makespan for proposed Algorithm.

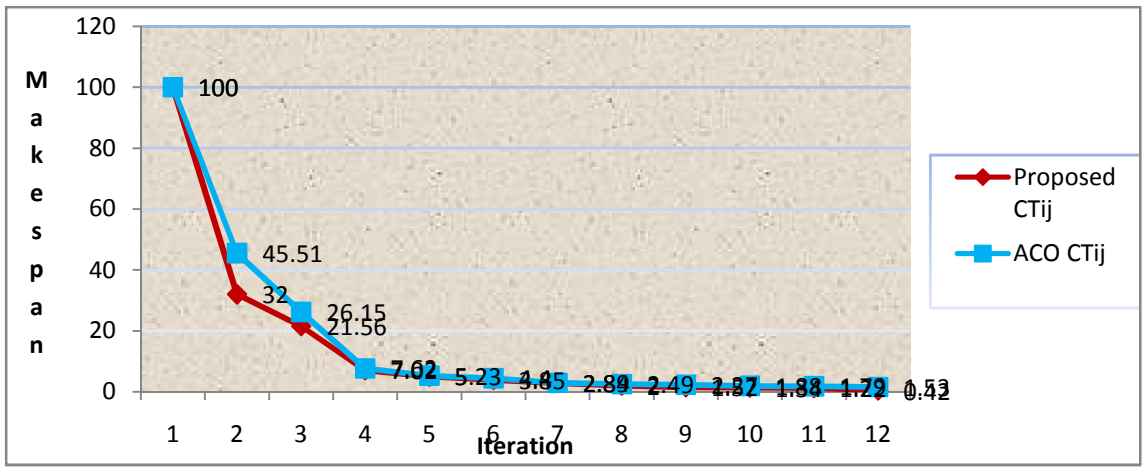

Fig 5: Compare the Completion time of makespans for ACO & Proposed Algorithm.

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 can not 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 resouse allocation accorrding 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 mechnisum of entire system needs .

REFERNCES

[1] SamanehHoseini Semnani, Kamran Zamanifar, Naser Nematbakhsh "New Heuristic in Ant ColonyOptimization to Solve Job Scheduling Problem in Grid", Dept. of Computer Science, University of Isfahan, Isfahan, Iran : 2009

[2] S_ Lorpunmanee, M_ Noor Sap, A_ Hanan Abdullah, and C_ Chompoo-inwai "An ant colonyOptimization for Dynamic Job Scheduling in Grid Environment", International Journal of Computer and Information Science and Engineering, 2007.

[3] He, X., X-He Sun and G.V. Laszewski, 2003. QoS Guided Min-min Heuristic for Grid Task Scheduling. Journal of Computer Science and Technology, 18: 442-451.

[4] *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

[5] Ke Liu1,2, Jinjun Chen1, Hai Jin2 and Yun Yang1 A Min-Min Average Algorithm for Scheduling Transaction-Intensive Grid Workflows, Swinburne University of Technology.

[6] Mr.K.Sankar1 and Dr. K.Krishnamoorthy2 MAX-MIN ANT OPTIMIZER FOR PROBLEM OFUNCERTAINITY, K. Sankar et al. / (IJCSE) International Journal on Computer Science and EngineeringVol. 02, No. 03, 2010, 473-480

[7]M. Maheswaran, S. Ali, H.J.Siegel, D. Hensgen, and R. Freund. Dynamic Matching andScheduling of a Class of Independent Tasks onto Heterogeneous Computng Systems. In 8thHeterogeneous Computing Workshop (HCW'99), Apr. 1999.

[8] 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.

[9]] A Heuristic Algorithm for Task Scheduling Based on Mean Load1 Lina Ni1,2, Jinquan Zhang1,2, Chungang Yan1, Changjun Jiang1 1 Department of Computer Science, Tongji University, Shanghai, 2000-92, China.

[10] 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

[11] 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.

[12]. J. D. Ullman, "NP-complete Scheduling Problems," Journal of Computer and System Sciences, vol. 10, pp. 384-393, 1975.

[13]. 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.

[14] 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

[15] 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

[16] 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.

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 a 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.