Efficient Load Balancing and Resource Scheduling for Optimizing Cost and Execution Time Using ACO-A*Algorithm

Priyanka Chauhan[1], Ritu Bansal[2]

[1]Research Scholar, M. Tech, MRIU, Faridabad

[2]Assistant Professor, Dept. of CSE, MRIU Faridabad

Abstract—**Grid computing is being accepted in a variety of areas from scholastic, business research to government use. Grids are becoming platforms for elevated concert and distributed computing. Resources are energetic in temperament so the load of resources varies with revolutionize in pattern of Grid so the Load Balancing of the tasks in a Grid milieu can appreciably authority Grid's performance. An underprivileged scheduling policy may leave many processors idle while a intelligent one may devour an unduly large portion of the total CPU cycles. The main goal of load balancing is to provide a distributed, low cost, scheme that balances the load across all the processors. To advance the overall throughput of Grid resources, effective and efficient load balancing algorithms are fundamentally important. A load balancing algorithm has been implemented and tested in a simulated Grid environment. In this paper, A* algorithm is being implied with ACO algorithm. The proposed algorithm helps to afford the more efficient load balancing scheduling in the grid computing environment. This algorithm is oriented on the risk assessment and execution time instead of the risk assessment used in the existing algorithms.**

Keywords—**Grid Computing, Load balancing, ACO algorithm, A* algorithm, Execution time**.

I. Introduction

Grid was originally conceived and designed in this community to allow access to computing resources that were geographically dispersed. The notion was that underutilized resources in places other than where the researchers were physically located could be used. Also fundamental in the formative thinking was the prospect of sharing access to data, typically in the form of files that were being jointly produced and used by collaborators in disparate locations.
Before discussing more about Grids lets go back to birth of distributed computing: In the early 1970's when computers were first linked by networks, the idea of harnessing unused CPU cycles was born.
An enterprise-computing grid is characterized by three primary features Client

a) Diversity
b) Decentralisation
c) Dynamism

a) Diversity:
A typical computing grid consists of many hundreds of managed resources of various kinds grid) being developed in a loosely coordinated manner throughout academia and the commercial sector.
The bottom horizontal layer of the Community Grid Model consists of the hardware resources that underlie the Grid. Such resources include computers, networks, data archives, instruments, visualization devices and so on. Moreover, the resource pool represented by this layer is highly dynamic, both as a result of new resources being added to the mix and old resources being retired, and as a result of varying observable performance of the resources in the shared, multi-user environment of the Grid.

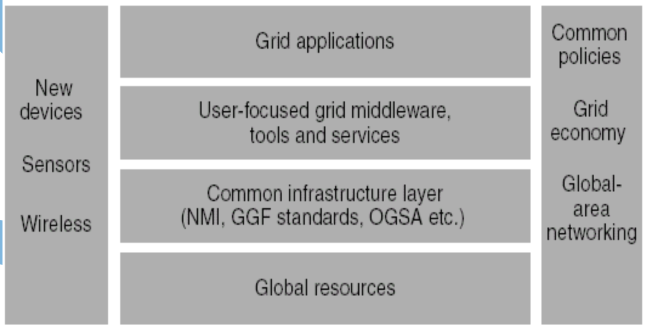

Figure 1 Elements of Grids

The next horizontal layer (common infrastructure) consists of the software services and systems, which virtualized the Grid. The key concept at the common infrastructure layer is community agreement on software, which will represent the Grid as a unified virtual platform and provide the target for more focused software and applications. The next horizontal layer (user and application-focused Grid middle-ware, tools and services) contains software packages built atop the common infrastructure. This software serves to enable applications to more productively use Grid resources by masking some of the complexity

b) Decentralization:
Traditional distributed systems have typically been managed from a central administration point. A computing grid further compounds these challenges since the resources can be even more decentralized and may be geographically distributed across many different data centers within an enterprise.

c) Dynamism:
Components of a traditional application typically run in a static environment without the needing to address rapidly changing demands. In computing grid, however, the systems and applications need to be able to flexibly adapt to changing demand. For instance, with the late binding nature

and cross-platform properties of web services, an application deployed on the grid may consist of a constantly changing set of components. At different points in time, these components can be hosted on different nodes in the network. Managing an application in such a dynamic environment can be a challenging undertaking [3].

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, grids are categorized by the type of solutions that they best address.

The three primary types of grids are

a) Computational grid:

A computational grid is focused on setting aside resources specifically for computing power. In this type of grid, most of the machines are high-performance servers.

b) Scavenging Grid:

A scavenging grid is most commonly used with large numbers of desktop machines. Machines are scavenged for available CPU cycles and other resources. Owners of the desktop machines are usually given control over when their resources are available to participate in the grid.

c) Data grid:

A data grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life science re-search, each with unique data. A data grid would allow them to share their data, manage the data, and manage security issues such as who has access to what data. Another common distributed computing model that is often associated with or confused with Grid computing is peer-to-peer computing. In fact, some consider this is another form of Grid computing

II. RELATED WORK

Tantawiet. al (1985) developed model for such a distributed computer system, in which the host computers and the communications network were represented by product-form queuing networks. In this model, a job might be either processed at the host to which it arrived or transferred to another host. In the latter case, a transferred job incurred a communication delay in addition to the queuing delay at the host on which the job is processed. It was assumed that the decision of transferring a job does not depend on the system state, and hence was static in nature. Performance was optimized by determining the load on each host that minimizes the mean job response time. A nonlinear optimization problem was formulated, and the properties of the optimal solution was the special case where the communication delay does not depend on the source-destination pair is shown. Two efficient algorithms that determined the optimal load on each host computer are presented. The first algorithm, called the parametric-study algorithm, generated the optimal solution as a function of the communication time. This algorithm was suited for the study of the effect of the speed of the communications network on the optimal solution. The second algorithm is a single-point algorithm; it yielded the optimal solution for given system parameters. Queuing models of host computers, communications networks, and a numerical example were illustrated [11].

Yagoubiet. al (2007) proposed strategy which was based on a neighbourhood load balancing whose goal was to decrease the amount of messages exchanged between Grid resources. As a consequence, the communication overhead induced by task transfer and workload information flow was reduced, leading to a high improvement in the global throughput of a Grid. The first experiment results of their strategy were very promising. In effect, they had obtained a significant improvement of the mean response time with a reduction of the communication cost [15].

Carreteroet. al (2007) presented Genetic Algorithms (GAs) based schedulers for efficiently allocating jobs to resources in a Grid system. Scheduling was a key problem in emergent computational systems, such as Grid and P2P, in order to benefit from the large computing capacity of such systems. They presented an extensive study on the usefulness of GAs for designing efficient Grid schedulers when makespan and flowtime are minimized. Two encoding schemes had been considered and most of GA operators for each of them are implemented and empirically studied. The extensive experimental study showed that GA-based schedulers outperform existing GA implementations in the literature for the problem and also revealed their efficiency when makespan and flowtime were minimized either in a hierarchical or a simultaneous optimization mode; previous approaches considered only the minimization of the makespan. Moreover, they were able to identify which GAs versions work best under certain Grid characteristics, which was very useful for real Grids. Our GA-based schedulers were very fast and hence they could be used to dynamically schedule jobs arrived in the Grid system by running in batch mode for a short time [17].

Yajun et. al (2008) addressed the load balancing problem by presenting a hybrid approach to the load balancing of sequential tasks under grid computing environments. their main objective was to arrive at task assignments that could achieve minimum execution time, maximum node utilisation and a well-balanced load across all the nodes involved in a grid. A first-come-first-served and a carefully designed genetic algorithm were selected as representatives of both classes to work together to accomplish our goal. The simulation results showed that theirr algorithm could achieve a better load balancing performance as compared to its 'pure' counterparts [18].

Saravanakumaret. al (2010) proposed Load Balancing on Arrival (LBA) for small-scale (intraGrid) systems. It was efficient in minimizing the response time for small-scale grid environment. When a job arrives LBA computed system parameters and expected finish time on buddy processors and the job was migrated immediately. This algorithm estimates system parameters such as job arrival rate, CPU processing rate and load on each processor to make migration decision. This algorithm also considered job transfer cost, resource heterogeneity and network heterogeneity while making migration decision [19].

Kumar et al. (2011) proposed a Load balancing algorithm for fair scheduling, and compared it to other scheduling schemes such as the Earliest Deadline First, Simple Fair Task order, Adjusted Fair Task Order and Max Min Fair Scheduling for a computational grid. It addressed the fairness issues by using mean waiting time. It scheduled the task by using fair completion time and rescheduled by using mean waiting time of each task to obtain load balance. This algorithm scheme tried to provide optimal solution so that it reduced the execution time and expected price for the execution of all the jobs in the grid system is minimized. The performance of the proposed algorithm compared with other algorithm by using simulation [26].

El-Zoghdy et al. (2012) addressed the problem of scheduling and load balancing in heterogeneous computational grids. They proposed a two-level load balancing policy for the multi-cluster grid environment where computational resources are dispersed in different administrative domains or clusters which were located in different local area networks. The proposed load balancing policy took into account the heterogeneity of the computational resources. It distributed the system workload based on the processing elements capacity which leaded to minimize the overall job mean response time and maximize the system utilization and throughput at the steady state. To evaluate the performance of the proposed load balancing policy, an analytical model was developed. The results obtained analytically were validated by simulating the model using Arena simulation package. The results showed that the overall mean job response time obtained by simulation was very close to that obtained analytically. Also, the simulation results showed that the performance of the proposed load balancing policy outperformed that of the random and uniform distribution load balancing policies in terms of mean job response time. The improvement ratio decreased as the system workload increased [27].

Keerthika et al. (2013) proposed a new scheduling algorithm for computational grids that considers load balancing, fault tolerance and user satisfaction based on the grid architecture, resource heterogeneity, resource availability and job characteristics such as user deadline. This algorithm reduced the makespan of the schedule along with user satisfaction and balanced load. A simulation was conducted using Grid Simulator Toolkit (GridSim). The simulation results showed that the proposed algorithm had better makespan, hit rate and resource utilization [31].

Chandran et al. (2014) demonstrated a genetic algorithm based resource scheduling strategy that focused on system load balancing. The genetic algorithm approach computed the impact in advance that it would have on the system after the new resource is deployed in the system, by utilizing historical data and current state of the system. It then picked up the solution, which would have the least effect on the system. By doing this it ensured the better load balancing and reduced the number of dynamic migrations. The approach presented in this project solves the problem of load imbalance and high migration costs. Usually load imbalance and high number of migrations occurred if the scheduling is performed using the traditional algorithms [32].

Patel et al. (2014) highlighted the Resource sharing and scheduling resources in Grid computing as a complex task due to the heterogeneous and dynamic nature of the resources. Resource sharing crisis brought Grid Technology that needed algorithms and mechanisms to be redesigned for resource handling. The analysis of algorithms is the determination of the number of resources (such as time and storage) necessary to execute them. In this paper, they could implement the MCT (minimum completion time) and MET (Minimum execution time) algorithm to increase the performance in terms of their speed of execution [33].

III. LOAD BALANCING APPROACHES

Whose goal is to keep all computing nodes busy, and load balancing which attempts to have an equal load on all the nodes. The design of a load redistributing algorithm depends on the performance objectives sought and the appropriate redistribution approach. The ultimate goal of these strategies is to minimise the system average and standard deviation of the response time with minimum adverse effect on individual users. A good handling of task partition, task allocation and load balancing can significantly increase a grid systems' efficiency. In this dissertation, balancing the loads in electrical grid systems and optimizing grid computing systems are analyzed. Unbalanced loads on feeders increase power system investment and operating costs. Three-phase lateral loads phase swapping is one of the popular methods to balance such systems. One way to provide load balancing in Grid Systems is with IP spraying, the load balancing mechanism used for spreading HTTP requests. The IP sprayer intercepts each HTTP request, and redirects them to a server in the server cluster.

1) Types

Depending on the type of sprayer involved, the architecture can provide scalability, load balancing and failover requirements. Load balancing schemas are characterized in terms of

- Allocation
- Agent
- Initiation and
- Policy types.

2) Initiation

Load balancing can be characterized based on the party that initiated the process

- *Sender Initiated*
  The sender makes a determination as to where a generated task or arriving task is to be executed. The queues of ready jobs tend to form at the target PE's. Job transfer decisions are made at task arrival time.
- *Receiver Initiated*
  In a receiver initiated process, a server or target PE determines which jobs at different sources, it will process. The ready jobs tend to queue at the source PE's. with job transfer decisions made at task completion time.

IV. ALLOCATION BASED LOAD BALANCING IN GRID COMPUTING

Grid Systems can be characterized based on its allocation. We will describe two only types of load balance allocation:

- Static
- Dynamic

1) Static allocation

These algorithms aim at finding an optimal assignment of tasks by clustering or co-scheduling, and are achieved by balancing the system loads periodically. They assume that the process behavior is known and use graph theory models to attempt a fair distribution of the load. The allocation decisions of the system components are based on pre-determined parameters. Early work on load balancing has been carried out along this approach but due to inherent drawbacks such as

1) the static nature of the algorithm does not allow these strategies to respond to short-term fluctuations in workload, 2) they require too much information such as arrival time and execution cost of each job or module to be implementable, and

3) they involve intensive computation to obtain the optimal schedule; the research effort has recently concentrated on the two other heuristic approaches which are implementable and achieve promising results. Quasi-static algorithms are a variant of this category. These algorithms ignore the current state of the system, but they tune their decision variables by adapting to slowly changing system characteristics such as the arrival rates of jobs. Static allocation are the simpler case: random, variations of round robin and master-worker.

Random Allocation

Requests are assigned to any server picked randomly. Counts on statistical average to have each server getting its share of the load due to the random selection.

- Pros: Simple to implement.
- Cons: Can lead to overloading of one server while under-utilization of others.

Round-Robin Allocation

Requests are assigned to one service from a list of the servers on a rotating basis.

- Pros: Better than random allocation because the requests are equally divided among the available servers in an orderly fashion.
- Cons: Round robin algorithm is not enough for load balancing based on processing overhead required and if the server specifications are not identical to each other in the server group.

Weighted Round-Robin Allocation

A weight is assigned to each server in the group. The most powerful gets higher weight. If server A has weight 2, and server B has 1, server A will receive twice the number of requests B does.

- Pros: Takes care of the capacity of the servers in the group.
- Cons: Does not consider the advanced load balancing requirements such as processsing times for each individual request.

"Worker" Pattern

One common pattern for doing this employs generic "workers" that are designed specifically to take executable entries from a space and run them. Using this model yields the added benefit of automatic load balancing: rather than having work pushed to them from a centralized load balancer, workers in this model each regulates its own load independently by taking entries at its own pace when it is ready to process them.

2) Dynamic Allocation

Here scheduling is seen as a job routing problem. These algorithms balance the loads upon the arrival of each job. This is achieved by a continuous assessment of the system load which is dynamic and unpredictable. The allocation of the job is done in real time following a fixed policy based on the recent system state information and currently perceived system load imbalance or bases their decisions on statistical averages. Extensive research work has been done in this category. For the second type of load balancing allocation complexity is exponentially higher. Two of the simplest cases are allocation by sampling and predictive.

Sampling

Uses a sampling on the past to select the running node (simpler). One other alternative is the use the average of the last execution times to select the next processor to receive the run. Some examples:

- Sampling of similar runs on the past
- Average of similar executions on the past
- Xsuffrage (task that will suffer the most if not executed on the fastest processor)
- Min-Min
- Min-Max.

Predictive

Preferably based on prediction methods, e.g. ES (Exponential Smoothing). ES is an equation that uses a constant alpha, the last prediction, and the last real value to calculate the future predicted value. This cannot be called "prediction methods" literarily, because you have to be wrong first to adjust your prediction. Uses sampling rate and a constant. Nodes exchange and re-arrange tasks as the load is predicted according to sampling rate. Some examples:

- Exponential Smoothing
- Norrish Equation
- Grover Model, etc

3) Adaptive Algorithms

Scheduling in this approach can be interpreted as an adaptive control problem. These algorithms, like dynamic algorithms, balance loads upon the arrival of each job, but also balance loads whenever anomalies appear in the workload of the system or individual nodes. They exhibit more flexibility by adjusting their policy to match the dynamic system characteristics. In the literature some algorithms with different degrees and approaches of adaptability have been reported. To support adaptability, most of these algorithms use preemptive scheduling.

Although the term dynamic scheduling and adaptive scheduling have often been used interchangeably in the literature by grouping any policy that is not static under the heading of dynamic, there is a clear distinction between the two. A dynamic algorithm has a fixed policy in dealing with its dynamic environmental inputs, whereas an adaptive algorithm uses the environmental stimuli to modify the scheduling policy itself.

V. PROPOSED ACO-A*ALGORITHM

The standard ACO algorithm is totally oriented on the risk assessment factor. This algorithm has totally ignored the concept of the execution time and resource cost. The proposed algorithm utilize the A* algorithm in which it calculate the evaluation value based on transition probability and MIPS rating. After considering these factors the proposed Heuristic Load Balancing Algorithm is presented below:

```
Phase 1 (Initialization Phase)
Begin: Initialize all parameters including resources (processing elements, MIPS rating), pheromone intensity, Task set, resource cost, no. of resources, no. of task, resource set
Phase 2 (Operational Phase)
While (Task set T! = Φ)
Begin: Select the next task t" from Task set T.
        Determine next resource Ri for task assignment having high evaluation value.
        (Evaluation value = Transition Probability + MIPS rating)
 Phase 3 (Result Phase)
        Schedule task t" to Ri and update Task set by T = T - {t}.
        If any node fails or complete its execution part update its pheromone intensity of that corresponding resource.
 END While
 END
```

VI. RESULT

On executing the proposed algorithm having 15 resources and 99 gridlets it show following output as given below:
Starting Execution of New Paper execution
Initializing GridSim package
Initialising...
Starting to create one Grid resource with *5 PE*of Rating 500
Starting to create one Grid resource with *4 PE*of Rating 650
Here we are using Time_shared allocation policy. We are going to compare following algorithm as below:

- Existing heuristic load balancing algorithm based on ACO algorithm
- Proposed heuristic load balancing algorithm based modified ACO algorithm

Existing heuristic load balancing algorithm based on modified ACO algorithm consist the combination of following factors instead of transition probability as given below:

- Transition probability
- MIPS Rating

Experiment: 1
The Total Execution Time of Existing algorithm compared with Proposed algorithm with the following parameters.
Resource Allocation Policy=TIME_SHARED
Number of Resources =25
Number of Tasks = 10 to

TABLE 1
EXECUTION TIME WITH VARYING NO. OF TASKS IN TIME SHARED ALLOCATION

Sr. No.	Number of Tasks	Execution Time using Proposed Algorithm(ns)	Execution Time using Existing Algorithm(ns)	Average Reductionin Execution Time%
1	10	5181	8910	71.97
2	20	18182	29597	62.78
3	30	38747	67431	74.02
4	40	66593	116923	75.57
5	50	101353	180192	84.82

Average Reduction in Total Execution Time is = 72.42 %. Figure 2.shows that as the number of tasks increases the difference between execution time taken by two algorithms decreases. The execution time is reduced due to selection of optimized node.

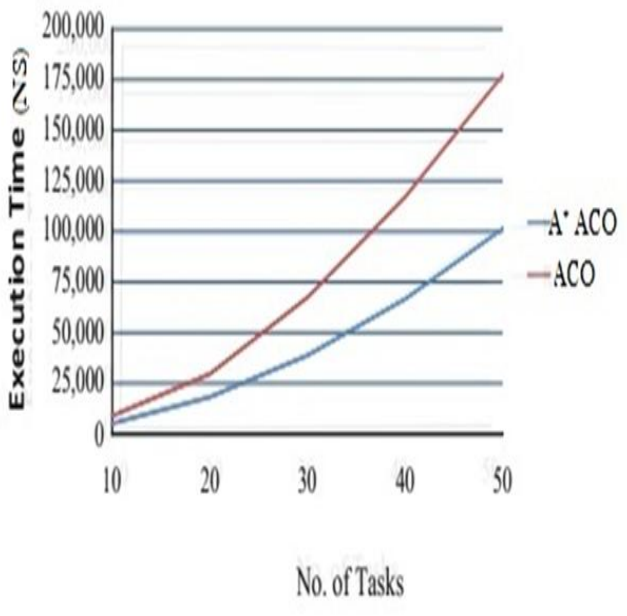

Figure 2.Number of Tasks Vs. Execution Time in TIME_SHARED Allocation

Experiment: 2
The Total Execution Cost ofExisting algorithm compared with Proposed algorithm with the following parameters.
Resource Allocation Policy=TIME_SHARED
Number of Resources =25
Number of Tasks = 10 to 50

TABLE 2
EXECUTION COST WITH VARYING NO. OF TASKS IN TIME SHARED ALLOCATION

Sr. No.	Number of Tasks	Execution Cost using Proposed Algorithm(G$)	Execution Cost using Existing Algorithm(G$)	Average Reduction in Execution cost%
1	10	457	565	23.63
2	20	1416	2062	45.62
3	30	3334	4452	33.53
4	40	6077	7823	28.73
5	50	9383	10907	16.24

Average Reduction in Total Execution Cost is = 29.55 %. Figure 3.shows that as the number of tasks increases the difference between execution cost the task acquired by two algorithms decreases. The more reliable resources are being selected.

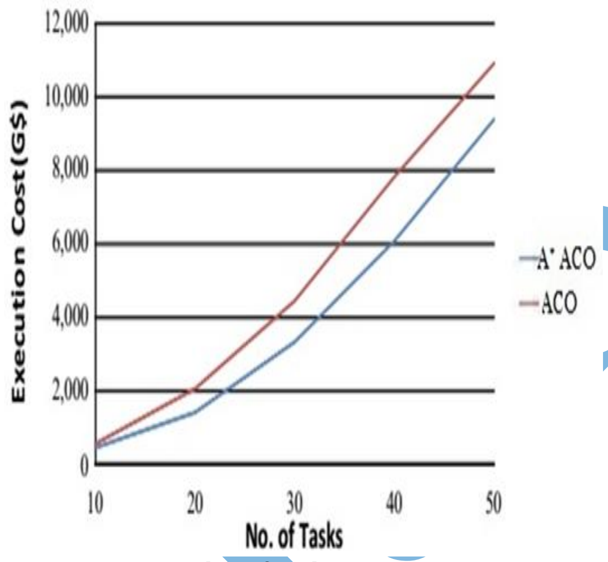

Figure 8.2 Number of Tasks Vs. Execution Cost in TIME_SHARED Allocation

Experiment: 3

The Total Execution Time of Existing algorithm with Proposed algorithm with the following parameters.

Allocation Policy=SPACE_SHARED

Number of Resources =10 to 50

Number of Tasks = 25

TABLE 3

EXECUTION TIME WITH VARYING NO. OF RESOURCES IN SPACE SHARED ALLOCATION

Sr. No.	Number of Resources	Execution Time using Proposed Algorithm(ns)	Execution Time using Existing Algorithm(ns)	Average Reduction in Execution Time%
1	10	21145	37919	79.32
2	20	28054	44408	58.29
3	30	30700	50736	65.26
4	40	39703	58385	47.05
5	50	56073	70702	26.08

Average Reduction in Total Execution Time is = 55.20%.

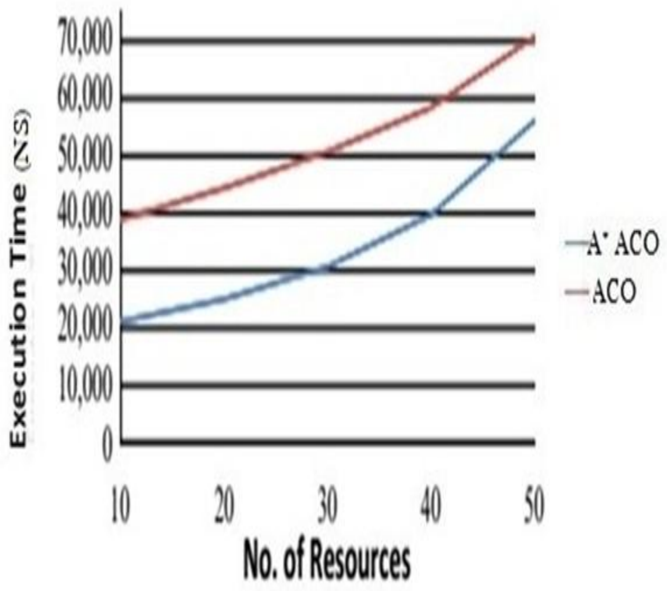

Figure 4 Number of Resources Vs. Execution Time in SPACE_SHARED Allocation

Experiment: 4

The Total Execution Time of Existing algorithm with Proposed algorithm with the following parameters.

Allocation Policy=SPACE_SHARED

Number of Resources =10 to 50

Number of Tasks = 25

TABLE 4

EXECUTION COST WITH VARYING NO. OF RESOURCES IN SPACE SHARED ALLOCATION

Sr. No.	Number of Resources	Execution Cost using Proposed Algorithm(G$)	Execution Cost using Existing Algorithm(G$)	Average Reduction in Cost%
1	10	4189	4473	0.06
2	20	3295	3934	19.39
3	30	2265	2887	27.46
4	40	1371	2307	68.27
5	50	990	1309	32.22

Average Reduced in Total Execution Cost is = 29.44 %. . The more reliable resources are being selected.

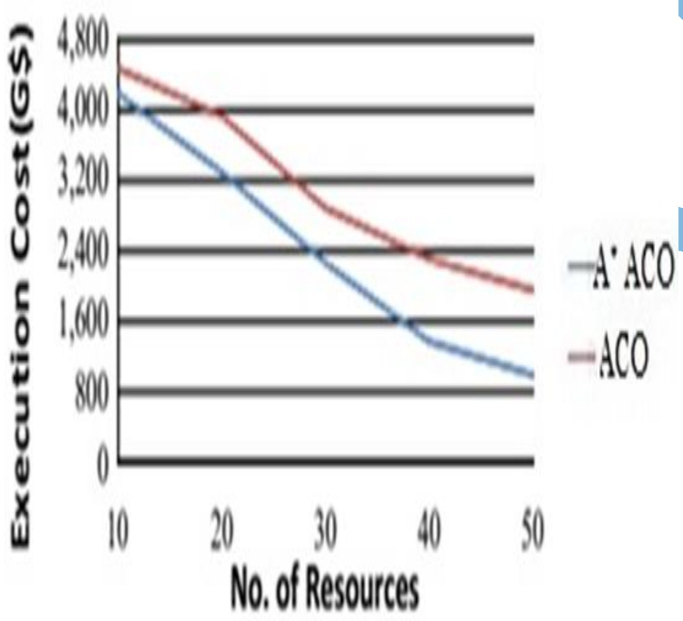

Figure 5: Number of Resources Vs. Execution Cost in SPACE_SHARED Allocation

VII. CONCLUSIONS

We have performed these experiments based on the ACO and ACO-A*Algorithm and it found that the proposed algorithm performs better than the Existing in terms of Execution Cost and Execution Time with both TIME_SHARED OR SPACE_SHARED allocation policies. Both the Execution time and Cost is reduced using ACO-A*. Result of these experiments can be summarized as:

TABLE 5
SUMMARY OF REDUCTION OF EXECUTION COST AND TIME

Experiment No.	Number of Tasks	Number of Resources	Allocation Policy	Reduction In Execution Time %	Reduction In Execution Cost %
1	10 - 50	25	TIME_SHARED	72.42%	29.55%
2	10 -50	25	SPACE_SHARED	55.20%	29.44%

VIII. FUTURE SCOPE

The work performed in this thesis can be used as the basis for an improved load balancing module in Condor. This not only improves the performance of grid application but also makes it more powerful, reliable and capable of handling more complex and large problems in Grid environment. A further extension to this work would be in making this Load balancing Module a middleware independent module. We may add other parameters like time delay etc.

REFERENCES

[1]. Akshay Luther, RajkumarBuyya, Rajiv Ranjan, and SrikumarVenugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", GRIDS Laboratory, The University of Melbourne, Australia

[2]. Asser N. Tantawim& Don Towsley, Optimal static load balancing in distributed computer systems, Journal of the ACM (JACM), Volume 32 Issue 2, April 1985 Pages 445-465

[3]. AzzedineBoukerche and Robson Eduardo De Grande, Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems, proceeding 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp. 174-182.

[4]. A.Oufimtsev, L. Murphy. Method Input Parameters And Performance Of EJB Applications. In Proceedings of 19th Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2004.

[5]. Berman F., fox G., Hey Y., “Grid Computing: Making the Global Infrastructure a Reality”, Wiley Series inCommunications Networking & Distributed Systems, 2003.

[6]. BelabbasYagoubi, YahyaSlimani, Load Balancing Strategy in Grid Environment, Journal of

Information Technology and Applications, Vol. 1 No. 4 March, 2007, pp. 285-296.

[7]. Belabbas Yagoubi & Meriem Meddeber, Distributed Load Balancing Model for Grid Computing, ARIMA Journal vol. 12 (2010), pp. 43-60.

[8]. B. Yagoubi , Task Load Balancing Strategy for Grid Computing .*IEEE Computer*, vol. 38, num. 11, pages 89–96, 2006

[9]. Buyya R., "A Grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing", www.buyya.com/gridsim/.

[10]. C. Campbell. Constructive learning techniques for designing neural network systems.In C. T. Leondes, editor, Neural Network Systems Technologies and Applications, San Diego, 1997.Academic Press.

[11]. C.-J. Wang, P. Krueger, M. T. Liu. Intelligent job selection for distributed scheduling. In Proceedings of the 13th IEEE International Conference on Distributed Computing Systems, pages 517--524, 1993.

[12]. D. Gupta, P. Bepari. Load sharing in distributed systems, In Proceedings of the National Workshop on Distributed Computing, January 1999.

[13]. El-Zoghdy, S.F.; Aljahdali, S., "A two-level load balancing policy for grid computing," Multimedia Computing and Systems (ICMCS), 2012 International Conference on , vol., no., pp.617,622, 10-12 May 2012

[14]. Foster I., Kesselman C. (editors), "The Grid2: Blueprint for a New Computing Infrastructure", Morgan Kaufmann (second edition), USA, 2004.

[15]. Francois Grey, MattiHeikkurinen, Rosy Mondardini, RobindraPrabhu, "Brief History of Grid", http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html.