Comparative Survey on Load Balancing Techniques in Computational Grids

R. Rajeswari, Dr. N.Kasthuri

Abstract— Grid is the system which provides a new, powerful and innovative platform that caters the need of massively computational or data intensive applications from its pool of resources like processors, memory, data, services etc. It differs from traditional computing systems because of its heterogeneous nature and back ground workloads. Performance and utilization of the grid rests on the optimal balancing of load among the available nodes which is very complex and highly dynamic in nature. Finding optimal solution in load balancing for such an environment using the traditional method is an NP-hard problem whereas heuristic approaches will provide near optimal solutions. Algorithms that could capture the dynamic need and complexity have to be developed for solving wide range of load balancing scenarios. Heuristic and artificial life techniques have the power of providing near by solutions from large search spaces since it deals real world scenarios with the capability of handling very large dataset and combinations. In this study, suitability and performance comparison are discussed with various heuristic and agent based techniques. Genetic Algorithm, Tabu Search, Ant Colony Optimization, Particle swarm Optimization are analyzed with their merits, demerits, solutions, issues and improvements towards load balancing in computational grid. Similarity in their nature towards load balancing motivates the attempts in the experimentation to get near optimal solutions from unpredictable information. Performance comparison is analyzed with algorithms like min-max, max-min and Sufferage embedded with Genetic Algorithm and Tabu search. Another heuristic method, Ant Colony Optimization algorithm is suitable for scheduling in grid environment which in tern balances the load. For the same purpose particle swarm optimization algorithm is also adopted. Particle Swarm Optimization is one of the latest evolutionary optimization techniques by nature which has the better ability of global searching leading to minimal makespan time due to the linear decreasing of inertia weight in it. From the literature, it could be understood that it was successfully applied in training the neural network and optimized result was been obtained. These techniques were studied with their successful results and analyzed. Agents can also be applied for handling grid resources and multi-agent approach can be applied for balancing the load through out the system. Agents can co-operate each other in making the decisions to balance the load among them through advertisement, discovery and distribution. Many results are proving that intelligent agents are effective enough to achieve resource scheduling, load balancing, execution performance and better resource utilization.

Index Terms— Agent, Ant Colony Optimization, Computational Grid, Genetic Algorithm, Load Balancing, Particle Swarm optimization, Tabu Search.

1 INTRODUCTION

Grid is defined as a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and quality-of-service. Grid technology is defined as the technology that enables resource virtualization, on demand provisioning and resource sharing between organizations. It can be confined to the network of computer workstations within a bigger organization like corporation or it can be a public collaboration.

Computational grid provides a better platform to acquire large amount of resources as a single storage which enables distributed processing of computational intensive applications. They are cost effective since it collects the information about idle resources and make them under the use on demand which enables pay for use. To fully utilize the resources with conditions and constraints, resource management, efficient allocation of resources to the jobs and balancing the load with internal and external requirements are to be effectively handled.

The load balancing mechanism aims at equally spreading the load to each and every computing node which in turn leads to maximize the utilization and minimize the total task execution time [1]. Fair distribution is needed among all the computing nodes to achieve the goals and objectives. Gradually the gap between heaviest and lightest load should be minimized. Makespan is a metric which defines the amount of time taken between the starting of the application and ending of the same in a computing node.

In the centralized approach central node acts as scheduler and makes all load balancing decisions where as all nodes are engaged in load balancing decisions and implementation in the decentralized approach [2],[3]. In static approach characteristics of nodes and jobs are known in advance whereas they are obtained on the fly in the dynamic approach [4].

From the literature three approaches are studied towards load balancing in the grid environment namely Min-min, Max-min, and Sufferage. For each and every task, minimum completion time (MCT) is calculated. In the Min-Min approach minimum MCT among all tasks is selected for load balancing where as it is reverse in Max-Min approach. Sufferage is the measure which is calculated as the difference between best MCT and second best MCT. Based upon the sufferage values load balancing is carried out among all the tasks.

- *Author R.Rajeswari is currently pursuing her Ph.D in Anna University, Chennai, Tamilnadu, India, E-mail:rajeswarisel@gmail.com*
- *Co-Author Dr. N. Kasthuri is currently working as Professor in Electronics and Communication Department,Kongu Engineering College, Perundurai, Erode, Tamilnadu, Indi*

The term heuristic is used for algorithms which find solutions among all possible combinations. These algorithms find a solution close to the best one and they find it fast and easily .In this paper, we analyze various heuristic methods like Genetic Algorithm (GA), Tabu Search (TS) , Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) for the power of real world nature in taking decisions and Agent based approaches for the power of decentralization. This is done towards their functionality, performance, issues, merits, demerits and scope to the improvement.

GA is often used to solve the problem in which possible solution involves searching a big search space of potential solutions. Since load balancing in computational grid has also a big solution search space, GA can be applied to derive the solution for the same. Tabu search has been successfully applied to a wide range of theoretical and practical problems, including graph coloring, vehicle routing, job shop scheduling, course scheduling, and maximum independent set problem [5],[6],[7],[8],[9]. One main ingredient of Tabu Search is the use of adaptive memory to guide problem solving. Tabu search uses a set of strategies and learned information to mimic human insights for problem solving.

The use of Genetic algorithm and Tabu Search provides efficient solutions to the load balancing of grid. The Sufferage algorithm is combined with GA, as well as TS, which improves the overall performance of the algorithms. A comparison on the performance improvement of each algorithm is also investigated.

The ant colony optimization algorithm is a heuristic algorithm for solving computational problems which can be reduced to find good paths through graphs. This algorithm is a member of swarm intelligence methods. This algorithm is aiming to search for an optimal path in a graph based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems. As a result, several problems have emerged, drawing on various aspects of the behavior of ants.

Particle swarm optimization is an another algorithm modeled on swarm intelligence, which finds a solution to an optimization problem in a search space, or model and predict social behavior in the presence of objectives. The PSO is a stochastic and population-based computer algorithm modeled on swarm intelligence. Swarm intelligence is based on social-psychological principles and provides insights into social behavior, as well as contributing to many engineering applications. The PSO algorithm is used to find an optimal solution to an objective function in a search space which is a direct search method depends only on the objective function. Hence it is more powerful.

In agent based approach, each agent is responsible for resource scheduling and load balancing across multiple hosts/processors in a local grid. The agent couples application performance data with iterative heuristic algorithms to dynamically minimize task makespan and host idle time, whilst meeting the deadline requirements for each task. The algorithm is based on an evolutionary process and is therefore able to absorb system changes such as the addition or deletion of tasks, or changes in the number of hosts/processors available in a local grid.

At the global grid level, each agent is a representative of a grid resource and acts as a service provider of high performance computing power. Agents are organized into a hierarchy and cooperate with each other to discover available grid resources for tasks using a peer-to-peer mechanism for service advertisement and discovery. Several metrics are considered to measure the load balancing performance of grid agents. A case study proves that intelligent agents, supported by application performance prediction, iterative heuristic algorithms and service discovery capabilities, are effective to achieve overall resource scheduling and load balancing, improve application execution performance and maximize resource utilization.

The following section deals about the grid system and its communication model that is used for this study. This is followed by the above said heuristic and agent based methods are analyzed to study the load balancing patterns. Experimentations are illustrated with the simulation tool kit GridSim. Result section provides the details of simulation and applicability of the approaches.

2 System Model

We consider the computational grid system as a set of sites which are connected by a communication network. Each site may contain many computing nodes and each and every node can contain many processors with different computing power and also heterogeneous in nature. A node may vary with other nodes in total number of users in local as well as in network, different capacity of memory, resources and other services. Each and every node will also have its local load and remaining part can only be spared as global resource.

In our context to constraint ourselves we take the jobs are in full, each node has single processor and sites are interconnected with message communication. Communication time depends on the transmission delay, bandwidth and size of the messages. Assuming our application model, we limit our assumptions that each and every application is independent, they do not require order of execution, they are computational intensive, no consideration in operating system level and no support of job migration and resource replication. Each and every task has different computation and communication time depends upon its own nature.

3 Load Balancing in Grid

Load balancing is a technique which enhances the usage of resources, utilizing parallelism, exploiting the improvisation of throughput and reduces the response time through the distribution of the applications in an appropriate fashion. Each and every load balancing method is designed to spread the load on resources equally and maximizes their utilization at the same time minimizes the total task execution time. Selection of optimal set of jobs for transfer has a significant role on the efficiency and effectiveness of the load balancing method

as well as grid resource utilization.

Load balancing algorithms are classified into many categories based on the criteria. They are like static or dynamic, sender or receiver initiated, global or local strategies, centralized or decentralized, co-operative or non-cooperative, Adaptive or non-adaptive and one time assignment or dynamic reassignment [12]. In static algorithms, the decisions are made at compile time where requirements are initially estimated. In dynamic load balancing, load balancer allocates/re-allocates resources at runtime and uses the system-state information to make its decisions. Adaptive load balancing algorithms adapt their activities by dynamically changing their parameters, policies and system state.

Methods used in load balancing can be divided into three classes namely centralized, decentralized or distributed and hierarchical. In a centralized approach, all jobs are submitted to a single schedule which is responsible for scheduling the jobs on the available resources. Since all the scheduling information is available at once, the scheduling decisions are optimal but this is not very scalable.

In a decentralized model there is no central scheduler and scheduling is done by the resource requesters and owners independently. This approach is scalable, distributed in nature, and suits vey well for present grid environment. But individual schedulers should cooperate with each other in scheduling decisions and the schedule generated may not be the optimal schedule. This category of load balancing is perfect for peer-to-peer architectures and dynamic environments.

In a hierarchical model, the schedulers are organized in a hierarchy. High level resource entities are scheduled at higher levels and lower level smaller sub-entities are scheduled at lower levels of the scheduler hierarchy. This model is a combination of the above two models.

In centralized load balancing, algorithms are used in three approaches. In the classical approach Random, Round-Robin, MET (Minimum Execution Time), MCT (Minimum Completion Time), Min-Min, Max-Min and Sufferage methods are used. In agent-based approach, intelligent agents and multi-agent approach, agent-based Grid management infrastructure with performance-driven task scheduler and agent with genetic algorithm-based scheduler are attempted. In Evolutionary Computing Approach, Genetic Algorithms, Tabu search, Ant Colony Optimization and Particle Swarm Optimization are used.

In the decentralized load balancing, again the algorithms are used in three approaches. In the classical approach, sender, receiver and stable symmetrically initiated adaptive algorithms, State Broadcast Algorithm (STB) and Poll when Idle Algorithm (PID) are used. In Ant Colony Optimization Approach, Ant Colony optimization, Anthill Framework (Messor system) and Multiple Ant Colony Optimization (MACO) are used. In the agent based approach, multi-agent systems with forward and backward routing are used. In the routing, ABC (Ant-Based Control system), Ant-Net (routing problem in datagram networks), ASGA (ant colony systems with genetic algorithm) are used. In another approach, Job Migration is also used with load balancing.

Genetic Algorithm and Tabu search are evolutionary search techniques to find solutions to optimization and search problems. These techniques are inspired from evolutionary biology and apply features such as inheritance, mutation, selection, and crossover. They have been proved to work better compared to classical algorithms such as Min-min, Max-min and Sufferage in terms of time makespan, which is the total completion time for all tasks. Each of these three algorithms selects a job from a set of tasks, calculates its completion time on each existing processor and assigns it to a resource iteratively.

In the Ant-Colony approach each job submitted to the Grid invokes an ant and the ant searches through the network to find the best node to deliver the job to. Ants leave information related to the nodes they have seen as pheromone in each node which helps other ants to find lighter resources more easily. In the particle swarm approach, each node in the network is considered to be a particle and tries to optimize its load locally by sending or receiving jobs to and from its neighbors. This process being done locally for each node, results in a move toward the global optima in the overall network.

In agent based approach, load balancing is carried out with combination of intelligent agents and multi-agent approaches [13]. At the global grid level, each agent is acting as a high-level representative of a grid resource and acts as a service provider of high performance computing power. Agents are organized into a hierarchy by assigning different roles. There are three roles in the system: Broker, Coordinator and Agent. They cooperate with each other to discover available resources for tasks using service advertisement and discovery. The hierarchical model can help when scalability problem arises. When the number of agents increases, the hierarchy can help in processing many activities in a local domain and does not have to rely on some central agents. Still their architecture of agents incorporates a central agent which coordinates the hierarchy at the highest level.

3.1 Heuristic and Evolutionary Methods

The following heuristic and evolutionary algorithms are discussed in detail towards their functionality and suitability for efficient and effective load balancing.

3.1.1 Genetic Algorithm

A Genetic Algorithm is a biologically inspired optimization and searching technique. It mimics the behavior of evolution of simple, single celled organisms. It is particularly useful in situations like the solution space to be searched is huge, making sequential search, computationally expensive and time consuming. GA is a type of guided random search technique, able to find efficient solutions which are not absolute optimal but use reasonable amount of time and resources within the constraints in a variety of cases. Effectiveness or quality of a GA can be judged by its performance against other historical known techniques in terms of solutions found, and time and resources used to find the solutions. GA has shown extremely effective in problems ranging from optimizations to machine

learning.

Some common terminologies used in GA are explained below. The relationships between various units used in the GA system are shown in Fig.1.

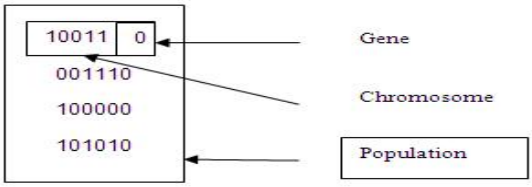

Fig.1. Genetic Algorithm System.

Population: A collection of chromosomes.
Chromosome: A collection of genes. Represents a solution or a parameter set for the particular problem.
Gene: A basic entity in GA, a single gene represents a particular feature or Characteristic of individuals in the GA system.
Allele: A value associated with a gene. Example: biologically speaking, a gene for eye color may have an allele value of 'brown'.
Locus: The position of a gene in the chromosome.

The gene in GA takes on a value from an alphabet of size r € Z+. Each chromosome / string then consists of a series of n genes:

$$s = \{b1,b2,\dots ,bn\} \quad (1)$$

resulting in a solution space of size rn. The most common and generally most effective string representation is the binary alphabet {0, 1}. In this case, each gene takes on a possible value of 0 or 1:

$$bi \in \{0,1\}\ i = 1,2, \dots , n \quad (2)$$

where by the solution space is now of size 2n. In most cases, the use of binary gene results in longer chromosome length. At the same time, this results in more genes available to be exploited by the GA, resulting in better performances in many cases. The evolution of the GA population from one generation to the next is usually achieved through the operators: selection, crossover, and mutation.

3.1.1.1 Selection

Selection is the process of selecting chromosomes from the current generation for processing to the next generation. Highly fit chromosomes are usually given a higher chance of being selected more often, thereby producing more offspring for the next generation. Some common techniques to achieve this objective include:

- Fitness-proportionate selection.
- Rank selection.
- Tournament selection.
- Elitism; this method is used in conjunction with the other selection methods.

3.1.1.2 Crossover

Once chromosomes are selected, crossover is applied to the chosen individuals. The crossover operator usually operates on two individuals / parents to produce two children. It ensures that characteristics of each parent are inherited in the children. Common crossover methods that are readily used for binary gene include one-point, two-point and uniform crossover. Most appropriate crossover method is selected depends on the particular problems, chromosome encoding and fitness function used.

In uniform crossover gene exchanges can occur at any position on the chromosome. For example, if we have two parents 000000 and 111111, then to produce two offspring we go through each of the six genes of one of the parent. For each gene, there is a probability that this gene will be exchanged with the other parent. As such, if an exchange happens at position 2 and 5, then the two offspring produced will be 010010 and 101101.

3.1.1.3 Mutation

While the crossover operator works on a pair or more of chromosomes to produce two or more offspring, the mutation operator works on each individual offspring. The mutation operator helps prevent early convergence of the genetic algorithm by changing characteristics of chromosomes in the population. Such changes in the chromosomes also results in the GA's ability to jump to far away solutions, hopefully to unexplored areas of the solution space. The mutation rate of the chromosomes should not be set too high, as too much mutations has the effect of changing the guided random search of the GA to a purely random search.

3.1.1.4 Genetic Algorithm Framework

The GA used for efficient solutions to the load balancing problem, considering the points discussed above, is shown below.

Algorithm 1 (Genetic Algorithm for Load Balancing):

Input : Parameters for the GA.
Output: Population of solutions, P. Each of these solutions can be used as a task schedule.
Begin
    Initialize the population, P.
    Evaluate P.
    While stopping conditions not true do
        Select Elite in P consisting of k (1 < k < population size) best individuals.
        Apply Selection from individuals in P to create Pmating, consisting of (population size - k) individuals.
        Crossover Pmating.
        Mutate Pmating.
        Copy the whole individuals of Pmating to P, replacing the worst (population size - k) individuals in P.
        Evaluate P.
        If escape condition true
            Then Escape.
    End While
End.

The parameters for the GA are shown in Table 1. First, a fixed number of tasks are considered for scheduling. This is to stop the scheduler from scheduling an excessive number of tasks at once. This also limits the computation time required by the scheduler. In this study, a window size of eight times, the number of nodes available is used. The window represents all the tasks in the queue that will be considered for scheduling. If the number of tasks in the queue is less than the window size then TaskSet consists of all the tasks in the queue. Otherwise TaskSet consists of all the tasks in the window. Note that a task remains in the scheduler's queue until it is completed. Tasks that have been sent or are executing on computing nodes still remain in the queue, as they may be reallocated to another node.

TABLE 1
GENETIC ALGORITHM PARAMETERS.

Window size w	8 × (number of nodes)
Population m	2 × (size of TaskSet)
Best cloned σ	5% of m
Tournament selection ps	0.8
Two-point crossover pc	0.8
Gene mutation pm	0.0005
Evolution period T	200
Max stale period Tstale	20
Escape mutation pmm	0.2

3.1.2 Tabu Search

Tabu search is also an evolutionary algorithm through its intimate relation to scatter search and path relinking [10],[11]. Tabu Search provides compete solutions for numerous computational studies. The improvements can be quite dramatic for some cases. It moves from one solution to a neighboring solution. In choosing the next solution, Tabu Search uses memory and extra knowledge endowed about the problem. A basic Tabu Search algorithm is shown below.

Algorithm 2 (Basic Tabu Search for Load Balancing):

```
Input  :    Parameters for the TS.
Output: A feasible solution to the problem.
Begin
      Generate an initial solution s.
      While stopping conditions not true do
            Select next solution neighboring s.
            Update memory.
      End While
End
```

For a given problem to be solved using Tabu Search, three information need to be defined: a set V of feasible solutions, a neighborhood structure N(s) for a given solution s € V, and a tabu list, TL. As shown in Algorithm 2, an initial solution s is chosen from the set of feasible solution V. This initial solution is usually chosen in a random fashion. Once an initial solution is chosen, the algorithm goes into a loop that terminates when one or more of the stopping conditions are met. In the next step of Algorithm 2, the next solution si+1 is selected from the neighbors of the current solution si, si+l € N(si). In a basic Tabu Search, all possible solutions s € N(si) is considered, and the best solution is chosen as the next solution si+1. The use of such selection rule may result in the algorithm going in circles. At the very worst, the algorithm will go back and forth between two solutions. To avoid this, memory is incorporated into Tabu Search.

3.1.2.1 Tabu list

The algorithm keeps a list of tabu solutions in a memory. The length of this tabu list may be varied, and a longer list will prevent cycles of greater length k. However, it may be impractical to incorporate such memory. Simpler type of memories is usually incorporated to remember the last k moves made. A move for a solution si can be viewed as changes to get to a new solution si+1 € N(si) and reverse also. The tabu search can then keep a tabu list of the last k reverse moves made, and avoid making these moves.

Such a list is not a perfect replacement for a solution list. The use of a move list does not guarantee that no cycle of length < k will occur. In other cases, its use results in a restrictive search pattern - an unvisited solution may be ignored because a move to that solution is in the tabu list. Sometimes the use of a move list results in both a loss of information and a more restrictive search pattern. To partially overcome the restriction imposed by using a move list, a tabu search usually incorporates aspiration conditions. If a tabu move leads to a solution that is better than the best solution found so far, then it should be selected. The important feature of tabu search is search intensification and diversification. For this purpose, a more elaborate memory structure needed to take into account of recency, frequency, and quality of solutions and moves made so far.

3.1.2.2 Search Intensification

In search intensification, exploration is concentrated on the vicinity, or neighbors of solutions that have historically been found to be good. During this process a modified fitness function is used to encourage moves that have historically been found to be good. Reward value may be added to the fitness function so that certain moves or solution's features become more attractive. Search intensification is usually carried out over a few numbers of iterations. If the process is not able to find a better solution than the best one found so far, a search diversification is usually carried out.

3.1.2.3 Search Diversification

The search diversification process attempts to spread out the search process to unvisited regions, and encourages moves that are significantly different from those that have been found. For this purpose, a modified fitness function is used with penalties applied to certain moves. For example, penalties may be given to frequently made moves, or common solution's features.

3.1.2.4 Tabu Search Framework

The Tabu Search implementation for the grid load balancing problem is graphically depicted in Fig.2. In the initialization phase, a starting solution is created. To this end, a Sufferage algorithm is used to generate the starting solution and the tabu search algorithm explores the solution space from this starting solution. During the exploration process, the best move is selected, and is then put into a tabu list, to prevent excessive cycling of solutions. Moves table is used instead of solutions table due to the impracticality of using solutions table.

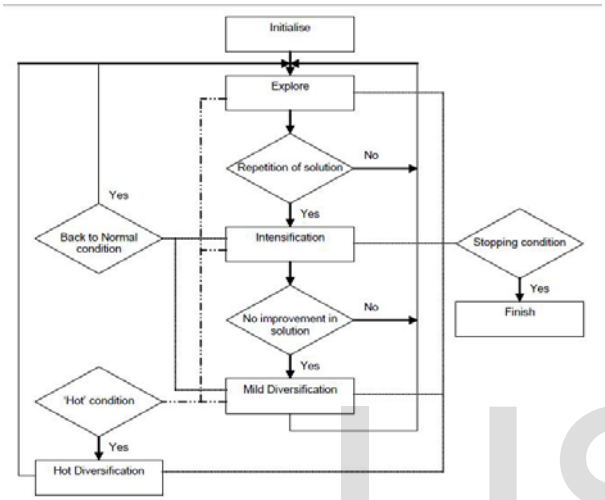

Fig.2. Tabu Search Procedure

Here, the tabu list keeps a record of the movement of a task to a particular computing node. Moves are kept in the tabu list for a period of between TLmin and TLmax. The exact period for each move is chosen randomly and uniformly from [TLmin, TLmax]. Further to this is an aspiration condition, whereby a move that's tabu will be accepted if it improves the best solution found so far. To complement the exploration process, search intensification and diversification are implemented. The inclusion of each type improves the performance of tabu search for the load balancing problem.

In search intensification, a modified fitness function is used to decide the next move to be made. In this implementation, moves with a history of good score are rewarded, so that such moves are likely to be chosen during the intensification period. To do this, a score is kept for each move made: the best move for the current iteration that gives a better solution than the solution from the previous iteration is considered good; the difference in value is then recorded. The reward for a move is then the average of the calculated sum. At the start of this intensification period, the tabu list is reset, and the period that moves stay in the tabu list is shortened to [TLI, min, TLI, max]. The search intensification is triggered when the best solution found is repeated. This intensification procedure would normally last for TI iterations.

Two types of search diversification procedure can be implemented which are mild and hot. A modified fitness function is again used in the mild diversification procedure. Here, frequently made moves are penalized, so that such moves are less likely to be chosen during this procedure. The ratio of the move to the total moves performed is calculated. The penalty for a move is then calculated as the fitness value ratio. This procedure is invoked when the search intensification is not able to improve the solution (Fig.2). The procedure lasts for a maximum period of TDm.

Finally, the hot diversification procedure is global in nature, and is executed when the solution found is not improved within a period Tstale. The procedure resets the tabu list, and a new random solution is generated. The move counts used in the mild diversification procedure are also reset. In the event that the conditions for search intensification and hot diversification are both satisfied, the search intensification takes precedence. Further, at the beginning of each search intensification and mild diversification, the countdown for Tstale is reset.

Each of the search intensification and mild diversification procedure would normally last for TI and TDm respectively. However, the procedure will exit and go back to normal exploration if a better solution is found. Finally, a global stopping condition is used to stop the tabu search. Here, the tabu search can be stopped after 200 iterations. An iteration can include a normal exploration, a search intensification, or a search diversification. Table 2 shows the tabu search parameter values used in the experiment:

TABLE 2
TABU SEARCH PARAMETERS.

Parameters	Values
Tabu period TLmin	5
Tabu period TLmax	10
Tabu period TLI,min	2
Tabu period TLI,max	5
Intensification period TI	13
Diversification period TDm	13
Max stale period Tstale	20

3.1.3 Ant Colony Optimization Algorithm

Ants are social beings with structured colonies based on their individual behavior. For computational purposes, it is relevant that the way they find paths between food sources and anthill. While walking ants place some amount of pheromone on the ground. Ants smell pheromone and when choosing their way, they tend in probability to the paths marked with stronger pheromone concentrations. When the time passes, the pheromone concentration decreases. Repeating the same behavior, they compose optimized trails that are dynamically defining and they use to find food sources and their nest. This environment is very similar to the Grid and can be used in a very direct way.

Algorithm 3 (Ant Colony Optimization for Load Balancing):

Input : Parameters for the ACO.
Output: Optimal solution to the problem.

Begin

```
Initialize the pheromone
While stopping criterion not satisfied do
        Position each ant in a starting node
        Repeat
                For each ant do
                        Chose next node
                        by applying the
                        state transition rate
                End For
        Until every ant has build a solution
        Update the pheromone
End While
End
```

Each edge between node (r, s) has a distance or cost associated δ(r, s) and a pheromone concentration τ (r, s). Equation (1) is the state transition rule, that is a probabilistic function for each node u, that has not been visited by each placed ant on node r.

We also have there the η(r, s) that is the reverse tau function.

The parameter β determine the relevance of the pheromone concentration compared with the distance or cost, δ(r, s) , and always β > 0.

The pheromone concentration on (3) is applied in each edge of the systems, for a global pheromone updating rule.

$$Pk(r,s) = \frac{[\tau(r,s)][\eta(r,s)]\beta}{\Sigma[\tau(r,s)][\eta(r,s)]\beta} \quad \text{……..} \quad (3)$$

$$\tau(r,s) = (1-\alpha)\tau(r,s) + \Sigma\Delta\tau k(r,s) \quad \text{----------} \quad (4)$$

Where α is the pheromone evaporation factor between 0 and 1. And Δτk(r, s) is the reverse of the distance or cost done by ant k, if (r,s) is its path and is 0 if it is not in the path.

The ACO-Grid flavour modified from the original ACO is as follows

Where,
Pij(t) -Probability to move along the path (i→j).
τij(t) -Trail intensity of the edge(i,j).
ηij(t) -Visibility (1 / distanceij).

- Every Grid request is an ant, when it finds its file object, the ant died.
- The Grid replies routing is done with traditional methods.
- ACO-Grid does not use global updating.
- Every time a request is processed on a Grid site, information is updated for all the site connections.
- The Grid distance or cost is defined on (5) as a function of network latency lt and bandwidth bw.

$$\delta(r,s) = ltr,s * c1 + (MAXBW - bw\ r,s) * c2 \quad \text{----(5)}$$

On (5) c1 and c2 are coefficients which balance the relative relevance between latency and bandwidth, they will fit with the bandwidth and latency values of the specific Grid infrastructure, and also fit with their measure relationship (ms. and MB/s.). For example: c1 = 1 and c2 = 0.2. Latency is always a constant, and bandwidth has a variable behaviour depending on sockets allocations and number of network request in a specific moment. MAXBW is the highest bandwidth of the Grid infrastructure. ACO-Grid realizes the performance due to its features like no control traffic, distributed optimization, localization and selection services, and autonomous management of each node [14].

3.1.4 Modified Ant Colony Algorithm

The modified ant colony optimization is used to solve large complex problems [15]. It requires grid scheduling to achieve high performance which is a complex problem. Hence better scheduling in grid systems can be achieved using heuristic approaches. The basic Ant algorithm involves Transition Probability and Pheromone Updating Rule. The modified ant colony algorithm has changed the basic Pheromone updating rule of original ant colony algorithm to improve the performance. The improved pheromone updating rule is given by (6).

$$\tau ij(t)new=[\{(1-)/(1+\rho)\}*\tau ij(t)old]+[\{\rho/(1+\rho)\}*\Delta\tau ij(t)] \quad \text{--------} \quad (6)$$

Where
τij(t) - Trail intensity of the edge(i,j).
ρ - Evaporation rate.
Δτij(t) - Additional pheromone when job moves from scheduler to resource.

(4)

The ants build a solution with the information stored in the pheromone trail and the best heuristic function. The probability of selecting job j to schedule next is given by the following (7). In the equation, α defines the relative weight given to the pheromone information and β defines the relative weight given to the heuristic information. If α is set to zero, then heuristic information is only used and the ants effectively perform a probabilistic search. If β is set to zero, pheromone information is only used.

The probability selection is obtained as

$$Pij(t)k = [\tau ij(t)]\alpha*[\eta ij(t)]\beta/\Sigma u\in allowed(k)\ [\tau iu(t)]\alpha*[\eta iu(t)]\beta \quad \text{…….} \quad (7)$$

The modified ant colony algorithm is as follows:

Algorithm 4 (Modified Ant Colony Optimization for Load Balancing):

Input : Parameters for the ACO.
Output: Improvised solution to the problem.

Begin

Initialize the pheromone
While stopping criterion not satisfied do
Position each ant in a starting node
Repeat
For each ant do
Chose next node by applying the state transition rate
Pij(t)k = [τij(t)]α*[ηij(t)]β /ΣuЄallowed(k) [τiu(t)]α*[ηiu(t)]β
End For
Until every ant has build a solution
Update the pheromone
τij(t)new=[{(1-)/(1+ρ)}*τij(t)old] + [{ρ/(1+ρ)}*Δτij(t)]
End while
End

3.1.5 Particle Swarm Optimization

Particle Swarm Optimization technique is also an evolutionary algorithm employed in many optimization and search problems due to its simplicity and ability to tackle these problems successfully [16],[17],[18],[19]. PSO optimizes an objective function by iteratively improving a swarm of solution vectors, called particles, based on special memory management technique. Each particle is modified by referring to the memory of individual swarm's best information. Due to the collective intelligence of these particles, the swarm is able to repeatedly improve its best observed solution and converges to an optimum.

Algorithm 5 (Particle Swarm Optimization for Load Balancing):

Input : Parameters for the PSO.
Output: Optimal solution to the problem.

Begin
Initialize the swarm from the solution space
Repeat
Evaluate fitness of individual particles
Modify gbest, pbest and velocity
Move each particle to a new position.
Until convergence or a stopping condition is satisfied.
End

The working principle of PSO is shown in the Fig.3

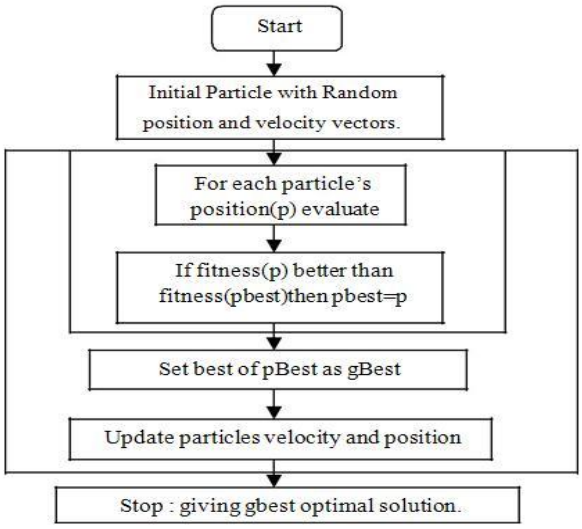

Fig.3. Flow chart for Particle Swarm optimization

3.1.5.1 PSO Algorithm

Particle Swarm Optimization is an algorithm modeled on swarm intelligence which finds a solution to an optimization problem in a search space, or models and predicts social behavior in the presence of objectives. The PSO is a stochastic and population-based computer algorithm which evaluates a solution in the form of a fitness function. A communication structure or social network is defined with nodes with their neighbors to interact with each other. Then a population of individual nodes defined as random guesses at the problem solutions is initialized. These individuals are candidate solutions which are known as the particles. An iterative process is carried out to evaluate the fitness to improve these candidate solutions is set in motion and remember the location where they had their best success. The individual's best solution is called the particle best (pbest) or the local best. Each particle makes this information available to their neighbors. They are also able to see where their neighbors have had success or not. Movements through the search space are guided by these successes up to the convergence of their population on a problem solution by the end of the trial.

Each particle represents a candidate solution which is based on the position of a particle is influenced by the best position visited by itself by its own experience and the position of the best particle in its neighborhood by the experience of neighboring particles. When the neighborhood of a particle is the entire swarm with the best position then it is the global best (gbest) of the particle which is shown in Fig.4. The performance of each particle measured using a fitness function varies problem to problem. Each Particle in the swarm is represented by its current position and velocity.

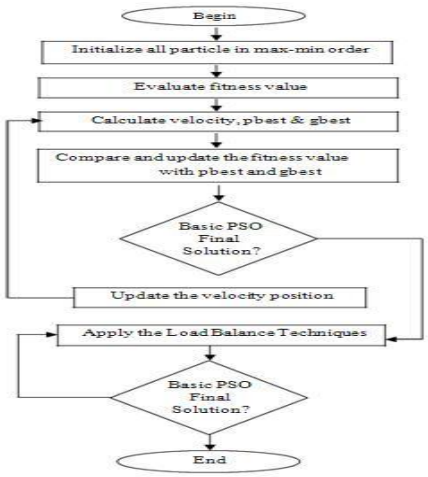

Fig.4. Flow chart for Particle Swarm optimization for Grid Environment

3.1.6 Enhanced Particle Swarm Optimization

Optimal task scheduling will enable the effective balancing of load. Representing a solution for task scheduling is one of the most important issues in grid environment PSO performance is tied with the solution representation. It is defined one particle as a possible solution in the population. The dimension n corresponds to n tasks. The position vector of each particle makes transformation about the continuous position. The Smallest Position Value (SPV) rule is used to find a permutation corresponding to the continuous position xik.

For the n tasks and m resources problem, each particle represents a reasonable scheduling scheme. The position vector xik has a continuous set of values and it is the position value of ith particle with respect to the nth dimension. sik is the sequence of task of ith particle in the processing order.

Then the operation vector rik is defined by the equation, R= sik mod m. The Initial population of particles is constructed randomly for PSO algorithm. The initialized continuous position values and velocities are generated by the formula,

$$Xk0 = xmin + (xmax - xmin) * r \quad \text{----- Eq (8)}$$

where, xmin = - 0.4 and xmax = 4.0 and r is the random number between 0 and 1.

$$Vk0 = Vmin + (Vmax - Vmin) * r \quad \text{----- Eq (9)}$$

where, Vmin = - 0.4 and Vmax = 4.0 and r is the random number between 0 and 1.

With these parameters enhanced PSO algorithm is described below.

Algorithm 6 (Enhanced Particle Swarm Optimization for Load Balancing):

Input : Parameters for the PSO.
Output: Improvised solution to the problem.

Begin
Initialize the contents for this PSO algorithm.
Define the active resource and the list of tasks.
Set the dimension as the number of tasks.
Initialize position vector and velocity vector of each particle randomly.

$$Xk0 = xmin + (xmax - xmin) * r$$
$$Vk0 = Vmin + (Vmax - Vmin) * r$$

Apply the SPV rule to find the permutation for the tasks.
Evaluate each particle in the swarm using an objective function.
Find the best fitness value and set the global best value.

Repeat
Update iteration variable.
Update inertia weight.
$w = wend + (wstart - wend) * \beta$
where, $\beta = (1 / 1 + (\alpha x / xmax)$
Update velocity.
$Vik+1 = wVik + c1\ rand1(\ ) x (pbesti-sik) + c2\ rand2(\ ) x (gbest-sik)$
and update velocity of each particle.
Update Position.
$Sik+1 = Sik + Vik+1$
and update position of each particle.
Apply the SPV rule to find the permutation.
Update personal best, by evaluating the particle.
Update global best.
Until number of iteration exceeds the maximum number of iteration
End

3.2 Agent based Approach

Software agents are used as powerful high-level abstraction for modeling of complex software systems and pure decentralization. This approach is used for building large-scale distributed systems with highly dynamic behavior. Agents are used for the implementation of resource management system for meta and grid computing. Each agent is responsible for resource scheduling and load balancing across multiple hosts / processors in a local grid is enabled with multiple agents. The agent couples application performance data with iterative heuristic algorithms to dynamically minimize task makespan and host idle time, whilst meeting the requirements for each task. The algorithm is based on an evolutionary process which will absorb system changes like addition or deletion of tasks and changes in the number of processors available in a local grid.

At the global level, each agent is a representative of a grid resource and acts as a service provider of high performance computing power. Agents are organized into a hierarchy and cooperate with each other to discover available grid resources for tasks using a peer-to-peer mechanism for service advertisement and discovery.

3.2.1. Agent Structure

Each agent is implemented for managing hosts / processors of a local grid resource and scheduling incoming tasks to achieve local load balancing. Each agent provides a high-level representation of a grid resource and characterizes these resources as high performance computing service providers. The architecture of each agent is illustrated in Fig.5 and explained below.

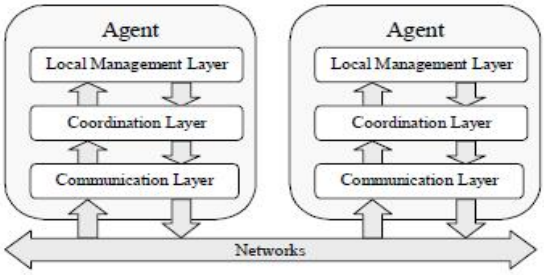

Fig.5. Agent Architecture.

- Communication Layer. Agents in the system must be able to communicate with each other or with users using common data models and communication protocols. This layer provides an agent with an interface to heterogeneous networks and operating systems.
- Coordination Layer. This layer decides how the agent should act on the request according to its own knowledge. If an agent receives a service discovery request, it must decide whether it has related service information.
- Local Management Layer. This layer performs functions of an agent for local grid load balancing. This layer is responsible for submitting local service information to the coordination layer for agent decision making.

3.2.2. Agent Hierarchy

Agents are organized into a hierarchy in a higher level global grid environment which is shown in Fig.6. The broker is an agent that heads the whole hierarchy. A coordinator is an agent that heads a sub-hierarchy. A leaf-node is actually termed an agent. The broker and coordinators are also agents which differentiate themselves with special positions in the hierarchy. All the agents have the same functionality despite their different positions. The broker does not have any more priorities than coordinators or agents. The hierarchy of homogeneous agents provides a high-level abstraction of a dynamic grid environment. New agents will join the hierarchy or existing agents will leave the hierarchy at any time. The hierarchy exists only logically and each agent can contact others as long as it has their identities.

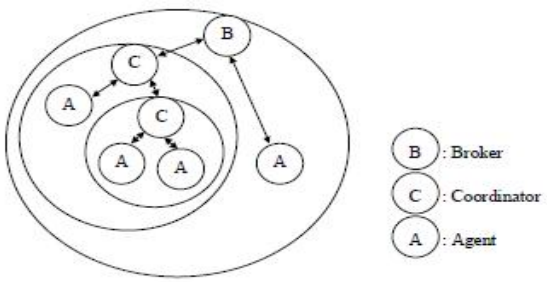

Fig.6. Agent Hierarchy.

The hierarchical model can address the problem of scalability to some extent. When the number of agents increases, the hierarchy may lead to many system activities being processed in a local domain through which it may scale well and does not need to rely on central agents. An agent can act as all abstractions like client, server and match maker, which provides a simple and uniform abstraction of the functions in the grid management. The service information provided at each local grid resource can be advertised throughout the hierarchy and agents can cooperate with each other to discover available resources.

In this study, local grid load balancing is performed in each agent using AI scheduling algorithms which is similar to that of AppLeS, Ninf and Nimrod. The main advantage of GA scheduling used is the quality of service (QoS) and multiple performance metrics support. This work also focuses on the cooperation of local grid and global grid levels of management and scheduling. Globus toolkit, is becoming a standard for grid service and application development, which is based on web services protocols and standards. Condor-G and Nimrod/G use the Globus toolkit to integrate with the grid computing environment though a centralized control structure is applied in both implementations. Legion is developed using an object-oriented methodology that provides similar functionalities to the Globus. Agents are used to control the query process and to make resource discovery decisions based on internal logic.

Agent-based grid management is also used in JAMM and NetSolve, where centralized broker/agents architecture is developed with peer-to-peer service advertisement and discovery to achieve global grid load balancing. The agent-based approach can provide a clear high-level abstraction of the grid environment that is extensible and compatible for integration of future grid services and toolkits.

In this work, GridSim toolkit has been used for exercising all the algorithms under simulation towards effective load balancing and better utilization of Grid.

4 GridSim: Grid Modeling and Simulation Toolkit

The GridSim toolkit provides a comprehensive facility for simulation of different heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains and distributed computing systems such as clusters and grids. Resource brokers perform resource discov-

ery, selection, and aggregation of a diverse set of distributed resources for an individual user. Each user has his own private resource broker and it can be targeted to optimize for the requirements and objectives of its owner. Whereas schedulers have complete control over the policy used for allocation of resources. All users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users.

4.1 Features

Salient features of the GridSim toolkit include the following:

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space- or time -shared mode.
- Resource capability can be defined in the form of MIPS.
- Resources can be located in any time zone.
- Leisure time can be mapped on resource's local time to model local workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- No limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time or space-shared. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded and analyzed using GridSim statistics analysis methods.

4.2 System Architecture

A layered and modular architecture for grid simulation is employed to leverage existing technologies and manage them as separate components [20]. A multi-layer architecture and abstraction for the development of GridSim platform and its applications is shown in Fig.7.

- The first layer is concerned with the scalable Java's interface and the runtime machinery, called Java Virtual Machine (JVM), whose implementation is available for single and multiprocessor systems including clusters.
- The second layer is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer.
- The third layer is concerned with modeling and simulation of core Grid entities such as resources, information services, application model, uniform access interface and primitives application modeling and framework for creating higher level entities. The GridSim toolkit focuses on this layer that simulates system entities using the discrete-event services offered by the lower-level infrastructure.
- The fourth layer is concerned with the simulation of resource aggregators called grid resource brokers or schedulers.
- The final layer focuses on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies, heuristics, and algorithms.

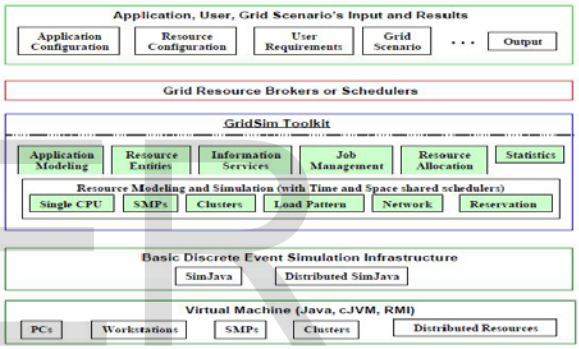

Fig.7. A Modular Architecture for GridSim Platform and Components.

5 Results from Literature

In the first set of experiment, the genetic algorithm (GA), tabu search (TS), along with Best-fit, Min-min, Max-min, Sufferage, and Random algorithms were applied to ten different networks and applications with the characteristics mentioned earlier [21]. The relative makespan percentages are calculated from the geometric mean of the makespans as obtained from each algorithm. From the graph (Fig.8) it can be seen that tabu search (TS) has the lowest makespans, followed closely by genetic algorithm (GA). Best-fit and Random gives the worst results of all the seven algorithms considered. Best-fit does not perform as well because it does not re-schedule tasks once they have been assigned, which is essential with varying background workloads in the computing nodes. We can see TS has a lower geometric mean of the maxspan, much lower average percentage deviation, and higher average rank than the other algorithms. Both the GA and TS has average percentage deviation that is at least two times smaller than the other algorithms (Fig.9). TS also have an average rank compared with other algorithms except GA.

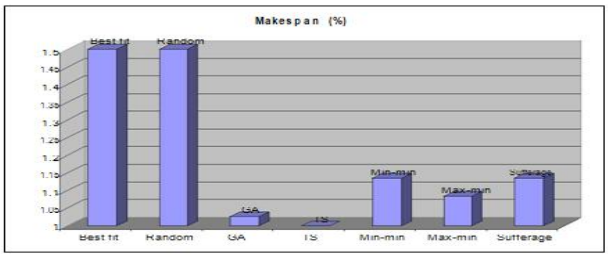

Fig.8. Results for GA, TS and other associated basic algorithms

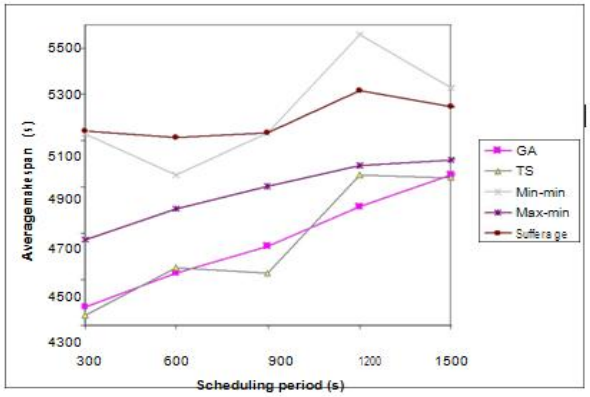

Fig.9. Effect of scheduling period on the geometric mean of the makespans

The modified Ant colony algorithm is a best suited method for tracking problem sets. The above approach simulated using GridSim toolkit and was found to be working efficiently and effectively [15]. Experimental test carried out for a varied range of input set to ascertain the efficiency of the algorithm. From the results it is clearly evident that the modified Ant colony algorithm offers better optimization a very fast rate. Table 3 shows the amount of tasks considered for each period of execution. The results are tabulated for interval of every 10 tasks, starting from 10 tasks to 100 tasks respectively.

TABLE 3
COMPARISON BETWEEN EXISTING AND MODIFIED ACO WITH NO. OF TASKS AND PERIOD OF EXECUTION

Number of tasks involved	Modified ACO (% of time taken for execution)	ACO (% of time taken for execution)
10	0.32	0.59
20	0.72	0.81
30	0.42	0.62
40	0.63	0.75
50	0.81	0.89
60	0.49	0.70
70	0.67	0.80
80	0.62	0.76
90	0.50	0.66
100	0.73	0.84

The following graph (Fig.10) shows the level of improvement with modified approach when compared with normal ACO.

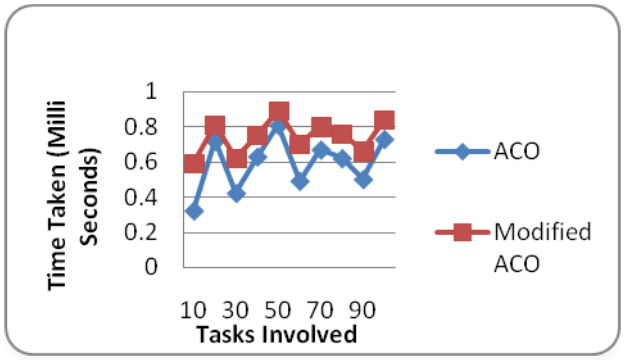

Fig.10. Comparison between Existing and Modified ACO with No. of tasks and period of execution

PSO and enhanced PSO algorithms are simulated in ALEA Grid simulation tool kit [22]. Experiments have been carried out with 5 iterations with the above said parameters. The experimental results show that the enhanced PSO algorithm is able to get the better schedule as shown in Table 4.

TABLE 4
COMPARISONS BETWEEN EXISTING AND ENHANCED PSO METHODS WITH ITERATIONS AND INERTIA VALUES

ITERATION	PSO Method e-(α x / xmax)	Enhanced PSO Method (1 / 1 + (α x / xmax))
1	1.4992	1.4992
2	1.4988	1.4985
3	1.4982	1.4977
4	1.4976	1.4970
5	1.4970	1.4962

The following graph (Fig.11) shows comparisons modified PSO and existing PSO method based on iterations and inertia values. The curve obviously shows the improvement.

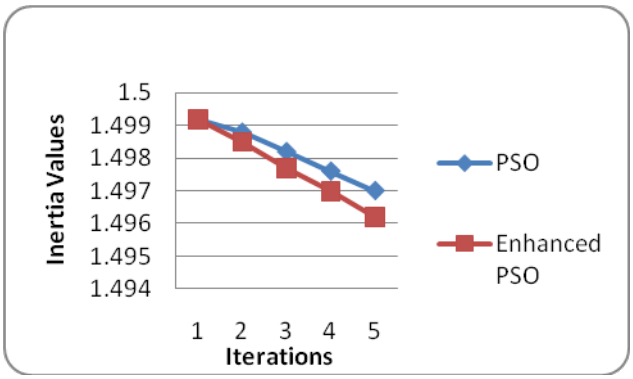

Fig.11. Comparison between iterations and inertia value.

For the agent based approach, experiments were conduct-

ed with 20 agents with centralized and distributed approaches [13]. The total task execution time decreases when the number of agents and grid resources increases. It is clear from the graph (Fig.12) that the centralized strategy leads to a bit better load balancing results, since tasks finish in a less time under the centralized control. This is more obvious when the number of the agents increases.

It is reasonable that a centralized strategy can achieve a better scheduling, because full service advertisement leads to full knowledge on the performance of all grid resources. However, under a distributed mechanism, each agent has only up-to-date information on its neighboring agents, which limit the scheduling effect. But for the dynamic environment, distributed approach is only suitable and can cater the information for load balancing.

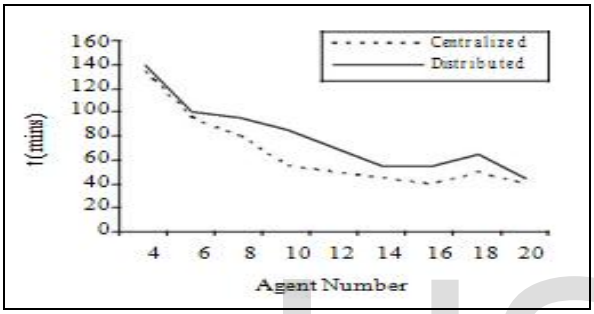

Fig.12. Comparison of total application execution time between the centralized and distributed strategies

6 Conclusion

This paper addressed the use of Genetic Algorithm and Tabu Search to solve the grid load balancing problem. Results of the experiment show that the two methods can be effectively used for grid load balancing. GA and TS shows similar performance results, and performs better than the Best-fit, Random, Min-min, Max-min, and Sufferage algorithms. As the scheduling period is decreased resulting in more scheduling runs, the performance gains from GA and TS increases. One drawback of the GA and TS algorithms is that they incur extra storage and processing requirement at the scheduling node. However they may be overcome by the ever-decreasing costs of storage and processing power.

It has been convincingly proved in the recent research papers that task scheduling on computational grids is best solved by heuristic approach. The project tried to cover the state-of-the-art studies about one such heuristic namely Ant Colony Optimization algorithm and its application to grid systems. The experimental results prove that the improved ant colony algorithm has effective role on grid scheduling. The modified pheromone updation rule makes the ant colony algorithm to work more efficiently than the original ant colony algorithm. Thus grid scheduling problems can be easily overcome using modified ant colony algorithm.

PSO is applied for task scheduling problem on computational grids. Task scheduling algorithms based on PSO algorithm can be applied in computational grid environment. Each particle represents a feasible solution. This project aimed at generating an optimal schedule so as to complete the tasks in a minimum time as well as utilizing the resources in an efficient way. The performance of the enhanced PSO has been improved compared with the existing approach. The future work may include other hybridization techniques to further minimize the execution time.

Agent based approach addresses grid load balancing issues using a combination of intelligent agents and multi-agent approaches. For local grid load balancing, the iterative heuristic algorithm is more efficient than the first-come first- served algorithm. For global grid load balancing, a peer-to-peer service advertisement and discovery technique is proven to be effective. The use of a distributed agent strategy can reduce the network overhead significantly and make the system scale well rather than using a centralized control, as well as achieving reasonable good resource utilization and meeting application execution deadlines.

All huristic and agent based approaches have been studied with the simulated environment GridSim. GA and TS are simple in nature and works in a restricted environment. They provide near by solutions from unpredictable data sets and their huge combinations and possibilities. They are suitable for the centralized approaches and resources are ready in the hand in the sense static requirements. Compared with GA and TS, ACO and PSO are doing better load balancing due to its dynamic and parallel in nature. ACO and PSO provide solutions on the way and better will be selected among them in the sense global solutions are selected from local solutions. Compared with ACO and PSO, Agent based approach has better performance due to its pure decentralization. Local scheduling is taken care by the agents and global load balancing is done with multiple agents by the synchronization, coordination and decentralization or distribution in nature.

7 Future Directions

Hybridization of the techniques will improve the load balancing and utilization of the grid further. Further experiments will be carried out using the bigger grid test bed in real time scenario since a large deployment of the system is impossible due to the absence of a large scale grid test bed. The grid modeling and simulation environment is under development to enable performance. All the heuristic methods are to be exercised with real time environment rather than simulated environment. The scalability of the agent system is to be investigated when thousands of grid resources and agents are involved. The next generation grid computing environment must be intelligent and autonomous to meet requirements of self management. Related research topics include semantic grids and knowledge grids [23],[24]. The agent-based approach described in this work is an initial attempt towards a distributed framework for building such an intelligent grid environment. Future work includes the extension of the agent framework with new features, like automatic QOS negotiation, self-organizing coordination, semantic integration, knowledge-based reasoning, and ontology-based service brokering

REFERENCES

[1] S. Salleh and A. Y. Zomaya, "Scheduling In Parallel Computing Systems: Fuzzy and Annealing Techniques", USA: Kluwer Academic Publishers, 1999.
[2] H.-C. Lin and C. S. Raghavendra, "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)", IEEE Transactions on Software Engineering,, pp. 148-58,1992.
[3] N.G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems", Computer, pp. 33-44.,1992
[4] C. Kim and H. Kameda, "An algorithm for optimal static load balancing in distributed computer systems", IEEE Transactions on Computers, vol. 41, pp. 381-84, 1992.
[5] M. A. S. Al, "Commercial applications of Tabu Search Heuristics", IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 2391-95, 1998.
[6] V. R. Alvarez, E. Crespo, and J. M. Tamarit, "Assigning students to course sections using Tabu Search", Annals of Operations Research, vol. 96, pp. 1-16, 2000.
[7] G. Barbarosoglu and D. Ozgur, "A Tabu Search algorithm for the vehicle routing problem", Computers & Operations Research, vol. 26, pp. 255-70, 1999.
[8] A. Hertz and W. D. de, "Using Tabu Search techniques for graph coloring", Computing, vol. 39, pp. 345-51, 1987.
[9] A. Hertz, E. Taillard, and D. Werra, "Tabu search in Local search in combinatorial optimization", E. Aarts and J. K. Lenstra, Eds. Chichester: John Wiley & Sons Ltd., pp. 121-36, 1997.
[10] F. Glover, J. P. Kelly, and M. Laguna, "Genetic Algorithms and Tabu Search: Hybrids for optimization", Computers & Operations Research, vol. 22, pp. 111-3, 1995.
[11] F. Glover, M. Laguma, and R. Marti, "Fundamentals of scatter search and path relinking", Control and Cybernetics, vol. 29, pp. 653-84, 2000.
[12] Paritosh Kumar, "Load Balancing and Job Migration in Grid Environment", M.E Thesis, 2009
[13] Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd, "Grid Load Balancing Using Intelligent Agents", 2001
[14] Vıctor Mendez Munoz1 and Felix Garcıa Carballeira, "Ant Colony Optimization for Data Grid Replication Services", Technical Report RR-06-08. DIIS. UNIZAR, 2004.
[15] P.Mathiyalagan, S.Suriya, Dr.S.N.Sivanandam, "Modified Ant Colony Algorithm for Grid Scheduling" (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 02, pp.132-139, 2010.
[16] Lei Zhang, Yuehui Chen, Bo Yang "Task Scheduling Based on PSO Algorithm in Computational Grid", Sixth International Conference on Intelligent Systems Design and Applications, 2006.
[17] M. Fikret Ercan "A hybrid particle swarm optimization approach for scheduling flow-shops with multiprocessor tasks", International Conference on Information Science and Security, 2008.
[18] Brian Ivers ,Gary G.Yen "Job Shop Optimization Through Multiple Independent Particle Swarms" IEEE Congress on Evolutionary Computation, 2007.
[19] Shih-Tang Lo, Ruey-Maw Chen, Der-Fang Shiau and Chung-Lun Wu "Using Particle Swarm Optimization to Solve Resource-constrained Scheduling Problems" IEEE Conference on Soft Computing in Industrial Applications, 2008.
[20] Rajkumar Buyya and Manzur Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing"
[21] Riky Subrata Albert Y. Zomaya Bjorn Landfeldt,"Artificial Life Techniques for Load Balancing in Computational Grids", 2006
[22] P.Mathiyalagan, U.R.Dhepthie, Dr. S.N.Sivanandam, "Grid Scheduling Using Enhanced PSO Algorithm", (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 02, pp.140-145, 2010.
[23] H. Zhuge, "Semantics, resource and grid, Future Generation Computer Systems", 20(1), pp.1-5, 2004.
[24] H. Zhuge, "China's E-Science Knowledge Grid Environment", IEEE Intelligent Systems 19(1), pp.13-17, 2004.