Comparative Analysis on Heuristic Based Load Balancing Algorithms in Grid Environment

Manthan Brahambhatt
M.E.Student
Computer Department,
St. Francis Institute of Technology
Mumbai, India

Dakshata Panchal
Assistant Professor
Computer Department,
St. Francis Institute of Technology
Mumbai, India

***Abstract*—The solution to increasing demands of computational resources for completing computational jobs were found in terms of grid computing. Grid computing is all new technology in the field of distributed systems, parallel computing and cluster computing. Grid environment is a collection of multiple resources, geographically widely distributed at multiple locations to reach over a common goal. This environment allows multi owner resources to solve large scale applications. The major concern and challenge in grid computing environment is to balance the load of jobs on resources over the entire network. The criticality lies in the issue of network aware load balancing algorithms that needs to be quick and dynamic. Taking dynamicity as a consideration, heuristic based algorithms are well suited for such applications. The objective of this paper is to analyze different heuristic based load balancing algorithms, such as Genetic Algorithm, ant Colony Optimization, Particle Swarm Optimization and Tabu Search.**

***Keywords*— Grid computing, load balancing, genetic algorithm,tabu search, particle swarm optimization, ant colony optimization.**

I. INTRODUCTION

Grid computing 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. A computational grid is a hardware and software infrastructure that provides a dependable, consistent and inexpensive access to high end capabilities. This grid environment supports sharing and coordinated use of resources, independently of their physical type and location that share the common goal. All resources come together to work upon a common task and achieve the maximum performance. This grid environment allows the use of geographically widely distributed and multi-owner resources to solve large scale applications.

In grid computing there are two major parties [1], namely resource consumers (users) and resource providers (facilitators). Resource consumers are the one who submit various applications to the environment, whereas resource providers are the one who share their resources with the consumers and help them to achieve their goal. With respect to the parties there are two objective functions set. These two objective functions can be classified as Application centric and Resource centric. Application centric: It functions at application level. It aims to optimize the performance of each individual application. E.g.: makespan. Resource centric: It functions at system level. It aims to optimize the performance of resources. E.g.: throughput, utilization etc.

There are two different types of grid computing environment

a.) Homogeneous environment

b.) Heterogeneous environment

Focusing on heterogeneous environment, heterogeneity exist in two categories

- Networks that are used to interconnect computational resources may differ significantly in terms of their communication protocols and bandwidth.
- Computational resources may differ and have different computer architectures, memory size, different hardware, CPU speed, RAM capability etc. and may also have different software applications such as operating system, grid management software and so on.

This grid computing environment is used to solve large scale applications such as meteorological simulations, data intensive applications, research of DNA sequences etc. The best examples of grid computing are Garuda by C-DAC India, seti@home by university of California, Berkeley.

A typical distributed system involves the use of geographically widely distributed network. This widely distributed network involves different nodes connected to each other in the network. Each node possesses some amount of initial load which represent amount of work to be performed. This work needs to be evenly balanced in the whole network and this is called *load balancing*.

This paper is organized as follows. In section II, different methods of performing load balancing for grid environment is introduced. In section III, heuristic based approach of load balancing is explained. In section IV, comparative analyses of all algorithms are done. Finally, in the next session, conclusion and future work is given.

II. DIFFERENT METHODS OF LOAD BALANCING FOR GRID ENVIRONMENT

As per the literature survey we got to know that load balancing mainly deals with distributing set of independent jobs among all the computing nodes of the grid such that the jobs are uniformly distributed over the entire grid computing environment and none of the nodes are under loaded or overloaded [2]. There are two methods of performing load balancing, *Static method*: In Static load balancing algorithms, it assumes that a priori information about all the characteristics of jobs the computing nodes, the computing nodes and the communication network is known. It has two major disadvantages,

- Workload distribution of many applications cannot be predicted before program execution.
- Assumptions made may not apply to distributed environment.

Because of static approach cannot respond to dynamic runtime environment, it may lead to load imbalance on some nodes and significantly increase the load balancing time.

Dynamic Method: In dynamic load balancing algorithms, it attempts to use the runtime state information to make more informative decisions in sharing the system load. Though dynamic algorithms have higher runtime complexity than static algorithms, they give better performance than latter. These kinds of algorithms are preferred for heterogeneous network where network elements may vary in capacity or number at runtime.

Dynamic load balancing algorithms are further classified into two different approaches.

a) Centralized Approach: In this method only one node acts as the central controller for the whole grid computing environment. Job allocation to the all slave nodes is done by this controller. It is a simple approach and beneficial when the communication cost is less significant. This kind of approach is mainly used for small grid computing environment. The main drawbacks of this kind of approach are, it limits the scalability of the grid by becoming bottleneck and Failure of central controller node causes entire system to fail.

b) Decentralized Approach: In this approach all nodes get involved in making load balancing decision. They are scalable and have better fault tolerance. This approach is preferred because elements of network may vary in capacity or number during run time. The major drawback of this approach is that it increases the communication overhead to a large extent.

   This approach is again classified into three different categories: *Receiver initiated* where heavily loaded nodes take the initiative, *Sender initiated* where lightly loaded nodes takes the initiative, *symmetrically initiated* combines the advantage of both the above types.

III. HEURISTIC BASED APPROACH FOR LOAD BALANCING ALGORITHM

This section describes different types of heuristics and evolutionary based algorithms. These search techniques are used to find solution to optimization and search problem [3]. These techniques are inspired from evolutionary biology and apply features such as inheritance, selection, crossover and mutation. These methods have been proved to work better as compared to classical algorithms. We will look four different evolutionary based load balancing algorithms those are Genetic Algorithm, Tabu Search, Particle Swarm optimization, Ant Colony Optimization.

Genetic Algorithm:

A Genetic Algorithm is a biologically inspired optimization and search technique [3] [4]. The behavior of Genetic Algorithm mimics the evolution of simple, single celled organisms. This algorithm is particularly useful in situations where the solution space to be searched is so huge, making sequential search, time consuming and computationally very expensive. It is a type of guided random search technique, able to find efficient solutions in variety of cases.

The evolution of the GA population from one generation to the next is usually achieved through the use of three operators that are fundamental in GA: *selection, crossover, and mutation*.

Selection: It is a process of selecting chromosomes from the current generation for processing to the next generation.

Crossover: Once chromosomes are selected, crossover is applied to the chosen individuals. The crossover operator usually operates on two individuals or parents to produce two children. It ensures that characteristics of each parent are inherited in the children.

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.

Algorithm:

```
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 $P_{mating}$, consisting of (population size-$k$) individuals.
        Crossover $P_{mating}$.
        Mutate $P_{mating}$.
        Copy the whole individuals of $P_{mating}$ to $P$, replace the worst (population size-$k$) individuals in $P$.
        Evaluate $P$.
        If Escape condition true then Escape
    End While
End
```

Tabu search:

Tabu search (TS) was first proposed in its current form by Glover. It 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. One main ingredient of Tabu search (TS) is the use of *adaptive memory* to guide problem solving. One may argue that *memory* is a necessary component for 'intelligence', and intelligent problem solving. Tabu search uses a set of strategies and learned information to 'mimic' human insights for problem solving, creating essentially an 'artificial intelligence' unto itself—though problem specific it may be. In its most basic sense, a Tabu search can be thought of as a local search procedure, whereby it 'moves' from one solution to a 'neighboring' solution. In choosing the next solution to move to, however, Tabu search uses memory and extra knowledge endowed about the problem. A basic Tabu search algorithm is shown below.

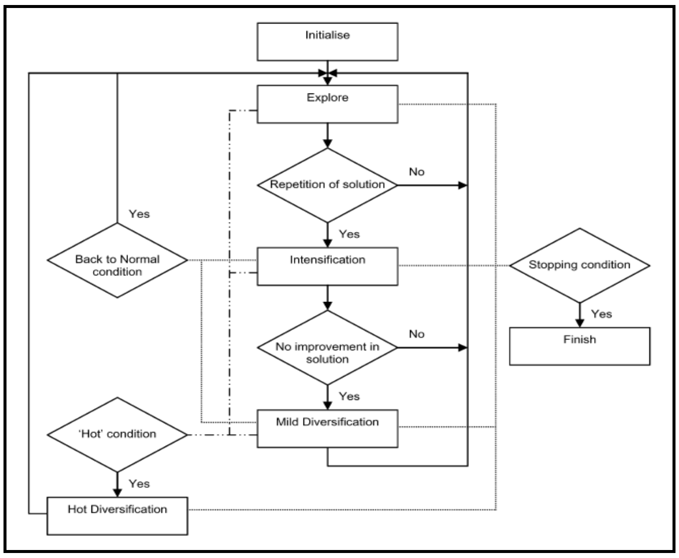

Fig 1. Tabu Search Framework

```
Algorithm
Input  : Parameter for the Tabu Search.
Output: A feasible Solution to the Problem.
Begin
        Generate an initial solution s.
                while stopping condition not true do
                        select next solution neighboring s.
                        update memory.
End
```

Three things are most important in Tabu search Framework i.e. Tabu list, Search Intensification, Search Diversification. For detail view please follow figure 1. of Tabu Search Framework.

Particle swarm optimization:

Particle swarm optimization (PSO) is an algorithm modelled on swarm intelligence, that finds a solution to an optimization problem in a search space, or model and predict social behaviour in the presence of objectives [5]. The PSO is a stochastic, population-based computer algorithm modelled on swarm intelligence. Swarm intelligence is based on social-psychological principles and provides insights into social behaviour, as well as contributing to engineering applications. The particle swarm optimization algorithm was first described in 1995 by James Kennedy and Russell C. Eberhart.

Algorithm:

- Step 1.:
  - Initialization: Set the contents for this PSO algorithm. Define the active resource and the list of tasks. The dimension of PSO algorithm is the number of tasks. Initialize position vector and velocity vector of each particle randomly by using equation

$$X_K^0 = x_{min} + (x_{max}-x_{min})* r$$
$$V_K^0 = V_{min} + (V_{max}-V_{min})*r$$

Where x_{max}, x_{min}, V_{max}, V_{min} are any random values and r is random number between 0 and 1. Apply SPV rule to find the permutation for the tasks

- Step 2.:
  - Update iteration variable
- Step 3.:
  - Update inertia weight
    $w = w_{end} + (w_{start}-w_{end}) * \beta$ where
    $\beta = (1/ 1+(\alpha x / x_{max}))$
- Step 4.:
  - Update velocity
    $V_i^{k+1} = wV_i^k + c_1rand_1() \ x \ (pbest_i-s_i^k) + c_2rand_2() \ x \ (gbest-s_i^k)$
    And update velocity of each particle
- Step 5.:
  - Update position
    $S_i^{k+1} = S_i^k + V_i^{k+1}$
    And update particle of each position
- Step 6.:
  - Apply the SPV rule to find the permutation.
- Step 7.:
  - Update personal best, by evaluating the particle.
- Step 8.:
  - Update global best
- Step 9.:
  - Stopping criterion. If the number of iteration exceeds the maximum number of iteration, then stop, otherwise go to Step 2.

Ant Colony Optimization:

It is a meta-heuristic using artificial ant to find desirable solutions to difficult combinatorial optimization problems [6]. The behaviour of artificial ants is based on the traits of real ants as described above, plus additional capabilities that make them more effective, such as a memory of past actions. Each ant of the "colony" builds a solution to the problem under consideration and uses information collected on the problem characteristics and its own performance to change how other ants see the problem.

Algorithm:

- Step 1.:
  - Initialize the value of α ,β ,ρ ,Δ, N, T, RU_i and also set pheromone trails for each resource.
- Step 2.:
  - Select the next task *t*.
- Step 3.:
  - Determine the transition probability (load) of each resource R_j as:

$$p_j(t) = \frac{[\tau_j(t)]^\alpha * [\eta_j]^\beta}{\sum_r [\tau_r(t)]^\alpha * [\eta_r]^\beta}$$

```
▪ Step 4.:
    – Find resource $R_i$ with high transition probability among all resources:
          $p_i(t) = \max_{l \in N} p_l(t)$
    – i.e. resource $R_i$ is having minimum load
▪ Step 5.:
    – Assign task $t$ to $R_i$ .
▪ Step 6.:
    – Set T = T - 1

▪ Step 7.:
    – Check whether any task completion or failure reported. If no, go to Step 11.

▪ Step 8.:
    – If (task completion at any resource $R_i$ ) then Increase pheromone of $R_i$ as:
          $\tau_i(t) = \tau_i(t)+\Delta$
    – reporting $R_i$ as lightly loaded.

▪ Step 9.:
    – $RU_i = RU_i + FT_i^t$
▪ Step 10.:
    – If (task failure at any resource $R_i$ ) then decrease pheromone of $R_i$ as:
          $\tau_i(t) = \tau_i(t) - \Delta$
    – reporting $R_i$ as heavily loaded
▪ Step 11.:
    – If (T>0) then go to Step 2.
▪ Step 12.:
    – For each resource $R_i$ , $1 \le i \le N$
      Compute $RU_i = \frac{RU_i}{\sum_{K=1}^{N} RU_K}$
    – Print resource utilization of $R_i$.
```

IV. Comparative Analysis

When we compare all the algorithms with their properties like working and stopping conditions the results are as follows in the following table

Table1. Comparison of Properties of Heuristics and Evolutionary Based Algorithm

PROPERTIES	GA	TS	PSO	ACO
WORKING	Fitness function	Fitness function and moves made	Objective function	Pheromone value
ITERATION/ INERTIA VALUE	Evolution period	Iterations	Iteration value ↑ Inertia value ↓	No of tasks
AGENTS	Not present	Not present	Present	Present
STOPPING CONDITIONS	Evolution period	Iterations	Iterations	No of tasks

In all the following algorithms there are two things very important and common in all of them.

1) If number of task increases then execution time also increases
2) If number of agent increases then execution time decreases

Whenever we increase number of tasks keeping all resources same and number of resources also same then resource utilization per resources also increases and hence execution time increases though overall effect of makespan of that grid remains same.

a) Comparing with respect to number of tasks, If number of task increases then

Table2. Comparison with respect to number of tasks

Properties	GA	TS	PSO	ACO
Resource utilization	↑	↑	↑	generalized
Execution time	↑	↑	↑	↑

Whenever we increase number of resources keeping number of tasks same then task distribution is done in such a manner that resource utilization per resource decreases and hence decreasing resource time effecting overall makespan time and decreases makespan too.

b) With respect to number of resources, if number of resources increases then

Table3. Comparison with respect to number of resources

Properties	GA	TS	PSO	ACO
Resource utilization	↓	↓	↓	↓
Execution time	↓	↓	↓	↓

V. Conclusion And Future work

Load balancing in grid computing is really important to be taken into consideration for any grid environment. It has been convincingly proved that load balancing and task scheduling is best solved by heuristics and evolutionary algorithms. These heuristics and evolutionary algorithm s give far better results than traditional load balancing algorithms like receiver broadcasting algorithms, bidding approach, dynamic scheduling using weights etc. It may happen that these algorithms may take extra processing requirement and incur extra storage space but eventually performs better load balancing than traditional algorithms.

In future, hybridization of the techniques will improve the load balancing and utilization of the grid further. This hybrid technique then can be implemented in the real world problem on large scale and then performance can be measured such that the next generation grid computing environment must be intelligent enough and autonomous to meet requirements of self-management.

References

[1] Belabbas Yagoubi, and Meriem Meddeber, "Distributed Load Balancing model for Grid Computing", Revue ARIMA journal volume. 12, pp.43-60, September 2010.

[2] Janhavi B., Sunil Surve, and Sapna Prabhu,"Comparison Of Load Balancing in a Grid", IEEE Computer Society, 2010 International

Conference on Data Storage and Data Engineering, pp. 20-23, doi:10.1109/DSDE.2010.13

[3] R. Rajeswari, Dr. N.Kasthuri ,"Comparative Survey on Load Balancing Techniques in Computational Grids ", International Journal of Scientific & Engineering Research (IJSER), Volume 4, Issue 9, September-2013 ISSN 2229-5518

[4] Riky Subrata, Albert Y. Zomaya, Bjorn Landfeldt "Artificial life techniques for load balancing in computational grids" Journal of Computer and System Sciences 73 (2007), Elsevier, doi:10.1016/j.jcss.2007.02.006

[5] Mr. P.Mathiyalagan, U.R.Dhepthie ,Dr. S.N.Sivanandam "Grid Scheduling using enhanced PSO algorithm", International Journal on Computer Science and Engineering(IJCSE), Vol. 02, No. 02, 2010, ISSN : 0975-3397, 140-145.

[6] Sandip Kumar Goyal , Manpreet Singh " Adaptive and Dynamic Load Balancing in Grid Using Ant Colony Optimization" International Journal of Engineering and Technology (IJET) ISSN : 0975-4024 Vol. 4 No 4 Aug-Sep 2012.