AN ENTROPIC OPTIMIZATION TECHNIQUE IN HETEROGENEOUS GRID COMPUTING USING BIONIC ALGORITHMS

Saad M. Darwish[1], Adel A. El-zoghabi[2] and Moustafa F. Ashry[3]

Information Technology Department, Graduate Studies and Research Institute, University of Alexandria. 163 Horreya Avenue, El-Shatby, 21526 P.O. Box 832, Alexandria, Egypt

ABSTRACT

The wide usage of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components have a deep impact on the way we use computers today, in such a way that these technologies facilitated the usage of multi-owner and geographically distributed resources to address large-scale problems in many areas such as science, engineering, and commerce. The new paradigm of Grid computing has evolved from these researches on these topics. Performance and utilization of the grid depends on a complex and excessively dynamic procedure of optimally balancing the load among the available nodes. In this paper, we suggest a novel two-dimensional figure of merit that depict the network effects on load balance and fault tolerance estimation to improve the performance of the network utilizations. The enhancement of fault tolerance is obtained by adaptively decrease replication time and message cost. On the other hand, load balance is improved by adaptively decrease mean job response time. Finally, analysis of Genetic Algorithm, Ant Colony Optimization, and Particle Swarm Optimization is conducted with regards to their solutions, issues and improvements concerning load balancing in computational grid. Consequently, a significant system utilization improvement was attained. Experimental results eventually demonstrate that the proposed method's performance surpasses other methods.

KEYWORDS

Grid Computing; Big Data; Bionic Algorithm; Load Balancing; Fault Tolerance; R-tree.

1. INTRODUCTION

Lately, the progressive expansion of large, complex, heterogeneous, and multi-dimensional data has put under spot the capacity of the existing data management software and hardware infrastructure. Nowadays, there exist many different types of data sources, such as sensors, scientific instruments, and the Internet, contributing to the data booming. The relatively slower development of new and efficient data models to process complex and large-scale data poses huge challenges that require straightaway attention from academia, research, and industry [1] [2]. As traditional data models, which are basically relational in nature, can't handle the today's data needs, a new technology in data science is produced that gives rise to a fast emergence of a wide range of non-relational data models, which, today, are popularly known as NoSQL and/or Big Data models [3]. Grid technology has emerged as a new era of large-scale distributed computing with high-performance orientation. Grid resource management is literally the process of identifying requirements, matching resources to applications, allocating those resources, scheduling and monitoring grid resources over time in order to run grid applications as efficiently as possible [4]. Resource discovery represents the first phase of resource management, whereas scheduling and monitoring is the next step. Scheduling process leads the job to the appropriate resource and monitoring process views the resources. Heavily loaded resources will act as server

of task, while lightly loaded resources will act as receiver of task. Task will migrate from heavily loaded node to lightly loaded node. As the resources are dynamic in nature, their load varies with change in configuration of grid resulting in a significantly influence of the load balancing of the tasks on grid's performance [5] [6].

Load balancing results in a low cost, dispensed scheme that balances the load among all the processors. Efficacious load balancing algorithms are basically important in enhancing the global throughput of grid resources. For applying load balancing in grid environment, multiple techniques, algorithms and policies have been introduced. Load balancing processes can be classified as centralized or decentralized, dynamic or static, and periodic or non-periodic [7]. Regarding the concept of fault tolerance, the main problems facing grid environments are *Feasibility, Reliability*, *Scalability, and Connectivity* [8] [9]. A fault can be tolerated depending on its behavior or the way of occurrence. To evaluate a fault tolerance in a system there are three different methods which are *Replication*, *Check-pointing, and Scheduling/Redundancy* [10].

Fractal transform based self similarity will always be a critical issue in the prospect of the cost of data storage and transmission times especially with the huge interest for images, sound, video sequences, computer animations and volume visualization [11]. A measurement of the inherent size of the data in a space can be represented by a fractal dimension of a cloud of points. This method has a significant disadvantage which is the long computing time, which can be reduced by several proposed methods. The most common approach for reducing the computational complexity is to organize the domain blocks into a tree-structure, which could lead to faster searching over the linear search [9]. This approach is able to reduce the order of complexity from *O(N) to O(log N)*. Instead of dividing space in some manner, it is also possible to group the objects in some hierarchic organization based on a rectangular approximation of their location. R-tree [12] is the most popular index structure for multi-dimensional data. Each type of R-tree could be distinctive in some aspects of performance such as query retrieval, insertion cost, application specific and so on. Accordingly the subsequent search in the domain pool is substituted by multi-dimensional nearest neighbor searching, run in logarithmic time.

In this paper a new method based on a balanced tree structure overlay over a grid network is proposed. Furthermore a fault tolerant technique, exploiting the replication of non leaf nodes to ensure the tree connectivity in presence of crashes is presented. This method is based on distributed R-tree with the concept of entropy of each node. Consequently, all ineffectual route paths will be discarded from the pool creating a more stable network. The proposed method improves the performance of the network and hence improves fault tolerance and load balancing algorithm when applying this approach to a grid computing networks. The rest of this paper is organized as follows: Section 2, mention a literature survey and related works. Section 3, presents the proposed method that improve both fault tolerance and load balance in grid computing, followed by experimental results and discussion in Section 4. In Section 5, conclusions of the present work are outlined.

2. RELATED WORK

At the service level of the grid software infrastructure, workload and resource management are two main functions provided. The problem of load balancing in grid computing is handled by assigning loads in a grid taking into account the communication overhead in gathering the load information. Load index is used as a decision aspect in the process of scheduling jobs within and among clusters. To solve this problem, a decentralized model for heterogeneous grid has been suggested as a collection of clusters by a ring topology for the Grid managers that are charged of managing a dynamic pool of processing elements, computers or processors [7].

Resource scheduling for tasks engages in the scheduling of a set of independent tasks [5]. Other works have been based on scheduling technologies using economic/market-based models [6]. As job scheduling is known to be NP-complete, the use of non-heuristics is an effective method that practically matches its difficulty. Single heuristic approaches for the problem include Local Search, Simulated Annealing, and Tabu Search [6]. In recent years, some new-type bionic algorithms are become hot research topics such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO) algorithm, Ant Colony Optimization (ACO) algorithm, Bees Algorithm (BA), Artificial Fish Swarm Algorithm (AFSA) [2]. In the Ant Colony algorithm, each job submitted issues an ant which searches through the network finding the best node to deliver the job to it. Ants catch information from each node they pass through as a pheromone in each node. This eventually helps other ants to find their paths more efficiently. In the particle swarm algorithm each node in the network individually acts as a particle which sends or receives jobs from its neighbors to optimize its load locally, resulting in a partially global optimization of the load within the whole network given time constraint limitations [17].

In [14], a hierarchical model of load distribution is introduced. It consists of a tree based Grid model together with three load balancing algorithms at various levels of the hierarchical model. The load balancing strategy is a hierarchical bottom up methodology where *intra-site* load is balanced first, followed by *intra-cluster* and finally *intra-grid* load balancing. To run complementary load balancing for batch jobs without specific execution deadlines, an agent-based self-organization scheme is proposed in [18]. In [19], a combination of intelligent agents and multi-agent scheme is applied for both global grid load balancing and local grid resource scheduling. Another approach proposed in [15], based on the fault tolerant hybrid load balancing strategy to reach job assignments with optimal computing node utilization and minimum response time.

In this paper, a tree based structure and the domain-range entropy is proposed to reduce the complexity of the grid computing network due to the similarity features with the tolerated and balanced R-tree index structure which can be run in logarithmic time. This will improve both load balancing and fault tolerance algorithm by enhancing connectivity, communication delay, network bandwidth, in addition to resource availability, and resource unpredictability.

3. THE PROPOSED METHOD

The optimization procedures depends on the complete adaptive proposed method is shown in Figure 1. The first step is to estimate Grid Computing Service (GCS) parameters then mapping this grid structure into DR-tree index structure enhanced by the Entropy method to reduce the completion time of the decision maker. Finally using threshold device to select the route path depending on two parameters load balance and fault tolerance controller. Migration controller is used to improve the fault tolerance and self-stabilizing controller is used to improve the load balance in cumulative condition way and this method is depend on first introduced in [20]. Three different optimization techniques applied on the herein proposed system to reach the optimum solution, namely: Genetic Algorithm, Ant colony optimization and Particle swarm optimization. Every user submits his computing jobs with their hardware requirements to the GCS. The GCS then replies to the user by sending the results after finishing the processing of the jobs.

At the first step GCS estimation will analyze the network parameters by determine the Three-level Top-Down view of the grid computing model depending on the method produced in [7] as shown in Figure 2. ***Level 0:*** *Local Grid Manager (LGM)*, the network is subdivided into geographical areas where any LGM manages a group of Site Managers (SMs). ***Level 1:*** *Site Manager (SM)*, every SM is assigned the management of processing elements (computers or processors) cluster which is dynamically configured (i.e., processing elements may join or leave

the cluster at any time). ***Level 2:*** *Processing Elements (PE)* any public or private PC or workstation can register within any SM to join the grid system and offer its computing resources to be exploited by the grid users. As soon as being adhered to the grid, a computing element triggers the GCS system which will return to the SM some information about its resources like CPU speed.

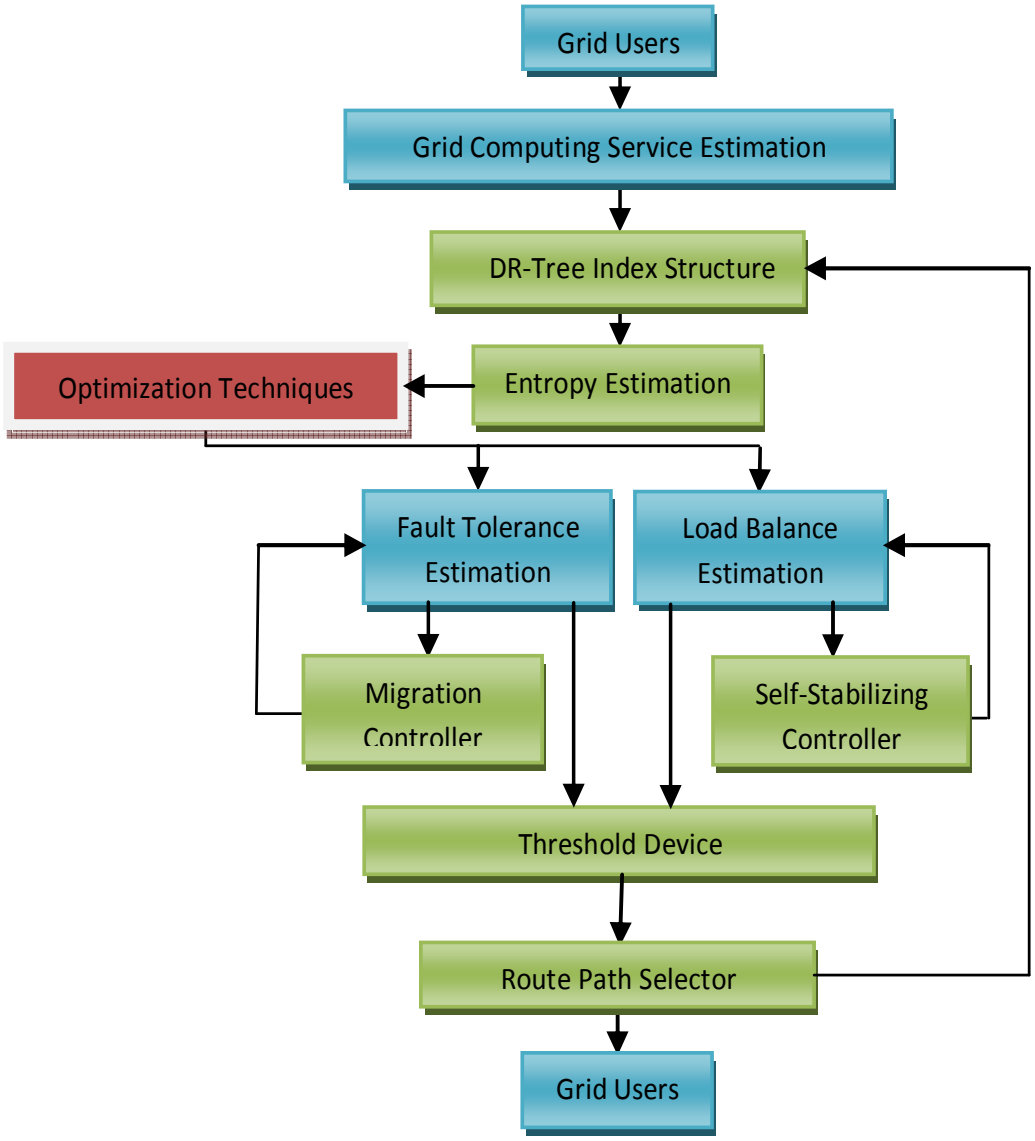

Figure 1. The complete adaptive proposed procedure.

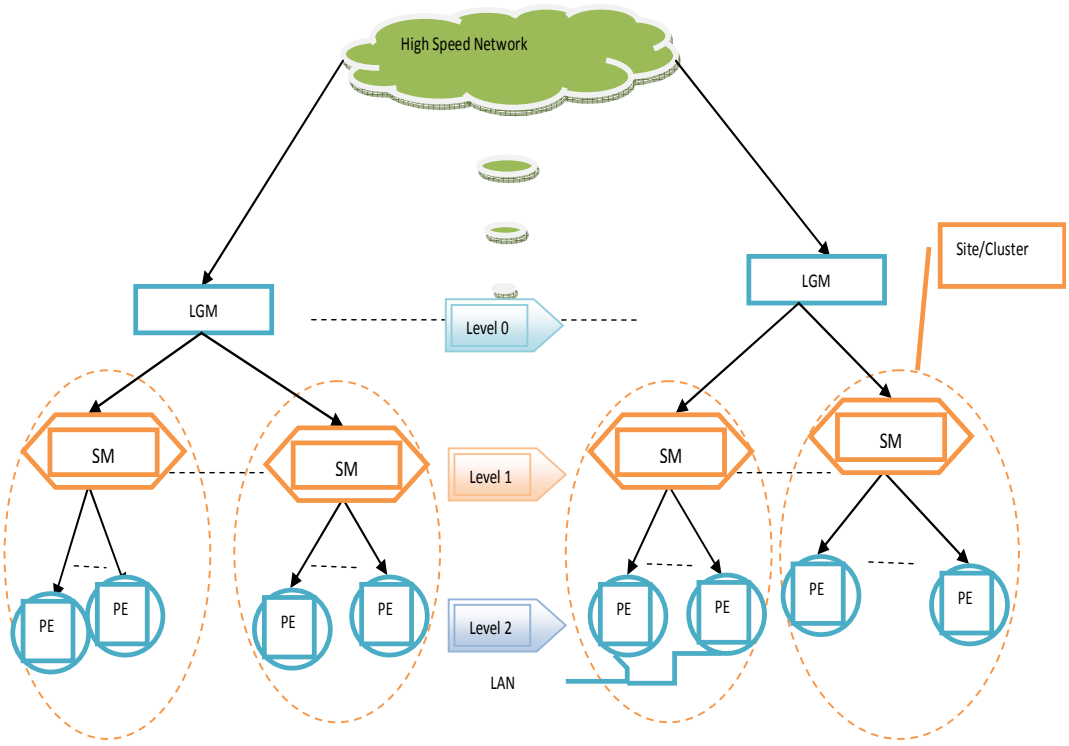

Figure 2. Grid computing model structure.

3.1. Distributed R-tree (DR-tree)

First we convert the tree model of grid computing nodes into DR-tree for the similarity features then dealing with the new tree with the DR-tree conditions. R-trees [13] are defined as height-balanced tree handling objects whose representation can be confined in a poly-space rectangle. In other words, R-tree can be depicted as a data structure capable of efficiently indexing a multi-dimensional space. R-tree has by the following structural properties: The root comprises 2 to *M* children, every internal node has from m to M children ($m \leq M/2$), and all leaves are equal in level.

Distributed R-trees (DR-trees) expand the R-tree index structures having similar self-organized nodes that are in a virtual balanced tree pave based on semantic relations. The structure maintains the R-trees index structure features: search time logarithmic in the network size and bounded degree per node. Physical machines connected to the system could be further referred to as *p-nodes* (shortcut for physical nodes). A DR-tree is defined as a virtual structure distributed over a set of *p-nodes*. In the following context, terms related to DR-tree will have the prefix: "*v-*". Consequently, DR-trees nodes will be denoted *v-nodes.* There are two distribution invariants properties in the implementation of DR-tree proposed in [16]. The main points in the composition of a DR-tree are the join/leave procedures. A *p-node* creates a *v-leaf* as soon as it is adhered to the system. After that, it contacts another *p-node* to embed its *v-leaf* in the existing DR-tree. During this insertion, some *v-nodes* may split as shown by Algorithm1.

Algorithm1

1: *Void* onSplit (*n:V_Node*)
2: **If** *n.is V_ Root* **then**
3: *Initiate new V_ Root = create n.V_Node.*
4: *Calculate n.v_ father = initiate new V_Root.*
5: **End if**
6: Calculate *m, where m = select n.v_children.*
7: Initiate then calculate *new V_Node = create m.V_Node.*
8: *Divide D = n.v_children.*
9: *Create new V_Node.v_children = D.*
10: *Create new V_ Node.v_ father then put n.v_ father into it.*

3.2. Entropy

According to Shannon, Entropy [11] is defined as a set of events S={ x_1 , x_2 ,... x_n }, where $p(x_i)$ $=p_i$ represents the probability of incidence of each event. These probabilities, P={ p_1 , p_2 ,... p_n }, are such that each $p_i \geq 0$, and $\sum_{i=1}^{n} p_i = 1$, which takes the form:

$$H(p_1, p_2, \ldots p_n) = H(s) = -\sum_{i=1}^{n} p_i \log p_i \qquad (1)$$

Entropy can be defined as the average self-information that is, the mean (expected or average) amount of information for an occurrence of an event x_i. In the context of message coding, entropy can be represented as the minimum bound on the bits average number for each input value. The function *H* has the following lower and the upper limits:

$$0 = H(1,0,0, \ldots 0) \leq H(p_1, p_2, \ldots p_n) \leq H\left(\frac{1}{n}, \frac{1}{n}, \ldots \frac{1}{n}\right) = log\ n \qquad (2)$$

Searching the pool of domain blocks is considered time consuming as good approximations are obtained when many domain blocks are allowed. This method consists of omitting the domain block having high entropy from the domain pool, and hence all the unnecessary domains will disappear from the pool to reach a higher production domain pool. This way will reduce the overhead of the network by decrease the number of searching nodes and then improve the performance of the grid computing networks. During GCS estimation, the grid manager will trigger Algorithm1 by picking up some parameter ε, subsequently Algorithm2 will be executed.

Algorithm2
1: **Step 1:** Initialization choose parameter ε ;
2: Divide the input grid into *n:V_Node*
3: **For** *(j =1; j ≤ n; j ++)* **{**
4: H = entropy (V_Node);
5: **If** *(H ≤ ε)*
6: **Step 2:** execute Algorithm 1**}**

3.3. Optimization Techniques

A load balancing algorithm should optimize the usage of available resources either in the Grid such as computational or data resources, in addition to time or cost related to these resources, etc. The Grid environment results in a dynamic search space and hence as this optimality represents a partial optimal solution that improves the performance. The proposed approaches optimize the parameter ε *by* combining genetic algorithm, ant colony and particle swarm optimization following the initialization process in the above mentioned algorithm2 to accomplish the optimum resource utilization. These three optimized methods were discussed earlier in details in [21].

3.3.1. Genetic Algorithm (GA)

GA is part of the group of Evolutionary Algorithms (EA). The evolutionary algorithms exploit the three key principles of the natural evolution: natural selection, reproduction, and species diversity, preserved by the differences of each generation with the previous. Genetic Algorithm usually works with a group of individuals, representing possible solutions of the task. Giving an individual assessment according to the desired solution, the selection principle is applied by using a criterion. The best-suited individuals create the next generation. The huge diversity of problems in the engineering sphere, equally as in other fields, implies the usage of algorithms from different type, with multiple characteristics and settings. Algorithm 3 shows the GA procedure used for select optimum parameter ε.

Algorithm 3

1: Initialize population of individual grid user jobs to both longest and smallest to fastest processor with 60 random schedules.
2: Evaluate the fitness of all individuals.
3: While termination condition not met do.
4: Select fitter individuals for reproduction at minimum execution time.
5: Crossover between individuals by two-point crossover.
6: Mutate individuals by simple swap operator.
7: Evaluate the fitness of the modified individuals that keeping relevant fitness.
8: Generate a new population
9: End while.
10: Initiate Authorized task for sequence *Si* For each processor *Pi*.
11: Concatenate sequences *Si*. A permutation sequence of tasks will be assigned to processors.

3.3.2. Ant Colony Optimization (ACO) algorithm

Ant Colony Optimization (ACO) is denoted as an analytical approach for dealing with optimization problems based on population met. ACO produces good solutions to the optimization problems by redistributing work among the nodes by means of ants. The ants traverse the grid network and leave the pheromones on the path. As soon as they reach the destination, the ants update the pheromones tables. The pheromone trail laying and following behavior of real ants represent the main source of ACO. The ants move from node to node, and consequently explore the information presented by the pheromones values and thus increasingly build the resultant solution. Algorithm 4 shows Ant Colony Algorithm procedure used for select optimum parameter ε.

Algorithm 4

```
1: Input: Parameters for the ACO.
2: Output: Optimal solution to the problem.
3: Begin
4:     Initialize the pheromone
5:     While stopping criterion not satisfied do
6:             Position each ant in a starting node
7:             Repeat
8:             For each ant do
9:             Chose next node
10:            by applying the state transition rate
11:            End For
12:            Until every ant has built a solution
13:            Update the pheromone
14:    End While
15: End
```

3.3.3. Particle Swarm Optimization (PSO) algorithm

As Particle Swarm Optimization technique is simple and able to successfully tackle these problems, it is also applied in many optimization and search problems. PSO optimizes an objective function by repeatedly amending a swarm of solution vectors, known as particles, depending on special memory management technique. Each particle is altered by attributing the memory of individual swarm's best information. The swarm can regularly amend its best observed solution and converges to an optimum by virtue of the collective intelligence of these particles. Every element ranges from 0 to 1 and vice versa. Additionally, each particle processes a D-dimensional velocity vector whose elements' range is [-Vmax,Vmax]. Velocities are interpreted according to probabilities that a bit will be in one state or the other. When the algorithm starts, a collection of particles with their velocity vectors are developed randomly. Then in some phase the algorithm's aim is to obtain the optimal or near optimal solutions referring to its predefined fitness function. The velocity vector is amended in each time step using two best positions, *pbest* and *nbest*, and then velocity vectors are used to update the position of the particles. *Pbest* and *nbest* are D-dimensional, with the elements composed of 0 and 1 exactly like particles position and represent the algorithm's memory. The personal best position, *pbest*, is the best position the particle has visited, whereas *nbest* is the best position visited by the particle and its neighbors since the first time step. At the time where all of the population size of the swarm becomes the neighbor of a particle, *nbest* is named global best (star neighborhood topology) and if the smaller neighborhoods are specified for each particle (e.g. ring neighborhood topology), then *nbest* can be called local. Algorithm 5 shows Particle Swarm Optimization Algorithm procedure used for select optimum parameter ε.

Algorithm 5

1: Randomly particle and position will store in a created and initialized *m x n* matrix.
Where
'm' denotes a number of node
'n' denotes a number of job
2: Calculate the estimated time to complete values of each node using Range Based Matrix.
3: Calculate the Fitness of each Particle and search for the Pbest and Gbest.
4: X_k is estimated and in case its value is greater than the fitness value of pbestk, pbestk is replaced with X_k. Where X_k is denotes the updated position of the particle.
5: Replace *nbest* with *pbest* because Fitness value of *nbest* is smaller than *pbest*.
6: Until to reach the max Velocity.
7: If it is satisfying the maximum velocity.
8: Stop.
9: End.
10: If the maximum velocity is not satisfied.
11: Update the Position Matrix.
12: Update the Velocity Matrix.
13:End.

3.4. Fault Tolerance

In the typical DR-tree performance [13], when a *p-node* fails, all its sub-trees are replaced in the non-faulty structure (*p-node* by *p-node*) to ensure the DR-tree invariants. The proposed algorithm guarantees fault-tolerance and also preserves the DR-tree design with its invariants by using the non leaf *v-nodes* reproduction. The pattern for *v-nodes* replication is represented as every *p-node* holds a replica of the *v-father* of its top *v-node* and on the other hand, the *p-root* holds no replica.

3.4.1. Fault Tolerance Estimation

This section investigates the cost of replication as presented in [9]. Then apply it to the proposed system. Let costs be defined as the cost of updating replicas in a split time. As each *v-node* has $M-1$ replicas and each update costs one message, we have:

$$cost_s = (m+2)(M-1) \quad (3)$$

A *p-node* associated with the system may change between 0 and $\lfloor log_m(N) \rfloor$ splits, adding its *v-leaf* to the *v-children* of another *v-node* that is denoted in the sequence the joined *v-node*. The latter has: Between *m* and *M v-children*; $\lceil log_m(N) - 1 \rceil$ *v-ancestors*; between $m-1$ and $M-1$ replicas. We will define hereafter an upper bound on the number of updates as long as each *v-node* has $M - 1$ replicas. A *v-node* can have m to *M v-children* and therefore has $M - m + 1$ possible numbers of *v-children*. The split will happen only when it has exactly *M v-children*. The probability for a *v-node* to split is *p,* where:

$$p = \frac{1}{M-m+1} \quad (4)$$

The probability for a *p-node* to generate *k* splits is the probability p_k that the associated *v-node* and its $k - 1$ first *v-ancestor* have literally *M v-children* while its *k-th* ancestor does not split. Hence:

$$p_k = p^k(1-p) \quad (5)$$

Let $cost_r$ denote the average cost of replicas updates when a *p-node* is associated with a DR-tree:

$$cost_r = p_o(M-1) + \sum_{k=1}^{\lfloor \log_m(N) \rfloor}(p_k\, k\, cost_s) \quad (6)$$

The first term represents the case where no splits are produced, i.e., $M-1$ replicas of the joined *v-node* are to be updated, while the second corresponds to the other cases. We could have identified the case where the *v-root* splits, with different probability as it has $M - 1$ possible *v-children*. However, for $m > 2$, this probability is smaller than *p* so in order to simplify the calculation, an upper bound is assumed.

3.4.2. Migration Controller

Reinsertion policy and *replication policy* [16] are used to evaluate DR-Tree insertion operations, which uses internal *v-nodes*. As soon as the crash of one non leaf *p-node* was generated for each DR-tree, the cost of system restoration in terms of number of messages and stabilization time is calculated based on both the reinsertion and replication policies.

(1) *Stabilization time:* The reinsertion mechanism is responsible of balancing the system in a number of cycles proportional to both, the tip of the participation graph and the level of the burst *p-node*. As the stabilization time is the time of the longest reinsertion, it is proportional to $log_m(N)$.

(2) *The message cost of the restoration phase:* denoted by the number of messages required to balance the system from a non leaf p-node burst. The costs are of different magnitude so with the reinsertion policy, the number of message distribution is much skewed which results in a high standard deviation.

3.5. Load Balance

As explained earlier in [14], every SM collects the information of any PE joining or leaving the grid system and then transmits it to its parent LGM. This means that a processing element only needs a communication when joining or leaving its site. The system workload between the processing elements will be balanced when using the collected information to efficiently utilize the whole system resources in order to minimize user jobs response time. By applying this policy, not only the connectivity will be improved but also, the system performance. This is accomplished by minimizing the communication overhead abstracted from capturing system information before making a load balancing decision. The following parameters will be defined for GCS model to formalize the load balancing policy:

1. **Job:** All jobs in the system will represented by a job Id, job size in bytes, and a number of job instructions.
2. **Processing Element Capacity (PEC_{ij}):** The *PEC* can be measured assuming an average number of job instructions using the PEs CPU speed and will be defined as number of jobs per second that the j^{th} PE at full capacity in the i^{th} site can be processed.
3. **Site Processing Capacity (SPC_i):** Defined as number of jobs that can be processed by the i^{th} site per second. Hence, the SPC_i can be measured by summing all the PECs for all the PEs managed the i^{th} site**.**
4. **Local Grid Manager Processing Capacity (*LPC*):** The *LPC* can be measured by summing all the *SPCs* for all the sites managed by that LGM and will be defined as number of jobs per second that LGM can be processed.

3.5.1. Load Balance Estimation

The load balancing policy as presented in [7] is a multi-level one and can be explained at each level of the grid architecture as follows:

A. Load Balancing At Level 0: Local Grid Manager

As mentioned earlier, the LGM sustains information about all of its responsible SMs in terms of processing capacity SPCs. LPC can be considered as the total processing capacity of a LGM obtained from the summation of all the SPCs for the total sites managed by that LGM. Depending on the total processing capacity of every site SPC, the LGM scheduler will control the workload balance between sites group members (SMs). Where *N* defined as the number of jobs arrived at a LGM in the steady state, the i^{th} site workload (S_iWL) is the number of jobs to be allocated to i^{th} site manager and is calculated as follows:

$$S_iWL = N \times \frac{SPC_i}{LPC} \qquad (7)$$

B. Load Balancing At Level 1: Site/Cluster Manager

Every SM stores some information about the PECs of all the processing elements in its cluster. The total site processing capacity SPC of all the processing elements is calculated from the sum of all the PECs included in that site. With the same policy used by the LGM scheduler to balance the load, the SM scheduler will be used where M is defined as the number of jobs arrived at a SM in the steady state. Under the policy of distributing site workload among the group of processing elements based on their processing capacity, the throughput of every PE will be maximized and also its resource utilization will be improved. On the other hand, the number of jobs to be allocated to i^{th} PE is defined as the i^{th} PE workload (PE_iWL) which is calculated as follows:

$$PE_iWL = M \times \frac{PEC_i}{SPC} \qquad (8)$$

3.5.2. Self-Stabilizing Controller

In order to calculate the mean job response time, we assume one LGM scenario as a simplified grid model. In this scenario, we will be examining the time elapsed by a job in the processing elements. Algorithm 6 will execute to calculate the traffic intensity ρ_{ij} and hence, the expected mean job response time:

Algorithm 6
1: Obtain λ,μ where, λ: is the external job arrival rate from grid clients to the LGM, μ: is the LGM processing capacity.
2: Calculate $\rho=\lambda/\mu$ is the system traffic intensity. For the system to be stable ρ must be less than 1.
2: **For** i= 1 to m
3: Calculate λ_i,μ_i where, λ_i is the job arrival rate from the LGM to the i^{th} SM which is controlled by that LGM, μ_i: is the i^{th} SM processing capacity.
4: Calculate the traffic intensity of the i^{th} SM, $\rho_i=\lambda_i/\mu_i$.
5: **For** j= 1 to n
6: Calculate λ_{ij},μ_{ij} where, λ_{ij}: is the job arrival rate from the i^{th} SM to the j^{th} PE controlled by that SM, μ_{ij}: is the j^{th} PE processing capacity which is controlled by the i^{th} SM.
7: Calculate the traffic intensity of the j^{th} PE which is controlled by i^{th} SM, $\rho_{ij}=\lambda_{ij}/\mu_{ij}$.
8: Calculate the expected mean job response time, $E[T_g]$.
9: End.
10: End.

The jobs arrive consecutively from clients to the LGM with a time-invariant Poisson process assumption. The inter-arrival times are not only independent, but also identically and exponentially allocated with the arrival rate λ *jobs/second*. Simultaneous arrivals are excluded. Each of PE in the non-static site pool will be represented by an *M/M/1* queue. Jobs arriving to the LGM will be naturally distributed on the sites regulated by that LGM with a routing probability, $PrS_i = \frac{SPC_i}{LPC}$.

Complying with the load balancing policy (LBP), if i is the site number $\lambda_i = \lambda \times PrS_i = \lambda \times \frac{SPC_i}{LPC}$. In a similar situation, the site i arrivals will be also naturally distributed on the PEs managed by that site with a routing probability $PrE_{ij} = \frac{PEC_{ij}}{SPC_i}$ according to the LBP, where j is the PE number and i is the site number $\lambda_{ij} = \lambda_i \times PrE_{ij} = \lambda_i \times \frac{PEC_{ij}}{SPC_i}$. As the arrivals to LGM are simulated to follow a Poisson process, the arrivals to the PEs will also follow a Poisson process. Let us consider that the service times at the j^{th} PE in the i^{th} SM are exponentially shared with fixed service rate μ_{ij} *jobs/second*. They will represent the PE's processing capacity (*PEC*) in our load balancing policy. The service control is First Come First Serviced. To calculate the expected mean job response time, we assume that $E[T_g]$ denotes the mean time spent by a job at the grid to the arrival rate λ and $E[N_g]$ denotes the total number of jobs in the system. Thus, the mean elapsed time by a job at the grid is given by equation 9 as follows:

$$E[N_g] = \lambda \times E[T_g] \qquad (9)$$

$E[N_g]$ can be measured by adding the mean number of jobs in every PE at all grid sites. So, $E[N_g] = \sum_{i=1}^{m}\sum_{j=1}^{n} E[N_{PE}^{ij}]$ where $i=1,2,\ldots,m$ is the number of site managers handled by a LGM, $j=1,2,\ldots,n$ is the number of processing elements handled by a SM and $E[N_{PE}^{ij}]$ is the mean number of jobs in a processing element number j at site number i. Because every PE is represented as an *M/M/1* queue, $E[N_{PE}^{ij}] = \frac{\rho_{ij}}{1-\rho_{ij}}$ where $\rho_{ij}=\lambda_{ij}/\mu_{ij}$, $\mu_{ij} = PEC_{ij}$ for PE number j at site number i. Referring to equation 9, the expected mean job response time is given by:

$$E[T_g] = \frac{1}{\lambda} \times E[N_g] = \frac{1}{\lambda} \times \sum_{i=1}^{m} \sum_{j=1}^{n} E[N_{PE}^{ij}] \quad (10)$$

Note that the stability condition for PE_{ij} is $\rho<1$.

3.6. Threshold Device

A novel *2-D Figure of Merit* is used to test the network performance. An example of the *2-D figure of merit* is shown in Figure 3. It divides the fault tolerance (FT) error – load balance (LB) error space into different performance conditions as follows: The FT estimation varies between three threshold intervals (FT error units are in numbers depending on the grid size) as follows: (1) From [0 … 30] which indicates a Good system. (2) From [30 … 60] which indicates a Medium system. (3) Above 60 which indicates a Bad system. The LB estimation varies between three threshold intervals (LB error units are in second depending on the mean job response time) as follows: (1) From [0 … 0.1] which indicates a Good system. (2) From [0.1 … 0.3] which indicates a Medium system. (3) Above 0.3 which indicates a Bad system. We can observe that Figure 3 is divided into 9 different areas as follows: (1) GG: this interval is good for both FT and LB estimation. (2) GM: this interval is good for FT estimation and medium for LB estimation. (3) GB: this interval is good for FT estimation and bad for LB estimation. (4) MG: this interval is medium for FT estimation and good for LB estimation. (5) MM: this interval is medium for both FT and LB estimation. (6) MB: this interval is medium for FT estimation and bad for LB estimation. (7) BG: this interval is bad for FT estimation and good for LB estimation. (8) BM: this interval is bad for FT estimation and medium for LB estimation. (9) BB: this interval is bad for both FT and LB estimation.

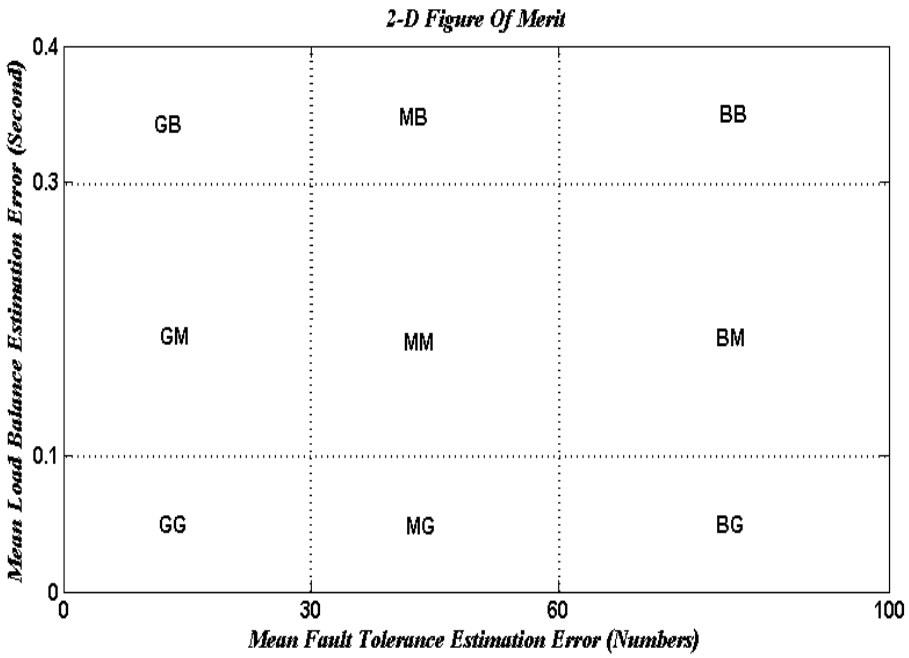

Figure 3**.** The proposed 2-D Figure of Merit.

Finally, to enhance fault tolerance estimation we decrease replication time and message cost and this will cause increase in probability of job completed. On the other hand, to enhance load balance estimation we decrease mean job response time and this will cause increase in number of *jobs/second*.

4. RESULTS AND DISCUSSION

A simulation model is built using MATLAB simulator to evaluate the performance of grid computing system based on the proposed algorithm. This simulation model consists of one local grid manager which manages a number of site managers which in turn manages a number of processing elements (Workstations or Processors). All simulations are performed on a PC (Dual Core Processor, 2.3 GHz, 2 GB RAM) using Windows 7 Professional OS. Figure 4 show the comparison between the load balance of the proposed method and the old algorithm mentioned before in [4] at different arrival rates. The mean job response time for different random distributions such as Exponential, Uniform, Normal and Poisson are calculated and after many trials it can be shown that the same results obtained for all distributions. The improvement ratio (gain) can be calculated and shown in Figure 5 and from this figure we can see that maximum improvement ratio is 98%.

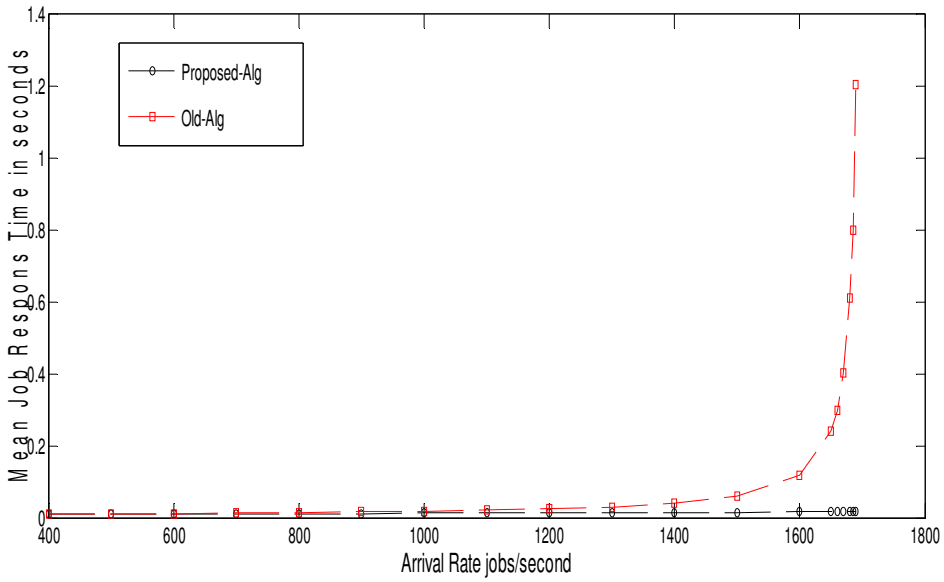

Figure 4. Load Balance Estimation Compared With The Old Algorithm.

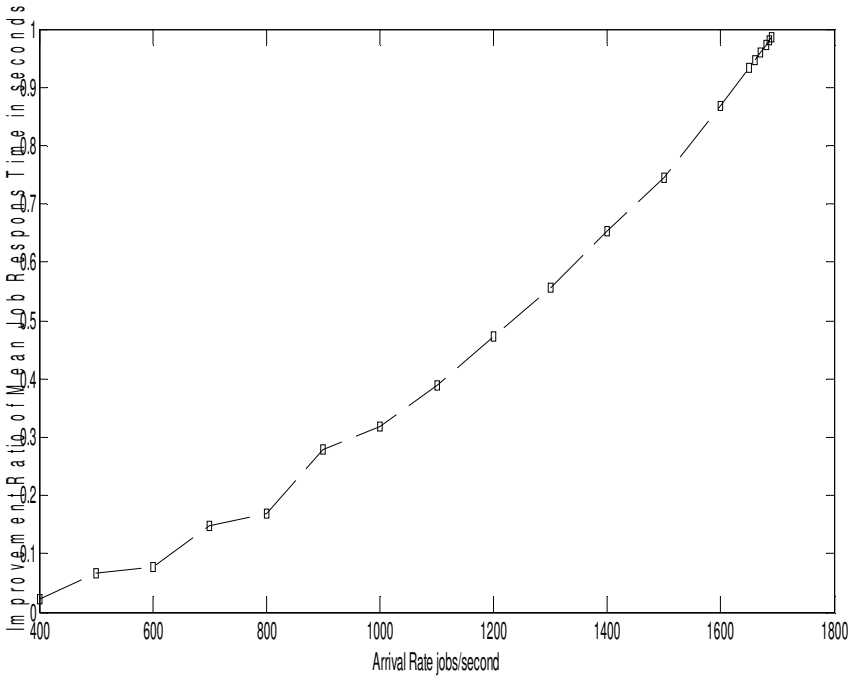

Figure 5. Load Balance Improvement Ratio.

The second experimental results is shown in Figure 6, here fault tolerance estimation is proportional to grid size and it can be shown that grid size for the proposed method is better than the old algorithm mentioned in [7] and the calculated values experimented at different levels of entropy values. The threshold device selects the best route path for the processing element from the all members of the grid then deciding the best system to be stable with respect to load balance and fault tolerance policies. When applying this best selected route path and feed back again to select the best value for the entropy threshold to improve both load balance and fault tolerance we found that the load balance still the same and this confirm the stability condition of DR-tree for the system. But the experimental results show that fault tolerance enhanced with different entropy threshold values and the improvement ratio for fault tolerance is 33%. The last experiment show that when decrease the entropy threshold under 80% of its values the stability of the system affected and the output result not accurate this could be shown in Figure 7 as the total final improvement ratios are 98% for load balance and 33% for fault tolerance which are very good enhancement ratios.

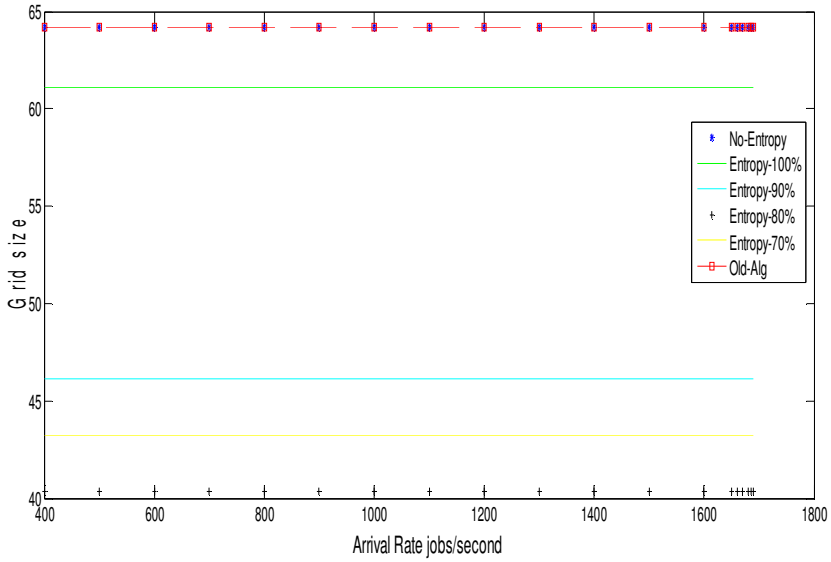

Figure 6. Fault Tolerance Estimation with different Entropy Levels Compared With The Old Algorithm.

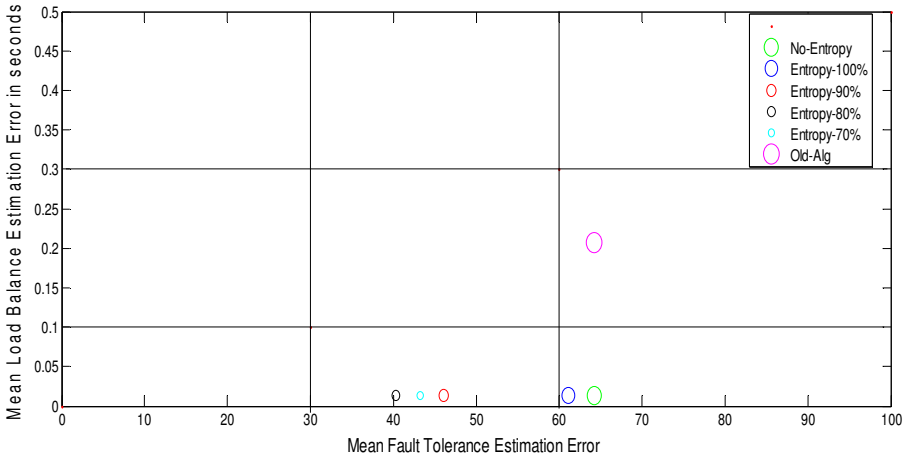

Figure 7. Path Selector with 2-D Figure of Merit with different Entropy Levels Compared With The Old Algorithm.

Finally, Applying optimization Algorithms to improve the performance of system utilization by using parameters as follows: No. of dimensions = 20, No of particles = 64, No. of iterations = 1000. Figure 8, 9, 10 show Fitness functions for GA, ACO and PSO respectively. When comparing the three different optimization algorithms with respect to system utilization the experimental study show that they almost the same but PSO algorithm give the best performance and the system utilization decrease about 75% as shown in Figure 11.

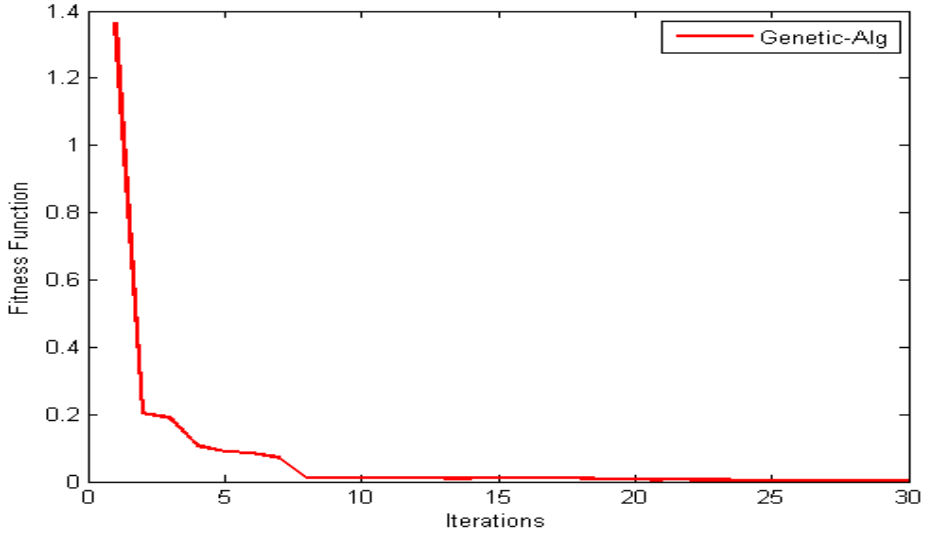

Figure 8. Fitness function of GA.

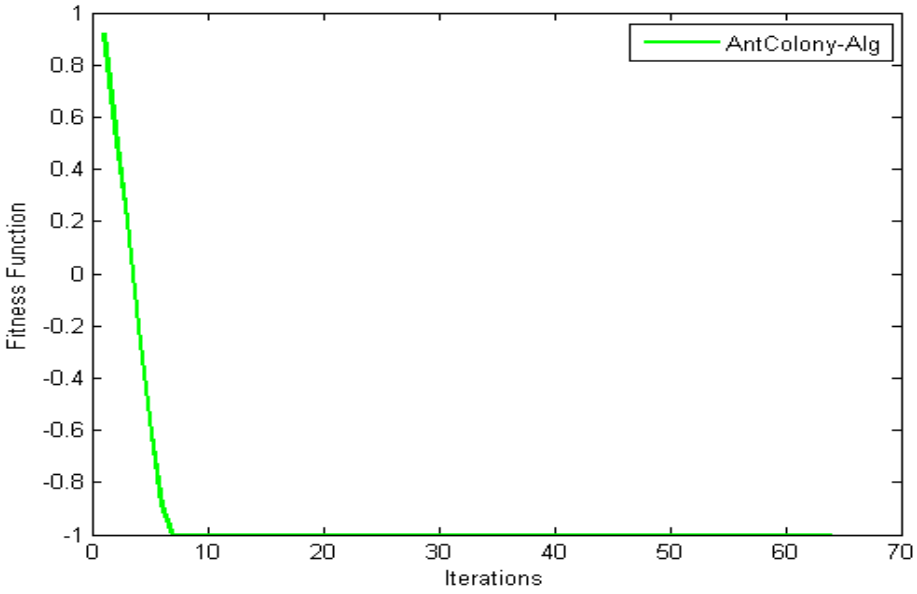

Figure 9. Fitness function of ACO.

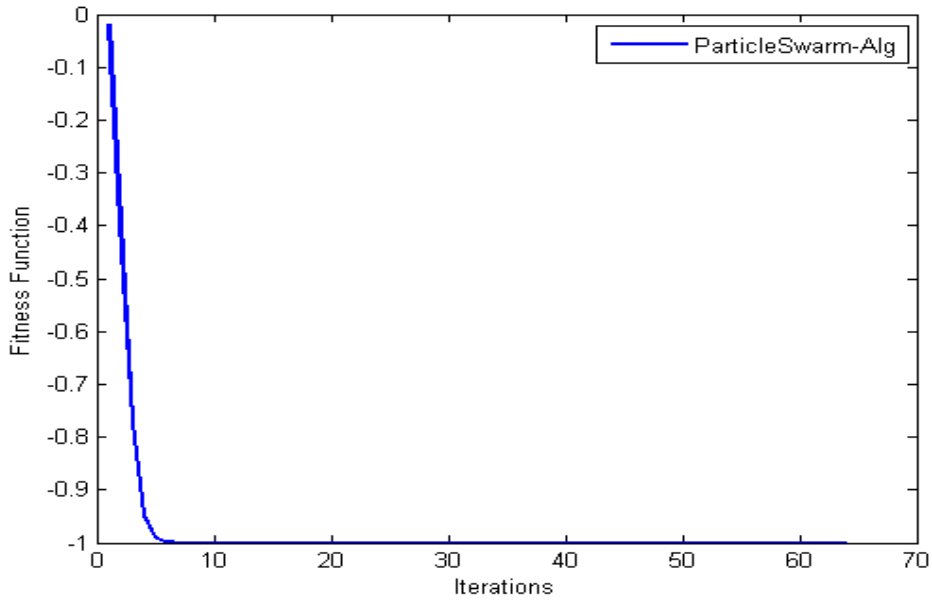

Figure 10. Fitness function of PSO.

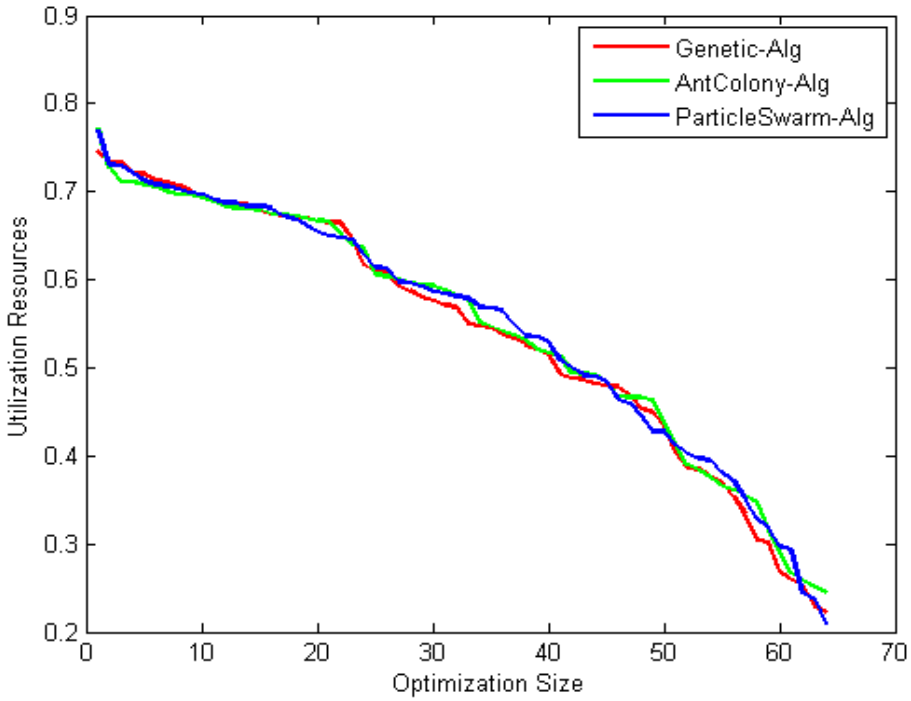

Figure 11. System utilization

5. CONCLUSION

This paper proposes a new adaptive procedure based on advanced fractal transform that enhances the tree model structure of grid computing environment to improve the network performance parameters affected by both fault tolerance and load balance. First, the network parameters fault tolerance and load balance estimation have been calculated based on fractal transform. In our work, simulator of fault tolerant approach for load balancing in grid environment is build. In this one local grid manager with 9 sites and each site have 1 to 3 processing elements and queue length of each computing element is from 1 to 50 and this was illustrated in a 2-D Figure of Merit. The grid computing routing protocol is enhanced by improving both fault tolerance and load balance estimation in a novel 2-D figure of merit. The improvement of the fault tolerance estimation is achieved by decrease replication time and message cost and this will cause increase in probability of job completed. On the other hand, the load balance estimation is enhanced by decrease mean job response time and this will cause increase in number of *jobs/second*. Finally, the improvement ratios are 98% for load balance and 33% for fault tolerance which are very good enhancement ratios. Also when comparing the system utilization by three different optimization algorithms GA, ACO and PSO they give almost the same results but PSO produce the best performance and decrease the system utilization to 75% which is high performance parameters. Further experimentation can be done by implementing our system on a real grid computing network and study its performance.

REFERENCES

[1] S. Sharma, U. S. Tim, J. Wong, S. Gadia & S. Sharma, (2014) "A Brief Review on Leading Big Data Models", Data Science Journal, Vol. 13, No. 0, pp. 138-157.

[2] S. Sharma, R. Shandilya, S. Patnaik & A. Mahapatra (2015a). Leading NoSQL models for handling Big Data: a brief review, International Journal of Business Information Systems, Inderscience, Vol. 18, No. 4.

[3] S. Sharma, U. S. Tim, J. Wong, S. Gadia, R. Shandily & S. Peddoju (2015b). Classification and Comparison of NoSQL Big Data Models, International Journal of Big Data Intelligence, Inderscience, Vol. 2, No. 2.

[4] F. Berman, G. Fox & A. Hey, (2003) Grid Computing: Making the Global Infrastructure a Reality, Wiley Series in Communications Networking & Distributed Systems, pp. 809-824.

[5] S. Elavarasi, J. Akilandeswari & B. Sathiyabhama, (2011) "A Survey on Partition Clustering Algorithms", International Journal of Enterprise Computing and Business Systems (Online) (IJECBS), Vol. 1, Issue 1, pp. 1-14.

[6] R. Sharma, V. Soni, M. Mishra & P. Bhuyan, (2010) "A Survey of Job Scheduling and Resource Management in Grid Computing", World Academy of Science Engineering and Technology, Vol. 64, pp. 461-466.

[7] S. El-Zoghdy, (2011) "A Load Balancing Policy for Heterogeneous Computational Grids", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 2, No. 5, pp. 93-100.

[8] A. Kumar, R. Yadav, Ranvijay & A. Jain, (2011) "Fault Tolerance in Real Time Distributed System", International Journal on Computer Science and Engineering (IJCSE), Vol. 3, No. 2, pp. 933-939.

[9] M. Valero, L. Arantes, M. Potop-Butucaru & P. Sens, (2011) "Enhancing Fault Tolerance of Distributed R-Tree", 5th Latin-American Symposium on Dependable Computing (LADC 2011), pp. 25-34.

[10] Y. Zhao & C. Li, (2008) "Research on the Distributed Parallel Spatial Indexing Schema Based on R-Tree", International Archives of the Photogrammetric, Remote Sensing and Spatial Information Sciences. Vol. XXXVII, Part B2, pp. 1113-1118.

[11] M. Hassaballah, M. Makky & Y. Mahdy, (2005) "A Fast Fractal Image Compression Method Based Entropy", Electronic Letters on Computer Vision and Image Analysis, Vol. 5, No. 1, pp. 30-40.

[12] V. Vaddella & R. Inampudi, (2010) “Fast Fractal Compression of Satellite and Medical Images Based on Domain-Range Entropy”, Journal of Applied Computer Science & Mathematics, Vol. 4, No. 9, pp. 21-26.
[13] L. Balasubramanian & M. Sugumaran, (2012) "A State-of-Art in R-Tree Variants for Spatial Indexing", International Journal of Computer Applications, Vol. 42, No. 20, pp. 35-41.
[14] B. Yagoubi & Y. Slimani, (2006) "Dynamic Load Balancing Strategy for Grid Computing", World Academy of Science Engineering and Technology, Vol. 19, pp. 90-95.
[15] J. Balasangameshwara & N. Raju, (2012) "A Hybrid Policy for Fault Tolerant Load Balancing in Grid Computing Environments", Journal of Network and Computer Applications (JNCA 2012), Vol. 35, Issue 1, pp. 412-422.
[16] S. Bianchi, A. Datta, P. Felber, & M. Gradinariu, (2007) "Stabilizing Peer-to-Peer Spatial Filters", 27th International Conference on Distributed Computing Systems (ICDCS'07), pp. 27-36.
[17] R. Mukhopadhyay, D. Ghosh, & N. Mukherjee, (2010) "A Study on the Application of Existing Load Balancing Algorithms for Large, Dynamic, Heterogeneous Distributed Systems", Recent Advances in Software Engineering, Parallel and Distributed Systems ACM 2010, pp. 238-243.
[18] J. Cao, (2004) "Self-organizing Agents for Grid Load Balancing", Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04), pp. 388-395.
[19] J. Cao, D. Spooner, S. Jarvis, & G. Nudd, (2005) "Grid Load Balancing Using Intelligent Agents", Future Generation Computer Systems, Vol. 21, Issue 1, pp. 135-149.
[20] Saad M. Darwish, Adel A. El-zoghabi, & Moustafa F. Ashry, (2015) "Improving Fault Tolerance and Load Balancing in Heterogeneous Grid Computing Using Fractal Transform", World Academy of Science, Engineering and Technology, International Journal of Computer and Information Engineering Volume 2, Number 5, 2015, International Science Index Volume 2, Number 5, 2015 waset.org/abstracts/18473, International Conference on Distributed Computing and Networking (ICDCN 2015), Tokyo, Japan, May 2015, pp. 820-840.
[21] R. Rajeswari & Dr. N.Kasthuri, (2013) "Comparative survey on load balancing techniques in computational grids", International Journal of Scientific & Engineering Research (IJSER), Vol. 4, Issue 9, pp. 79-92.

AUTHORS

Saad M.Darwish received his Ph.D. degree from the Alexandria University, Egypt. His research and professional interests include image processing, optimization techniques, security technologies, and machine learning. He has published in journals and conferences and severed as TPC of many international conferences. Since Feb. 2012, he has been an associate professor in the Department of Information Technology, Institute of Graduate Studies and Research, Egypt.

Adel A.El-Zoghabi received his Ph.D. degree from the Alexandria University, Egypt. His research and professional interests include image processing, optimization techniques, security technologies, and machine learning. He has published in journals and conferences and severed as TPC of many international conferences. Since April. 2013, he has been a head of the IT Department of Information Technology, Institute of Graduate Studies and Research, Egypt.

Moustafa F.Ashry received his M.Sc. degree in communication engineering from the Arab Academy for Science And Technology (AASTMT), Egypt. His research, published and professional interests include communication networks, Ad-Hoc networks, routing protocols, optimization techniques, and security technologies. Since Feb. 2012, he has been a Ph.D. student in the Department of Information Technology, Institute of Graduate Studies and Research, Egypt.