Detection of False Data Injection Attacks and Genetic Algorithm Based Dynamic Jobs Scheduling in Grid Computing

Shruti B Patel
Computer Science Engineering,
VIT Vellore Institute of Technology,
Vellore, India,
shrutibpatel50@gmail.com

Prof. Manikandan K
Computer Science Engineering,
VIT Vellore Institute of Technology,
Vellore, India,
kmanikandan@vit.ac.in

ABSTRACT
A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by low latency and high bandwidth networks. The sharing of computational resources is a major aspect of grids. Scheduling is a key problem in emergent computational systems, such as Grid and P2P, in order to benefit from the large computing capacity of such systems. Our approach is to dynamically generate an optimal schedule to complete the different tasks in a minimum period of time as well as utilizing the resources in an efficient way. There are so many approaches for scheduling like Genetic Algorithm (GA), Simulated Annealing (SA) and Ant Colony optimization (ACO). In this paper, we would like to present Genetic Algorithms (GAs) based schedulers for efficiently allocating jobs to resources in a Grid system. We would also like to implement GAs for designing efficient Grid schedulers when makespan is minimized. Our GA-based schedulers are very fast and hence they can be used schedule jobs arrived in the Grid system. Adding to this, increased connectivity of grid helps for bidirectional communications presents extreme security vulnerabilities. The proposed system also provides approach for false data detection in smart grids, like MD5 message-digest algorithm used as cryptographic hash function for message authentication and to verify the content of the message.

Keywords:
False data detection, Genetic Algorithm, Makespan, Minimum completion time, Fitness.

1. INTRODUCTION
Grid computing has emerged as an important field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and high-performance orientation. In grid, computing scheduling is challenging job. Therefore, we used GAs for designing efficient Grid schedulers when makespan is minimized. The GA operation is based on the Darwinian principle of "survival of the fittest". It implies that the fitter individuals are more likely to survive and have a greater chance of passing their good genetic features to the next generation. In genetic algorithm, each individual that is a member of the population represents a Potential solution to the problem. GA starts with initial population of individuals (chromosomes).In [9] each individual is evaluated using fitness function to produce a value known as goodness of the solution. In [3], then a new population is generated by selecting best individuals from the current population and applying crossover operator to produce new offspring, which would inherit good features of parents. Then each offspring is mutated in order to prevent GA to be trapped in local optima. Best individuals among current population and new population are carried forwarded in the next generation. The process is repeated until stopping condition met and best solution in the current generation is returned. We have used Genetic Algorithm based approach for our paper because GA can search for optimal/nearly optimal solution for scheduling quickly. It is well understood and applicable to many real life problems. In [9] GA can easily be combined with other meta-heuristic approaches for multiple objectives.

2. PROPOSED SYSTEM DESIGN
I used genetic algorithm to find optimal/nearly optimal schedule when makespan is minimum which efficiently utilize the resources. Proposed GA can quickly search solution space in parallel to find optimal/nearly optimal solution in very less time. It uses dynamic information received from Grid Information System to determine optimal/ nearly optimal solution. It can work with larger sized problems. We are going to present a job-scheduling algorithm, which can perform well and adding to this, We present a security framework, which can detect various attacks including random attacks, denial-of-communication attacks, replay attacks, and false data injection attacks in the smart grid.

3. PROBLEM FORMULATION
Our GA is based on Expected Time to Compute (ETC) Model. An ETC for any job j on any resource (machine) r is expected execution time of job j on r if j is scheduled on r. The problem for grid scheduling consists of following:

- n – the number of jobs to be schedule at particular instance of time. Any job has to be processed entirely in unique resource.

- m – the number of heterogeneous resources(machines) available in the Grid for an execution of a given set of jobs
- $N = \{j_1, \ldots, j_n\}$ a set of n jobs
- $R = \{r_1, \ldots, r_m\}$ set of available m resources.
- The workload Wi of each job i.
- The computing capacity CC_r of each resource (in millions of instruction per second) r.
- The expected time to compute ETC matrix of size n×m(number of jobs * number of resources).ETC[j][r] indicates the expected execution time of job j in resource r.

I considered the scenario in which jobs submitted to the Grid are independent and are not preemptive.

A. Fitness of a Schedule*:* We used uni-criteria optimization case for computing optimal/nearly optimal schedule of a set of jobs on a set of heterogeneous resources as per [5]. The fundamental criterion is that of minimizing the makespan.

B. Makespan: The time when latest job finishes. It is calculated as follows:

$$makespan = \min_{S_i \in Schedules} \left\{ \max_{j \in N} F_j \right\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

In eq.(1) F_j denotes time when job j finalizes, Schedules denotes the set of all possible schedules and N denotes the set of all jobs to be scheduled. The goal of scheduler is to maximize resource utilization and minimize makespan. Completion time of machine i is denoted by completion[r] and it is expressed as a total time needed for the resource r to finalizing its previously assigned jobs and jobs which are actually scheduled to this resource. We can compute ETC and completion time completion[r] for resource r as follows:

$$ETC[j][r] = \frac{W_j}{CC_r} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$completion[r] = ready_r + \sum_{[j \in N | Schedules[j] == r]} ETC[j][r] \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Where,
ETC[r] [j] =expected time to compute job j on resource r.
W_j = workload of job j
CC_r = computing capacity of resource r.
Completion[r] =completion time for resource r.
Readyr=time when resource r finishes previously assigned jobs to it.

The makespan of eq. (1) can be redefined as the maximal completion time and can be calculated as follows: A criteria makespan can be integrated in several ways to establish the desired priority among them. In the multi-objective optimization, two fundamental models are the hierarchical and the simultaneous approach. In hierarchical approach, the optimization criteria are sorted by their importance. The process starts by optimizing most important criterion. When further improvements are not possible, the second criterion is optimized while keeping optimized value of first important criterion unchanged. In grid scheduling, makespan may be considered as most important criterion. We used simultaneous approach to compute objective function or fitness function.

Fitness = 1 / makespan … … … … … … … …(5)

4. OVERALL SYSTEM ARCHITECTURE

We implemented GA based grid scheduler that maximizes resource utilization by minimizing makespan. It also determines schedules based on the current resource information (dynamic and static information). Hence can easily react to dynamism involved in grid environment. For same overall system, architecture is shown in figure.1.
We designed our system in 3 major modules.

4.1 Monitoring & Discovery Service (MDS) Module:

This module is used to discover the new grid resources and to monitor already discovered resources. When MDS process starts first time it reads /var/grid resources file to get list of the resources available initially. It also creates a thread to periodically poll already discovered grid resources to get current information about each of these grid resources. The information includes static information about resources such as processor family/architecture, number of CPUs/resource, CPU frequency, total RAM, total swap area etc., and dynamic information such as resource computing status busy/free, resource up/down status, free RAM, load, number of free CPUs etc. It also periodically receives resource information from grid resources. This information is sent to manager process as well as GA based grid scheduler as and when needed. GA scheduler uses current resource information to compute optimal/nearly optimal solution to assign jobs to resources. It also receives update information from manager process and updates its data

4.2 Manager Module:

This module is the central part of our implementation. It receives commands from users. It implements following functions.

- Command processing & Scheduler invocation
- Job queue management
- Job management
- Job monitoring

The Manager receives command requests (such as submit a job, query jobs, delete a job) from users as shown in figure 1. When a user submits a job using gsub command, it sends job submission request to manager. When manager receives a job submission request, unique job id is generated for a job and its description is appended to job queue. If a command request is to query jobs(gstat), it simply loop through job queue and send information such as job id, job status, job name, job executable, assigned resource if it is already scheduled etc. If command is to delete a job (gdel) and job is scheduled then job management components forward request to gatekeeper of the assigned host to clean the job. Once the job is deleted on the resource, it will be removed from the job queue otherwise an error is reported. This component periodically checks if there are unscheduled jobs in the job queue. If there are some jobs, it connects to GA Grid scheduler, send

information about jobs to GA grid scheduler and wait for optimal/nearly optimal mapping of jobs to suitable resources from the scheduler. Once it receives, a optimal/nearly optimal schedule from scheduler, for each (job, resource) pair in the schedule, it submits to local resource manager for execution purpose.

4.3 Scheduler Module

This module uses Genetic Algorithm to find optimal/nearly optimal solution by minimizing makespan. It receives information about list of jobs from manager and information about available resources from MDS server. It then creates initial population of k schedules using Minimum Completion Time heuristics. It then evaluates the current population by computing fitness function for each of k. It then creates a new population by repeating selection, crossover, and mutation and assignment steps until the size of new population becomes k. It then evaluates the new population and carries forward best schedules of the current population as well as the new population in the next generation in order to get optimal/nearly optimal solution quickly. The algorithm evolves generation by generation until termination criteria met. The Scheduler then return best schedules in current population. This schedule will then be sent to manager. Manager submits this job description to the assigned resource.

5. SYSTEM DESIGN

This section presents actual design of our system which is Job scheduler using Genetic Algorithm in grid computing. In [9] Dynamic task scheduling using Genetic Algorithm in a computational grid, resources are shared by many users, who submit their applications concurrently. We implemented Genetic Algorithm based Grid scheduling using following steps.

5.1 Schedule Encoding

We used direct representation to encode each possible schedule in a chromosome. We used array chromosome of n(number of jobs) integer to represent a chromosome(a schedule) as shown in fig. 2. Chromosome[j] represents the resource number where job j is scheduled.

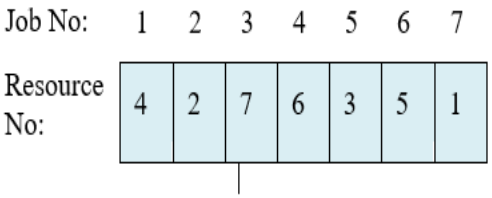

Figure. 1: Encoding of a schedule (a chromosome)

5.2 Generation of Initial population

In [7], initial population is usually generated randomly. But to guide the searching process and to get optimal/nearly optimal solution in fewer generations, several problem specific heuristics may be used such as Min-Min, Minimum Completion Time (MCT) etc. We used MCT heuristics to guide a searching process for finding optimal/nearly optimal schedule quickly in fewer generations. In the MCT heuristic, each job is assigned to the resource where job completes in minimum time. Jobs are considered for allocation at random.

5.3 Compute Fitness function

The scheduler aims to maximize resource utilization by minimizing makespan. Good chromosomes have higher fitness values. The fitness of each chromosome (schedule) is computed using eq. (5).

5.4 Selection operator

Selection operator is used to select parents to which crossover operator is applied to produce new offspring. In general, selection is directly proportional to the fitness of chromosomes. Several selection methods exist to select chromosomes for crossover such as linear ranking, roulette wheel selection etc. We used roulette wheel selection technique to select good schedules to produce new offspring. In roulette wheel selection method, the probability that a chromosome selected is directly proportional to its fitness value. Higher the fitness, higher chances the chromosome will be selected. In this method, each schedule or chromosome gets portion on the roulette wheel according to its fitness value. Chromosomes with higher fitness value get larger slice on roulette wheel. Selection is done by spinning a roulette wheel. Since fittest schedule has larger portion on the roulette wheel, they will have higher chance of being selected. Circumference of roulette wheel represents the total fitness of all chromosomes. Pseudo code for roulette wheel selection method is shown below. The roulette wheel selection of among 4 chromosomes is shown in figure 2. Chromosome 3 has higher chance of getting selected as shown in figure. 2.

Pseudo code Roulette Wheel Selection

```
RouletteWheelSelection()
        {
        total_fitness=0.0;     running_sum=0.0;
        for each chromosome k in a current population
                total_fitness=fitness(k);
        r=select random number r in the range [0,total_fitness-1]. for each chromosome k in a current population
                running_sum=running_sum+fitness(k);
                if(running_sum >= r)
                    return(k);
        }
```

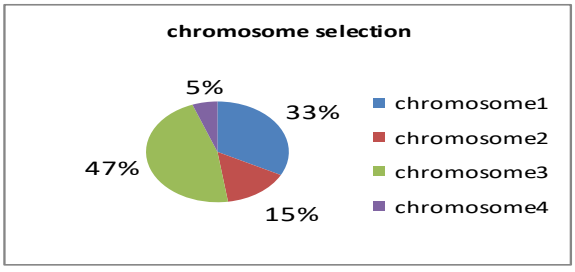

Figure 2: Chromosome selection as per roulette Wheel

5.5 Crossover operator

With crossover operator, two selected parent chromosomes can interchange their genes and produce new offspring (children). The aim is to obtain better quality solution and explore a new region of

solution space that has not been yet explored. One may use several different types of crossover such as one-point crossover, two-point crossover, uniform crossover etc. In [10], one-point crossover operator to produce offspring schedules. In this method, first, random crossover point between 1 and n (number of jobs) is selected, and then first parts of two parents are interchanged to produce two offspring (schedules). Same way, exchanging second parts of two parents to produce two new offspring (schedules) which are same as those produced by exchanging first parts. One point crossover

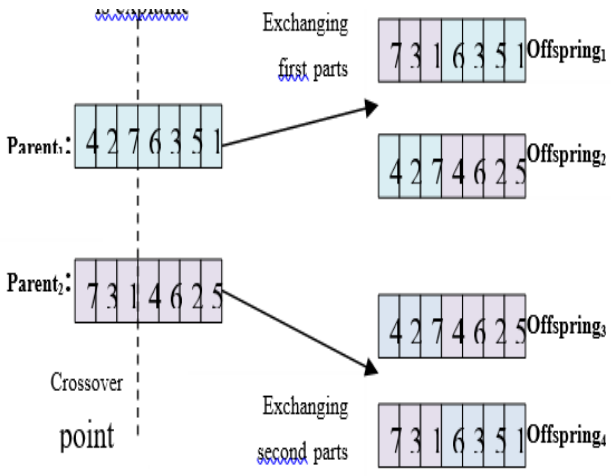

Figure. 3 one point crossover operator

5.6 Mutation operator

Mutation randomly changes gene(s) to different values. It is used to provide diversification by changing some gene(s) randomly and thereby prevent GA search process getting stuck in to local optima. There are several types of mutation such as move, swap etc., applied to a schedule. We used move mutation which randomly selects a job in a schedule (a chromosome) and assign it to another machine as shown in fig 4.

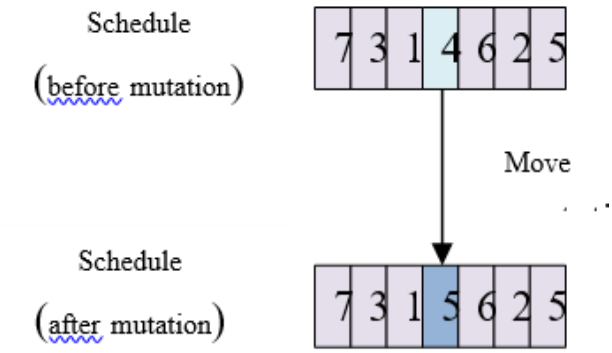

Figure. 4 Mutation operation

5.7 Replacement operator

Replacement operator determines which of the chromosomes (schedules) survives in the next generation. Two kinds of replacement usually used to carry forwards chromosomes to next generation (a) Generational replacement (b) Partial replacement. In a generational replacement, the current population is entirely replaced by new population while in partial replacement worst chromosomes in a current population are replaced by good chromosomes of new population. We used partial replacement strategy in which k best chromosomes from combined current and new population are carried forward to the next generation. First, fitness function is computed for each offspring. Let CP(t) be the current population in generation t and NP(t) be new population in generation t, then current population of next generation t+1 will be

CP(t+1)=k best schedules from (CP(t) U NP(t))

5.8 Termination Criteria

Termination criteria could be

(i) Maximum number of generations or iterations: the genetic search process is terminated after fixed number of generation [2].

(ii) Number of iterations without improvement: the optimization process is terminated after some fixed number of iterations without any improvement [2].

We used (i) termination criterion for our genetic algorithm based grid scheduler in which search process terminates after 300 generations. If termination criterion is not satisfied goto, step 3 and repeat the process. In general, this genetic search process of finding optimal/nearly optimal solution can be summarized as follows:

GAGridScheduling ( ) {

1. ENCODING: Represent a schedule(a chromosome) using array of n(number of jobs) integer chromosome such that chromosome[i] represents the resource on which job is scheduled
2. INITIALIZATION: Generate a initial population CP(t=0) of k schedules using
   MCT (Minimum Completion Time) heuristic.
3. FITNESS: Evaluate schedule in CP(t) using eq. (5)
4. TERMINATION CRITERIA: Check if termination criteria satisfied, if 'yes' return the best solution from current population CP (t).
5. NEW POPULATION: Repeat following steps until size of new population NP (t) becomes k.
   (a) Selection: Select two parents schedules p1 & p2 from CP (t) using roulette wheel method.
   (b) Crossover: With crossover probability p_c Perform one-point crossover to produce two new offspring schedules o1 & o2.
   (c) Mutation: With very low mutation probability $p_{m,}$ change the assignment of randomly chosen job to new grid resources in each offspring o1 and o2.
   (d) Assignment: Place o1 & o2 in NP(t)

   NP(t)=NP(t) U{ o1,o2}
6. FITNESS: Evaluate schedule in NP (t) using eq.(5).
7. REPLACEMENT:
   (a) Select k best schedules from CP(t) and NP(t) to carry forward in the next generation. CP(t+1)=k best schedules from (CP(t) U NP(t))

(b) Increment generation count

t=t+1

Goto Step 4

}

6. RESULTS AND ANALYSIS

For the experimental purpose, consider following problem instance consisting of 10 grid resources and 20 jobs. List of grid resources with existing workload is shown in the Table-1.

Table-1: List of grid resources with corresponding computing capacity

Resource No.	Computing Capacity (MIPS)	Existing workload (pending processing in ms)
1	3380	72.88
2	931	43.44
3	2969	69.92
4	3120	97.47
5	3728	47.61
6	1815	32.22
7	3170	22.67
8	2084	46.86
9	2014	26.48
10	3318	46.09

Table-2: List of jobs with corresponding workload

Job No	Workload
1	126
2	233
3	759
4	858
5	829
6	255
7	789
8	898
9	547
10	110

To find out optimal/nearly optimal solution for this problem instance, we tuned our genetic algorithm based scheduler with following parameters

Number of Generations=300
Size of population=256
Crossover probability (P_c) =0.90
Mutation probability (P_m) =0.0001

We got makespan=26.0183 in generation number 189 and then it retains this value until last generation. So if we reduce number of generations to less than 189, we got makespan=26.6066.

7. CONCLUSION

We presented an extensive study on the usefulness of Genetic Algorithms (GAs) for designing efficient Grid schedulers when makespan parameter is minimized under hierarchic and simultaneous approaches. The experimental study reveals the quality of the proposed GA-based schedulers as compared well to the existing GA-schedulers in the literature. Our GA-based schedulers can be used to design dynamic schedulers. A dynamic scheduler would run our GA in batch mode to schedule jobs arrived in the system since last activation of the scheduler.

REFERENCES

[1] Abraham, A. H. Liu, W. Zhang and T. G. Chang, Job scheduling on computational grids using fuzzy particle swarm algorithm, Proc. of the 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, B. Gabrys et al. (eds.): Part II, Lecture Notes on Artificial Intelligence 4252, 500507, Springer, 2006.

[2] Abramson, D., R. Buyya and J. Giddy, A computational economy for grid computing and its implementation in the Nimrod-G resource broker, Future Generation Computer Systems Journal, vol.18,no.8, pp.1061-1074, 2016.

[3] Alba, E., F. Almeida, M. Blesa, C. Cotta, M. Daz, I. Dorta, J. Gabarr, C. Le, G. Luque, J. Petit,C. Rodrguez, A. Rojas and F. Xhafa, Efficient parallel LAN/WAN algorithms for optimization, Parallel Computing, vol.32, no.5-6, pp.415-440, 2006.

[4] Buyya, R., Economic-based Distributed Resource Management and Scheduling for Grid Computing, Ph. D. Thesis, Monash University, Melbourne, Australia, 2017.

[5] Buyya, R., D. Abramson and J. Giddy, Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid, Proc. of the 4th International Conference on High Performance Computing, Asia-Pacific Region, China, 2015.

[6] Javier Carretero, Fatos Xhafa, Ajith Abraham. Genetic algorithm based schedulers for grid computing systems. In International Journal of Innovative Computing, Information and Control ICIC International °c 2005 ISSN 1349-4198 Volume 3, Number 6, December 2017.

[7] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In The 8th IEEE

International Conference on Advanced Computing and Communications, India, 2016.

[8] Jia Yu and Rajkumar Buyya. Workflow Schdeduling Algorithms for Grid Computing Grid Computing and Distributed Systems (GRIDS) Laboratory Department of Computer Science and Software Engineering The University of Melbourne, Australia.

[9] Guangchang Ye, Ruonan Rao, Minglu Li. A Multiobjective Resources Scheduling Approach Based on Genetic Algorithms in Grid Environment. In Fifth International Conference on Grid and Cooperative Computing Workshops (GCCW'06) IEEE computer society.

[10] Taras S. Shapovalov, Alexey G. Tarasov. Genetic Algorithm Based Parallel Jobs Scheduling. In program "Research and scientific-pedagogical personnel of innovative Russia"(project No. 02-740-11-0626) and Grant of Russian Foundation for Basic Research and Far eastern branch of Russian academy of sciences No. 10-III-B- 01I-009.

[11] Wei Sun , Yuanyuan Zhang , Yanwei Wu, and Yasushi Inoguchi Practical Task Flow Scheduling for High Throughput Computational Grid. In International Conference on Parallel Processing Workshops (ICPPW'06) 0-7695-2637-3/06 $20.00 © 2006 IEEE computer society.

[12] T. Casavant, and J. Kuhl, A Taxonomy of Scheduling in General-purpose Distributed Computing Systems, in IEEE Trans. on Software Engineering Vol. 14, No.2, pp. 141--154, February 2016.

