International Journal of Computer Applications (0975 – 8887)
Volume 48– No.23, June 2012

A New Approach to Grid Scheduling using Random Weighted Genetic Algorithm with Fault Tolerance Strategy

Chinmoy Kar
West Bengal University of Technology
BF-142, Bidhan Nagar, Kolkata-700064, India

Vineet Kumar Rakesh
Variable Energy Cyclotron Centre
1/AF, Bidhan Nagar, Kolkata-700064, India

Tapas Samanta
Variable Energy Cyclotron Centre
1/AF, Bidhan Nagar, Kolkata-700064, India

Sreeparna Banerjee
West Bengal University of Technology
BF-142, Bidhan Nagar, Kolkata-700064, India

ABSTRACT
Grid provides us a huge amount of computational resources in a distributed manner, using which we can perform our tasks over these grid environments. These resources are geographically distributed around the globe and are dynamically available. Hence, to schedule them for actual use we need to consider various points like, availability, fault tolerance, and response time etc. In this paper we consider a grid scheduling strategy with respect to multiple objectives. We have followed a multi objective genetic algorithm which is basically a Random Weighted Genetic Algorithm (RWGA) considering the checkpoint based fault tolerance mechanism to prevent resource failure.

General Terms
Distributed systems

Keywords
Genetic Algorithm (GA), Multi objective Genetic Algorithm (MOGA), weighted sum approach, Grid scheduling, Checkpoint, Fault Tolerance.

1. INTRODUCTION
Multi objective evolutionary algorithms (MOEA) are the efficient algorithm to solve the multi objective optimization problem [1]. In real world there are many problems with several objectives having no single optimal solution [2][14]. Therefore, the decision maker is required to select a solution from a finite set of solution. One of best approach of MOEA is Multi objective genetic algorithm (MOGA) [3][13]. GA is inspired by the evolutionist theory explaining the origin of species. In GA terminology, Chromosomes are made of discrete units called genes. Each gene controls one or more features of the chromosome [4]. GA uses two operators to generate new solutions from existing population: crossover and mutation. The crossover operator is the most important operator of GA. In crossover, generally two chromosomes, called parents, are combined together to form new chromosomes, called offspring, by iteratively applying the crossover operator, genes of good chromosomes are expected to appear more frequently in the population, eventually leading to convergence to an overall good solution. Mutation is generally applied at the gene level. In typical GA implementations, the mutation rate (probability of changing the properties of a gene) is very small and depends on the length of the chromosome.
In our proposed scheduling approach using Random Weight Genetic Algorithm we have implemented the same genetic approach for job scheduling. We also consider fault tolerance mechanism by introducing checkpoint strategy. We have implemented our check pointing strategy such a way that if a job failed after certain time then we can start that job on another resource from its last saved checkpoint value.

2. RELATED WORKS
In past few years many multi objective genetic algorithm proposed by different researchers. In [7], the authors proposed an algorithm called Multi-Objective Resource Scheduling Approach - MORSA, which is a combination between NPGA and NSGA Algorithms. They combine the sorting algorithm of non dominated solutions with the process of Niche Sharing to ensure diversity. Another interesting proposal is presented in where the NSGA-II is used as base algorithm [8]. Some authors are investigated workflow scheduling after optimize three conflicting objectives simultaneously by NSGA-II [9], to solve the workflow scheduling problem in grid. Simultaneously fault tolerance mechanism is important aspects of grid computing.

3. GRID SCHEDULING USING RANDOM WEIGHTED GENETIC (RWGA) ALGORITHM WITH FAULT TOLERANCE STRATEGY
Resource discovery, resource selection, job pool construction, scheduling and fault tolerance mechanism in case of resource failures are our main objectives.

3.1 Resource Discovery
Resource discovery is a process by which a grid can select resources by filtering a pool of available resources. The filtering process is concerned with the fault index value of resources, from which we get from fault index manager and some characteristics of resources that do not change often (static), such as the amount of physical memory of a computing node or the processor type and speed. In our proposed architecture, Grid information server (GIS) gives us the information about various static resource requirements which are given as an input to the resource discovery phase. The output of this phase is a set of available resources, which are sorted in terms of Fault index value and GIS information (processing speed etc).

3.2 Resource Selection
Resource selection determines the set of resources which are selected to be executed from the available resource set. In our

proposed architecture we select the first and last resource from the set of available resource list.

3.3 Job pool construction Using Random Weight Genetic Algorithm

RWGA based on a weighted sum of multiple objective functions where a normalized weight vector randomly generated for each solution during the selection phase at each generation [5][6].
E = external archive to store non-dominated solutions found during the search so far;
nE= number of elitist solutions immigrating from E to P in each generation.

Step 1: Generate a random population.

Step 2: Assign a fitness value to each solution $x \in P_t$ by performing the following steps:

Step 2.1: Generate a random number u_k in [0,1] for each objective k, k = 1,…,K.

Step 2.2: Calculate the random weight of each objective k as $w_k = (1/u_k) \sum_{k=1}^{n} u_k$

Step 2.3: Calculate the fitness of the solution as

$$f(x) = (a_0 + \sum_{n=1}^{k} (W_k Z_k(x)))$$

Step 3: Calculate the selection probability of each solution $x \in P_t$ as follows

$P(x) = (f(x)-f^{min})^{-1} \Sigma_{y \in Pt}(f(y)- f^{min})$

Where $f^{min} = \min\{f(x) | x \in P_t \}$

Step 4: Select parents using the selection probabilities calculated in Step 3. Apply crossover on the selected parent pairs to create N offspring. Mutate offspring with a predefined mutation rate. Copy all offspring to Pt+1. Update E if necessary.

Step 5: Randomly remove nE solutions from Pt+1and add the same number of solutions from E to Pt+1.

Step 6: If the stopping condition is not satisfied, set t =t + 1 and go to Step 2. Otherwise, return to E.
In the below section equations (1) to (3) we shows multiple objective functions. The grid resource broker is responsible for resource discovery, deciding allocation of job to a particular resource. To formulate the problem we consider Jn independent user jobs n={1,2,….N} on Rm heterogeneous resource m={1,2,….M} with an objective of minimizing the completion time and utilize the resource effectively.
To formulate our objective define Cj as the completion time of job J, the makespan of job define by Cmax=max {Cj, J=1,2,…3}, and T=ΣCj is the flow time, Fi is the fault index value of job i. The fundamental criterion is that minimize makespan, flow time and fault index. The objective functions are:

Minimize $C_{max} = \max(C_j)$ (1)

Minimize $T = \Sigma C_j$ (2)

Minimize $F = \Sigma F_i$ (3)

Subject to $C_{max} < D$ and $T < B$

Where D is the deadline and B is the budget.

3.4 Mapping RWGA with grid scheduling

We create a population of chromosomes; basically the chromosomes are collection of jobs. We have assigned some randomly generated weight between 0 to 1 for above mention three objectives. Calculate the selection probability of each solution. Select the best two chromosomes depending on selection probability. Apply crossover and mutation and form another population. The above steps are performs until some stopping criteria matched.

4. PROPOSED SYSTEM COMPONENTS

The major components of our proposed architecture are:

1) GIS (Grid information server) contains information about all available grid resources. It maintains details of the resource such as process speed, memory available, load and so on. All grid resources that joined and leave are monitored by GIS.

2) Fault Index Manager maintains the fault index value of each resource which indicates the failure rate of the resource. The fault index of a grid resource is incremented every time the resource does not complete the assigned job within the deadline and also on resource failure. The fault index of a resource is decremented whenever the resource completes the assigned job within the deadline. Fault index manager updates the fault index of a grid resource using fault index update algorithm.

3) Grid Scheduler plays the role of an important component in this architecture; it is basically considered to be the main function that maps the job in resources and schedules them in resources depending on the chromosome which we get by RWGA.

4) Checkpoint Manager receives the scheduled job from the scheduler and sets checkpoints based on the failure rate of the resources on which it was scheduled. Then it submits the job to the resource. Checkpoint manager receives the job completion message or job failure message from the grid resource and responds to that accordingly. If a failure occurs during execution, the job is rescheduled from the last checkpoint instead of running from the scratch. Checkpoint manager implements checkpoint setter algorithm to set checkpoints of the jobs.

5) Resource pool is a table of information of actual location of all resources.

The logical architecture of grid scheduling is described in fig. 1.

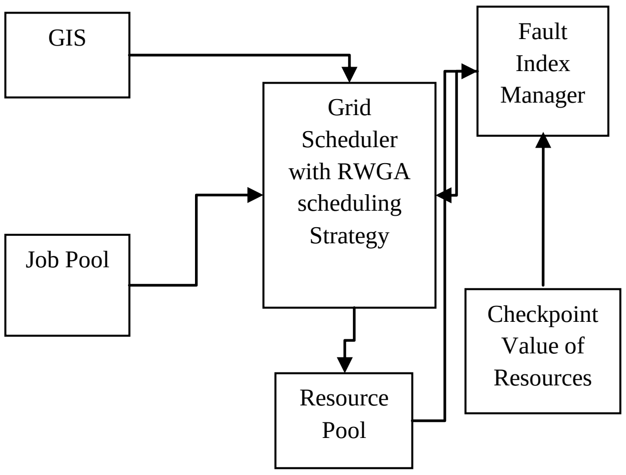

GIS
Fault Index Manager
Grid Scheduler with RWGA scheduling Strategy
Job Pool
Checkpoint Value of Resources
Resource Pool

Fig 1: Grid Scheduling Architecture with RWGA and Check-pointing

4.1 System Flow

1) The priority of the resources has been set depending upon fault index value. Less fault index value has higher priority and high fault index value has less priority.

2) If the two resources have the same priority then the priority setter checks GIS information of those resources. The resource having high load and low processing speed has less priority and a resource with high speed and low load has high priority.

3) Scheduler selects highest and lowest priority resource from the priority list for job execution. These two selected resources execute the job concurrently.

4) Using RWGA we have created a job string which returns best result after scheduling. RWGA consider multiple objectives and assign a fixed random value to all other objectives. We have to select two possible solution strings then perform crossover and mutation operation. If the resultant string satisfies the stopping criterion then algorithm stop otherwise we have to generate new population.

5) If a high priority resource fails then the job restarts with last saved checkpoint value in another low priority resource, if and only if the last saved check point value for that resource is less than low priority resource.

6) If the low priority resource fails then the job restarts with last saved checkpoint value in the other high priority resource, if and only if the last saved check point value for that resource is less than the high priority resource.

7) After failure, if the resource is a high priority resource then next resource is placed immediately next of that resource form the priority list.

8) After failure if the resource is low priority resource then next resource is immediately prior to of that resource form the priority list.

5. PROPOSED AND IMPLEMENTED ALGORITHMS

5.1 Resource Discovery and Selection:

```
1) Set the priority for each and every resource using PrioritySet() function.
2) The above function returns the highest priority resource that includes lowest fault index value and highest GIS information among the resources which have same fault index value.
3) Select MAX, MIN from that sorted array of resources where MAX represents the highest priority resource and MIN is the lowest priority resource.
4) Select a job from job pool and assign a job concurrently on these two resources.
5) If a resource fails then
    a) If the value of the FailedResource(MAX,MIN) function return MAX then
        i) If last saved checkpoint value from CheckpointValue(MAX,MIN) returns MAX.CheckpointValue then
            (1) Set the update the checkpoint value MIN.CheckpointValue by MAX.CheckpointValue.
            (2) Select the available maximum priority resource using SelectMax()
            (3) Restart the job with new checkpoint value.
        ii) Else
        Do nothing.
```

b) If the value of the **FailedResource(MAX,MIN)** function returns MIN then
   i) If last saved checkpoint value from **CheckpointValue(MAX,MIN)** return MIN.CheckpointValue then
      (1) Set the update the checkpoint value MAX.CheckpointValue by MIN.CheckpointValue.
      (2) Select the available minimum priority resource using SelectMin()
      (3) Restart the job with new checkpoint value.
   ii) Else
      Do nothing.

6) If no resource fails then
   a) Send the process completion signal.
   b) Continuing the same algorithm for another job.

5.2 PrioritySet() Function

1) Sort the resources based on fault index value.
2) If the fault index values the of resources are same then priority is assigned depending on load information and processing speed.
3) Return the resource list with their priorities.

5.3 FailedResource(MAX,MIN)

```
If the resource which fails is MAX then return MAX
Else
Return MIN
```

5.4 CheckpointValue(MAX,MIN)

```
If(MAX.CheckpointValue >  MIN.CheckpointValue)
    Return (MAX.CheckpointValue)
    Else
    Return (MIN.CheckpointValue)
```

5.5 SelectMax()

```
Index = MAX.Index
/* MAX.Index is an array index of the resource*/
While ((Index + 1) < N) //N is number of resources
Index = Index + 1
If (Resource with this Index is free)
Return Resource which index is Index.
Else
Return no Resource available
```

5.6 SelectMin()

```
Index = MIN.Index
/* MIX.Index is an array index of the resource*/
While ((Index-1) < N) //N is number of resources
Index = Index-1
If (Resource with this Index is free)
Return Resource which index is Index.
Else
Return no Resource available
```

6. EXPERIMENTAL RESULT

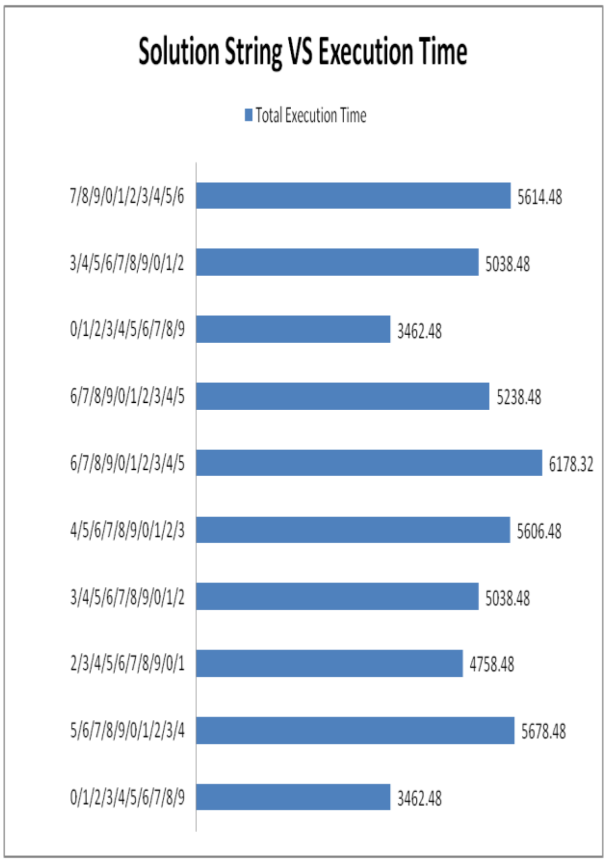
1) We have created ten jobs and put the references of each job (We called each job gridlet) in a array called chromosome.
2) We have created another chromosome with same jobs.
3) We have sorted resources depending on speed, means high speed resource come first then second and then third.
4) After that we have generated a solution string which consists of ten job depending on RWGA.
5) We have also implemented the functions in case of resource failure.
6) We have performed this algorithm ten times, each a every time algorithm returned a chromosome of ten gridlets after that we submitted those jobs on three resources.
7) We use Gridsim[11] [12] for our simulation.

Table 1.Total Time taken for a solution chromosome in 10 different Run

Run	Iteration	Sequence of Job	Total Time
1	52	0/1/2/3/4/5/6/7/8/9	3462.48
2	6	5/6/7/8/9/0/1/2/3/4	5678.48
3	13	2/3/4/5/6/7/8/9/0/1	4758.48
4	31	3/4/5/6/7/8/9/0/1/2	5038.48
5	8	4/5/6/7/8/9/0/1/2/3	5606.48
6	22	6/7/8/9/0/1/2/3/4/5	6178.32
7	35	6/7/8/9/0/1/2/3/4/5	5238.48
8	54	0/1/2/3/4/5/6/7/8/9	3462.48
9	26	3/4/5/6/7/8/9/0/1/2	5038.48
10	12	7/8/9/0/1/2/3/4/5/6	5614.48

6.1 Discussion on Results

In above table we can see that when numbers of iteration increased we get a sequence of job or chromosome which gives minimum execution time, so from Table 1 we can easily say that maximum iteration gives better result.



7. ASSUMPTIONS

We have assumed some important characteristics to get the result successfully. Like we have take only ten job which is called gridlets[12] and run them depending on processing speed of lower priority resource is grater than high priority resource. In traditional resource allocation mechanisms scheduler assigns a job in that resource which has low fault index value. They are not considering the speed of that resource at all, which gives an unnecessary delay in completion of job. Our proposed architecture considers this as the main issue and tries to solve this problem. In our assumption we consider that if the high fault index value comes up with the higher processing speed then the check point value becomes larger than the other low fault index value resource at any time, when a failure occurs, and hence we get a substantial amount of benefits by completion time.

8. CONCLUSION

In this paper, our focus has been on finding a scheduling algorithm which is easy to implement and which also satisfy the multiple objectives related to grid scheduling and we also try to implement a checkpoint mechanism which can handle the situation like resource failure. In Table 1 we have mention the total time to schedule a set of jobs which does not include the iteration time and restart time from its last saved checkpoint value(in case of job failure). If we say that total time taken by a scheduler for schedule a job is T_{total}, iteration time is T_{itr} and restart time for a set of job is $T_{restart}$ then total time for scheduling is T:

$$T = T_{total} + T_{itr} + T_{restart}$$

If we assume that $T_{total} >> T_{itr}$ and if number of failure is zero then $T_{restart} = 0$ or if number of failure is less then $T_{total} >> T_{restart}$, then

$$T = T_{total}$$

9. FUTURE WORK

In this paper we try to map between a RWGA and checkpoint mechanism with grid scheduling. After we implement RWGA on grid architecture we also try to implement a checkpoint mechanism. This checkpoint mechanism approach is a straightforward and implementation friendly, it need much more complex environment for getting best result. In our future we will enhance and implement this algorithm on many jobs with sufficient resources with different characteristics. We also concentrate on this checkpoint mechanism with flexible environment for enhancing this approach. We also try to compare this algorithm with other MOGA which are used for grid scheduling in a generalize test environment.

10. ACKNOWLEDGMENTS

Our special thanks to Director, Variable Energy Cyclotron Center, Kolkata and Head, Computer and Informatics Group, VECC for granting permission to use the infrastructural facility. Without which, the project could not be materialized.

11. REFERENCES

[1] Zitzler, E., Thiele, L. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE. 3(4):257–71.

[2] Srinivas, N., Deb, K. 1994. Multi-objective optimization using non-dominated sorting in genetic algorithms. J Evol Comput. 2(3):221 48.

[3] Murata, T., Ishibuchi H. 1995. MOGA: multi-objective genetic algorithms. In: Proceedings of the 1995 IEEE international conference on evolutionary computation, 29 November–1 December, 1995. Perth, WA, Australia: IEEE 1995.

[4] Holland, JH. 1975. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press.

[5] Jones, DF., Mirrazavi, SK. Tamiz, M. 2002. Multiobjective meta heuristics: an overview of the current state-of-the-art. Eur J Oper Res. 137(1):1–9.

[6] Murata, T., Ishibuchi, H. 1996. Tanaka H. Multi-objective genetic algorithm and its applications to flowshop scheduling. ComputInd Eng. 30(4):957–68.

[7] Guangchang, Ye., Ruonan, Rao. and Minglu Li, A Multiobjective Resources Scheduling Approach Based on Genetic Algorithms in Grid Environment. Hunan, China: Fifth International Conference on Grid and Cooperative Computing Workshops, 2006.

[8] Grimme, C., Lepping, J. and Papaspyrou, A. 2008. "Discovering Performance Bounds for Grid Scheduling by using Evolutionary Multiobjective Optimization," in Proceedings of the 10th annual conference on Genetic and evolutionary computation. Atlanta, GA, USA, ACM. pp. 1491-1498.

[9] Garg, R, Singh, A.K. May, 2011. Multi-objective optimization to workflow grid scheduling using refernce

point based evolutionary algorithm, International Journal of Computer Application (0975-8887), Vol. 2(6).

[10] Nandagopal, M. & Dr. Uthariaraj, V.R. 2010. Fault tolerant scheduling strategy for computational grid environment.International Journal of Engineering Science and Technology. Vol. 2(9), 4361-4372.

[11] Buyya, R., Murshed, M., Abramson, D. 2002. A deadline and budget constrained cost time optimization algorithm for scheduling task farming applications on global grids, In Proceedings of the international conference on parallel and distributed processing techniques and applications, Las Vegas, USA, pp. 24–27.

[12] Buyya, R. GridSim: A Toolkit for Modeling and Simulation of Grid Resource Management and Scheduling, http://www.buyya.com/gridsim.

[13] Grosan, C., Abraham, A., Helvik, B. 2007. Multiobjective Evolutionary Algorithms for Scheduling Jobs on Computational Grids. In: International Conference on Applied Computing, Spain. ISBN 978-972-8924-30-0, 459-463.

[14] Camelo, M., Donoso, Y., Castro, H. 2010. A Multi-Objective Performance Evaluation in Grid Task Scheduling using Evolutionary Algorithms. In: Applied Mathematics and Informatics, ISBN: 978-960-474-260-8.