# Task Optimization in Grid Computing using Genetic Algorithm

*Ujjwal Prajapati*[1], *Subarna Shakya*[2]
*Department of Electronics and Computer Engineering*
*Central Campus, IOE, Tribhuvan University, Nepal*
*meujjwal @gmail.com*[1], *drss@ioe.edu.np*[2]

*Abstract*—**Task scheduling is a key problem in Grid computing in order to benefit from the large computing capacity of such systems. The need of allocating a number of tasks to different resources for the efficient utilization of resources with minimal completion time and economic cost is the essential requirement in such systems. The problem is multi-objective in its general formation, with the objectives being the minimization of makespan of the system along the economic cost. An optimal scheduling could be achieved minimizing the completion time and economic cost using the heuristic approach, which is chosen to be Genetic Algorithm. The ability of Genetic Algorithm to simultaneously search different regions of a solution space makes it possible to find a diverse set of solutions for difficult problems. Each individual is represented as possible solution. The solutions are the schedulers for efficiently allocating jobs to resources in a Grid system**

**Keywords— Task Scheduling; Grid Computing; Distributed Computing; Makespan; Economic Cost; Genetic Algorithm**

## I. Introduction

A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by heterogeneous networks [3]. A computational grid contains resource management, task scheduling, security problems, and information management and so on. Task scheduling is one of the fundamental issues which play an important role in the operation of distributed computing systems. Task scheduling in distributed computing systems is defined as the process of assigning the tasks of a distributed application into the available processors, and specifying the start execution time of the tasks assigned to each processor [1]. The problem of task allocation in distributed computing system is the need to allocate a number of tasks to different processors for execution. A task is an atomic unit to be scheduled by the scheduler and assigned to a resource [2]. A task scheduling is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains [2]. In the case of static scheduling, information regarding all resources in the Grid as well as all the tasks in an application is assumed to be available by the time the application is scheduled [2]. The application centric objective function in Grid computing could be either *make span*, which is the time spent from the beginning of the first task in a job at the end of the last task of the job, or *economic cost* that an application needs to pay for resource utilization [2]. Job Scheduling is known to be NP-complete; therefore the use of heuristics is the de facto approach in order to cope in practice with its difficulty [3]. The meta-heuristics run on static instances of the problem and therefore in this approach static schedulers are obtained. A Genetic Algorithm is a meta-heuristic search technique which allows for large solution spaces to be partially searched in polynomial time, by applying evolutionary techniques from nature [6].

## II. Literature Review

In Distributed Computing System, an allocation policy may be either static or dynamic, depending upon the time at which the allocation decisions are made. In a static task allocation, the information regarding the tasks and processor attributes is assumed to be known in advance, before the execution of the tasks. Distributed Computing Systems have become a key platform for the execution of hydrogenous applications. The major problem encountered when programming such a system is the problem of task allocation. Task allocation problem is known to be NP-hard problem in complexity, where required an optimal solution to the problem. The easiest way to finding an optimal solution to the problem is an exhaustive enumerative approach. But it is impractical, because there are $n^m$ ways of allocation m-tasks to n-processors.

*Ahmed Younes. Hamed* [4] presents a genetic algorithm, considering distributed computing system with heterogeneous processors in order to achieve optimal cost by allocating the tasks to the processors, in such a way that the allocated load on each processor is balanced. The algorithm is based on the execution cost of a task running on different processors and the task communication cost between two tasks to obtain the optimal solution. The proposed algorithm tries to minimize the processor execution cost and inter processor communication.

*Javier Carretero, Fatos Xhafa* [3] presents an extensive study on the usefulness of Genetic Algorithms for designing efficient Grid Schedulers when makespan and flowtime are minimized under hierarchic and simultaneous approaches. Two encoding schemes have been considered and most of GA operators for each of them are implemented and empirically studied.

*Mohammad I. Daoud and Nawwaf Kharma* [1] proposed customized genetic algorithm to produce high-quality task schedules for Heterogeneous Distributed Computing Systems. Also, the performance of the scheduling algorithm is compared to two leading scheduling algorithms which is based on both randomly generated task graphs and task graphs of certain real-world numerical applications, exhibits the supremacy of the new algorithm over the older ones, in terms of schedule length, speedup and efficiency.

*Prateek Kumar Singh, Neelu Sahu* [5] proposed compact genetic algorithm, which aims to generate an optimal schedule so as to get the minimum completion time while completing the tasks.

## III. Genetic Algorithm and Scheduling

A Genetic Algorithm is a meta-heuristic search technique which allows for large solution spaces to be partially searched in polynomial time, by applying evolutionary techniques from nature. Genetic Algorithm is high level algorithms that integrate other methods and genetic operators, therefore in order to implement it for a problem, we have to use the template for the method and design the inner methods, operators and appropriate data structures [5].

### A. Schedule Encoding

In grid scheduling, we have a set of tasks and a set of resources as input and a sequence, which informs that which task is to be operated on which resource and in which order as output. So, a population approach is acceptable with each individual in a population representing the scheduling solution. If we represent a set of task as $X = \{t_1, t_2, t_3, \ldots t_n\}$ and set of resources as $P = \{P_1, P_2, P_3, \ldots P_n\}$, then the sequence can be represented as shown in Fig 1. [4]

The chromosome is represented as string of integers. The length of chromosome is given by the number of tasks that should be allocated. Every gene in the chromosome represents the processor or resource which the task is running on.

From the allocation as shown, it is known that task $t_1$ should be run on resource $P_1$, task $t_2$ should be run on resource $P_3$ and so on.

### B. Initialization

The initial population is generated randomly. Given the population size, the random strings of integers are formed of definitive chromosome length evaluated from the number of task set to form the initial population.

### C. Fitness Function

The execution cost of a task running on different processors are different and it is given in the form a matrix of order m * n, named as execution cost matrix (ECM) [4]. Similarly, the inter task communication cost between two tasks is given in the form of a symmetric matrix named as inter task communication cost matrix (ITCCM), of order m * m [4]. The makespan is expressed in terms of Expected Time to compute matrix (ETC) of size m*n.

### D. Execution Cost (EC)

The execution cost ($ec_{ik}$) of a task $t_i$ , running on a processor $P_k$ is the amount of the total cost needed for the execution of $t_i$ on that processor during the execution process. If a task is not executable on a particular processor, the corresponding execution cost is taken to be infinite [4].

| $t_1$ | $t_2$ | $t_3$ | $t_4$ | .. | .. | $t_m$ |
|---|---|---|---|---|---|---|
| $P_1$ | $P_3$ | $P_1$ | $P_2$ | .. | .. | $P_n$ |

Fig 1. Task Allocation in the form of chromosome [4]

### E. Communication Cost (CC)

The communication cost $cc_{ij}$ incurred due to the inter task communication is the amount of total cost needed for exchanging data between ti and $t_j$ residing at separate processor during the execution process. If two tasks executed on the same processor then $cc_{ij} = 0$ [4]

### F. System Cost

Given a task allocation $X = \{x_{ik}\}$, i = 1,2,3,….m, k=1,2,3,.. n, the execution cost of all processors can be computed by the following equation: [4]

$$\mathbf{PEC(X)} = \sum_{k=1}^{n}\sum_{i=1}^{m} ec_{ik}\, x_{ik} \qquad (1)$$

The inter processor communication cost for all processors can be computed as follows: [4]

$$\mathbf{IPEC(X)} = \sum_{k=1}^{n}\sum_{i=1}^{m}\sum_{j>i}\sum_{b\neq k} ec_{ik}\, x_{ik}\, x_{jb} \qquad (2)$$

The system cost which is defined as the sum of the execution and communication cost is computed as follows:

$$\mathbf{C(X) = PEC(X) + IPEC(X)} \qquad (3)$$

### G. Expected Time to Compute

An Expected Time to compute makes an estimation of the computational load of each job, the computing capacity of each resource, and an estimation of the prior load of each one of the resources. Each position in ETC[t][m] indicates the expected time to compute job t in resource m [3].

$$\mathbf{M(X)} = \min \{\textstyle\sum_{\{j\, \varepsilon\, jobs \mid schedule[j] = m\}} ETC[j][m] \} \qquad (4)$$

Thus, we need to minimize the system cost and the makespan which is to allocate each of the m tasks to one of the n processors. Hence our fitness function is:

$$\mathbf{Min\ \{C(X) + M(X) = PEC(X) + IPEC(X) + M(X)\}} \qquad (5)$$

### H. Crossover

Crossover provides the important operation in evolutionary algorithm. The crossover operation is used to obtain new individuals (descendants) by selecting individuals from the parental generation and interchanging their genes. The aim is to obtain descendants of better quality that will feed the next generation and enable the search to explore new regions of solution space not yet explored [3].

### I. One-point crossover

Given two parent solutions, this operator, first chooses a position between 1 and n; where n is the chromosome length of a chosen individual. The resulting position serves as a ‘cutting point’ splitting each parent into two segments. Then, the two first parts of the parents are interchanged yielding two new descendants [3]. Also, the first part and the later part of the parents could be exchanged to form two new descendants; the converse is true. The illustration is as shown in Fig 2.

### *J. Mutation*

The mutation operation is performed on a single gene strand basis. The mutation operation will perform if the mutation ratio ($P_m$) is verified. The point to be mutated is selected randomly. The illustration is as shown in Fig 3.

### *K. Selection*

Selection operators are used to select the individuals to which the crossover operators will be applied. The fitness proportionate selection, also known as roulette wheel selection is chosen for recombination.

In fitness proportionate selection, as in all selection methods, the fitness function assigns fitness to possible solutions or chromosomes. The fitness level is used to associate a probability of selection with each individual chromosome.

If $f_i$ is the fitness of individual $i$ in the population, its probability of being selected is

$$pi = \frac{fi}{\sum N\, fj}$$
$$j = 1$$

where N is the number of individuals in the population [7].

This could be imagined similar to a Roulette wheel. Usually a proportion of the wheel is assigned to each of the possible selections based on their fitness value. This could be achieved by dividing the fitness of a selection by the total fitness of all the selections, thereby normalizing them to 1. The illustration is as shown in Fig 4.

### *L. Stopping Criteria*

Stopping Criteria is fulfilled from either one of the below:

i. Generation
ii. Required Fitness

The generation is set at some fixed value like 100, 1000 for example on fulfillment of which the algorithm stops.

Since we don't have fix value of required fitness, a minimum arbitrary value could be set on fulfillment of which the algorithm stops.

↓ Cutting point

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Parent 1 | 1 | 2 | 1 | 1 | 2 | 1 | 2 | 3 |
| Parent 2 | 2 | 3 | 1 | 2 | 1 | 3 | 2 | 1 |
| Child 1 | 1 | 2 | 1 | 1 | 1 | 3 | 2 | 1 |
| Child 2 | 2 | 3 | 1 | 2 | 2 | 1 | 2 | 3 |

Fig 2.One-point Crossover

| | | | | | | |
|---|---|---|---|---|---|---|
| Parent | 1 | 3 | **2** | 1 | 2 | 3 |
| Child | 1 | 3 | **3** | 1 | 2 | 3 |

Fig 3. Mutation

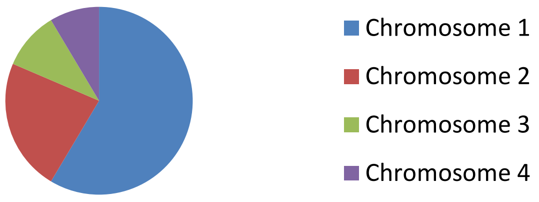

Fig 4. Roulette Wheel Selection

## IV. Result Analysis

The result set shows the optimal solution for the input data set using the genetic algorithm. The solutions are derived using different parameter set. Each parameter set comprises of the necessary parameters for the algorithm. The variant population size in the parameter set provides the variant solution space which enables the algorithm to find a diverse set of solutions from the larger solution space. The input data-set can be considered as shown in Fig 5 and 6. The input data-set is taken from the reference papers and then arbitrarily replicated to test the performance of the algorithm. The data is expressed in the units of cost for execution cost matrix and in the units of time for expected time to compute matrix.

| Processor | P1 | P2 | P3 |
|---|---|---|---|
| Task | | | |
| t1 | 174 | 176 | 110 |
| t2 | 95 | 15 | 134 |
| t3 | 196 | 79 | 156 |
| t4 | 148 | 215 | 143 |
| t5 | 44 | 234 | 122 |
| t6 | 241 | 225 | 27 |
| t7 | 12 | 28 | 192 |
| t8 | 215 | 13 | 122 |
| t9 | 211 | 11 | 208 |

Fig 5. Data-set: Execution Cost Matrix [4]

| Processor | P1 | P2 | P3 |
|---|---|---|---|
| Task | | | |
| t1 | 25137.5 | 52468 | 150206 |
| t2 | 30802.6 | 42744.5 | 49578.3 |
| t3 | 242727.1 | 661498.5 | 796048.1 |
| t4 | 68050.1 | 303515.9 | 324093.1 |
| t5 | 6480.2 | 42396.7 | 98105.4 |
| t6 | 175953.8 | 210341.9 | 261825.0 |
| t7 | 116821.4 | 240577.6 | 241127.9 |
| t8 | 36760.6 | 111631.5 | 150926 |
| t9 | 383709.7 | 442605.7 | 520276.8 |

Fig 6. Data-set: Expected Time to Compute Matrix

| Run | Population | Cost | Make span |
|---|---|---|---|
| 3 | 121113122 | 640 | 1318023 |
| 4 | 212113122 | 605 | 1752183 |
| 15 | 121111111 | 1256 | 1098384.9 |
| 34 | 322113122 | 459 | 1861862.9 |

Fig 7. Summary of Output Solution for Test Data set with Parameter Set 1

| Run | Population | Cost | Make span |
|---|---|---|---|
| 6 | 221113122 | 642 | 1345353.5 |
| 17 | 121111111 | 1256 | 1098384.9 |
| 25 | 322113232 | 584 | 2024913.6 |
| 42 | 322313222 | 470 | 2241662.1 |

Fig 8. Summary of Output Solution for Test Data set with Parameter Set 2

The input data-set is fed into the algorithm taking different GA parameters as shown in Fig 10 or Fig 11. The algorithm is ran multiple instances and with each instance, it tries to find the suitable solution. For the input data-set with the different parameter set, the output can be summarized as shown in Fig 7 and Fig 8.

As we can see, we have the diverse set of solutions with the optimal values. The optimum could be from cost perspective or make span perspective. If we see for cost values, then we achieve the cost of *640 units* at the make span of *1318023 units*, now if we further optimize the cost to 605 units, the make span increment to *1752183 units*. If we try to optimize make span, we have the make span of *1098384.9 units*, and our cost increments to *1256 units* which is quite drastic if we try to achieve both cost and make span. One other solution with the cost *459 units* and make span *186186.9 units* seems quite optimal in comparison to other solutions as the cost is quite low and we don't have to sacrifice much for make span as well. The optimal solution for the input data set can illustrated as shown in Fig 9.

Using different parameter set provides variant optimal solutions. With the increase in population size, the solution space increases which provides more efficient and optimal solutions. The population size herein is doubled in case of second parameter set. With the larger solution space, it is more likely to find the diverse set of solutions, which enables more optimality and efficiency.

With the increase in crossover probability, it is likely to find the diverse set of solutions as individuals undergoes interchanging their genes. The more interchange takes place, the more diversity is achievable. With crossover, it does not only guarantee the optimality but also could decrease it. It is likely to find the fit individual after crossover but sometimes, the fit individual could undergo crossover and end up in worst or unfit individual. So, the best individual needs to be preserved with generations which guarantee elitism. The crossover probability is chosen between 60-80% to guarantee the best individual is preserved. The comparison of the output between the parameter sets can be shown in Fig 10, 11 and 12. The horizontal x-axis in Fig 12 represents the cost value in the units of cost and vertical y-axis represents the make span in the units of time.

| t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | t9 |
|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 2 | 1 | 1 | 3 | 1 | 2 | 2 |

Fig 9. Optimal Solution for Test Data Set

| GA Parameters | Value |
|---|---|
| Population Count | 10 |
| Crossover Probability | 0.8 |
| Mutation Probability | 0.01 |
| Chromosome Length | 9 |
| Chromosome | 123 |
| Processor Count | 3 |
| Task Count | 9 |
| Generation | 100 |

Fig 10. Parameter Set 1

| GA Parameters | Value |
|---|---|
| Population Count | 20 |
| Crossover Probability | 0.8 |
| Mutation Probability | 0.01 |
| Chromosome Length | 9 |
| Chromosome | 123 |
| Processor Count | 3 |
| Task Count | 9 |
| Generation | 100 |

Fig 11. Parameter Set 2

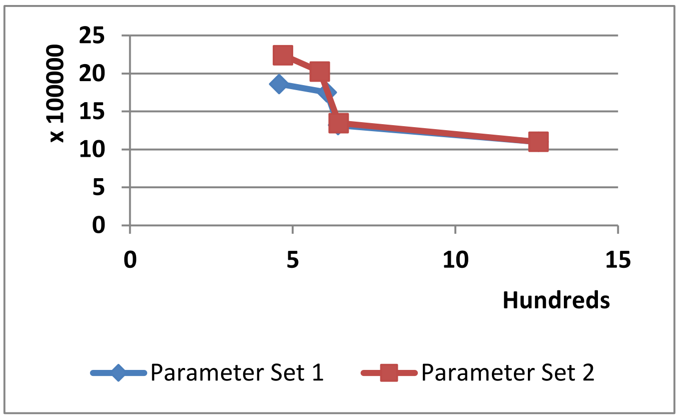

Fig 12. Comparison of Parameter Sets for Test Data Set

From the analysis we saw that, the optimality can be seen from different perspectives in case of multiple objectives and there isn't a single fixed optimal solution. We suggested the pool of optimal solutions and chose the best among them to be optimal. Also, with parameter tuning we formed the larger solution space to find the diversity which enables to find more diverse optimal solutions. The users are allowed to choose one solution among the others which meet their requirement and necessity. It isn't essential to have only one solution in case of multi-objective optimization.

## V. CONCLUSION

In this paper, we implemented the genetic algorithm to optimize the task scheduling problem both from time and cost perspective, providing the optimal solutions which are the schedulers for efficiently allocating jobs to resources in a grid system. The algorithm only considered static schedulers and does not consider grid characteristics such as consistency of computing, heterogeneity of resources and jobs.

## ACKNOWLEDGMENT

Ujjwal P. would like to express his gratitude to all the faculty members of Electronics and Computer Engineering Department who are directly/indirectly involved in inspiring and motivating him to carry out this research, along with his colleagues for their continuous idea sharing and discussion, which have definitely paved the path to succeed this research.

## REFERENCES

[1] Mohammad I. Daoud and Nawwaf Kharma, "An Efficient Genetic Algorithm for Task Scheduling in Heterogeneous Distributed Computing Systems", July 2006.

[2] Fangpeng Dong and Selim G. Akl "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", January 2006.

[3] Javier Carretero, Fatos Xhafa "Genetic Algorithm Based Schedulers for Grid Computing Systems", Vol 3, No. 6, December 2007.

[4] Ahmed Younes. Hamed "Task Allocation for Minimizing Cost of Distributed Computing Systems using Genetic Algorithms", Vol 2, Issue 9, September 2012.

[5] Prateek Kumar Singh, Neelu Sahu "Task Scheduling in Grid Computing Environment Using Compact Genetic Algorithm", Vol3, Issue 1, January 2014.

[6] Andrew J. Page and Thomas J. Naughton "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms", Department of Computer Science, National University of Ireland, Maynooth, County Kildare, Ireland. 2005.

[7] https://en.wikipedia.org/wiki/Fitness_proportionate_selection