TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

CENTRAL CAMPUS

THESIS NO: 069/ MSCS/ 669

Task Scheduling in Grid Computing Using Genetic Algorithm

by

Ujjwal Prajapati

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

NOVEMBER, 2014

Task Scheduling in Grid Computing Using Genetic Algorithm

by

Ujjwal Prajapati

069/MSCS/669

Thesis Supervisor

Prof. Dr. Subarna Shakya

Department of Electronics and Computer Engineering

Institute of Engineering

Central Campus

A thesis submitted to the Department of Electronics and Computer Engineering in partial fulfillment of the requirements for the degree of Master of Science in Computer System and Knowledge Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Central Campus

Tribhuvan University

Lalitpur, Nepal

November, 2014

COPYRIGHT ©

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

CENTRAL CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certify that they have read and recommended to the Institute of Engineering for acceptance, a thesis entitled **"Task Scheduling in Grid Computing Using Genetic Algorithm"** submitted by **Mr. Ujjwal Prajapati** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"**

……

Supervisor: **Prof. Dr. Subarna Shakya**
Department of Electronics and Computer Engineering
Institute of Engineering
Central Campus

……

External Examiner:

DEPARTMENTAL ACCEPTANCE

The thesis entitled **"Task Scheduling in Grid Computing using Genetic Algorithm"** submitted by **Ujjwal Prajapati** in partial fulfillment of the requirement for the award of the degree of **"Master of Science in Computer System and Knowledge Engineering"** has been accepted as a bonafide record of work independently carried out by him in the department.

Dr. Diwakar Raj Panta

Head of the Department

Department of Electronics and Computer Engineering,

Central Campus,

Institute of Engineering,

Tribhuvan University,

Nepal.

ACKNOWLEDGEMENT

I would like to express my gratitude to all the faculty members of Electronics and Computer Engineering Department who are directly/indirectly involved in inspiring and motivating me to carry out this thesis. I am very grateful to my thesis Supervisor **Professor Dr. Subarna Shakya** for his guidance and encouragement throughout the thesis. This would be never succeeded without his support and inspiration. A special word of gratitude to my program co-ordinator **Dr. Sanjeeb Prasad Panday** and **Professor Dr. Shashidhar Ram Joshi** whose continuous support and guidance has led me to come up with this title of thesis.

Also, I would like to express a word of thanks to all my classmates for their continuous idea sharing and discussion, which have definitely paved the path to undertake this thesis.

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 and flowtime 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

TABLE OF CONTENTS

LIST OF TABLES

TABLE **PAGE**

CHAPTER ONE

INTRODUCTION

1.1 Introduction

A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by heterogeneous networks. 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. A task scheduling is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains. 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. 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.

1.2 Problem Statement

Job sharing is one of the major difficult tasks in a computational grid environment. Unlike scheduling problems in conventional distributed systems, this problem is much more complex as new features of Grid systems such as its dynamic nature and the high degree of heterogeneity of jobs and resources must be tackled. When there are more tasks than available resources, the problems arises for the minimization of the completion time and utilize the resources effectively with minimum cost.

1.3 Objectives

- To implement Genetic Algorithm for the multi-objective optimization of make span, flow time and economic cost of the system
- To see and compare the possible performance change considering different encoding schemes, operators and parameter tuning

1.4 Scope of Work

In any of the Grid computing where task scheduling is a necessarily complicated, this solution could be implemented. The solution could be run in Grid Scheduler or Grid Resource Manager which provides the functionality for discovery and publishing of resources along with scheduling, submission and monitoring of jobs.

CHAPTER TWO

LITERATURE REVIEW

2.1 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.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Methodology

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.

begin

Initialization: Generate the initial population *P (t=0)* of n individuals

Fitness: Evaluate the fitness of each of the population.

Evaluate (P (t))

while (not termination condition) **do**

Selection: Select a subset of *m* pairs from *P (t)*.

Let P_1 (t) = Select (P (t))

Crossover: With probability p_c, cross each of the m chosen pairs.

Let P_2 (t) = Cross (P_1 (t)) be the set of offspring.

Mutation: With probability P_m, mutate each offspring in P_2 (t).

Let P_3 (t) = mutate (P_2 (t))

Fitness: Evaluate the fitness of each offspring. Evaluate (P_3 (t))

Replacement: Create a new generation from individuals in P (t) and P_3 (t).

Let P (t+1) = Replace (P (t), P_3 (t)); t = t+1

fwhile

return Best found solution;

end

3.1.1 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,….t_n} and set of resources as P = {P_1, P_2, P_3,… P_n}, then the sequence can be represented as:

t_1	t_2	t_3	t_4	..	..	t_m
P_1	P_3	P_1	P_2	..	..	P_n

Table 1 Task Allocation in the form of chromosome

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.

3.1.2 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.

3.1.3 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. 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. The makespan is expressed in terms of Expected Time to compute matrix (ETC) of size m*n.

3.1.3.1 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.

3.1.3.2 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 $cc_{ij} = 0$

3.1.3.3 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:

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

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

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

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

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

3.1.3.4 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.

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

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:

Min {C(X) + M(X) = PEC(X) + IPEC(X) + M(X)}

3.1.4. 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.

3.1.4.1. 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. Also, the first part and the later part of the parents could be exchanged to form two new descendants; the converse is true.

				↓ 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

Table 2 One-point Crossover

3.1.5 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.

Parent	1	3	**2**	1	2	3
Child	1	3	**3**	1	2	3

Table 3 Mutation

3.1.6 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

$$pi = \frac{fi}{\sum_{j=1}^{N} fj}$$

where N is the number of individuals in the population.

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.

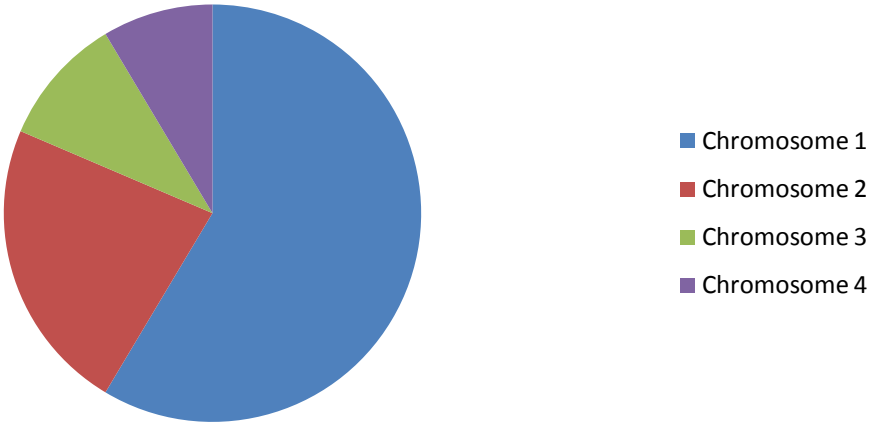

Table 4 Roulette Wheel Selection

3.1.7. 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.

3.2 Data Collection

I have considered a typical program made up by 9-executable tasks {t1, t2, t3, t4, t5, t6, t7, t8, t9} and 4 executable tasks {t1, t2, t3, t4} to be executed on the Distributed Computing System having three processors {P1, P2, P3}. I have taken the execution cost of each task on different processors and inter task communication cost between the tasks in the form of matrices Execution Cost Matrix and Inter-Task Communication Cost Matrix. Also, the expected time to compute matrix is formulated. All the matrices are shown in table below.

3.2.1 Data Set 1

Processor / Task	P1	P2	P3
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

Table 5 Data-set 1: Execution Cost Matrix [4]

Tasks	t1	t2	t3	t4	t5	t6	t7	t8	t9
t1	0	8	10	4	0	3	4	0	0
t2	8	0	7	0	0	0	0	3	0
t3	10	7	0	1	0	0	0	0	0
t4	4	0	1	0	6	0	0	8	0
t5	0	0	0	6	0	0	0	12	0
t6	3	0	0	0	0	0	0	0	12
t7	4	0	0	0	0	0	0	3	10
t8	0	3	0	8	12	0	3	0	5
t9	0	0	0	0	0	12	10	5	0

Table 6 Data-Set 1: The Inter Task Communication Cost Matrix [4]

Processor Task	P1	P2	P3
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

Table 7 Data-set 1: Expected Time to Compute Matrix

3.2.2. Data Set 2

Processor Task	P1	P2	P3
t1	9	2	6
t2	3	8	7
t3	7	10	3
t4	3	4	9

Table 8 Data-set 2: Execution Cost Matrix [4]

Tasks	t1	t2	t3	t4
t1	0	1	4	6
t2	1	0	2	0
t3	4	2	0	8
t4	6	0	8	0

Table 9 Data-set 2: The Inter Task Communication Cost Matrix [4]

Processor Task	P1	P2	P3
t1	392348.2	399562.1	441485.5
t2	58268.1	58987.9	85213.2
t3	915235.9	925875.6	978057.6
t4	841877.3	856312.9	861314.8

Table 10 Data-set 2: Expected Time to Compute Matrix

CHAPTER FOUR

RESULT ANALYSIS AND COMPARISON

4.1. Result

The collected dataset is fed to the algorithm which outputs the solution which is the schedulers for the grid computing. Multiple solutions are shown with different cost and make span which enables users to chose one among the others which fit best to their requirement.

4.1.1. Test Dataset 1:

For a given input dataset 1 as shown in Table 5, 6 and 7, we feed the input to the algorithm with multiple test parameters as shown in Table 11 and 20. The population size is chosen different to test the output solution. Given the variant population size, the output solution space varies giving the larger solution set. The output solution is tabulated at different run of the program to provide optimal values. From the output solution derived, task allocation is performed giving the system cost and make span.

4.1.1.1. Parameter Set 1

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

Table 11 Parameter Set 1: GA Parameters

i. Program Run 3

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

Table 12 Program Run 3: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t1, t3, t4,t5, t7	P1	640	1318023
t2, t8, t9	P2		
t6	P3		

Table 13 Program Run 3: Task Allocation

ii. Program Run 4

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

Table 14 Program Run 4: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2, t4, t5, t7	P1	605	1752183
t1, t3, t6, t8, t9	P2		
t6	P3		

Table 15 Program Run 4: Task Allocation

iii. Program Run 15

t1	t2	t3	t4	t5	t6	t7	t8	t9
1	2	1	1	1	1	1	1	1

Table 16 Program Run 15: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t1, t3, t4, t5, t6, t7, t8, t9	P1	1256	1098384.9
t2	P2		
	P3		

Table 17 Program Run 15: Task Allocation

iv. Program Run 34

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

Table 18 Program Run 34: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t4, t5, t7	P1	459	1861862.9
t2, t3, t8, t9	P2		
t1, t6	P3		

Table 19 Program Run 34: Task Allocation

4.1.1.2. Parameter Set 2

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

Table 20 Parameter Set 2: GA Parameters

i. Program Run 6

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

Table 21 Program Run 6: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t3, t4,t5, t7	P1	642	1345353.5
t1, t2, t8, t9	P2		
t6	P3		

Table 22 Program Run 6: Task Allocation

ii. Program Run 17

t1	t2	t3	t4	t5	t6	t7	t8	t9
1	2	1	1	1	1	1	1	1

Table 23 Program Run 17: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t1, t3,t4, t5, t6, t7, t8, t9	P1	1256	1098384.9
t2	P2		
	P3		

Table 24 Program Run 17: Task Allocation

iii. Program Run 25

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

Table 25 Program Run 25: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t4, t5	P1	584	2024913.6
t2, t3, t7, t9	P2		
t1, t6, t8	P3		

Table 26 Program Run 25: Task Allocation

iv. Program Run 42

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

Table 27 Program Run 42: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t5	P1	470	2241662.1
t2, t3, t7, t8, t9	P2		
t1, t4, t6	P3		

Table 28 Program Run 42: Task Allocation

4.1.2. Test Dataset 2:

For a given input dataset 1 as shown in Table 8, 9 and 10, we feed the input to the algorithm with multiple test parameters as shown in Table 29 and 38. The population size is chosen different to test the output solution. Given the variant population size, the output solution space varies giving the larger solution set. The output solution is tabulated at different run of the program to provide optimal values. From the output solution derived, task allocation is performed giving the system cost and make span.

4.1.2.1. Parameter Set 1

GA Parameters	Value
Population Count	10
Crossover Probability	0.8
Mutation Probability	0.01
Chromosome Length	4
Chromosome Gene	123
Processor Count	3
Task Count	4
Generation	100

Table 29 Parameter Set 1: GA Parameters

i. Program Run 1

t1	t2	t3	t4
1	1	1	1

Table 30 Program Run 1: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t1, t2, t3, t4	P1	22	2207729.5
	P2		
	P3		

Table 31 Program Run 1: Task Allocation

ii. Program Run 3

t1	t2	t3	t4
2	1	1	2

Table 32 Program Run 3: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2, t3	P1	16	2229379
t1,t4	P2		
	P3		

Table 33 Program Run 3: Task Allocation

iii. Program Run 4

t1	t2	t3	t4
2	1	3	2

Table 34 Program Run 4: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2	P1	12	2292200.7
t1, t4	P2		
t3	P3		

Table 35 Program Run 4: Task Allocation

iv. Program Run 15

t1	t2	t3	t4
2	1	3	1

Table 36 Program Run 15: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2, t4	P1	11	2277765.1
t1	P2		
t3	P3		

Table 37 Program Run 15: Task Allocation

4.1.2.2. Parameter Set 2

GA Parameters	Value
Population Count	20
Crossover Probability	0.8
Mutation Probability	0.01
Chromosome Length	4
Chromosome Gene	123
Processor Count	3
Task Count	4
Generation	100

Table 38 Parameter Set 2: GA Parameters

i. Program Run 1

t1	t2	t3	t4
2	1	3	2

Table 39 Program Run 1: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2	P1	12	2292200
t1, t4	P2		
T3	P3		

Table 40 Program Run 1: Task Allocation

ii. Program Run 2

t1	t2	t3	t4
2	1	3	1

Table 41 Program Run 2: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2, t4	P1	11	2277765.1
t1	P2		
t3	P3		

Table 42 Program Run 2: Task Allocation

iii. Program Run 5

t1	t2	t3	t4
2	1	1	1

Table 43 Program Run 5: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
t2, t3, t4	P1	15	2214943.4
t1	P2		
	P3		

Table 44 Program Run 5: Task Allocation

iv. Program Run 7

t1	t2	t3	t4
1	1	1	1

Table 45 Program Run 7: Output Solution

Task Allocation		System Cost	Make span
Tasks	Processors		
	P1	22	2207729.5
t2, t3	P2		
t1, t4	P3		

Table 46 Program Run 7: Task Allocation

4.2 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 solutions of the algorithm are actually the schedulers for efficiently allocating jobs to resources in a grid system.

For the input data set 1, we can tabulate the summarized output as follows:

Run	Cost	Make span
3	640	1318023
4	605	1752183
15	1256	1098384.9
34	459	1861862.9

Table 47 Summary of Output Solution for Test Data set 1

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 at the make span of 1318023, now if we further optimize the cost to 605, the make span increment to 1752183. If we try to optimize make span, we have the make span of 1098384.9, and our cost increments to 1256 which is quite drastic if we try to achieve both cost and make span. One other solution with the cost 459 and make span 186186.9 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. We can consider this as our optimal solution for the input data set 1.

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

Table 48 Optimal Solution for Test Data Set 1

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 between parameter sets can be shown in the chart below:

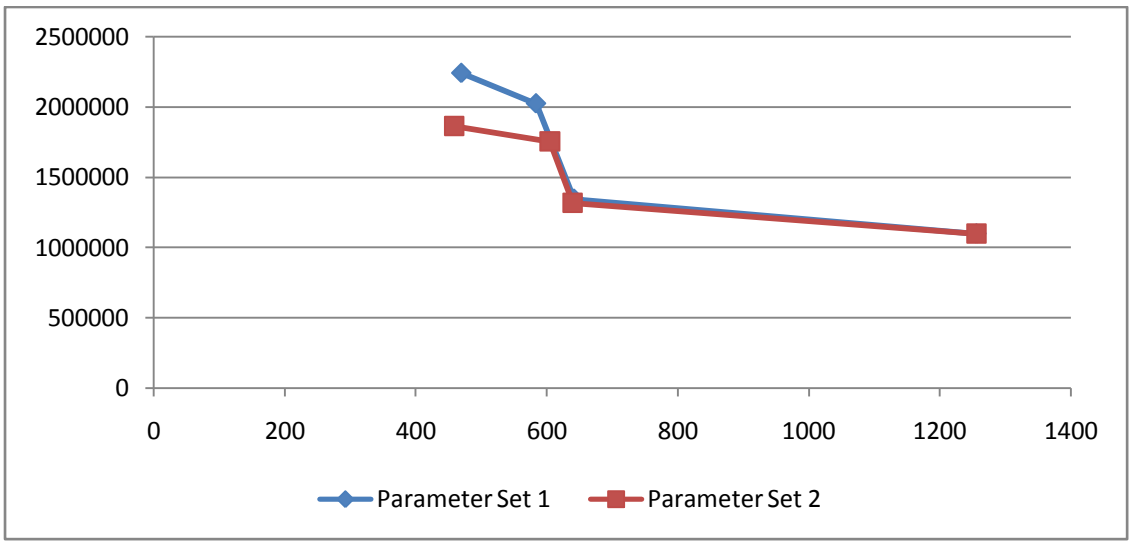

Table 49 Comparison of Parameter Sets for Test Data Set 1

For the input data set 2, we can tabulate the summarized output as follows:

Run	Cost	Make span
1	12	2292200
2	11	2277765.1
5	15	2214943.4
7	22	2207729.5

Table 50 Summary of Output Solutions for Test Data Set 2

From the diverse set of solutions for data set 2, we tabulated the optimal values both from cost and make span perspectives. We have the make span of 2277765.1 at the cost of 11 and make span of 2207729.5 at the cost of 22 which seems double from cost perspective but only few decrement for make span. From both cost and make span, we can see the cost of 15 at the make span of 2214943.4 which seems feasible and optimal. We can consider this solution as our optimal solution.

t1	t2	t3	t4
2	1	1	1

Table 51 Optimal Solution for Data Set 2

The variant population size and crossover probability provides the diverse set of solutions. The diversity and large solution space enables more efficient and optimal solutions. Comparing the output with different parameters, it is likely to find the more diverse solutions with the larger population size and crossover probability. With more diverse set of solutions, it is likely to find more optimal and efficient solutions. The comparison between the parameter sets can be shown in the chart below:

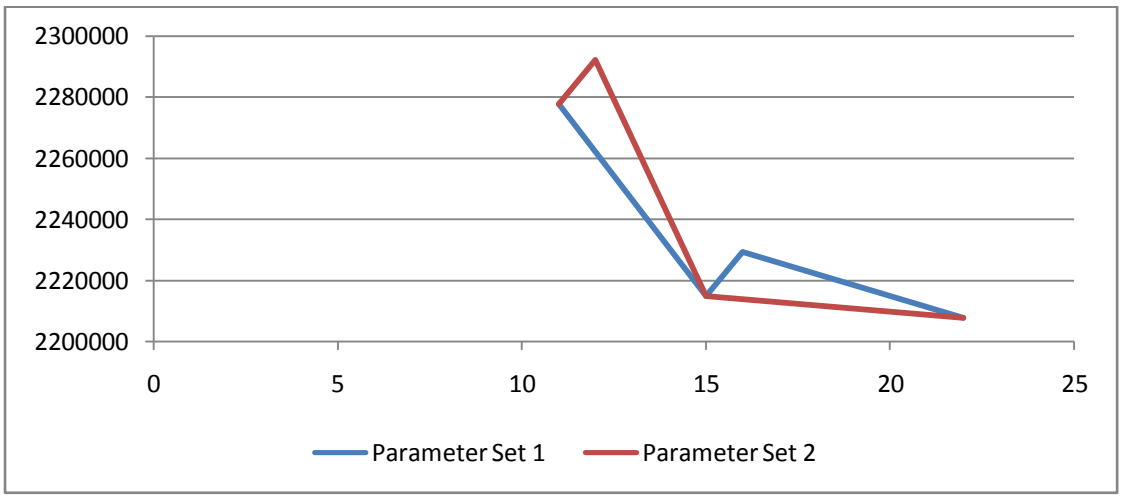

Table 52 Comparison of Parameter Sets for Test Data set 2

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.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Task scheduling problem in grid computing could be addressed by implementing genetic algorithm in order to find the efficient solution to the problem. The need of allocating a number of tasks to different resources for the efficient utilization of resources with minimal completion time and economic cost formulate the problem in multi-objective fashion. Given the problem in multi-objective formulation, use of the algorithm seems efficient enough to provide the solution. Optimizing the problem both from time and cost perspective and providing the multiple solutions seems possible using the algorithm.

Using genetic algorithm for multi-objective optimization of the task scheduling problem seems to be possible and efficient. Given 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 task scheduling problem. The solutions are the schedulers for efficiently allocating jobs to resources in a grid system.

5.2 Recommendation

The algorithm only considered static schedulers. The algorithm could be extended to use for dynamic schedulers in future. The algorithm does not consider grid characteristics such as consistency of computing, heterogeneity of resources and jobs. The algorithm only considered the input data set which is feed statically into the code. There is always a place for performance improvement and enhancement.

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