Asian Journal of Information Technology 15 (15): 2584-2593, 2016
ISSN: 1682-3915
© Medwell Journals, 2016

Design and Implementation of Non-Dominated Sorting Genetic Algorithm Scheduler Using Mapreduce Model

D. Rajeswari and V. Jawahar Senthil Kumar
Department of ECE, Anna University, Chennai, Tamil Nadu, India

Abstract: Parallel and distributed system plays a key role in the development of high performance systems. To achieve the high performance of a system, task scheduling is an important issue. At common, the problem of task scheduling has been considered to be NP-hard. Several algorithms put into practice to find the optimal schedule for task scheduling. Evolutionary algorithms are one of the best. In an evolutionary kind of algorithms, the time taken to find an efficient schedule is high. This study presents the implementation of Non-dominated Sorting Genetic Algorithm (NSGA-II) with MapReduce model. In a distributed system, most of the task scheduling problem is formulated as multi-objective optimization problem. For multi-objective problem formulation, minimization of makespan and flowtime is considered. MapReduce model can automatically parallelize the execution of NSGA-II. The algorithm is tested on a set of benchmark instances. Experimental results show that NSGA-II with MapReduce model minimizes the amount of time taken, makespan and flowtime than a Weighted Sum Genetic Algorithm (WSGA) with MapReduce model which is also implemented in this study for better comparison.

Key words: Non-dominated sorting, NSGA-II, distributed system, multi-objective, MapReduce, optimization

INTRODUCTION

Distributed and parallel computing environment has a collection of heterogeneous computing systems interconnected by communication channels to process complex computational problems. Efficiency and the use of Distributed Heterogeneous Computing System (DHCS) depend on the capacity of the system to satisfy computational requirements of complex computational problems. As the computing nodes are heterogeneous in a multiprocessor environment, execution time of tasks varies on each processor. Scheduling of tasks is a key issue, to achieve the high usability of supercomputing capacity of distributed computing environment. To ensure efficient utilization of resources, suitable scheduling algorithms are used to assign the tasks to the available processors efficiently. Scheduling methods are static or dynamic. Static scheduling methods are useful to analyze the heterogeneous computing environment (Braun *et al*., 2001). Static methods are also used in grids and clouds to distribute the computing resources to tasks (Foster and Kesselman, 2003). Hence, this study focuses on static scheduling.

To effective utilization of resources several factors like response time, resource usage, throughput, reliability, network traffic and others are to be optimized. This study considers the minimization of makespan and flowtime. Makespan is defined as the time when last task is finished and flowtime is the sum of finalization time of all the tasks (Liu *et al*., 2010). Since, both the objectives are contradicted with each other, multi-objective formulation is needed for this problem. Multi-Objective Optimization Problem (MOOP) has two approaches. First method combines the multiple objectives into a single objective before optimization using scalar cost function. Second method finds the set of Pareto optimal solutions or a subset. The second method is preferable for real life applications (Chankong and Haims, 2008).

In general, task scheduling on DHCS is complex due to the computational complexity of tasks. Multi-objective evolutionary methods are more suitable for this kind of environments. GA is one of the promising evolutionary techniques to schedule the complex computational task. GA is used in many areas like scheduling, biology, chemistry and CAD/CAM. GA finds an optimal solution in a complex search space (Goldberg, 1989). GA considers the entire problem space as individuals and use optimization method to search near optimal individuals by generating offspring. This is a time consuming process for complex tasks. To overcome this, Parallel Genetic Algorithm (PGA) was developed (Lim *et al*., 2007). The PGA used to divide the complex search space into smaller subspaces and find the sub optimal solution for each small space and then this sub optimal solution forms the

Corresponding Author: D. Rajeswari, Department of ECE, Anna University, Chennai, Tamil Nadu, India

optimal solutions. A lot of development difficulties like communication and synchronization between distributed computing resources are there in the PGA. This encourages the research on implementing NSGA-II with MapReduce programming model.

The MapReduce model is used to develop distributed applications. It provides the parallel design method to simplify the distributed application development. It divides the complex problem into small and automatically parallelizes the execution of small tasks. Google proposed this model to process data intensive applications in a large number of resources easily. It allows users to run the complex application in a distributed environment without worrying about the communication and synchronization among the distributed resources. The coordinator is used to perform all the sequential works. In this study, WSGA and NSGA-II with MapReduce programming model is applied to the complex scheduling problem. This technique will ensure time reduction to find the best optimal schedule without facing the difficulty of coordinating distributed component.

Related work: In the past, a lot of heuristic methods are implemented for scheduling independent tasks in a distributed environment. To find optimal schedule, all the heuristic methods for scheduling tasks to processors depends on some perception and performance differs depending on the conditions used (Izakian *et al*., 2009). A few well known heuristic methods are min-max (Munir *et al*., 2007), suffrage (Maheswaran *et al*., 1999), min-min, max-min (Freund *et al*., 1998) and LJFR-SJFR (Abraham *et al*., 2008). These above heuristic methods are more time consuming process. In recent, several meta-heuristic methods are developed to solve complex computational problems. The most popular methods are GA, Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Simulated Annealing (SA). The description of eleven heuristics and comparison on the various distributed environment was done by Braun *et al*. (2001) and illustrates the effectiveness of GA with others. All the above meta-heuristic methods considered single objective and aimed to minimize the makespan.

There are some methods considered multiple objectives, while scheduling tasks in distributed environments. Izakian *et al*. (2009) compares five heuristics depends on the machine and task characteristics for minimizing both makespan and flowtime but calculated separately. Several nature inspired meta-heuristic methods like GA, PSO, ACO and SA for scheduling tasks in a grid computing environment by using single and multi-objective optimization was done by Abraham *et al*. (2008). Carretero *et al*. (2007) implemented GA based scheduler. All the above methods convert the multi-objective optimization problem into a scalar cost function which makes single objective before optimization.

To minimize the amount of time to find the best optimal schedule (Lim *et al*., 2007) implemented PGA, Durillo implemented parallel execution of NSGA-II and these methods have the difficulties to make communication and synchronization between the resources in a distributed environment. Chao implemented MRPGA to combine GA and MapReduce, but they extend the basic MapReduce model to MapReduceReduce. Verma *et al*. (2009) shows GA can be combined with the MapReduce model without extension. This study implements WSGA and NSGA-II with MapReduce programming model to find the best optimal schedule. It makes the task scheduling as an efficient real time multi-objective optimization problem.

Table 1: An example ETC matrix

Tasks	Processor 1	Processor 2
Task 1	5	3
Task 2	2	6
Task 3	4	1
Task 4	3	7

Problem statement: Real world heterogeneous computing systems are complex combinations of hardware, software and network components. The problem is formulated as follows. M is set of p machines in a heterogeneous computing environment and T is set of n tasks assigned to the machines. As scheduling is performed for independent tasks, there is no communication among tasks and a task is assigned to a processor exclusively. The pre-emption of task is not allowed. As scheduling is performed statically, computing capacity of machine, computational load of the task and prior load of the entire machine is estimated. Expected Time to Compute matrix (ETC) can be built by having the workload of task and computing capacity of machines. An ETC matrix is a n×p matrix where each position, ETC[n][p] indicates the expected time to compute task n in machine p. Each row of ETC matrix has the execution time of a task on each processor and each column specifies the estimated execution time of a processor for all the tasks. An example ETC matrix is given in Table 1 for 4 tasks and 2 processors. ETC matrix is explicitly provided and MapReduce programming model is used to reduce the time taken to find the optimal schedule.

Fitness function formulation: The objective is minimized of makespan and flowtime. Makespan is the time when last task is finished and flowtime is the sum of finalization time of all the tasks described as follows:

$$\text{makespan} = \min_{s_j \varepsilon Sch}\left(\max_{\text{tetasks}} F_t\right) \quad (1)$$

$$\text{flowtime} = \min_{s_j \varepsilon Sch}\left\{\sum_{\text{tetasks}} F_t\right\} \quad (2)$$

Where:

F_t = Completion time of jth task

Sch = Overall schedule and tasks is set of all tasks want to be scheduled

MATERIALS AND METHODS

Multi-objective GA: GA is fitted for multi-objective problem in a population based approach. GA has the ability to search the different regions in a solution space for finding a diverse set of solutions to the complex problems. The crossover operator produces the new individual by interchanging the genes of the parents. Mutation alters one or more genes in an individual.

WSGA: The weighted sum method is used to make a multiple objective into single objective. GA is computerized optimization and search algorithm based on the mechanics of natural genetics and natural selection which comes under the type of optimization techniques and non-traditional search used for searching large solution spaces. According to the survival of the fittest, natural populations evolved after many generations. The real world problems have been encoded suitably for GA to find the solutions (Abraham *et al*., 2008). This study uses the following template of GA (Doulabi *et al*., 2012).

Algorithm templates of GA:

Initialization : Produce the random initial population (pop) of n individuals.
Fitness : Calculate the each individual fitness (pop (t)).
While (not met stopping criteria) do
{
Selection : Select m pairs from pop (t),
pop1 (t) = select (pop (t)).
Crossover : Perform crossover on pairs of individuals (m), with probability pc,
pop2 (t) = crossover (pop1 (t)). Pop2 (t) has offspring.
Mutation : Mutate each offspring in pop2 (t) with probability pm.
Fitness : Calculate the fitness of offspring.
pop3 (t) = calc (pop3 (t))
}
Return best solutions

Schedule encoding: In evolutionary algorithms, encoding of an individual in the population is a key issue. The individuals are solution of the problem. Encoding determines the type of operators used for evolution. The behavior of evolutionary algorithm depends on representation and reproduction operators.

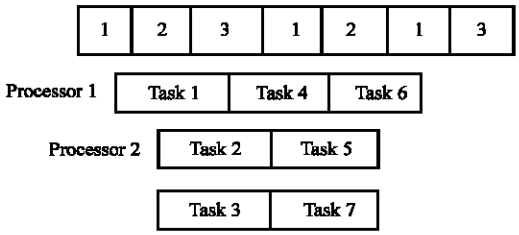

Fig. 1: Task assignment and schedule representation

Representation: Each individual is represented as a vector which is also called as schedule. The length of a schedule is equal to the number of tasks. The value available in position n in the schedule represents the resource to which the task n is allotted. The values of the schedule are numbers in the range (1, number of resources). The resource number can appear more than once in the schedule. A task assignment and schedule representation of 7 tasks and 3 resources in Fig. 1 (Subashini and Bhuvaneswari, 2010).

Fitness function: As makespan and flowtime are considered for calculating fitness function, weighted sum method is used. Fitness value is calculated as:

$$\text{Minimize fitness} = w1.\text{makespan} + w2 \times \text{mean flow time Minimize} \quad (3)$$

Subject to:

$$w1+w2 = 1$$

where, w1 and w2 are the weights assigned. To minimize the combined objective function, the individual with lower fitness value is selected.

Selection: Selection operator is used to select best offspring for the next iteration. Large tournament selection produces better results. The N individuals are participating in the tournament and the best will be selected for the next iteration.

Genetic operators: In this study one point crossover and swap mutation is implemented for WSGA.

One point crossover: It selects the pair of individuals and chooses a random position between 1 and a number of tasks. The random position is known as cut point. It splits

each individual into two parts. The first part of each individual is interchanged and produces new offspring.

Swap mutation: It interchanges the tasks between two resources in the individual. To apply this mutation operator the two tasks should be assigned to different resources.

After completing one generation, the available individuals are considered as the parent for the next generation. This procedure is executed continuously till the number of iterations given.

Elitist Non-Dominated Sorting Genetic algorithm: At first, Goldberg (1989) proposed Pareto based fitness assignment. The basic idea is assigning equal probability for producing all non-dominated individuals in the population. NSGA-II has fast, non-dominated sorting method with complexity O (k μ2) to provide the solution close to Pareto optimal solution. A crowding distance assignment and combined ranking method ensure the range of the individuals in the population. Elitism concept is used to preserve the best parent solution for the next iteration. NSGA-II is one of the efficient methods for multi-objective optimization problem. This study presents the following template of NSGA-II (Deb *et al*., 2002):

NSGA algorithm:

Step 1: Initialize the population of size N randomly (pop (t)).
Step 2: Perform non-dominated sorting of the population to classify it into a number of fronts.
Step 3: Perform crowded tournament selection by assigning crowding distance.
Step 4: Perform crossover and mutation to produce the offspring of size N. child (t) = mutate (cross (pop (t))).
Step 5: Combine parent and offspring of size 2N. total (t) = pop (t) U child (t)
Step 6: Update the population by copying the individuals from lowest front to size N. Small crowding distance individual will be dropped in the tie.
Step 7: Stop the process, once the stopping criteria are met. Otherwise, go to step2

Genetic operators: This study implements one point crossover and swap mutation for NSGA-II.

Non-dominated sorting: It is used to find the solution to the next iteration by classifying the population. The procedure for non-dominated sorting is given below (Deb *et al*., 2002):

Elitist Non-Dominated Sorting Genetic algorithm:

Step a: For each solution x in population N
Step b: For each solution y in population N
Step c: If x and y are not equal
Compare x and y for all the objectives
Step d: For any x, y is dominated by x, mark solution y as dominated
First non-dominated set is formed from unmarked solutions
Step e: Repeat the procedure till the entire population is divided into fronts

Selection: Crowded tournament selection operator is used. An individual x wins the tournament with another individual y, if any of the following is true:

- An individual x has better rank, i.e., rank x<rank y
- The individual x and y have the same rank (rank x = rank y) then the individual x has better crowding distance (in less crowded areas, i.e., d x = d y) than individual y

Crowding distance calculation: Crowding distance is used to break the tie between individuals which are having the same rank (Deb *et al*., 2002). The following steps are used to calculate the crowding distance:

Crowding distance algorithm:

Step a: Initialize the number of individuals (x) in the front (Fa)
Step b: Set the crowding distance, d i = 0, i = 1, 2...x
Step c: Sort the individuals (x) in front (Fa) based on the objective function (obj). obj = 1, 2...m. m is the number of objectives and S = sort (Fa, obj)
Step d: Set the distance of boundary individuals as S (d1) = ∞ and S (d x) = ∞
Step e: Set k = 2 to (x-1) and calculate S (d2) = S (d x-1) as follows:

$$S(d_k^m) = S(d_k) + \frac{s(k+1)f_m - S(k-1)f_m}{f_m^{max} - f_m^{min}} \quad (4)$$

$S(k)f_m$ is the kth individual value in S for m th object function

Multi-objective GA with MapReduce programming model: Hadoop is a framework used to provide a reliable, distributed storage and processing. The storage portion of Hadoop is called as Hadoop Distributed File System (HDFS) and processing part is called as MapReduce. Several other components are available in Hadoop for different purpose. Hadoop helps the users to have customized scripts to analyze complex data sets.

HDFS: It is a primary distributed storage used by Hadoop applications. Hadoop works with any distributed file system which is mounted by underlying OS but it needs to know which servers are near to the data. HDFS is developed with Hadoop for locality, fault tolerance and reliability. HDFS is a part of Hadoop cluster and it can be act as stand-alone general purpose distributed file system. HDFS splits the large file into a number of chunks and stored it in the different nodes. Each chunk is replicated at the nodes of Hadoop cluster. At the time of failure data is re-replicated by the active Hadoop monitoring system. HDFS consists of name node and data node. Name node coordinates HDFS and it has metadata information and secondary name node has a replication of metadata.

MapReduce: It supports distributed processing of terabytes of data in the nodes of the cluster with fault

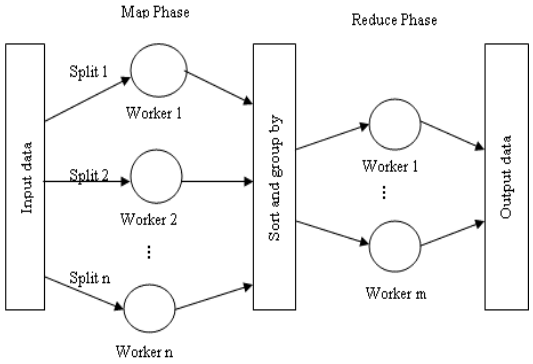

Fig. 2: MapReduce architecture

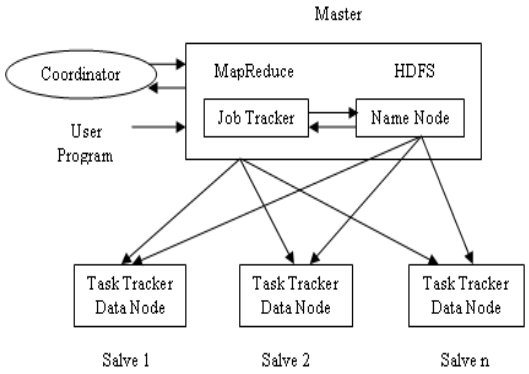

Fig. 3: Workflow on MapReduce and HDFS

tolerant and reliable manner. It splits the processing into sub processes to execute simultaneously on different nodes that are available on the same server or remote server. In general, it is master slave architecture. The architecture of MapReduce is presented in Fig. 2.

Master splits the task into sub tasks and assign to different slaves. Map phase divides a user program into sub tasks and generates a set of key-value pairs. It will be submitted to the reducer after a shuffle. The reduce phase performs user supplied reduce function on same key values to generate single entity. The reduce phase is also called as merge phase. MapReduce has job tracker and task tracker. The job tracker is available in the master. All the nodes in the Hadoop cluster have a task tracker. The workflow of MapReduce is presented in Fig. 3. The MapReduce function is represented as, map:: (input _ record) => list (key, values); reduce:: (key, list (values) => key, aggregate (values).

WSGA with MapReduce: WSGA is implemented as MapReduce programming in two ways:

- Global parallelism
- Local parallelism

Global parallelism: In this method, the fitness evaluation of WSGA alone has done parallel in the number of workers available in the map phase. The evaluation of individuals is executed parallel because the fitness calculation is independent from others in a population:

- The initial population is seeded into the coordinator
- The coordinator produces the offspring population
- The job tracker divides the offspring population into sub populations and assigns to workers in the map phase
- The workers perform the fitness evaluation for assigned individuals concurrently
- The workers of reduce phase collect the fitness values and perform merge operation then send it back to the coordinator for the next generation

Local parallelism: This approach can parallelize the crossover, mutation and fitness evaluation of individuals. The crossover operation performed on a pair of individuals belongs to same worker. The mutation operation and fitness of an individual is independent of other individuals in a population. Workflow for local parallelism of WSGA with MapReduce consists of following steps:

- Initial population is stored in the coordinator
- Coordinator evaluates the fitness and perform selection
- After selection, job tracker splits the population and assigns to workers of map phase
- The workers perform crossover, mutation and produce offspring, then evaluate the fitness of offspring assigned
- The workers of reduce phase collect the fitness value and perform merge operation, then send it back to the coordinator for the next generation

NSGA-II with MapReduce: Like WSGA, NSGA-II can be implemented as MapReduce program.

Global parallelism: The fitness evaluation of offspring alone has done parallel in the workers available in the map phase. Non-dominated sorting, crowded tournament selection is performed on an entire population. So, it cannot be fit into concurrent process:

- Initial population is loaded into coordinator
- Coordinator evaluates the fitness value; perform non-dominated sorting, crowded tournament selection, crossover and mutation. The offspring generated by coordinator sends to job tracker

- The job tracker splits the offspring population and send to workers of map phase to the evaluate fitness value in parallel manner and send it to reduce phase
- Shuffle operation is performed between map and reduce phase
- The workers of reduce phase aggregate the fitness value and send it to the coordinator

Local parallelism: This approach can parallelize the execution of crossover, mutation and fitness evaluation. The crossover performed on a pair of individuals in same worker. Mutation and fitness evaluation of individuals are independent from others in a population. The flow of local parallelism for NSGA-II with MapReduce has following steps:

- The initial population is stored in coordinator
- The coordinator performs a fitness evaluation, non-dominated sorting and crowded tournament selection and send it to job tracker
- The job tracker splits the population and send to the worker nodes
- The workers perform the crossover, mutation and fitness evaluation of different individuals concurrently
- The reduce phase collects all the fitness values and send it to the coordinator

RESULTS AND DISCUSSION

Experimental setup: Expected Time to Compute (ETC) matrix for 512 tasks and 16 processors are simulated (Abraham *et al*., 2008) to compare the performance and execution time of global and local parallelism of WSGA and NSGA-II. In this study, the MapReduce programming model is used to implement the WSGA and NSGA-II then performance and execution time are compared.

Hadoop setup: Hadoop 1.2.1 stable version is used to setup 4 node clusters which is backed up by HDFS. Hadoop cluster is running on Ubuntu Linux platform and Java 1.6 is used for writing the code. All the 4 systems have i5 processor, 4GB RAM and 500GB hard disk.

Data set description: At first, ETC matrix for 512 tasks and 16 processors is simulated randomly using the following specifications. Using repeatedly choosing m uniform random floating point values between 1 and φt an mx1 baseline column vector A is generated. Each value A (i) in A is multiplied by a uniform random numbers which has an upper bound of φs. Each row in the ETC matrix is then given by A (i) xs one row requires n different values of s.

Table 2: Parameter setting

Parameters	WSGA	NSGA-II
Population size	200	200
Number of iterations	1000	1000
Crossover probability (p_c)	0.8	0.8
Mutation probability (p_m)	0.01	0.01
Crossover type	Single point	Single point
Mutation type	Swap	Swap
Selection type	Large tournament	Large tournament
Weights used	w1 = 0.75, w2 = 0.25	-

This process is repeated for each row until the m×n matrix is full. The vector A is not used in the actual matrix. Hence, the values in the ETC matrix are within the range (1, φt, φs).

To get real heterogeneous environment, different kind of ETC matrices is simulated based on task heterogeneity, processor heterogeneity and consistency. The possible variance in execution time of tasks is known as task heterogeneity. High task heterogeneity was considered as φt = 3000 and low task heterogeneity φt = 100. Variation in execution time of a task in all the processors is called as processor heterogeneity. The high processor heterogeneity was φs = 1000 and low task heterogeneity φs = 10. Three various ETC consistencies are consistent, inconsistent and semi consistent which are used to capture the real heterogeneous environment. An ETC matrix is said to be consistent, if a processor Pm executes task j faster than processor Pn, all the tasks should be executed faster by Pm than Pn. Inconsistency means that a processor Pm executes some task faster and some tasks slower than Pn. A semi consistent ETC matrix is set in an inconsistent matrix which has a predefined size of consistent sub matrix. To generate a consistent matrix, each row of the matrix was sorted independently with processor Pn executes task j slower than processor Pm. Inconsistent matrices are generated randomly. To model semi consistent matrix, in each row even column elements are sorted and odd column elements are left as same. The various instances are labelled as x-yy-zz that represents the following:

- x-Consistency type (c-consistent, i-inconsistent and s-semi consistent)
- yy-Task heterogeneity (hi-high and lo-low)
- zz-Processor heterogeneity (hi-high and lo-low)

Parameter description: All the heuristic methods are depending on the parameter. Extensive simulations are used to find the values of parameters. In this study, Table 2 represents the parameter used which are found by preliminary simulation.

Test results: The algorithm, global and local parallelism WSGA and NSGA-II apply to all 12 problem instances and

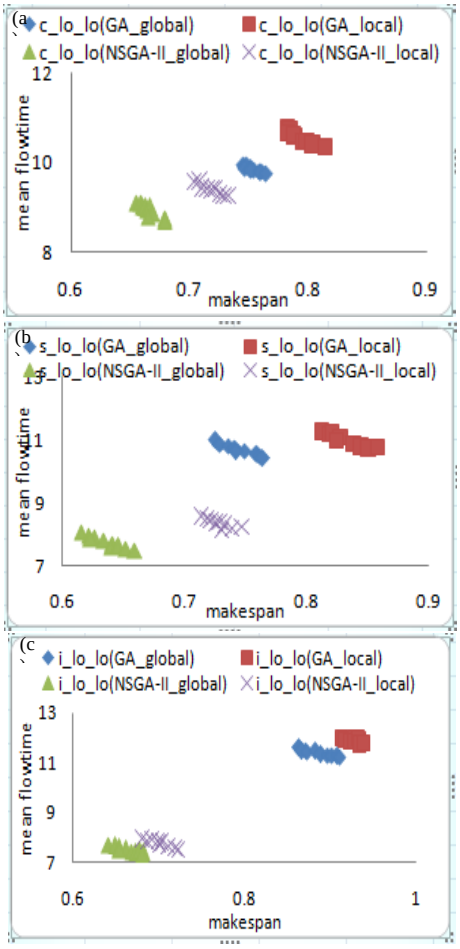

Fig. 4: Local and global parallelism of WSGA and NSGA-II comparison of low task, low processor heterogeneity; a) consistent; b) semi consistent; c) inconsistent

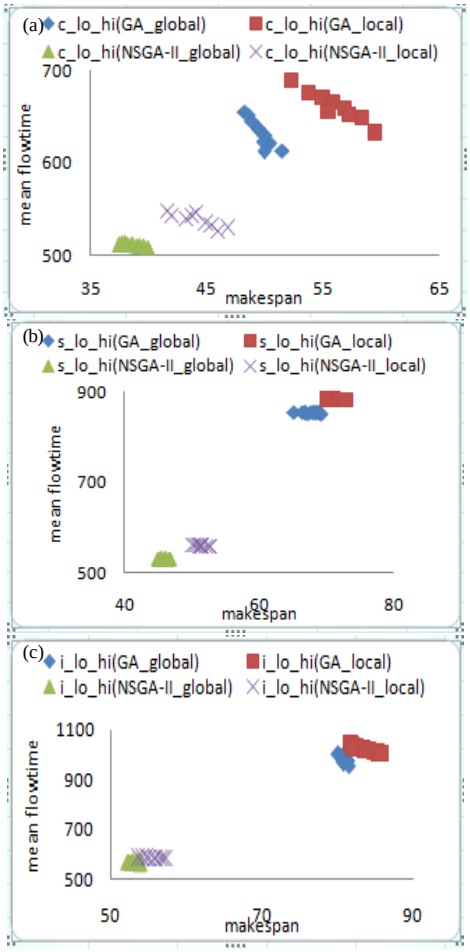

Fig. 5: Local and global parallelism of WSGA and NSGA-II comparison of low task, high processor heterogeneity; a) consistent; b) semi consistent; c) inconsistent

simulated using the above parameter settings specified in Table 2. To compare the performance of multi-objective scheduling algorithm, the Pareto optimal solutions produced by each algorithm are plotted in Fig. 4-7 for all the instances.

The values of makespan and mean flowtime are measured in same time units and obtained Pareto optimal solutions are plotted on a scale of ten thousands of time unit. The plotted graphs indicate that global parallelism of NSGA-II with MapReduce model produces best schedule in terms of the minimization of objectives for all cases compared to other methods. The algorithms are run for 1000 iterations and 200 initial populations were taken. It is also noted that the number of solutions obtained in NSGA-II increases by increasing number of population and number of iterations.

Performance comparison of WSGA and NSGA-II: A fuzzy based approach is used to choose best compromise solution for the obtained non-dominated set of solutions. The fuzzy sets are defined using a triangular membership function. Consider fmax and fmin are maximum and minimum values of each objective function and kth objective function of a solution in a Pareto set fk is represented by a membership function μ_k defined as:

$$\mu_k = \begin{cases} 1, & f_k \le f_{min} \\ \dfrac{f_k^{max} - f_k}{f_k^{max} - f_k^{min}}, & f_k^{min} < f_k < f_k^{max} \\ 0, & f_k \ge f_{max} \end{cases} \quad (5)$$

The value of membership function indicates how far a non-dominated solution has satisfied the objective. In

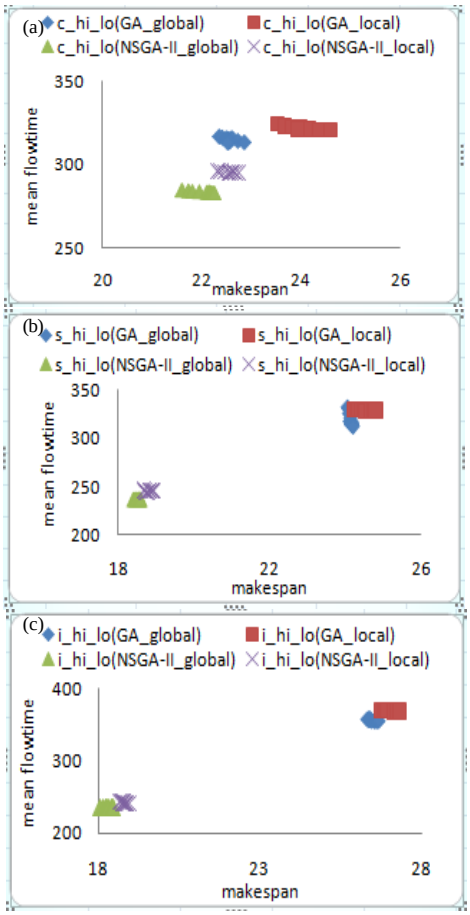

Fig. 6: Local and global parallelism of WSGA and NSGA-II comparison of high task, low processor heterogeneity; a) consistent; b) semi consistent; c) inconsistent

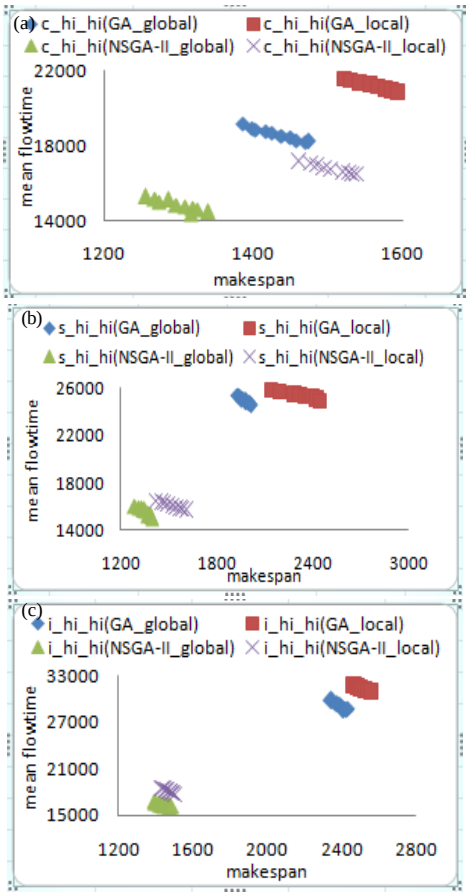

Fig. 7: Local and global parallelism of WSGA and NSGA-II comparison of high task, high processor heterogeneity; a) consistent; b) semi consistent; c) inconsistent

order to measure the performance of each solution to satisfy the objective, the sum of membership function values μ_k is computed where k = 1, 2,..., m objectives. The performance of each non-dominated solution can be rated with respect to the entire N non-dominated solutions by normalizing its performance over the sum of the ability of N non-dominated solutions as follows:

$$\mu^i = \frac{\sum_{k=1}^{m} \mu_k^i}{\sum_{i=1}^{n}\sum_{k=1}^{m} \mu_k^i} \quad (6)$$

Where:
n = No. of solution
m = No. of objective functions

The solution has the maximum value of μ^i is the best solution. The makespan and mean flowtime value for the best compromise solution obtained for global parallelism is listed in Table 3 and local parallelism is listed in Table 4. The percentage of reduction in makespan and mean flowtime NSGA-II over WSGA is calculated as:

$$\text{Makespan reduction} = 1 - \frac{MSP_{NSGA-II}}{MSP_{WSGA}} \times 100 \quad (7)$$

$$\text{Mean flowtime reduction} = 1 - \frac{MFT_{NSGA-II}}{MFT_{WSGA}} \times 100 \quad (8)$$

Where:
MSP = Makespan
MFT = Mean flowtime

The makespan and mean flowtime reduction percentage for global parallelism is presented in Table 3 and local parallelism is presented in Table 4. NSGA-II

Table 3: Comparison of global parallelism of WSGA and NSGA-II

	WSGA (global parallelism)		NSGA-II (global parallelism)		Percentage of reduction in	Percentage reduction in mean flowtime
Instances	Makespan	Mean flowtime	Makespan	Mean flowtime	makespan by NSGA-II	by NSGA-II
c_lo_lo	7453.29	99471.16	6546.28	90838.43	12.17	8.68
c_lo_hi	482742.86	6554734.22	376851.95	5133760.26	21.94	21.68
c_hi_lo	223592.4	3167581.74	216064.67	2846845.19	3.37	10.13
c_hi_hi	13856432.42	191843113.60	12548739.42	152960763.32	9.44	20.27
s_lo_lo	7230.64	109855.63	6145.27	80376.11	15.01	26.83
s_lo_hi	652923.17	8547249.07	449285.18	5304352.69	31.19	37.94
s_hi_lo	240591.15	3327845.62	184273.52	2368543.50	23.41	28.83
s_hi_hi	19264052.79	254051003.09	12864789.63	160436981.84	33.22	36.85
i_lo_lo	8623.04	115783.24	6389.37	76943.33	25.90	33.55
i_lo_hi	801363.16	9987848.35	522843.21	5679384.21	34.76	43.14
i_hi_lo	263219.12	3569280.18	180437.35	2344622.15	31.45	34.31
i_hi_hi	23395748.84	298917438.50	13901746.88	167435439.50	40.58	43.99

Table 4: Comparison of local parallelism of WSGA and NSGA-II

	WSGA (global parallelism)		NSGA-II (global parallelism)		Percentage of reduction in	Percentage reduction in mean flowtime
Instances	Makespan	Mean flowtime	Makespan	Mean flowtime	makespan by NSGA-II	by NSGA-II
c_lo_lo	7835.70	107538.39	7046.58	95616.65	10.07	11.09
c_lo_hi	522865.28	6886582.03	416359.15	5477872.14	20.37	20.46
c_hi_lo	235263.09	3247896.12	223180.67	2958973.26	5.14	8.90
c_hi_hi	15219391.38	215490836.86	14612699.98	171819265.18	3.99	20.27
s_lo_lo	8103.61	112471.83	7087.60	84968.77	12.54	24.45
s_lo_hi	702865.79	8825134.91	502881.89	5591758.23	28.45	36.64
s_hi_lo	242531.97	3297138.69	186832.54	2457632.42	22.97	25.46
s_hi_hi	21425760.16	258606633.30	14249348.26	164273685.36	33.49	36.48
i_lo_lo	9135.21	119563.56	6781.38	78953.64	25.77	33.97
i_lo_hi	817628.14	10455602.47	537054.62	5853643.89	34.32	44.01
i_hi_lo	267865.12	3680847.23	186359.13	2417565.89	30.43	34.32
i_hi_hi	24653628.75	316794482.50	14367126.70	184449678.10	41.72	41.78

Table 5: Comparison of execution time

Methods	WSGA (sec)	NSGA-II (sec)	Percentage of reduction
Sequential	37.923	34.680	8.55
Global parallelism with MapReduce	32.256	31.714	4.64
Local parallelism with MapReduce	30.337	29.098	6.92

achieves a reduction for global parallelism in makespan and flowtime by 24 and 29% and for local parallelism in makespan and flowtime by 22 and 28%, over the values of WSGA. NSGA-II out performs for global and local parallelism over WSGA.

Execution time comparison of WSGA and NAGA-II: Sequential WSGA and NSGA-II are executed in a single machine. Global and local parallelism with the MapReduce programming model for WSGA and NSGA-II is executed in 4 node Hadoop cluster.

The time taken by all the algorithms to find the optimal schedule is listed in Table 5. It is noted that local parallelism with MapReduce of NSGA-II has less execution time than other methods. As the number of nodes in a Hadoop cluster is increased, the execution time of these algorithms will be reduced.

CONCLUSION

In distributed and parallel computing systems, efficient allocation of tasks to the machines with minimal amount of time is a key step for better utilization of resources and task execution. In this study, global and local parallelism with the MapReduce programming model for WSGA and NSGA-II is implemented and their execution times are compared. The aim of these algorithms schedules independent tasks in a heterogeneous computing environment by minimizing makespan and flowtime in less execution time. From the obtained results, it is noted that global parallelism with MapReduce of NSGA-II produces an efficient schedule and local parallelism with MapReduce of NSGA-II has minimal execution time. The experimental results also imply that the number of nodes in a Hadoop cluster is in direct proportional to execution time of the algorithm.

Future work could be extended by implementing all the evolutionary kind of algorithms with the MapReduce programming model to execute its parallel without coordination issue. Also, the master of the MapReduce programming model may replace the coordinator.

REFERENCES

Abraham, A., H. Liu, C. Grosan and F. Xhafa, 2008. Nature Inspired Meta-Heuristics for Grid Scheduling: Single and Multi-Objective Optimization Approaches. In: Metaheuristics for Scheduling in Distributed Computing Environments. Xhafa F. and A. Abraham (Eds.). Springer Berlin Heidelberg, Heidelberg, Germany, ISBN: 978-3-540-69260-7, pp: 247-272.

Braun, T.D., H.J. Siegel, N. Beck, L.L. Boloni and M. Maheswaran et al., 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput., 61: 810-837.

Carretero, J., F. Xhafa and A. Abraham, 2007. Genetic algorithm based schedulers for grid computing systems. Int. J. Innovative Comput. Inform. Control, 3: 1053-1071.

Chankong, V. and Y.Y. Haims, 1983. Multiobjective Decision Making: Theory and Methodology. Elsevier, North-Holland, Amsterdam.

Deb, K., A. Pratap, S. Agarwal and T. Meyarivan, 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput., 6: 182-197.

Doulabi, S.H.H., M. Avazbeigi, S. Arab and H. Davoudpour, 2012. An effective hybrid simulated annealing and two mixed integer linear formulations for just-in-time open shop scheduling problem. Intl. J. Adv. Manuf. Technol., 59: 1143-1155.

Foster, I. and C. Kesselman, 2003. The Grid 2: Blueprint for a New Computing Infrastructure. 2nd Edn., Morgan Kaufmann, UK., ISBN-13: 9780080521534, Pages: 748.

Freund, R.F., M. Gherrity, S. Ambrosius, M. Campbell and M. Halderman et al., 1998.. Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. Proceedings of the 7th Heterogeneous Computing Workshop, March 30, 1998, Orlando, FL., USA., pp: 184-199.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. 1st Edn., Addison-Wesley Professional, Boston, MA., USA., ISBN-13: 9780201157673, Pages: 412.

Izakian, H., A. Abraham and V. Snasel, 2009. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. Proceedings of the International Joint Conference on Computational Sciences and Optimization, Volume 1, April 24-26, 2009, Sanya, Hainan, pp: 8-12.

Lim, D., Y.S. Ong, Y. Jin, B. Sendhoff and B.S. Lee, 2007. Efficient hierarchical parallel genetic algorithms using grid computing. Future Gener. Comput. Syst., 23: 658-670.

Liu, H., A. Abraham and A.E. Hassanien, 2010. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Gener. Comput. Syst., 26: 1336-1343.

Maheswaran, M., S. Ali, H.J. Siegel, D. Hensgen and R.F. Freund, 1999. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput., 59: 107-131.

Munir, E.U., L. Jian-Zhong, S. Sheng-Fei and Q. Rasool, 2007. Performance analysis of task scheduling heuristics in grid. Proceedings of the 6th International Conference on Machine Learning and Cybernetics, August 19-22, 2007, Hong-Kong, pp: 3093-3098.

Subashini, G. and M.C. Bhuvaneswari, 2010. A fast and elitist bi-objective evolutionary algorithm for scheduling independent tasks on heterogeneous systems. ICTACT, J. Soft Comput., 1: 9-17.

Verma, A., Llora, X., D.E. Goldberg and R.H. Campbell, 2009. Scaling genetic algorithms using mapreduce. Proceedings of the 9th International Conference on Intelligent Systems Design and Applications, November 30-December 2, 2009, IEEE, Pisa, Italy, ISBN: 978-1-4244-4735-0, pp: 13-18.