2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems

Multi-objective Evolution based Dynamic Job Scheduler in Grid

Debjyoti Paul
Dept. of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, India
debpaul@cse.iitk.ac.in

Sanjeev K Aggarwal
Dept. of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, India
ska@cse.iitk.ac.in

***Abstract*—Grid computing is a high performance computing environment to fulfill large-scale computational demands. It can integrate computational as well as storage resources from different networks and geographically dispersed organizations into a high performance computational & storage platform. It is used to solve complex computational-intensive problems, and also provide solution to storage-intensive applications with connected storage resources. Scheduling of user jobs properly on the heterogeneous resources is an important task in a grid computing environment. The main goal of scheduling is to maximize resource utilization, minimize waiting time of jobs, reduce energy consumption, minimize cost to the user after satisfying constraints of jobs and resources. We can trade off between the required level of quality of service, the deadline and the budget of user. In this paper, we propose a Multi-objective Evolution-based Dynamic Scheduler in Grid. Our scheduler have used Multi-objective optimization technique using Genetic algorithm with pareto front approach to find efficient schedules. It explores the search space vividly to avoid stagnation and generate near optimal solution. We propose that our scheduler provides a better grip on most features of grid from perspective of grid owner as well as user. Dynamic grid environment has forced us to make it a real time dynamic scheduler. A job grouping technique is proposed for grouping fine-grained jobs and for ease of computation. Experimentation on different data sets and on various parameters revealed effectiveness of multi-objective scheduling criteria and extraction of performance from grid resource.**

***Keywords*-Multi-objective, Job scheduling, GA, Grid computing, Pareto, Job grouping**

I. Introduction

Grid is a parallel and distributed processing architecture or system that aggregates geographically dispersed resources to act as a high performance computing environment for computing extensive jobs. It enables sharing and selection of resources dynamically depending on their performance, capability, availability, user's quality of service requirement and cost [8]. Grid is heterogeneous in nature where resources are connected through internet or private networks and their computational capabilities can vary a lot. Grid features include high performance, high throughput and scalability; facilitating inexpensive access to wide range of high performance resources. Grid also has a feature of choosing a resource in some specific manner while submitting jobs on it [10].

Grid performance can be improved in terms of job processing time by confirming that all the resources are utilized efficiently and optimally using a good job scheduling algorithm. Job scheduler exists in many conventional distributed environment systems. However dynamic nature, high heterogeneity of resources, high variance in jobs granularity, interconnection networks, existence of local policies on resources of grid makes grid job scheduling different from conventional approach and more challenging [27].

Grid scheduler follows a series of following steps [20] : (1) Collecting information of jobs submitted, (2) Collecting available resource information, (3) Find scheduling strategy i.e. mapping of jobs to feasible resources, (4) Job allocation according to the mapping, and (5) Monitoring jobs running status and completion.

Maximum utilization of grid resources is the most cogitated

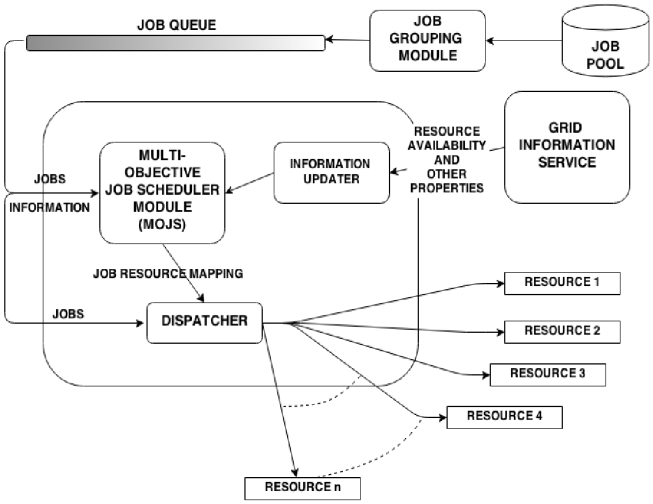

Figure 1: Simplified Scheduling Model

objective in scheduling literatures. However other factors like maintaining QoS constraints, cost effectiveness, energy efficient scheduling were either discussed separately or not acknowledged. Fair amount of importance should be given to user satisfaction, time and cost deadline of jobs. In 2007, Gartner estimated that the Information and Communication Technology industry is liable for 2% of the global CO_2 emission annually, which is equal to that from the aviation industry [21]. Above mentioned facets clearly show an ur-

978-1-4799-4325-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CISIS.2014.50

gency for a multi-objective grid scheduler, dealing them on their gravity of importance.

The Job scheduling in Grid is correlated with a classical problem, Flexible Job-shop Scheduling problem(FJSP) [7] with dynamic changes of resources and their availability. Besides these grid jobs need to be scheduled as soon as possible after they are enqueued in the job queue, granting the scheduler only a few minutes of time to find the scheduling strategy. FJSP consists of routing subproblem and the scheduling subproblem [26]. Routing problem is to assign each job with a resource among a set of resources and scheduling problem is to obtain a feasible and satisfactory sequence of jobs within the resources.

Computationally, FJSP is as hard as JSP which is an NP Hard problem [11]. So finding near optimal solution in polynomial time is our aim. The problem becomes even more interesting when multiple objectives are there to be taken care of.

Finding near optimal solution for FJSP problem with more than one objective in a time efficient way is a difficult task. Grid environment being dynamic in nature, reallocation of jobs is quite evident in it.

We present a multi-objective Job scheduler based on an evolutionary algorithm. The aim of this work is to give grid administrators a better scheduler, which will give better grip on the trade off among cost, utilization, energy efficiency and QoS. The scheduler can cope up with the dynamic behavior of resources, resource constraints and predecessor job constraints. A job grouping mechanism is proffered for fine grained jobs.

Our approach of solving the above problem using Non-dominated sorting evolutionary algorithm for minimization of multiple objectives, is well enough to find near optimal scheduling strategy in time. Non-dominated Sorting Genetic Algorithm II [9] is used as a basic framework for our Job scheduler module. Non-dominating sorting mechanism with our avant-garde crossover and mutation operator enables the scheduler to explore the search space minutely. The running time complexity of algorithm is $O(GMN^2)$ where G is the number of generations or iterations, M is the number of objectives and N is the population size of the chromosomes or scheduling strategies to run the algorithm.

The organization of the rest of the paper and a brief outline of the sections is as follows. In section II, some related works on job scheduling in grids and their merits and demerits have been discussed. In section III, Multi-objective Evolution based Dynamic Job Scheduler in Grid has been presented. Here problem definition, job-grouping strategy, problem formulation, MOJS module and algorithms are described. In section IV, implementation details and experimental results are given. Section V sums up the work with conclusion and future work.

II. Related Work

Job shop scheduling problem has been proven to be an NP-complete problem in 1979 [16]. Many researcher have used heuristic based solving approach to address the problem. Local Search [23], Tabu search [1], simulated Annealing [28] [1] are single heuristic based approach.

In Tabu search, one solution s moves to another solution s' located in the neighborhood with a slight modification possible from s. Its performance largely depends on the parameters and heuristic used in formulating the problem.

In simulated annealing technique, each solution is mutated and if the mutant spawned exceeds threshold it is rejected, and if less than or equal to the energy of the parent, the difference of threshold and energy of mutant is added to Energy Bank(EB). The threshold is changed when EB reaches a certain value and population moved to new generation. Simulated annealing in Multiobjective domain e.g. AMOSA [4] requires many parameters and domination factor to find near optimal solutions, which are hard to established in dynamic environment of grid scheduling.

There are also some hybrid approaches like Tabu search with Ant colony Optimization [22] [24], GA's with Simulated annealing [30]. Other predictive model approaches for the problem are Particle Swarm optimization [18] [2], Fuzzy based scheduling [17]. AI based scheduling algorithms like Max-min (Task with more computation time has higher priority), Suffrage (Task with higher sufferage value is given higher priority, its value is determined as the difference of computational time between best and second best resources on which job can be allocated) [14]. All the above work have focused on single objective i.e. minimizing the makespan, which in turn maximizes the utilization of resources and resource constraint was also not taken into consideration.

Job grouping based scheduling algorithm is used for fine-grained jobs & light-weight jobs which increase the resource utilization [19] [3]. However they have not taken care of predecessor job completion constraint and dynamic behavior of resources in grid.

GA based scheduler can act as a real time scheduler due to increase in computational capability of processors in last five years. EDSA is a GAs searching technique in which the crossover and mutation rates are changed dynamically depending on the variances of the fitness values in each generation [29] . The scheduling consider minimization of makespan. In our work based on multi-objective evolutionary algorithm we have converted resource scheduling problem in grid into *resource-constrained project scheduling problem*. We have incorporated dynamic scheduling mechanism, advanced crossover and mutation operator & minimizing five objectives with pareto front technique. The GA structure of Non-dominating Sorting Genetic Algorithm II proposed by K.Deb *et al.* have helped us in creating the MOJS module [9]. MOJS with graph based job grouping strategy for fine-grained jobs eases the process.

A comparable work with matching constraints could not be found in literature, only few publications deal with multi-objective scheduling [12] but their platform is different from ours. So in result section we experimented our scheduler and produced result on the performance based on various parameters.

III. The Multi-objective Job Scheduler module

A. Problem Definition

A dynamic scheduler for grid environment solving flexible job shop scheduling problem based on multi-objective optimization technique. The problem is defined in three sections:

- Multi-objective optimization based FJSP.
- Dynamic scheduler for frequent change in resource availability status.
- A job grouping technique for grouping fine grained jobs helping scheduler to yield scheduling strategy in time.

1) Formulation of problem: Here the problem is formulated with the notations described in scheduling literature [6], [5], [15]. Given are a set $M = \{M_1, M_2, M_3, ...M_m\}$ of resources, a set $J = \{J_1, J_2, J_3, ...J_j\}$ of application jobs, and a set O of grid jobs. The n Grid Jobs of application job J_i are denoted by $O_{i1}, ..., O_{in}$, a set $W = \{W_1, W_2, \ldots, W_m\}$ denotes normalized energy dissipation factor of resources. Table I gives a concise definition of the notations have been used. Optimization is done with start time of grid jobs

Table I: Notation Symbol and their definitions

Notation	Definition
M_i	Resource with ID i
J_i	Application job with ID i
O_{ij}	jth Grid Job or task of Application job J_i
W_i	Energy dissipation factor of Resource M_i, normalized with the max value from set W
$p(O_{ij}, O_{ik})$	p is precedence function, if O_{ij} precedes O_{ik} it is $TRUE$ else $FALSE$
μ_{ij}	set of all combinations of resources which can execute O_{ij}
R_{ij}	R_{ij} represent mapping of job O_{ij} on a machine in M, $R_{ij} \in \mu_{ij}$,
$t(O_{ij}, R_{ij})$	Processing time of O_{ij} mapped to resource R_{ij}
$c(O_{ij}, R_{ij})$	Cost of O_{ij} mapped to resource R_{ij}
$s(O_{ij})$	O_{ij} start time
$e(O_{ij})$	O_{ij} end time
d_{ij}	Time limit for completion of O_{ij}
c'_{ij}	Cost limit for O_{ij}
$l(M_i)$	Last grid job executed in M_i
$tsum(M_i)$	Running time or Uptime of M_i

$s(O_{ij}) \in \mathbb{R}$ and allocating them on resources $R_{ij} \in \mu_{ij}$. If the following restrictions are met a solution is said to be valid:

1) Each grid job is allocated in such a way that no conflict happens while demanding resource:
   - $\forall O_{ij} : \exists s(O_{ij}) \in \mathbb{R}, R_{ij} \in \mu_{ij} : \forall M_j \in R_{ij}$:
   - M_j is in $[s(O_{ij}); s(O_{ij}) + t(O_{ij}), R_{ij}]$ exclusively allocated by O_{ij}
2) Each grid job can start only after its predecessor jobs finish execution:
   - $\forall i, j \neq k : p(O_{ij}, O_{ik}) \Rightarrow s(O_{ik}) \geq s(O_{ij}) + t(O_{ij}, R_{ij})$

Exceeding the time limit and budget cost will affect QoS of grid jobs. A penalty factor is imposed when jobs violates following constraints.

1) All grid jobs O_{ij} have time limit d_{ij} which must be adhered to:
   - $\forall O_{ij} : d_{ij} \geq s(O_{ij}) + t(O_{ij}, R_{ij})$:
2) Optimize allocation of grid jobs on resources such that cost limit c'_{ij} of grid job O_{ij} is more than cost it is incurring on current resource mapping:
   - $\forall O_{ij} : c'_{ij} \geq c(O_{ij}, R_{ij})$

This work focuses on achieving near-optimal scheduling strategy on following objective functions:
Minimize $y = f(x) = (f_1(x), f_2(x), f_3(x), f_4(x), f_5(x))$
where, $x \in \mathbb{V}, y \in \mathbb{R}^5$
i.e., x is decision vector in search space $\mathbb{V}$, y is objective vector with 5 objectives.

1) Minimizing makespan, $e(O_{ij})$ is the end time of grid job O_{ij}

$$\begin{aligned} f_1 &= makespan \\ &= max\{e(l(M_1)), e(l(M_2)), \ldots, e(l(M_m))\end{aligned}$$

2) Maximizing utilization of resources i.e. minimizing f_2

$$\begin{aligned} f_2 &= non - utilization \\ &= \frac{1}{m}\sum_{j=1}^{m}\{e(l(M_j)) - tsum(M_j)\}\end{aligned}$$

3) Minimizing time limit penalty (minimizing number of jobs completing after due date)

$$f_3 = \frac{1}{j * n}\sum_{\forall i,j}\varphi_1(e(O_{ij}) - d_{ij})$$

where $\varphi_1(x)$ is a non-negative continuous exponential non-decreasing function, if $x > 0$ else 0.

4) Minimizing cost penalty

$$f_4 = \frac{1}{j * n}\sum_{\forall i,j}\varphi_2\{c(O_{ij}, R_{ij}) - c'_{ij}\}$$

where $\varphi_2(x)$ is a non-negative continuous linear non-decreasing function, if $x > 0$ else 0.

5) Minimizing Overall Energy consumption

$$f_5 = \sum_{i=1}^{m} tsum(M_i) * W_i$$

2) Chromosome model: A scheduling strategy or mapping of jobs on resources satisfying the constraints is represented by chromosome. A chromosome stores parameters as follows
(i) Resource id corresponding to each job
(ii) Job start time
(iii) Job end time
(iv) Predecessor job ID of each job
(v) Five objective function values
(vi) Rank of chromosome, see section III-A3
(vii) Crowding distance, see section III-A4
Start time for execution of jobs is calculated according to heuristic rules

(i) Schedule grid job as early as its precedent job is completed.
(ii) Schedule grid jobs according to shortest due date.

Algorithm 1 Multi-objective Job Scheduler

Input: Jobs[NUM_JOBS],
Resource[$NUM\_RESOURCES$],
n,$num\_iteration$
Initialization: Generate initial population P_0 of n chromosomes
Fitness Calculation:
for $i = 1 \rightarrow n$ **do**
Evaluate(chromosome[i] from P_i)
end for
for $i = 1 \rightarrow num\_iteration$ **do**
Selection: Select a subset of even number of chromosomes from P_i
$P_{i_1} = Select(P_i)$
Crossover: With probability P_c crossover every two chromosome from P_{i_1}
$P_{i_2} = crossover(P_{i_1})$
Mutation: With probability P_m mutate chromosome from P_{i_2}
$P_{i_3} = mutate(P_{i_2})$
Fitness Calculation:
for $i = 1 \rightarrow n$ **do**
Evaluate(chromosome[i] from P_{i_3})
end for
$P_{i_4} = P_i + P_{i_3}$
Assign non-domination rank to each chromosome, Non-dominating_Sort(P_{i_4})
Calculate_crowding_distance(P_{i_4})
Sort based on Crowding distance of each chromosome Crowding_distance_sorting(P_{i_4})
Replacement: Create population for new generation
Forward 1st n chromosomes from sorted set P_{i_4} to P_{i+1}
end for
return Chromosomes with non-domination rank 1 i.e. First pareto front

3) Non-Dominated Ranking (R_x): A chromosome a is said to be dominated by chromosome b iff $\forall i \in \{1, 2, \ldots, k\} : f_i(a) \leq f_i(b)$ and $\exists i \in \{1, 2, \ldots, k\} : f_i(a) < f_i(b)$. A chromosome a is said to be Non-dominated if there does not exist any chromosome $b \in \mathbb{V}$ search space that dominates a. A set of such non-dominated chromosome in objective space is called pareto optimal front. After removing the pareto optimal front, a second pareto optimal front can be obtained. We assign a rank to each of the chromosome according to their occurrence in the pareto front. Then the algorithm sorts the population according to their rank and crowding distance (discussed in section III-A4) for selecting population for next generation.

4) Crowding distance: Crowding distance ($dist_x$) of a particular chromosome x in population measures the density of chromosomes surrounding it [9].
After non-dominated sorting is completed, each chromosome's crowding distance is calculated as the sum of normalized distance between its adjacent neighbors corresponding to each objective. Crowding distance for first and last individual is infinite.
$dist_x = \sum_{j=1}^{k} \frac{f_j(x_{left}) - f_j(x_{right})}{f_j^{max} - f_j^{min}}$ where f_j is jth objective function, and number of objectives is k.

5) Partial order on chromosomes: A partial order $\prec$ between chromosomes are defined as:
$a \prec b$ if $R_a \prec R_b$
or $(R_a = R_b)$ and $(dist_a \succ dist_b)$
This means that a chromosome with lower rank is preferred. Again, a chromosome from less crowded region in search space is preferred when compared with chromosomes having same rank. It is desired that the evolutionary algorithm maintains a good spread of solutions in the population, so that sustainable diversity in the population remains and solutions are not restricted to local optimization.

6) Crossover: Crossover operator is applied on two chromosomes selected from mating pool; interchanging their genes (resource mapping) to obtain new individuals and satisfying the constraints. The aim is to obtain new individual/chromosome with better fitness function and that will help in exploring new regions in search space not explored yet. P_c is the probability with which crossover operator is applied.
k-point crossover: Two or more cutting points i.e. $k \geq 2$ are randomly chosen and segments are interchanged alternately generating two new descendants. If the value of k is very large it can explore the solution space thoroughly. But it will destroy the inherited nature from the parent and optimization may take longer time to converge than usual.
Fitness based Crossover: In this operator fitness or any other external function can be used. Our approach yield two descendants. The crossover is computed as follows. Here $c_{parent}[i]$ represents the ith gene of parent chromosome.

$$\forall i, c_{child_1}[i] = \begin{cases} c_{parent_1}[i] & \text{with probability } p = \frac{g_1[i]}{g_1[i]+g_2[i]} \\ c_{parent_2}[i] & \text{with probability } 1-p \end{cases}$$

$$\forall i, c_{child_2}[i] = \begin{cases} c_{parent_1}[i] & \text{with probability } p = \frac{h_1[i]}{h_1[i]+h_2[i]} \\ c_{parent_2}[i] & \text{with probability } 1-p \end{cases}$$

Each resource in grid has processing capability and energy efficiency parameter. They almost counteract each other and need a tradeoff between them to find optimal schedule.
Here $g_1[i], g_2[i]$ are energy efficiency parameter of $c_{parent_1}[i]$ and $c_{parent_2}[i]$ resources respectively. Similarly $h_1[i], h_2[i]$ are processing capability parameter of $c_{parent_1}[i]$ and $c_{parent_2}[i]$ resources respectively.

7) Mutation: P_m is the probability with which mutation operator is applied. Mutation operators used as follows.
Move: This operator randomly assigns a resource to the job. Care is taken that resource type is same i.e. resource belongs to same set.

Swap: This operator randomly chooses two jobs and swap their assigned resources if they belong to same set.
Rebalancing: This operator chooses most overloaded resource and randomly pick a job assigned to it. Then the job is moved to a resource which is less overloaded.
In the process of crossover and mutation it is possible that some good chromosomes might be lost. Elitism is a mechanism to preserve these chromosomes. A small percentage of the fittest population i.e. first pareto front in multi-objective search space is forwarded to be the part of new population for next iteration.

B. Dynamic scheduling

Application jobs queued in grid are fed to the MOJS module in batch. The output is the elitist pareto front of chromosomes comprises near optimal schedules. Soft constraints and weighted sum approach is applied on multiple objectives to finalize a chromosome as scheduling strategy. Some jobs are queued on their respective resource while others are again fed to MOJS module. The number of jobs queued depend on the fitness of the chromosome and time available for the scheduler to re-run and find a better schedule. Progress is ensured by setting a minimum number of jobs to be queued on a single run of module.
As this is a real time scheduling problem and resources are dynamic in nature it can participate and leave the system any time, it is of higher importance to make the scheduling dynamic in nature. By dynamic we mean re-allocation of already scheduled jobs which were not completed. So change in the resource pool can trigger running of scheduler which can either reschedule the jobs whose resources have left the grid or can request processing of new jobs on addition of one or more resources or both of them. Jobs whose predecessors is present in the set of rescheduled jobs are also rescheduled. There is very little scope for this paper to solve the issue where a job suffers starvation and penalty due to failure of resource. To handle this issue accounting of Mean Time to Failure (MTTF) with log mining can help.

C. Job Grouping for Fine-grained Jobs

Fine grained jobs are grouped to form a single job. Following are the constraints considered while job grouping.

- Jobs grouped as single job should be of same type i.e. either computational jobs or storage intensive jobs.
- Prevail same job precedence rule after job grouping.

Workflow and precedence of jobs is represented through Directed Acyclic Graph (DAG), where nodes are jobs and directed edge represents precedence.
A directed edge from a to b is drawn, when b awaits for the completion of job a and a is said to be the predecessor of b and b is the successor of a. Nodes having common edge are defined as adjacent nodes. We define a, b to have same job type if their resource type requirement is same.
Before we present our heuristic algorithm for job-grouping, we define few terms as follows:

1) Entry job: A job without any predecessor but has atleast one successor is called entry job. If there are multiple entry jobs for a DAG component then we add a zero size job/node and new directed edges are drawn from zero size job to entry jobs. Hence we have a single entry job denoted as j_{entry}.

2) Exit job: A job without any successor but has atleast one predecessor is called exit job. If there are multiple exit jobs for a DAG component then we add a zero size job/node and new directed edges are drawn from exit jobs to zero size job. Hence we have a single exit job denoted as j_{exit}.

3) Job size: Size of a computational job is measured in Million Instructions(MI) and storage jobs in Megabytes(MB). It is denoted as $job\_size(j)_{computational}$ or $job\_size(j)_{storage}$.

Algorithm 2 job grouping

Input: Job pool with DAG representation
  Compute $crit_{up}(j)_{<type>}$ for each job j according to the equation in section III-C4
  Compute $crit_{down}(j)_{<type>}$ for each job j according to the equation in section III-C4
  Compute $crit_{<type>}$ for each job j according to the equation in section III-C4
  while Job $a \in$ job pool exists, where a is unprocessed fine-grained job **do**
    $flag \leftarrow 0$
    while a is fine-grained job **and** $flag = 0$ **do**
      for each $b \in adjacent\_node(a)$ **and** same type i.e. computational or storage **do**
        Temporary merge adjacent node b and a to form t
        Calculate new $crit_{up}(t)_{<type>}$, $crit_{down}(t)_{<type>}$ and $crit(t)_{<type>}$
        if new $crit(t)_{<type>} \leq crit(j_{entry})_{<type>}$ **and** $crit(t)_{<type>}$ is minimum till now **then**
          $merge\_node \leftarrow b$
        end if
      end for
      if $merge\_node$ is found **then**
        Permanently merge $merge\_node$ with a to form a'
        Change parent and child relation accordingly
        if a' is not fine-grained job **then**
          $flag \leftarrow 1$
        end if
      else
        $flag \leftarrow 1$
      end if
    end while
  end while

4) Critical length : Critical length denoted as $crit(j)$ refers to the longest distance from j_{entry} to j_{exit} passing through the job j. There are two types of jobs viz., computational intensive and storage intensive jobs. Hence we

consider $crit(j)_{computational}$, $crit(j)_{storage}$ accordingly for calculation in Algorithm 2 depending on the job type.
The Upward Critical length of job j is the longest distance from j to the exit job j_{exit}. It is denoted as $crit_{up}(j)_{<type>}$ where $_{<type>}$ is computational and storage. Upward critical length is computed with the following equation starting from j_{exit} and moving upward towards j.

$$crit_{up}(j)_{<type>} = job\_size(j)_{<type>} + max_{j' \in succ(j)}(crit_{up}(j')_{<type>})$$

Similarly, the Downward Critical length of job j is the longest distance from the entry job j_{entry} to j. It is denoted as $crit_{down}(j)_{<type>}$ where $_{<type>}$ is computational and storage. Downward critical length is computed with the following equation starting from j_{entry} and moving downward towards j.

$$crit_{down}(j)_{<type>} = job\_size(j)_{<type>} + max_{j' \in pred(j)}(crit_{down}(j')_{<type>})$$

$$crit(j)_{<type>} = crit_{up}(j)_{<type>} + crit_{down}(j)_{<type>}$$

IV. Experiment and results

Standard grid workload from Grid Workload Archive [13] have been used in this experiments. 60% of grid jobs have precedence dependencies. Gridload SHARCNET and DAS-2 are the two traces that have been used for experiment. Traces shows that execution time of jobs varies widely making scheduler task difficult. A sample of job characteristic is given in Table II.
The *resource manager* simulates the dynamic behaviour of resources in grid. We have assumed 10% of the resources will show anomalous behaviour; i.e. 10% of resources can leave the grid without notifying the grid. The *dispatcher* send jobs to corresponding resources and collect the results after completion.
Our scheduler outputs a set of near optimal non-dominating scheduling strategies based on five objectives. For our result section equal weighted sum on each objective approach is used to choose a schedule strategy from set of near optimal strategies. Regarding the processing time, the scheduler can produce scheduling strategy for 200 jobs in 21 seconds with 400 chromosomes and 200 iterations.
In any stochastic algorithm randomize function plays an important role. A very fast random number generator Mersenne Twister of period $2^{19937} - 1$ is used in different parts of the code and has a better equidistibution property. It generates integer in the range 0 to $2^{32} - 1$ and real number range $[0, 1)$ with a precision of 2^{32} [25].

Table II: User job queue

JOB_ID	*JOB_SIZE* in MI/MB	*TIME_LIMIT* in seconds	*JOB_COST* in $	*PRED_ID*	*JOB_TYPE*
⋮	⋮	⋮	⋮	⋮	⋮
42	24,000 MI	63.0	5.41	29	computational
43	130,000 MI	107.0	4.86	-1	computational
44	10,500 MI	106.0	5.99	24	computational
47	530.0 MB	150.0	5.04	29	storage
48	240,200 MI	133.0	5.40	31	computational
⋮	⋮	⋮	⋮	⋮	⋮

A. Scalability

This section tests the scalable property of the scheduler with increase in jobs and resources. Independent jobs are considered for nullifying precedence constraint effects on scheduling during execution.
Result given in Table III shows that irrespective of the large variation of granularity in grid jobs in average 99%+ utilization performance has been achieved. The scheduler scalability has been tested with 10000, 15000, 20000, 25000 independent jobs; and on 10, 15, 20, 25 resources. This shows that scheduler can process large amount of gridlets and resources without compensating on the Makespan and utilization of resources.

Table III: Scalabiltity with jobs and resources

#JOBS	#RESOURCES	Average Utilization	Lowest Utilization	Standard deviation
10000	10	99.61	99.03	0.33
15000	10	99.94	99.47	0.16
20000	10	99.94	99.46	0.17
25000	10	99.49	98.76	0.47
10000	20	99.29	97.55	0.83
15000	20	99.92	99.33	0.18
20000	20	99.47	98.91	0.26
25000	20	99.65	99.04	0.26
10000	25	99.08	94.71	1.26
15000	25	99.69	95.99	0.86
20000	25	99.59	97.93	0.49
25000	25	99.43	98.39	0.44

B. Tradeoff energy, performance and pricing

This section shows the effect of introduction of energy and performance parameters on the scheduler. Resource configuration of the machines used are given in Table IV. Result in Table IV with 10000 jobs explains the scheduling behavior w.r.t performance and energy consumption. Table IV shows that very less jobs are scheduled on Pentium 4 which is poor in performance and energy dissipation among other resources. Now comparing resources Core 2 X6800 with Core 2 QX6700, they have almost same MIPS specification but Core 2 X6800 consumes less power. The scheduler have allocated more jobs in Core 2 X6800 which is reasonable. Same logic can be applied for resources Core i7 920 and Core i7 3960X. They have same energy dissipation factor but Core i7 3960X performance is better. As a consequence scheduler have scheduled more jobs on Core i7 3960X. Best resource of the lot is Core i7-2600. The scheduler have uniformly distributed jobs among Core 2 X6800, Core i7 3960X and

Table VI: Resource utilization after appending pricing model (DAS-2)

ID	Resource	Pricing model ($/unit time)	Makespan %	Utilization %
1	Pentium 4	0.05	99.20	91.35
2	Pentium 4	0.05	97.34	90.71
3	Pentium 4	0.05	97.12	90.95
4	Intel Core i7 920 (Quad core)	0.1	99.28	89.93
5	Intel Core i7 920 (Quad core)	0.1	99.31	92.29
6	Intel Core 2 Extreme X6800	0.09	98.74	89.29
7	Intel Core 2 Extreme X6800	0.09	98.59	90.45
8	Intel Core i7 Extreme Edition	0.15	98.65	94.55
9	Intel Core i7 Extreme Edition	0.15	97.94	91.82
10	Intel Core 2 Extreme QX6700	0.165	100.00	94.28
11	Intel Core 2 Extreme QX6700	0.165	99.90	94.16
12	Intel Core 2 Extreme QX6700	0.165	98.71	94.30
13	Intel Core 2 Extreme QX6700	0.165	98.08	97.95
14	Core i7-2600	0.18	99.63	98.28
15	Core i7-2600	0.18	99.59	98.33

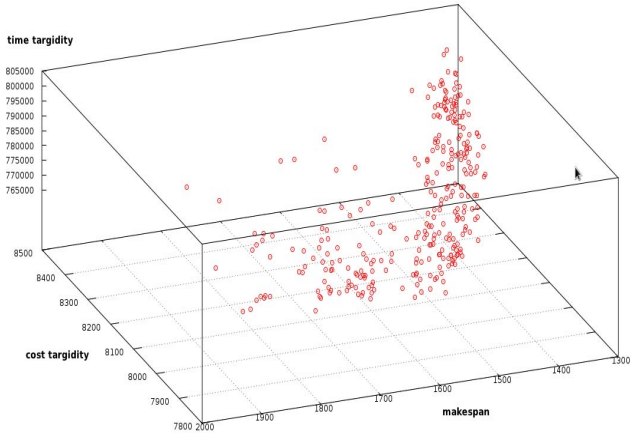

Figure 2: Pareto front with makespan, time targidity and energy efficiency as objectives

Core i7-2600 to have a minimum makespan. Jobs whose deadline are far is judicious enough to be scheduled on low end resources.

Since workload SHARCNET has large variation in the granularity of jobs the result gives worst case analysis in their utilization.

Now result in Table V reflects the change in scheduling

Table IV: Resource configuration for experiment

ID	Machine	Frequency GHz	Energy Watts	Performance MIPS/core	Utilization or Uptime %
1,2	Pentium 4 Extreme	3.2	92.1	9,726	33.5, 29.2
3,4	Intel Core 2 X6800	2.93	75	13,539	100.0, 99.1
5,6	Intel Core 2 QX6700	2.66	95	12,290	44.6, 47.3
7,8	Intel Core i7 920	2.667	130	20,575	50.8, 50.8
9,10	Intel Core i7 3960X	3.3	130	29,621	99.8, 98.8
11,12	Core i7-2600	3.4	95	32,075	99.8, 99.9

behavior after appending the pricing model. The result shows that low end resources have been utilized well enough. Jobs having low job cost or high time limit can afford to run on these cheap resources whereas high priority jobs demanding high performance run on costly resources. Trade off among performance, time of execution and cost have allowed jobs to be scheduled on various resources uniformly. In table V & VI it is observed that resources have adhered to the makespan and all have 98%+ execution time. Utilization percentage shows actual uptime or running time of resources. This reveals that all resources have been utilized properly. Even resources like Pentium 4 have 95% utilization on average. The utilization of resource with workload DAS-

Table V: Resource utilization after appending pricing model (SHARCNET)

ID	Resource	Pricing model ($/unit time)	Makespan %	Utilization %
1	Pentium 4	0.05	98.01	98.04
2	Pentium 4	0.05	98.95	92.90
3	Pentium 4	0.05	98.95	96.07
4	Intel Core i7 920 (Quad core)	0.10	97.99	93.57
5	Intel Core i7 920 (Quad core)	0.10	98.00	97.52
6	Intel Core 2 Extreme X6800	0.09	98.01	97.39
7	Intel Core 2 Extreme X6800	0.09	98.92	96.83
8	Intel Core i7 Extreme Edition	0.15	98.94	93.91
9	Intel Core i7 Extreme Edition	0.15	98.92	96.85
10	Intel Core 2 Extreme QX6700	0.165	98.92	97.56
11	Intel Core 2 Extreme QX6700	0.165	98.89	98.52
12	Intel Core 2 Extreme QX6700	0.165	98.92	96.43
13	Intel Core 2 Extreme QX6700	0.165	98.92	98.11
14	Core i7-2600	0.18	100.00	97.28
15	Core i7-2600	0.18	98.92	99.16

2 is less when compared with workload SHARCNET. Jobs in DAS-2 is more fine grained in nature and to maintain predecessor relation among them is has to sacrifice a little on utilization. Figure 2 shows how our scheduler gives a better grip to the administrator to trade off between user objectives and grid administrator objectives. Each point on the space represents a scheduling strategy. In a 3D co-ordinate system we represents 3 objectives which are needed to be minimized namely (i) makespan (ii) energy efficiency parameter, (iii) time targidity. Any point in the first pareto front can be chosen for scheduling strategy. This gives grid administrator wide range of choices and cope up with dynamic behaviour of grid.

V. Conclusion

The main motive of this work is to provide a flexible scheduler keeping multiple objectives into consideration. The scheduler module yields best scheduling strategies on various parameters in a pareto front. This is upto grid administrator and dynamic grid environment to choose a scheduling strategy of its choice. For experimentation purpose we have put equal weights on each objective for choosing best scheduling strategy.

Our results clearly shows that our scheduler produces optimized schedule on multi-objective optimization environment. The scheduler is scalable with resources and can process an infinite queue of jobs. The scheduler responded well with the change of constraints and behaviour of grid and job model. All resources have adhered to the makespan, and utilization rate is also high inspite of precedence constraint. It is difficult to display minimization of cost and time targidity parameter in graph or table. However a live demo of search space or or pareto graph with schedules/chromosomes on successive iteration of genetic algorithm can verify its authenticity.

REFERENCES

[1] A. Abraham, R. Buyya, and B. Nath. Natures heuristics for scheduling jobs on computational grids. In *The 8th IEEE international conference on advanced computing and communications (ADCOM 2000)*, pages 45–52, 2000.

[2] A. Abraham, H. Liu, W. Zhang, and T.-G. Chang. Scheduling jobs on computational grids using fuzzy particle swarm algorithm. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 500–507. Springer, 2006.

[3] T. Ang and W. Ng. A bandwidth-aware job scheduling based scheduling on grid computing. *Asian Network for Scientific Information*, 8(3):372–277, 2009.

[4] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb. A simulated annealing-based multiobjective optimization algorithm: Amosa. *Evolutionary Computation, IEEE Transactions on*, 12(3):269–283, 2008.

[5] K. Brucker. *Book: Complex Scheduling*. Springer, 2006.

[6] P. Brucker. *Book: Scheduling algorithms*. Springer, 2004.

[7] P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990.

[8] R. Buyya and S. Venugopal. Article: A gentle introduction to grid computing and technologies. *Journal: Database*, 2:R3, 2005.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions*, 6(2):182–197, 2002.

[10] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[11] M. R. Garey and D. S. Johnson. *Book: Computers and intractability*, volume 174. freeman New York, 1979.

[12] C. Grosan, A. Abraham, and B. Helvik. Multiobjective evolutionary algorithms for scheduling jobs on computational grids. In *International Conference on Applied Computing*, pages 459–463, 2007.

[13] GWA. The grid workloads archive, 2013. Online; accessed 10-April-2013, gwa[dot]ewi[dot]tudelft[dot]nl/pmwiki/.

[14] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[15] W. Jakob, A. Quinte, K.-U. Stucky, and W. Süß. Fast multi-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm. In *Parallel Problem Solving from Nature–PPSN X*, pages 1031–1040. Springer, 2008.

[16] M. T. C. S. JIS. Computers and intractability a guide to the theory of np-completeness. 1979.

[17] K. P. Kumar, A. Agarwal, and R. Krishnan. Fuzzy based resource management framework for high throughput computing. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 555–562. IEEE, 2004.

[18] H. Liu, A. Abraham, and A. E. Hassanien. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Generation Computer Systems*, 26(8):1336–1343, 2010.

[19] N. Muthuvelu, J. Liu, N. L. Soe, V. Srikumar, A. Sulistio, and R. Buyya. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In *Conferences in Research and Practice in Information Technology Series*, volume 108, pages 41–48, 2005.

[20] J. Nabrzyski and J. Schopf, Jennifer M. *Book: Grid resource management: state of the art and future trends*, volume 64. Springer, 2004.

[21] C. Pettey. Gartner estimates ict industry accounts for 2 percent of global co2 emissions, 2007.

[22] G. Ritchie. Static multi-processor scheduling with ant colony optimisation & local search. *Master of Science thesis, University of Edinburgh*, pages 1–101, 2003.

[23] G. Ritchie and J. Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. 2003.

[24] G. Ritchie and J. Levine. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. 2004.

[25] M. Saito. An application of finite field: Design and implementation of 128-bit instruction-based fast pseudorandom number generator. *Master's thesis, Hiroshima University*, 2007.

[26] L. Wang, G. Zhou, Y. Xu, and M. Liu. An enhanced pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, pages 1–13, 2012.

[27] F. Xhafa and A. Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[28] A. YarKhan and J. J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment. In *Grid ComputingGRID 2002*, pages 232–242. Springer, 2002.

[29] K.-M. Yu and C.-K. Chen. An evolution-based dynamic scheduling algorithm in grid computing environment. In *Intelligent Systems Design and Applications, 2008. ISDA'08. Eighth International Conference on*, volume 1, pages 450–455. IEEE, 2008.

[30] S. Zheng, W. Shu, and L. Gao. Task scheduling using parallel genetic simulated annealing algorithm. In *Service Operations and Logistics, and Informatics, 2006. SOLI'06. IEEE International Conference on*, pages 46–50. IEEE, 2006.