Nature's Heuristics for Scheduling Jobs on Computational Grids

Ajith Abraham, Rajkumar Buyya* and Baikunth Nath

School of Computing and Information Technology
Monash University, Gippsland Campus
Churchill, VIC 3842, Australia

*School of Computer Science and Software Engineering
Monash University, Caulfield Campus
Caulfield, VIC 3145, Australia

Email: {Ajith.Abraham, Rajkumar.Buyya, Baikunth.Nath}@infotech.monash.edu.au

Abstract

Computational Grid (Grid Computing) is a new paradigm that will drive the computing arena in the new millennium. Unification of globally remote and diverse resources, coupled with the increasing computational needs for Grand Challenge Applications (GCA) and accelerated growth of the Internet and communication technology will further fuel the development of global computational power grids. In this paper, we attempt to address the scheduling of jobs to the geographically distributed computing resources. Conventional wisdom in the field of scheduling is that scheduling problems exhibit such richness and variety that no single scheduling method is sufficient. Heuristics derived from the nature has demonstrated a surprising degree of effectiveness and generality for handling combinatorial optimization problems. This paper begins with an introduction of computational grids followed by a brief description of the three nature's heuristics namely Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS). Experimental results using GA are included. We further demonstrate the hybridized usage of the above algorithms that can be applied in a computational grid environment for job scheduling.

Keywords: Computational grid, grid computing, scheduling, resource management, global optimization algorithms, genetic algorithm, simulated annealing, tabu search, nature's heuristics and hybrid algorithm.

1 Introduction

Grid Computing (GC) is the ultimate framework to meet the growing computational demands in the new millennium [1-3]. To meet the growing needs of the computational power, geographically distributed resources need to be logically coupled together to make them work as a unified resource (see Figure 1). The continuous exponential growth of the Internet and World Wide Web (WWW), ever improving high-bandwidth communications, wide spread availability of powerful computers and low-cost components will further enhance transformation of computational grids to a reality. Computing resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints. Every resource owners will have a unique way of managing and scheduling resources and the grid schedulers are to ensure that they do not conflict with resource owner's policies. In the worst-case situation, the resource owners might charge different prices to different grid users for their resource usage and it might vary from time to time. Traditionally, most of the schedulers followed system centric approach in resource selection and often completely ignore the user requirements. In an economic-based approach an optimal schedule often relies on a trade off between cost and the user specified deadline [4]. Our approach is to dynamically generate an optimal schedule so as to complete the tasks in a minimum period of time as well as utilizing the resources in an efficient way.

In recent years, several analogies from the natural and social systems have been widely accepted to form powerful heuristics, which have proven to be highly successful in solving several NP hard global optimization problems [5][12][18]. Some of the common characteristics of nature's heuristics are the close resemblance of a phenomenon existing in nature, non-deterministic; present implicitly a parallel structure and adaptability. In section 3, 4 and 5 we will briefly discuss the features of genetic algorithm, simulated annealing and tabu search and in section 6 we demonstrate how it can be used to formulate scheduling independent tasks in a grid environment.

2 Grid Resource Management and Scheduling Issues

Figure 1 depicts the general framework for grid computing focusing on the interaction between grid resource broker, Domain Resource Manager (DRM) and the grid information server. The grid resource broker is responsible for resource discovery, deciding allocation of a job to a particular resource, binding of user applications (files), hardware resources, initiate computations, adapt to the changes in grid resources and present the grid to the user as a single, unified resource. It finally controls the physical allocation of the tasks and manages the available

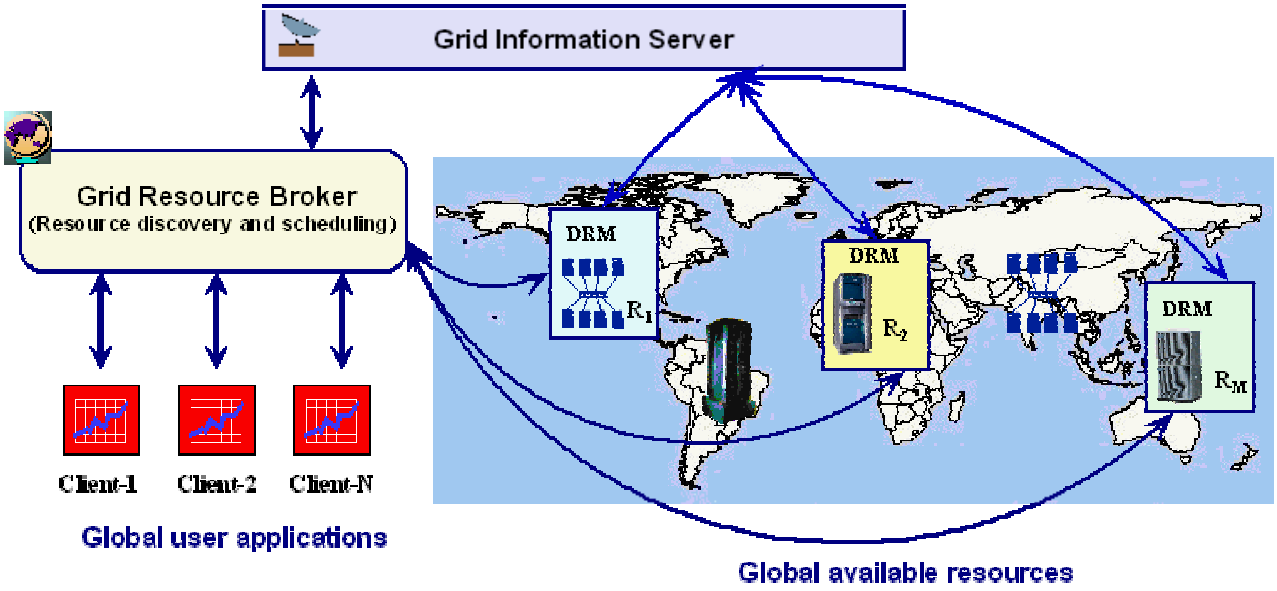

Figure 1. General Framework of a Computational Grid

resources constantly while dynamically updating the grid scheduler whenever there is a change in resource availability. In a grid environment knowing the processing speeds of the available resources and the job length of user applications is a tedious task. Usually it is easy to get information about the speed of the available resources but quite complicated to know the computational processing time requirements from the user. When the computing power demand is much greater than the available resources only dynamic scheduling will be useful. To conceptualize the problem as an algorithm, we need to dynamically estimate the job lengths from user application specifications or historical data. Soft computing techniques like fuzzy logic and artificial neural networks might be of useful aid in the parameters estimation process especially in times of uncertainty and vague data.

To formulate the problem, we consider J_n independent user jobs n={1,2,….N} on R_m heterogeneous resources m={1,2,….,M} with an objective of minimising the completion time and utilizing the resources effectively. The speed of each resource is expressed in number of cycles per unit time, and the length of each job in number of cycles. Each job J_n has processing requirement P_j cycles and resource R_m has speed of S_i cycles/unit time. Any job J_n, once allocated to a resource R_m, has to be processed on this resource until completion.

To formulate our objective, define C_j as the completion time the last job j finishes processing. Define C_{max} = max $\{\Sigma C_j /M, j=1,\ldots,N\}$, the makespan and ΣC_j , as the flowtime. An optimal schedule will be the one that optimizes the flowtime and makespan [6]. The conceptually obvious rule to minimize ΣC_j is to schedule Shortest Job on the Fastest Resource (SJFR). The simplest rule to minimize C_{max} is to schedule the Longest Job on the Fastest Resource (LJFR). Minimizing ΣC_j asks the average job finishes quickly, at the expense of the largest job taking a long time, whereas minimizing C_{max}, asks that no job takes too long, at the expense of most jobs taking a long time. In summary, minimization of C_{max} will result in maximization of ΣC_j or vice versa [17].

3 Genetic Algorithm (GA)

GAs are adaptive methods that can be used to solve optimization problems, based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "Survival of the Fittest", first clearly stated by Charles Darwin in "The Origin of Species". By mimicking this process, GAs are able to "evolve" solutions to real world problems, if they have been suitably encoded. GA search is constrained neither by the continuity of the function under investigation, nor the existence of a derivative function [7]. Figure 2 illustrates the functional block diagram of a GA. It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as genes) are joined together to form a string of values (known as a chromosome). The particular values the genes represent are called its alleles. The position of the gene in the chromosome is its locus. Encoding issues deal with representing a solution in a chromosome and unfortunately, no one technique works best for all problems. A fitness function must be devised for each problem to be solved. Given a particular chromosome, the

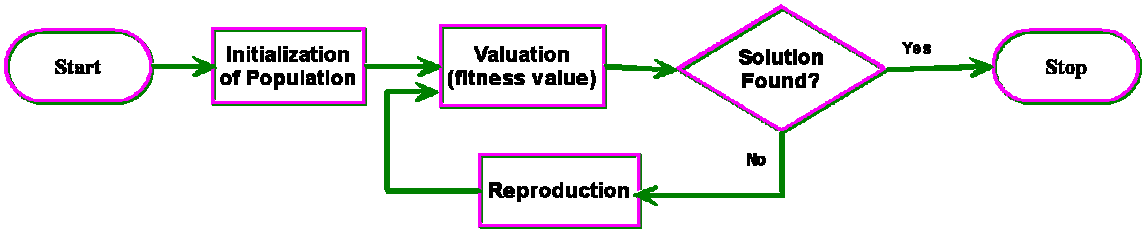

Figure 2. Flowchart of GA Iteration

fitness function returns a single numerical fitness or figure of merit, which will determine the ability of the individual, which that chromosome represents. Reproduction is another critical attribute of GAs where two individuals selected from the population are allowed to mate to produce offspring, which will comprise the next generation. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. Traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and helps ensure that no point in the search space has a zero probability of being examined. If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum. Selection is the conservation of the fittest individuals for the next generation and is based on 3 parts. The first part involves determination of the individual's fitness by the fitness function. The second part involves converting the fitness function into an expected value followed by the last part where the expected value is then converted to a discrete number of offspring. To avoid premature convergence of GAs due to interference from mutation and genetic drift, sharing and crowding may be used to decrease the amount of duplicate schemata in the population. Elitism may be incorporated to keep the most superior individuals (and superior schemata) within the population. The use of GA in formulating the scheduling algorithm is discussed in section 6.1.

4 Simulated Annealing (SA)

SA exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. Figure 3 shows a flowchart of SA iteration. The annealing schedule, i.e., the temperature-decreasing rate used in SA is an important factor, which affects SA's rate of convergence.

The algorithm employs a random search, which not only accepts changes that decrease objective function "f", but also some changes that increase it. The latter are accepted with a probability $p = \exp\left(-\frac{\delta f}{T}\right)$, where δf is the increase in objective function, and "f" and T are control parameters. Several SAs have been developed with annealing schedule inversely linear in time (Fast SA), exponential function of time (Very Fast SA) etc. We explain a SA algorithm [10], which is exponentially faster than Very Fast SA whose annealing schedule is given by $T(k) = \frac{T_0}{\exp(e^k)}$, where T_0 is the initial temperature, $T(k)$ is the temperature we wish to approach to zero for $k=1,2,....$

If the generation function of the simulated annealing algorithm is represented as:

$$g_k(Z) = \prod_{i=1}^{D} g_k(z_i) = \prod_{i=1}^{D} \frac{1}{2(|z_i| + \frac{1}{\ln(1 / T_i(k))}) \ln(1 + \ln(1 / T_i(k)))}$$

Where $T_i(k)$ is the temperature in dimension i at time k. The generation probability will be represented by

$$G_k(Z) = \int_{-1}^{z_1} \int_{-1}^{z_2} .....\int_{-1}^{z_D} g_k(Z) dz_1 dz_2 ....dz_D = \prod_{i=1}^{D} G_{ki}(z_i)$$

Where $G_{ki}(z_i) = \frac{1}{2} + \frac{\mathrm{sgn}(z_i) \ln(1 + |z_i| \ln(1 / T_i(k)))}{2 \ln(1 + \ln(1 / T_i(k)))}$

It is straightforward to prove that an annealing schedule for

$$T_i(k) = T_{0i} \exp(-\exp(b_i k^{1/D}))$$

A global minimum, statistically, can be obtained. That is,

$$\sum_{k=k_o}^{\infty} g_k = \infty$$

Where $b_i > 0$ is a constant parameter and k_0 is a sufficiently large constant to satisfy the above equation if the generation function is adopted.

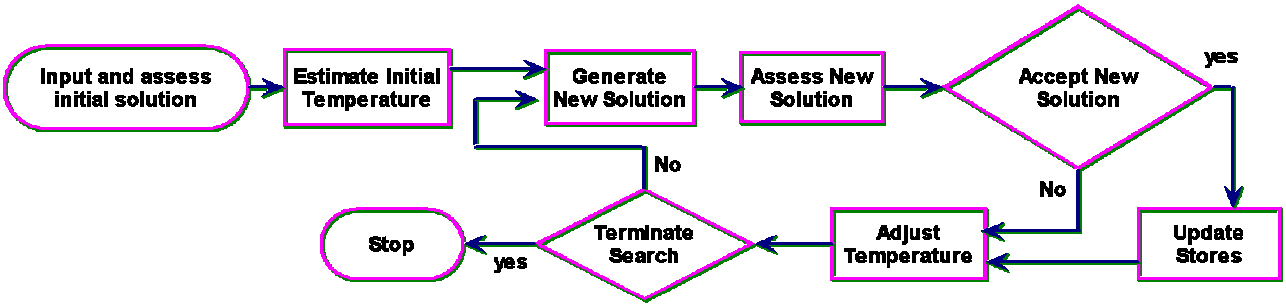

Figure 3. Flowchart of SA Iteration

5 Tabu Search (TS)

TS is a meta-strategy for guiding known heuristics to overcome local optimality and has now become an established optimization approach that is rapidly spreading to many new fields. The method can be viewed as an iterative technique which explores a set of problem solutions, denoted by *X*, by repeatedly making moves from one solution *s* to another solution *s′* located in the neighborhood *N(s)* of *s*. These moves are performed with the aim of efficiently reaching optimal solution by the evaluation of some objective function f(s) to be minimized [8][9][13]. In the sequel we sketch the basic ingredients of TS.

Let us define the notion of neighborhood N(s) for each solution *s* in X. By definition N(s) is a set of solutions in X reachable from *s* via a slight modification *m*.

$$N(s) = \{s' \in X \mid s' = s \oplus m, m \in M\}$$

Where M contains all possible modifications and $s' = s \oplus m$ means that *s′* is obtained by applying modification *m* to *s*. TS starts from an initial solution randomly generated in X and moves repeatedly from a solution to a neighbor. At each step of the procedure, a subset *V** of the neighborhood of the current solution *s* is generated and the local optimization problem $min\{f(x) \mid x \in V^* \subseteq N(s)\}$ is solved. In order to escape from local minima, the idea is to move to the best neighbor *s′* in *V** even if *f(s′) > f(s)*. Following a steepest descent / mildest ascent approach, a move may result in a best possible improvement (or a least possible deterioration) of the objective function value. Without additional control, however, such a process can cause a locally optimal solution to be re-visited immediately after moving to a neighbor, or in a future stage of the search process. To prevent the search from endlessly cycling between the same solutions, a tabu list T is introduced. This list keeps track of the reverses of the last $|T|$ modifications that have been done enacted during the search process. A move from *s* to *s′* will be considered tabu if it is performed via a modification contained in T. However the concept of tabu list sometimes appears to be restrictive. Since only parts of the neighborhood are explored, it might be worth returning after a while to a solution visited previously to search in another direction. An aspiration function *A* deals precisely with the rigidity of the tabu list. It permits the tabu status of a move to be dropped under certain favorable circumstances.

We define an aspiration level *A(z)* for each value *z* of the objective function. Then a tabu move from *s* to *s′* is permitted if *f(s′)<A(f(s))*. Initially *A(z)=z* , for all possible values of *z=f(s)*. This aspiration function is updated as follows whenever we move from *s* to *s′*.

A(f(s))=min(A(f(s)),f(s')) and *A(f(s'))=min(A(f(s')),f(s))*

The above shows that the reverse move from *s′* to *s* is considered in the updating of *A* even though it was not done explicitly. Generally the process is stopped as soon as a given number of iterations have been performed without improving the best solution obtained.

6 Chromosome Representation and Issues

For applying GAs directly or coupled with other meta-heuristics, problem (chromosome) representation is very important and it directly affects the performance of the proposed algorithm. The first decision a designer has to make is how to represent a solution in a chromosome. We assume that the jobs and resources are arranged in an ascending order according to the job lengths and processor speeds. The information related job lengths may be derived from historical data, some kind of strategy defined by the user or through load profiling. Figure 4 depicts the chromosome representation which can be used in the 3 heuristic algorithms presented in sections 6.1-6.3. Each block in the chromosome represents a gene, coding a particular sequence of jobs. Job J_1 is allocated to resource R_1, J_2 to R_2 and J_3 to R_3 and so on. When J_1 is completed, resource R_1 is empty and job J_N is allocated. This procedure goes on until all the jobs are allocated.

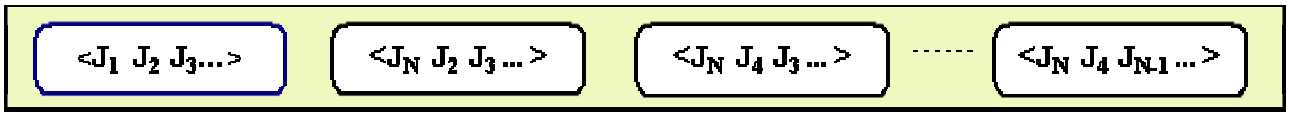

Figure 4. Representation of a chromosome

To optimize the makespan and flowtime we propose to swap the usage of LJFR and SJFR heuristic alternatively every time a new job is allocated to a resource. In sections 6.1-6.3, we present a pure genetic algorithm, hybrid genetic-simulated annealing and hybrid genetic-tabu search approach for scheduling the jobs. If the number of jobs is less than the number of resources, we propose to allocate the jobs based on a First-Come-First-Serve basis and LJFR heuristic (if possible). In a grid environment, a scheduler might have to make a multicriteria decision analysis (access policy, access cost, resource requirements, processing speed, etc.) for selecting an optimal resource.

To formulate the algorithm, we propose the following job lists and resource lists. JList1 and Rlist1 are to be dynamically updated through load profiling, grid resource health status, and forecasted load status, etc. along with grid information services. The entire job and the resource lists are to be arranged in the ascending order of the job lengths and processing speeds/access-cost (based on multicriteria decision analysis). Frequency of updating the lists will very much depend on the grid condition, availability of resources and jobs.

JList1 = Job list maintaining the list of all the jobs to be processed.
Jlist2 = Job list maintaining only the list of jobs being scheduled.
Jlist3 = Job list maintaining only the list of jobs already allocated.
Rlist1 = List of available resources (including time frame).
Rlist2 = List of resources already allocated to jobs.
Rlist3 =List of free resources = (Rlist1-Rlist2).

6.1 GA Approach for Job Scheduling on the Grid

1. If the grid is active and (*Jlist1=0*) and no new jobs have been submitted, wait for new jobs to be submitted. Update *Rlist1 and Jlist1*.

2. If (*Rlist1=0*), wait until resources are available. If *Jlist1>0, update Jlist2*. If *Jlist2 < Rlist1* (available resources) allocate the jobs on a first-come-first-serve basis and if possible allocate the longest job *J* on the fastest resource *M* according to the *LJFR* heuristic. If *Jlist1 > Rlist1*, job allocation is to be made by following the heuristic algorithm detailed below. Take jobs and available resources from Jlist2 and Rlist3.

3. At t =0; generate an initial population with *P* chromosomes $Pop_i(t)$, encoding the schedules. Feasibility of each chromosome is to be checked and makespan for each schedule represented by the chromosome is to be calculated. In certain cases, illegal offspring's (duplicated jobs, missing jobs, jobs outside the list) are generated due to genetic operators that require to be repaired immediately to ensure that each job appears only once in the sequence.

4. Begin GA loop

   While (until the specified fitness value is achieved) ***do****;*

   a. For each chromosome (i=1 to P), first allocate the jobs to the available resources based on the LJFM heuristic and once a resource is free (due to job completion), a job is allocated based on the SJFM heuristic. There after LJFR – SJFR heuristic is applied alternatively after completion of every job. Calculate the make span and total flowtime for the generated schedule. Also calculate fitness value of each chromosome, $Fitness_i=FitPop_i(t)$*;*

   b. For (i=1 to P), Create new population New$Pop_i(t+1)$ *which is* choosen randomly based on the fitness value of each chromosome $Pop_j(t)$ *in the current population Pop(t).* The probability for selection a chromosome for the next generation may be defined as $p_j = \frac{Fitness_j}{\sum_{k=1}^{P} Fitness_k}$ or according to the selection strategy adopted;

   c. Apply crossover operator on the population according to the probability selected, Crospop(t+1) = recombined chromosomes of the population $NewPop_i(t+1)$*;*

   d. Apply mutation operator on the population according to the probability selected, *MutPop(t+1)* = mutated population *CrosPop(t+1).*

5. Evaluate fitness of each individual and when the $Fitness_i$ has reached the required value *end loop.*

6. Check the feasibility of the generated schedule with respect to resource availability and user specified requirements. Then allocate the jobs to the resources and update Jlist2, JList3, RList2 and Rlist3. Un-allocated jobs (infeasible schedules or resource non-availability) shall be transferred to JList1 for re-scheduling or dealt with separately.

7. Repeat steps 1-6 as long as the grid is active.

We define that a job schedule is feasible only if the makespan (C_{max}) does not exceed the user specified completion time. We suggest the use of a penalty function to distinguish a feasible schedule from the non-feasible and relate as (*User specified completion time - makespan of the generated schedule).* If makespan exceeds the *required completion time* the fitness value will be negative. The fitness function is set to the inverse of the flow time (ΣC_j) from the generated schedule. The sequences of the jobs could be coded in a sequence of integer arrays. In our scheduling experimentations using GAs [6][15], we have achieved good results using 2-point crossover operator and mutation operators with a selection probability of 1.0.

6.2 Hybrid GA-SA Approach for Job Scheduling on the Grid

GA-SA is a hybrid random searching technique fusing GA and SA, inheriting the convergence property of SA and parallalization capability of GA. Each genotype is assigned an energy threshold, initially equal to the energy of the randomized bit string to which it is assigned. If the energy of the mutant exceeds the threshold of the parent that spawned it, the mutant is rejected and a new genotype is considered. However if the energy of the new genotype is less than or equal to the energy of the parent, the mutant is accepted as a replacement for its progenitor. GA-SA uses an Energy Bank (EB) to keep track of the energy liberated by the successful mutants. Whenever a mutant passes the threshold test, the difference between the threshold and the mutant's energy is added to the EB for temporary storage. Once the quantum of energy is accounted, the threshold is reset so that it equals the energy of the accepted mutant and move on to next member of the population. After each member has been subjected to a random mutation, the entire population is reheated by changing the threshold. The rate of reheating is directly proportional to the amount of energy accumulated in the EB (from each member of the population) as well as designer's choice of coolant rate (refer to section 4). Annealing results from repeated cycles of collecting energy from successful mutants and then redistributing nearly all of it by raising the threshold energy of each population member equally [11]. The GA-SA algorithm for job scheduling on computational grid can be formulated as follows:

1 and 2 are the same as in section 6.1

3. Generate an initial population of P schedule vectors and for i =1 to P, initialize the i^{th} threshold, $T_h(i)$, with the energy of the i^{th} configuration. For each schedule (i=1 to P), first allocate the jobs to the available resources based on the LJFM heuristic and once a resource is free (due to job completion), a job is allocated based on the SJFM heuristic. There after LJFR – SJFR heuristic is applied alternatively after completion of every job.

4. *Begin the cooling loop*
   - Energy bank (EB) is set to zero and for i = 1 to N randomly mutate the i^{th} schedule vector.
   - Compute the Energy (E) of the resulting mutant schedule vector.
     - If $E > T_h(i)$, then the old configuration is restored.
     - If $E \leq T_h(i)$, then the energy difference ($T_h(i)$ –E) is incremented to the Energy Bank (EB) = EB+ $T_h(i)$ –E. Replace old configuration with the successful mutant

   End cooling loop.

5. *Begin reheating loop.*
   - Compute reheating increment $eb = \frac{EB * T_i(k)}{N}$, for i= 1 to N. (Ti(k)=cooling constant).
   - Add the computed increment to each threshold of the schedule vector.

   End reheating loop.

6. Go to step 4 and continue the annealing and reheating process until an optimum schedule vector is found.

7. Same as step 6 and 7 as mentioned in section 6.1.

6.3 Hybrid GA-TS Approach for Job Scheduling on the Grid

Contrary to the SA, there exists no convergence theorem for TS. The research and experimentation's have proven TS is powerful, very flexible and easy to implement. However performance of TS very much depends on the selection of parameters and a good knowledge of the problem to be solved is required to design the parameters appropriately. Hybridization of TS with GA makes the algorithm more robust. In the hybrid GA-TS approach, reproduction, crossover and mutation

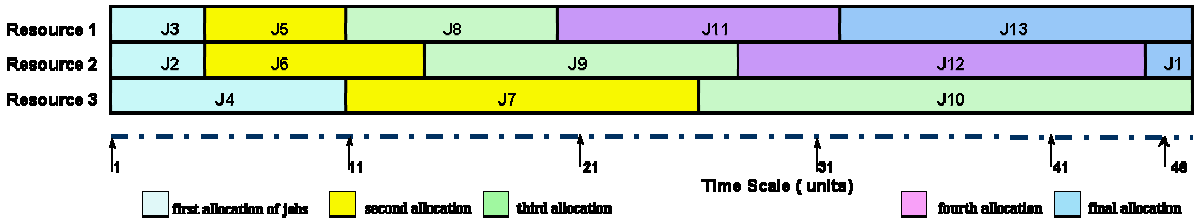

Figure 5. Simulation results demonstrating resource usage and job allocation schedule using pure GA approach.

in GA is replaced by reproduction, crossover and Tabu search. Instead of random mutation changes each member of the population undergoes a separate optimization process described by a tabu algorithm [12].[13]. The application of the hybrid algorithm for job scheduling on computational grid can be formulated as follows:

1, 2 and 3 are the same as in section 6.1.

4. *Begin GA-TS loop.*

   a, b and c are the same as mentioned in section 6.1.

   d. Begin tabu algorithm

   Initialization

   i. Evaluate neighborhood N (S) for every individual of *CrosPop(t+1).* Generate maximum number of feasible *schedule* for each *CrosPop(t+1)* individual at each step of the search.

   ii. For every *schedule,* evaluate the makespan. Let s*=s; for the best solution reached so far and set iteration counter *(iteration)*= 0. For the best moves (schedules created) set the iteration counter *bestiteration =0, set tabu list (T=0)* and initialize the aspiration function *A*.

   Begin loop

   iii. ***While*** *(iteration-bestiteration < nmax)* ***do***
   iteration= iteration+1;
   generate a set of feasible V* solutions $s_i = s \oplus m_i$,such that $m_i \notin T$
   or $f(s_i) < A(f(s));$
   Choose the best solution s′ in V*; update tabu list T and A(f);
   If *(f(s′) < f(s*))* ***then***
   s*=*s′*
   bestiteration=iteration;
   s=*s′*
   End loop when number of iterations has reached *nmax*.

5. the same as steps 6 and 7 as mentioned in section 6.1.

7. Algorithm performance demonstration

Simulation using pure GA was carried out to demonstrate the proposed scheduling method using the data given in table 1. To simplify the simulation, we considered a finite number of resources and assumed that the processing speeds of the resources (Cycles Per Unit Time-CPUT) and the job lengths (processing requirements in cycles) are known.

Table 1: Test data (13 jobs and 3 resources)

Resource Speed (cput)	Job lengths (cycles)	
$R_1 = 4$	J_1=6	J_7=30
$R_2 = 3$	J_2=12	J_8=36
$R_3 = 2$	J_3=16	J_9=40
	J_4=20	J_{10}=42
	J_5=24	J_{11}=48
	J_6= 28	J_{12}=52
	J_{13}=60	

Figure 5 demonstrates one of the optimal ways of allocating the 13 jobs to the 3 resources such that all the resources are utilized efficiently and all the jobs are completed in minimum time. From the simulation results, all the resources were equally utilized for 46 time units (makespan) and the total time taken for completion of all the jobs is 138 time units (flowtime). Initially J_3, J_2 and J_4 are allocated to R_1 R_2 and R_3 respectively. After the completion of the allocated jobs (J_3, J_2 and J_4) next set of jobs are assigned to the resources and the complete schedule is in figure 5.

8 Conclusions and Future Work

In this paper, we attempted to address the hybridization of the three of the popular nature's heuristics namely GA, SA and TS for job scheduling on large-scale distributed systems. Conceptually, when compared to pure GA, the GA-SA algorithm should provide a better convergence and GA-TS algorithm should improve the search efficiency of GA. A number of

related works have shown that hybrid heuristic algorithms do perform better than the classical GA approach [15].

Global optimization algorithms attract considerable computational effort. In a grid environment, the main emphasize will be to generate the schedules at a minimal amount of time. Especially as the demand increases, when the number of jobs and the resources starts towering up, conventional GAs become time consuming. Fortunately GAs work with a population of independent solutions, which makes it easy to distribute the computational load among several processors. The design of parallel GA's involves choices such as using one population or multiple populations. In both cases, the size of population or populations must be determined carefully, and when multiple populations are used, one must decide how many to use. In addition, the populations may remain isolated or they may communicate by exchanging individuals. Communication involves extra costs and additional decision on topologies, on how many individuals are exchanged, and on the frequency of communications [14].

Currently a large number of research projects worldwide are exploring different approaches to the development of grid technologies and global scheduling systems [16]. Even though significant progress has been made in modeling the infrastructure for grid computing, a close review clearly indicates that not much progress is made in formulating an efficient, globally optimized, grid-scheduling algorithm for allocating jobs. Therefore, we plan to explore this space in depth along with the implementation and evaluation of the heuristics and algorithms in the context of global computational grids.

References

[1] Foster I, Kesselmann C, (Eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.

[2] Buyya R, Abramson D, Giddy J, *Grid Resource Management, Scheduling, and Computational Economy*, International Workshop on Global and Cluster Computing, Japan, 2000.

[3] Baker M, Buyya R, Laforenza D, *The Grid: International Efforts in Global Computing*, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, Rome, Italy, 2000.

[4] Buyya R, Abramson D, Giddy J, *An Economy Driven Resource Management Architecture for Global Computational Power Grids*, International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, 2000.

[5] Abraham A, Nath B, *Hybrid Heuristics for Optimal Design Of Artificial Neural Networks*, Third International Conference on Recent Advances in Soft Computing (RASC2000), England, June 2000.

[6] Nath B, Lim S, Bignall R J*, A Genetic Algorithm For Scheduling Independent Jobs On Uniform Machines With Multiple Objectives*, Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, Australia, pp. 67-74, 1998.

[7] Goldberg DE, *Genetic Algorithms in Search, Optimization and machine learning*, Addison-Wesley Publishing Company, Inc., 1989.

[8] Taillard.E, *Parallel Tabu Search Technique for the Job Shop-Scheduling Problem*, ORSA Journal of Computing, (6):108-117, 1994.

[9] Glover F, Taillard E, De Werra D, *A User's Guide to Tabu Search*,. In: Hammer PL (Ed.) Annals Of Operations Research, Volume 41, 3-28, pp 3-27, 1993.

[10] Yao. X, *A New Simulated Annealing Algorithm*, International Journal of Computer Mathematics, 56: pp.161-168, 1995.

[11] Price KV, *Genetic Annealing*, Dr. Dobbs Journal, Vol.220, pp. 127-132, Oct 1994.

[12] Colorni A., M.Dorigo, F.Maffioli, V. Maniezzo, G. Righini & M. Trubian, *Heuristics from Nature for Hard Combinatorial Problems,* International Transactions in Operational Research, Vol (3) 1, pp 1-21, 1996.

[13] Costa D., *An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem*, INFOR Vol. 33, No.3, 161-178, 1995.

[14] Cantu-Paz E, *Designing Efficient and Accurate Parallel Genetic Algorithms*, Technical Report No. 99017, Illinois Genetic Algorithms Laboratory, UIUC, USA, July 1999.

[15] Nath B, *A Hybrid GA-SA Algorithm for Flowshop Scheduling Problems*, In Proceedings of the International Conference on Computer Integrated Manufacturing, ICCIM97, pp 462-471, 1997.

[16] Grid Computing Information Centre*:* http://www.gridcomputing.com.

[17] Nath B & Abraham A, *Parallel Machine Scheduling using Genetic Algorithms*, International AMSE conference on Computer Modeling, Simulation and Communication, (CMSC-99), India, 1999.

[18] Wall B W, *A Genetic Algorithm for Resource Constrained Scheduling*, PhD Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, USA, 1996.