A Heuristic Genetic Algorithm based Scheduler for 'Clearing House Grid Broker'

Pramod Kumar Konugurthi[†*], Krishnan Ramakrishnan[†] and Rajkumar Buyya[*]

[†] Project Implementation & High Performance Computing Section
Advanced Data Processing Research Institute (ADRIN)
Department of Space, Govt. of India, Hyderabad, India
Email:{pramod, krishnan}@adrin.res.in

[*]Grid Computing and Distributed Systems (GRIDS) Lab
Department of Computer Science and Software Engineering
University of Melbourne, Australia
Email:{pramod, raj}@csse.unimelb.edu.au

Abstract- **In the recent past there has been an increasing demand from users for QoS based resource selection especially in Enterprise Grids. Hence, 'brokers' which schedule jobs based on the QoS requirements have been designed and implemented in projects like Condor-G, Gridbus, GRUBER and Nimrod/G. Although several mechanisms have been suggested in these projects, all of them either consider optimization for the jobs submitted to the local node or use a conventional FCFS and Greedy approaches for scheduling the jobs.**

In this paper we propose and demonstrate results that such brokers are not only non-scalable but also fail to deliver optimal schedules. We then propose the need for a 'third party broker' and suggest a Clearing House Broker which uses Publish/Request mechanisms to schedule the jobs on Grid Federations or Virtual Organizations. The final execution and control still happens at the user end, which is decentralized. First, we present the architecture of such a broker and discuss the requirements and the functionality of the Clearing House Broker. Later, we discuss the pros and cons of different implementations of Clearing House Broker and finally propose a heuristic based Genetic Algorithm for an optimal mapping of application jobs to suitable resources. We simulate all the existing broker schedulers used in Gridbus, Nimrod/G and GRUBER, and compare the results of these with our proposed scheduling algorithm.

Keywords: *Third Party Broker, Scheduler, Quality of Service, Budget and Deadline, Genetic Algorithm*

I. INTRODUCTION

The Grid [1] has started from the realization of scientific computations over geographically distributed systems and has been an emerging technology in recent years. A 'Virtual Organization' (VO) in the Grid is defined as a set of individuals and institutions forming an ad-hoc partnership to solve a common problem by sharing the resources [1] [2]. Although the Grid has been visualized as a cooperative and coordinated sharing of resources, with the recent success of the commercial grids like Amazon Elastic Computing Cloud (EC2) [3], where the users are willing to pay for the quality of service (QoS) and also with the increasing demand from all scientific/commercial community for solving more complex solutions, Grid is becoming a main stream infrastructure. Hence Grid Brokers, which do an efficient resource management and scheduling considering the Grid economy models [4] are going to play a very critical role in achieving the optimal utilization of resources, thereby increasing the revenue for the Grid providers and also minimize the cost for Grid users for the desired QoS.

Present Grid resource management and scheduling systems such as Gridbus[5], Nimrod/G[6] and Condor-G[9] implement the brokering at the user end, trying to optimize the cost or time for its local users by considering the other parameter (cost or time) as the constraint. In subsequent sections we present the results, that these approaches result in resource under utilization, thereby failing to achieve the cost/time optimizations. The next section discusses more of the related works in this area. It presents a case for a clearing house concept of scheduling, which does the scheduling at a centralized place, yet the actual job submission/execution is done at the local user site itself. The third section discusses the third party broker architecture namely, Clearing House Broker. We also present the mathematical model for the broker. The fourth section discusses the simulation studies of various brokers that are implemented in Nimrod/G, Gridbus and GRUBER brokers using GridSim[13] and compares the results. Finally, we conclude by proposing our future work in this area in section five.

II. RELATED WORK

Previous works [5][6] in the area of economy based schedulers considered the personalized version of brokers, which resides at the user end and tries to optimize cost or time by keeping the other parameter as constraint, i.e., if cost optimization is performed then time of completion(deadline) is considered as the constraint and similarly for time

optimization. But the disadvantage with these approaches is when multiple users try to submit the jobs at the same time, it would result in contention for the same resource. This issue has been addressed by many researchers and have proposed Service Level Agreement (SLA) based or contract based negotiation before executing the jobs [7] [10] [14] [15] [27]. In [30] authors have suggested an auction based policies for VO, while [7] proposed a case for Grid Federation Agents, which uses a decentralised resource information sharing protocols and scheduling followed by SLA negotiations. These systems work fine for reservation based scheduling in Grids, where the probability of concurrent users or simultaneous job submissions is very low. These systems fail to scale especially when the concurrent users increase [25] and result in resource under utilization. In [8] policy based resource sharing has been suggested, but it does not address how to handle user QoS.

GRUBER [10] and DI-GRUBER [11] have also a centralized decision point and decentralized job execution environment, but the actual scheduling policies utilised are still classical ones such as FCFS, Round Robin, etc. Similarly the other resource management brokers listed in [20] [21] [22] [23] [24] use similar scheduling policies and are aimed at either minimizing the job completion time, and/or maximizing resource utilization. The limitations of these systems are two folds, first they have very limited support for user level QoS parameters and second, they use conventional scheduling algorithms which are suitable for per user based. In our previous work [12] we presented the drawbacks in such systems and presented a fuzzy based solution for combined/group scheduling. Although the schedule was 'one-to-one', i.e., one machine could execute one job per schedule, the next schedule would consider the same machine for scheduling if sufficient resources were available. In this paper we also consider multiple jobs being submitted to a single node, and bring in QoS parameters, i.e., budget and deadline constraints. Then we model the scheduling problem as Generalized Assignment Problem (GAP) as discussed in [17] [27]. GAP is a well known NP-hard problem and several people have tried to address heuristic based solutions for solving GAP. In [17] authors, proposed a heuristic improvement based on repairing the infeasible solutions before going for next breeding. In [19] an LP/IP based initial solution and followed be an intelligent replacement of offspring based on [18]. But these models do not consider the deadline and budget constraints and hence have to be modified accordingly.

III. CLEARING HOUSE BROKER

A. Architecture

Clearing House Broker offers a de-centralized control for grid consumers and providers, at the same time supports coordinated, optimal resource mapping for all the users and leaves individual user level brokers to manage the job submission and execution. Fig. 1 shows the architecture of the Clearing House Broker. Each user[1] and owner[2] registers with the broker. The registration process yields generation of keys required for PKI (Public Key Infrastructure), which is used for negotiation and contract finalization (discussed later in this section). Grid owners publish their 'resource-publish' on their gateway node or at a common place from where broker can fetch them. Broker obtains the latest 'resource-publish' from all the grid resources at every scheduled period. A 'resource-publish' consists of, the number of resources or processing elements (PEs) available for usage, corresponding load, memory, operating system, Grid middleware used, MIPS of each processing element and the cost per unit resource. Note that the publish information is dynamic and the Grid resource owners can vary any or all of the parameters at any time based on either their local requirements or based on economy models such as supply and demand. Finally, publish can also be merged with Grid Information Services (GIS) services also to make the implementation simpler.

Flow 1 (hereafter denoted as (1)) in Fig. 1 indicates the resource publish obtained by the broker. Grid users submit their job requests along with their QoS demands (2) to the broker. The QoS demands consist of budget, deadline, Grid middleware, if any and preferred/authorized list of resources that broker should look for, if any. Since the broker itself does not do the job submission and execution, the authentication and credentials are kept outside the broker's prerogative and hence they are considered as QoS parameters by the broker. We assume that the user obtains required credentials and authentications from the Grid owners before submitting the jobs to the broker. The idea behind this is to keep Grid owners and users, absolutely independent of the broker and the broker acts as a service provider for doing the optimal scheduling. This provides local autonomy, no centralised control and distributed ownership to all the users and Grid resource owners, which is a mandatory requirement for the Grid.

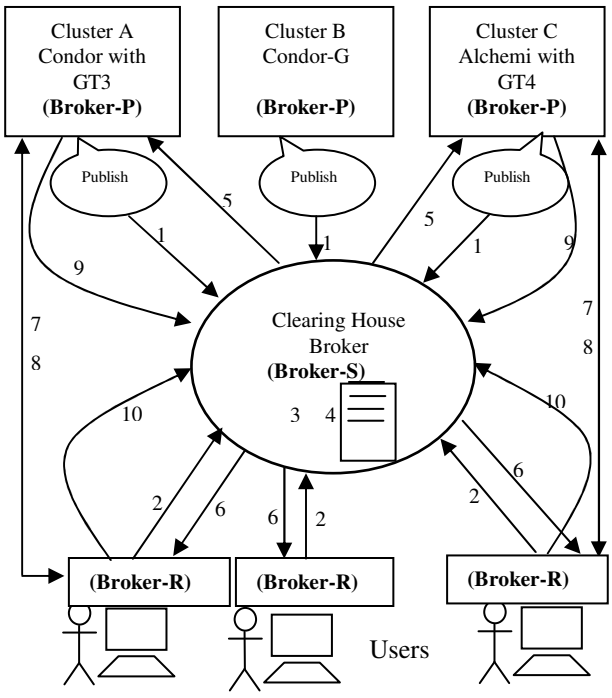

Fig. 1: Clearing House Broker Architecture

[1] user and requestor are used interchangeably and both mean the Grid user

[2] owner and provider are used interchangeably both mean the owner of the resource

The Clearing House Broker does HGA based scheduling as discussed in section 3.3 and generates combined schedule for all the jobs pending at the given instance(3). Then based on the schedule list the broker generates 'Tickets' for each valid job schedule in the schedule list (4) and the job is removed from the pending job list. 'Ticket' plays a critical role in establishing the contract and authentication between the Grid user and Grid owner. Ticket is a combination of broker id, broker's secret code, user id, job id, provider id, number of processing elements to be used, start time, end time and the cost negotiated by the broker. Note that, the problem is asymmetric, i.e., the total number of resources required by all the users need not be equal to the total number of resources available in the Grid. Hence, all the jobs need not necessarily be scheduled. Jobs not scheduled in the current schedule period are automatically passed over for the next period of scheduling.

As there is a commercial angle to the entire problem, we propose a PKI (Public Key Infrastructure) as followed in Grid Security Infrastructure (GSI) to enable secure functioning of the Grid scheduling, job execution and payment. Therefore the ticket $T_{i,j}$ implies ticket generated by broker for executing Q_j job on node P_i. First the broker's secret code is encrypted using its public key, i.e., $\Omega(S_{i,j})$ and then the ticket is encrypted by provider r_i's public key, i.e., $\Gamma_i(T_{i,j})$ and both the strings are sent to the provider r_i (5). The string thus generated can be represented as follows

$$\Omega(S_{i,j}) + \Gamma_i(T_{i,j}) \quad (A)$$
$$\Omega(S_{i,j}) + \Psi_j(T_{i,j}) \quad (B)$$

string (A) is sent to provider r_i (5) and (B) is sent to requestor u_j (6).

TABLE 1: LIST OF TERMS USED AND WHAT THEY CONTAIN

Term	What it contains
Publish[3] P_i	Resource Id, PEs available, Grid Middleware, and Cost per node per unit time for each user, MIPS rating of PEs (r_i, n_i, g_i, $c_{i,j}$, v_i)
QoS[1] Q_j	User Id, Job Id, Budget, Deadline, List of resources, Average estimated time for execution, credentials, Number of nodes required. (u_j, j_j, b_j, d_j, t_d L_j, A_j, m_j)
Ticket $T_{i,j}$	Serial Number, User Id, Resource Provider Id, PEs allotted, Start Time, End Time, cost negotiated (sn, u_j, j_j, r_i, m_j, ts_j, te_j, $c_{i,j}$)
Code $S_{i,j}$	Serial Number, Broker Id, Secret Code (sn, B, X)
$TS_{i,j}$	$S_{i,j} + T_{i,j}$
Γ_i	PKI Encryption function of provider 'i'
Ψ_j	PKI Encryption function of user 'j'
Ω	PKI Encryption function of the broker

At requestor end the second part of the string, i.e., $\Psi_j(T_{i,j})$ is decrypted using its private key to obtain ticket, $T_{i,j}$. The string thus obtained is as follows:

$$\Omega(S_{i,j}) + T_{i,j} \quad (C)$$

After obtaining the $T_{i,j}$, r_i is extracted from the ticket by the requestor u_j. Similarly at the provider's end also the same process is applied. Note that both the provider and requestor have generated the same string (C). Then the provider extracts requestor's id and number of processing elements along with the time, from the ticket and accordingly blocks the necessary resources for the requestor u_j. Then the requestor makes the claim for job submission and execution by sending the following string to provider r_i, i.e., the string obtained by encrypting (C) using the public key of provider r_i (7).

$$\Gamma_i(\Omega(S_{i,j}) + T_{i,j}) \quad (D)$$

Provider r_i upon receiving (D) decrypts the entire string using its private key. The string thus obtained is checked for equality with the string obtained after decrypting the string sent by the broker. If the strings are matched then an acknowledgment is sent, which triggers the job execution (8) of user u_j on resource provider r_i. After the job execution is completed then both the provider and requestor notify the broker about the completion of transaction by sending the first part of string A (9) and first part of string B (10), which contains the broker's secret code. Broker decrypts the strings obtained by the providers and users using his private key, and then checks for the consistency/equality of the strings obtained from both the owner and user. If the strings are found to be same and consistent then the broker initiates the payment action. The incentive payment and auditing is outside the scope of this paper.

For realization of this architecture we propose three components, viz., Broker-S, Broker-P and the Broker-R.

Broker-S: This module resides at the Clearing House Broker. Its main functionality is as follows:

- User and Owner Registration: This creates the certificates and also stores the credentials.
- Receives all the 'resource-publish' from the Grid owners and updates the hash table.
- Receive QoS demands from the users, i.e., Broker-R.
- Perform 'Heuristic Genetic Algorithm (HGA)' scheduling as discussed in the next section.
- Encryption and sending the schedules to corresponding provider, 'Broker-P' and requestor, 'Broker-R'.

Broker-P: This module resides at the provider end and its main functionalities are as follows:

- Registration with 'Broker-S'
- Update 'publish' parameters based on either supply demand or based on manual decisions.
- Receive schedule information from the 'Broker-S'
- Receive Job claims from 'Broker-R'
- Encryption and Decryption
- Claim for payment

Broker-R: This module resides at the requestor/user and has following functions:

- User registration with 'Broker-S'
- Submit job requests and manage execution as per schedule given by the Clearing House Broker

[3] *Ideally Processor Architecture, Operating System, Memory and Disk space available should be included, but for simplicity and for present simulations we assume all of these criteria are satisfied.*

- Receive schedule information from the 'Broker-S'
- Encryption and Decryption
- Job submission environment for various clusters

The present version of Gridbus broker has most of the 'Broker-S', 'Broker-R' components under one umbrella (i.e., except the encryption/decryption and HGA based scheduling) and we propose to refractor them and break into three components to support the Clearing House Broker.

Pros and Cons compared to the present implementations of the Broker

One of the limitations of the present implementations of the brokers in Nimrod/G, Gridbus, etc, which we refer to as 'personalized broker', is that each of the users try to optimize for the local submitted jobs. Here we consider the case of concurrent users (i.e., multiple users which are trying to use the Grid which is very common, especially in Production Grids), when each user tries to optimize for their jobs, then it would result in a situation where all the users select the same resource. Hence, when the users try to establish a contract or SLA, the resource owner would select only the first user and would send a 'fail SLA' for all other users. Failed-SLA jobs are re-submitted by local broker which would result in the same situation with reduced users as compared to previous schedule, provided no other user enters the grid. The process would be repeated until the job eventually finds a resource where SLA is established. Meanwhile if the deadline is over then the job is automatically thrown out. This is a live lock situation, where, even if the resources are available the jobs get omitted. Our simulation studies show more interesting results of this argument. The reason for this is the lack of coordination.

On the other hand, the personalised broker gives much more autonomy, is very simple to implement and also ideal for reservation based jobs and jobs with relaxed deadlines. The disadvantages are resource under utilization, less throughput, lower incentives to the Grid owners, and most importantly not suitable for jobs with tighter deadlines and concurrent user environments.

B. Problem Statement

The above discussed points are the motivation factors for our work. One of the vital requirements of the Broker-S is scheduling. In a Grid, multiple users would be submitting their jobs and similarly many nodes may join and leave in a very short duration. Hence, the possibility of simultaneous or concurrent users exists especially in the case of production Grids [25]. The users demand QoS and are willing to pay for the services they get, on the other hand the Grid resource owners would like to maximize their return on investment, in terms of incentives for sharing their resources. Thus, there is a requirement of optimizing for both, minimizing cost for the users and maximize the resource utilization for the providers (more the usage more incentives for the provider). Hence, a conventional FCFS, Greedy approach of scheduling adopted by various resource management brokers, does not yield better results as the scheduling of each job is done independent of the other jobs in the queue.

Thus we state the objectives of scheduling for a Clearing House Broker environment as follows:

- Scheduling based on QoS, i.e., budget and deadline
- Maximize the resource utilization, thereby increasing the owners payoff function
- Minimize cost for the users by meeting the budget and deadline constraints

In all our later discussions we assume that the basic requirements of processor architecture, memory, operating system, and other constraints are checked before scheduling. The broker categorises the requirements based on all these requirements and puts them in different queues. Similarly, it categorises the nodes also and does optimization for each such category.

C. Mathematical Formulation

Let P_i represent the 'resource-publish' of Grid resource i at a given instance T.

P_i: $(r_i, n_i, g_i, c_{i,j}, v_i)$, where

- r_i is the Resource Id
- n_i is number of free processing elements or resources available
- g_i is the Grid Middleware used at the local site
- $c_{i,j}$ is the cost of using a single resource per second, for the job/user j.
- v_i is the MIPS speed of one of the PE. In the case of varying MIPS each node is considered as a different resource

and $i \in I = \{1..m\}$

Let Q_j represent the 'QoS' demand from the user j at a given instance T

Q_j: $(u_j, j_j, b_j, d_j, L_j, A_j, M_k, t_j, m_j)$ where

- u_j is the user id
- j_j is the job id, since one user can submit multiple jobs, hence combination (u_j,j_j) is unique.
- b_j is the budget constraint the user specifies
- d_j is the deadline constraint for finishing the job
- L_j is the vector of resource ids, where the user has required access and necessary credentials for executing the jobs
- A_j is the Authorisation/Credential vector,
- M_k size of each task in the job in terms of MIPS (for every task k in m_j)
- t_j is the estimated duration of execution on an average processing capability node
- m_j is the number of resources user requires for executing the job.

and $j \in J=\{1..n\}$, $k \in K=\{1.. m_j\}$

Here we assume that the authorization credentials, Grid middleware, etc., are already checked and filtered out before

coming to the scheduling stage, therefore it does not appear in the problem formulation:

$$c(x) = \sum_{i=1}^{i=m} \sum_{j=1}^{j=n} c_{i,j} x_{i,j} r_{i,j} \quad (1)$$

$$\ni \sum_{j=1}^{j=n} r_{i,j} x_{i,j} \le n_i, \ \forall \quad i \in I \quad (2)$$

$$\ni \sum_{i=1}^{i=m} r_{i,j} x_{i,j} = m_j, \quad \forall \quad j \in J \quad (3)$$

$$\ni x_{i,j} \in \{0,1\}, \quad \forall \quad i \in I, j \in J \quad (4)$$

$$\ni \sum_{i=1}^{i=m} r_{i,j} c_{i,j} \le b_j, \ \forall \quad j \in J \quad (5)$$

$$\ni \text{MAX} \ (M_k / v_i) * x_{i,j} \le d_j, \forall \ i \in I, j \in J, k \in K \quad (6)$$

$$\ni c_{i,j}, r_{i,j}, b_j \text{ and } d_j > 0 \ \forall \quad i \in I, j \in J \quad (7)$$

$r_{i,j}$ denotes the number of resources allotted to a node. Note that the value would be either *0* or m_j (the number of resources required by the job *j*. Equation (2) denotes the node's capacity constraints. Equations (3) and (4) denote the assignment matrix, which indicates that a job can be assigned to a maximum of one node. Equation (5) is added for satisfying the budget constraints and equation (6) is added for the time constraints. Equation (1) is solved for minimization to obtain optimal schedule. Both equations (5) and (6) are specific to the Grid broker, which are additions on top of the classical Generalized Assignment Problem. All the quantities i.e., the resources, cost, deadline and budget are greater than zero, hence the equation (7). Note that Equations (3), (4) and (6) make the problem a 'NP-hard problem', thus an intelligent heuristics are required to reduce the time complexity.

D. Genetic Algorithm formulation

Genetic algorithms (GAs) [26] provide robust search techniques that allow a high-quality solution to be derived from a large search space in polynomial time, by applying the principle of evolution. A genetic algorithm combines the exploitation of best solutions from past searches with the exploration of new regions of the solution space. Any solution in the search space of the problem is represented by an individual (chromosomes). A genetic algorithm maintains a population of individuals that evolves over generations. The quality of an individual in the population is determined by a fitness-function. The fitness value indicates how good the individual is compared to others in the population. A typical genetic algorithm consists of the following steps:

1. Create an initial population consisting of randomly generated solutions.
2. Generate new offspring by applying genetic operators, namely selection, crossover and mutation, one after the other.
3. Compute the fitness value for each individual in the population.
4. Repeat steps 2 and 3 until the algorithm converges.

Convergence can be a different criterion for different problems, but generally 'a no change in the solution for *n* generations' is considered as convergence. *n* could be application specific again, in our implementation *n* is assigned a value of 10.

The most important aspect in GA is the solution space encoding and the fitness function. The genetic operations, i.e., crossover, mutation, inversion are standard operation in GA.

1) Problem Encoding

The solution for the scheduling is an assignment vector *S*, and each element s_j in the vector represents the node/resource onto which the job is scheduled. Since each job can be assigned to a maximum of one node, it can be denoted as a simple single dimensional integer vector of size equal to the number of jobs. Note that the node number s_j can be the same for multiple jobs, which means that, if there are sufficient resources then the node can execute multiple jobs.

1	3	8	0	5	1	2	1	3	7

Fig. 2: Chromosome representation or Problem encoding

$$S = \{ s_1, s_2, ..., s_n \}$$

where $s_j \in I$

2) Fitness Function

The fitness function we use is the cost function on the same lines of equation (1) and is denoted as follows:

$$F(S) = \sum_{j=1}^{n} \sum_{k=1}^{m_j} c_{s_{ji},j} * M_k / v_{s_j} + C * p_j \quad (8)$$

p_j is the priority of job *j* (priority sequence number, i.e., 1..n) and *C* is a 'cost constant' chosen such that it is sufficiently large, and also the fitness value of a least priority (higher value of p_j) job assigned to real node is always less than a higher priority (lower value of p_j) job scheduled to a dummy node. Addition of p_j into the fitness function gives preference to the jobs, which promise better QoS, and constraints on *C* ensures that the solution does not prematurely end in a local minimum. Priority is computed based on the QoS index discussed in the next section.

If constraints listed in equation (2), (5) and (6) are not satisfied then an infeasible cost or high cost is assigned to 'S'.

3) QoS index

QoS index is used to assign priority to the jobs. Generally priority is personal or provider's preference, but in an ideal market situation priority is linked with QoS, i.e., priority is given to the users who are willing to pay more incentives, and provide relaxed deadlines. Therefore it can be represented as follows:

$$\alpha_j = C * b_j / MAX \ (1, d_j - t_j) \quad (9.1)$$

where *C* is a constant value. QoS index is a fairness index computed for each job based on the deadline and budget parameters of the job. Ideally the user pays either the entire budget or amount proportional to its priority. As we are considering a 'fair market' broker scheduling, where user pays only for the resources he/she used, we use equation 9.2 for computing the priority index.

$$\text{if } (d_j > (T + t_j))$$
$$\alpha_j = 1 / MAX\ (1, d_j - (T + t_j)) \qquad (9.2)$$
$$\text{else}$$
$$\alpha_j = 0$$

T is the current schedule time. This function assigns priority to the jobs/users, which have lesser time for finish and also offer better incentive. Note that as the time increases, even the relaxed deadline jobs gain priority and similarly, if the user provides a tight deadline then there is a danger of getting eliminated from scheduling very quickly.

4) Proposed HGA Algorithm

Genetic Algorithms are known for their robust searching algorithms, but at the same time have a drawback of taking a long time and also more iterations before convergence. An initial solution to the problem significantly reduces the search space and converges fast [18]. Hence we suggest a priority based greedy scheduling to obtain a good initial solution. Priority is computed based on fairness index as denoted in equation (9.2). Then we iterate using genetic operations to get the best solution.

TABLE 2: HGA ALGORITHM

```
1. while (current_time < next_schedule_time) do
    1.1  Receive 'resource-publish' (Pj) from the providers
    1.2  Receive 'QoS requests' (Qj) from all the users
2. scheduleList:do HGAScheduling()
    2.1  Add a dummy node(S) with more PEs and high cost to P
    2.2  PLIST:sort(P(ascending order of cost))
   2.2  for each job 'j' in Q compute QoS index as given in (8)
    2.3  JLIST=sort(Q(descending order of QoS index))
    2.4  // greedy scheduling
      for each job 'j' in JLIST do
        for each node in PLIST do
          if(JLIST[i].PEs<PLIST[j])
            If (check_deadline_budget() is true)
              Add (j,i) to the SCHEDULE_LIST
              PLIST[j].freePEs:=PLIST[j]-JLIST[i].PEs
              break; // process next job
            else
              continue; // check for the next node  else
          continue; //go for next node
        continue; //go for next job
    2.4 add SCHEDULE_LIST to the initial population POPU_LIST.
    2.5  generateNextGenerationPopulation()
        2.5.1 for each i in (population_size-size (POPU_LIST)
              Generate random sequence chromosome and add to
        POPU_LIST)
        2.5.2 Compute Fitness function based on equation (7) and  Select
        best chromosome.
        2.5.3 if (termination) return best chromosome
        2.5.4 doSelection()
              Select best 'crossover_rate%' chromosomes based on
        'Roulette wheel selection policy'
              Do crossover()
              Do Mutation()
              Add to POPU_LIST
        2.5.2 repeat thru 2.5.1
3. Generate SCH_LIST from best chromosome from step 2
    3.1  for each element in SCH_LIST do
            3.1.1 Notify user
            3.1.2 Notify provider.
4. Repeat thru step 1 at 'the next scheduling' time.
```

One limitation to the problem formulation is that if the number of resources requested is more than the number of resources available for the broker, then the problem results in an infeasible solution. Therefore, we propose addition of dummy nodes with infinite capacity and having more cost than any of the nodes available on the Grid. This would enable the algorithm to converge and assign some of the jobs to the dummy nodes. The fitness function forces the jobs with lower priority to be scheduled on the dummy nodes rather than the jobs with higher priority. Dummy node jobs are rolled over to the next schedule period, as they cannot be executed. Ideally priority is directly proportional to the cost, which means that the higher priority users would pay more incentives than the lower priority users. Here we assume a fair policy priority based on user QoS demands using equation (9.2) and also assume that the users would not under/over quote for their jobs. We now formulate the algorithm as given in Table-2:

IV. SIMULATION AND RESULTS

We used GridSim toolkit for simulating a Grid environment. We simulated 50 concurrent users, who submit jobs to the brokers. Grids having resources as small as 2 nodes to as large as 200 resources are generated. We simulated the Grid with heterogeneous resources having different MIPS ratings and each resource having different processing elements (PEs) in the range of 4 to 12. The cost of each resource is varied, between 4G$-5G$ with mean cost of 4.5G$. The cost of using a resource is kept the same for all the users. The jobs are also simulated with varying QoS requirements. Jobs with average PE requirements of 4 and having 10-50% variations[4] are considered and all the jobs are submitted within 20 seconds of simulation start time. As far as QoS parameters, viz., budget and deadline are concerned, three simulations are done for different deadlines which are listed as below:

Simulation-1: Tight deadline (estimated time + 50 seconds with 20% variation)

Simulation-2: Medium deadline (estimated time + 250 seconds with 20% variation)

Simulation-3: Relaxed deadline (estimated time + 500 seconds with 20% variation)

As far as budget is concerned, all the jobs are given relaxed budget constraints (i.e., twice the cost of average job execution time). The above jobs are submitted separately to the three broker implementations, viz.,

1. Personalized broker implementations as in Nimrod/G, Gridbus and GRUBER
2. Greedy implementation of Clearing House Broker and GRUBER
3. Heuristic GA algorithm for Clearing House Broker

The results for three different simulation scenarios are plotted in Fig 3.a to Fig. 6.

In the following comparison plots two result sets are shown. In the first set of results, i.e., Fig. 3a, 4a, and 5a, on X-axis grid resources are plotted while on Y-axis jobs completed

[4] Until or otherwise specified variation implies Gaussian distribution

(out of the 50 concurrent jobs submitted) are plotted. In the second set of results, i.e., Fig. 3b, 4b and 5b, aggregated revenue earned for the completed jobs are plotted on Y-axis.

Fig. 3a clearly indicates that inspite of having more resources, the personalized broker could not execute all the jobs. For example, when 200 grid resources were available the personalized broker could only complete 9 jobs out of the possible 42 jobs (8 jobs had very tight deadline, hence cannot be completed within the deadline provided), whereas in the case of Clearing House Broker all the jobs were finished when the nodes available were around 50 or more. The reason behind this is the contention, leading to live lock situation created due to lack of coordination in the case of personalized broker. It is noticed (thru Fig. 6) that the broker executes more SLAs before scheduling the jobs, hence resulting in resource under utilization. The situation is found to be the same whenever there were more jobs and is independent of the number of nodes available.

Another interesting observation that can be made is that the greedy approach and HGA algorithm perform equally or with marginal difference until the number of nodes are less than 25, but when the number of nodes increases then the HGA algorithm outperforms all the implementations by completing more jobs and also yeilding less aggregated cost for the same number of completed jobs. For example, in the first simulation when the Grid size was 50, the greedy implementation of Clearing House Broker could only complete 35 jobs, whereas the HGA implementation could complete all 42 jobs which explains the higher aggregated revenue (see Fig. 3b). When both of the Clearing House Brokers complete the same number of jobs the HGA implementation reduces the aggregated revenue. The personalized broker billed less revenue than the HGA implementation but the number of jobs completed is also significantly less compared to other brokers. Therefore, HGA implementation produces better optimised schedule in terms of cost per job. The same trend can be noticed in all the simulations as shown in Fig. 3b, 4b and 5b.

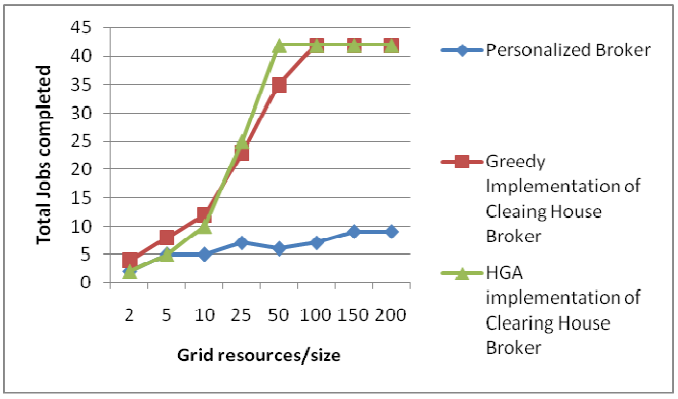

Fig. 3a: Jobs completed in 'tight deadline'

It can be noticed from Fig. 3a to 4a and 4a to 5a the curves get smoother because as the deadlines get relaxed, more jobs get completed with the same number of resources. For example, in case of tight deadline for 25 resources, the number of jobs completed is less than 25 for HGA implementation, while in the case of relaxed deadline it is more than 40. Hence the curves become more smoother and peak early compared to tighter deadlines (Fig 3a, 4a and 5a).

If the deadline for the jobs is relaxed, all the brokers including the personalised broker try to complete more tasks, but far less compared to the Clearing House Broker. This behaviour is understandable because when the deadline is too relaxed the jobs find time to finish the tasks within the deadline even after executing multiple SLAs. Both the implementations of Clearing House Brokers complete all the 50 jobs when number of resources available are around 50 or more.

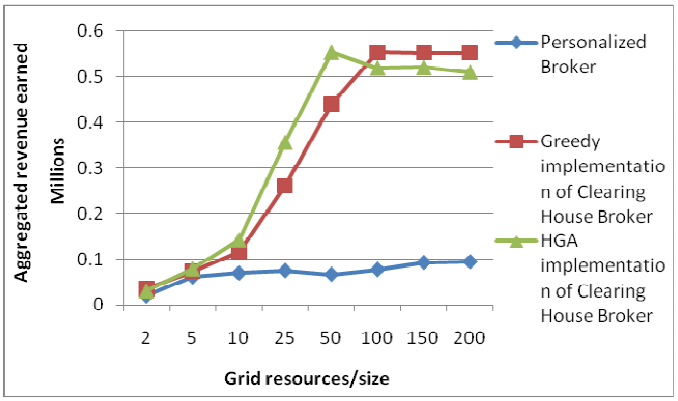

Fig. 3b: Aggregated revenue billed for the jobs completed in 'tight deadline'

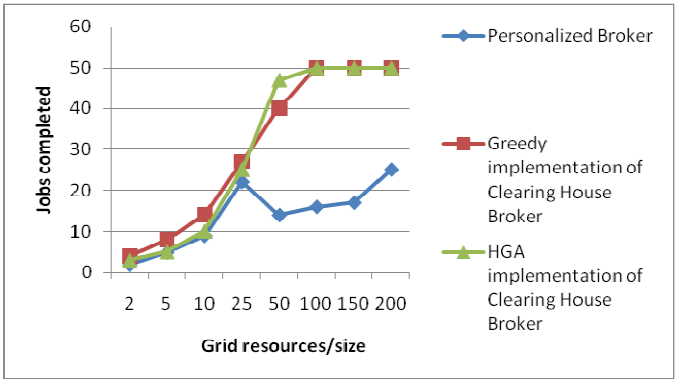

Fig.4a: Jobs completed in 'medium deadline'

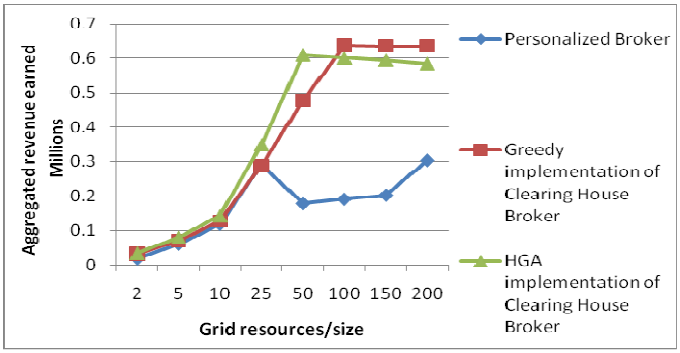

Fig.4b: Aggregated revenue earned in 'medium deadline'

It can also be noticed that the aggregated revenue for completing all the jobs in the case of tighter deadline is just above 500K G$, while it is about 600K G$ in the case of medium/relaxed deadline. The reason for this is that in the tight deadline, maximum jobs completed are only 42, while in

medium and relaxed deadlines all the 50 jobs were completed. Since the cost variations within the grid resources are not significant (i.e., 4.5 G$ with ± 0.5 G$) only 10% cost benefit was noticed. We believe it would be much more if the variations are higher.

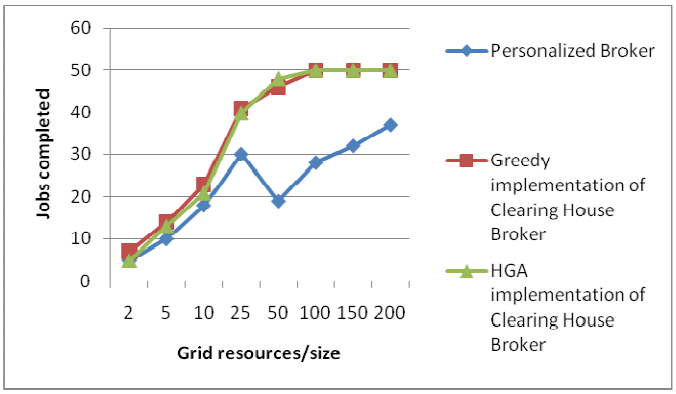

Fig.5a: Jobs completed in 'relaxed deadline'

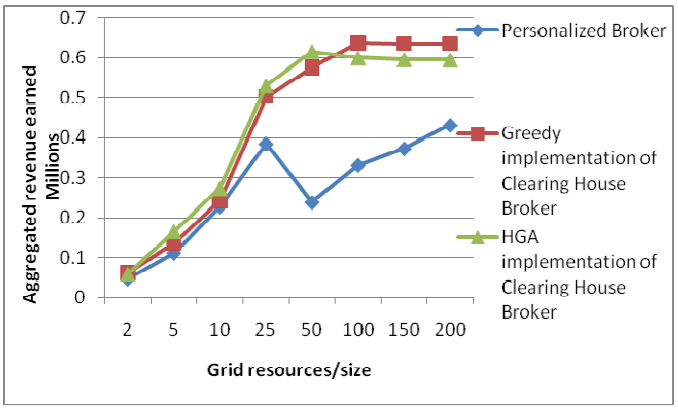

Fig.5b: Aggregated revenue earned in the 'relaxed deadline'

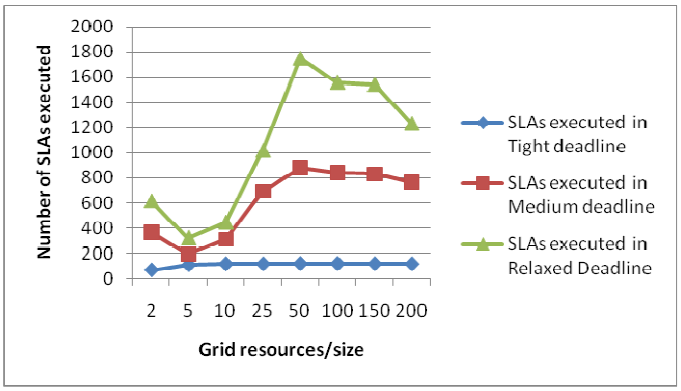

Fig.6 SLAs executed in different simulations

Next simulation study presents the advantages of using heuristics in the proposed HGA algorithm. As the conventional GA was taking more number of iterations for larger problem sizes, we reduced the problem size to 25 and varied the grid size from 15 to 50. In order to ensure that the solution is not trapped in the local minima, we also increased the iterations to a maximum of 10000 with convergence criterion of 200 successive generations. The results are shown in Table 2 and Fig. 7, which indicate that the addition of heuristics not only reduced the number of iterations but also reduced the aggregate revenue billed. In some cases, HGA algorithm completed more jobs compared to the conventional GA algorithm, which again explains the aggregated revenue being higher than the conventional GA in the corresponding cases. This result also indicates that despite taking more iterations the conventional GA could not find the global/better optimum.

The conventional GA resulted in better result, when the Grid size was 15 and 25. The reason for this was investigated and found to be that the conventional GA could not converge to a solution for 6 schedule periods i.e., from 150 to 400, during the seventh schedule interval (450), some of the jobs got eliminated because of the deadline constraints. The elimination of jobs resulted in better schedule in the case of conventional GA, whereas the HGA algorithm produced result in the first schedule period itself (150) without any elimination.

TABLE 2: COMPARISON OF CONVENTIONAL GA WITH HGA

GA						
#Providers	#Jobs	Sch Time	jobs Scheduled	Tasks Scheduled	iterations	Total Cost
15	25	450	22	115	1479	227599
20	25	150	25	129	2040	182015
25	22	450	22	113	4004	136124
35	25	150	25	129	6593	120984
40	25	150	25	129	2877	107756
50	25	150	25	129	2640	57458
HGA						
15	25	150	22	114	2559	228267
20	25	150	25	129	1354	178242
25	25	150	25	129	1565	159764
35	25	150	25	129	1331	119719
40	25	150	25	129	1629	101389
50	25	150	25	129	1545	52190

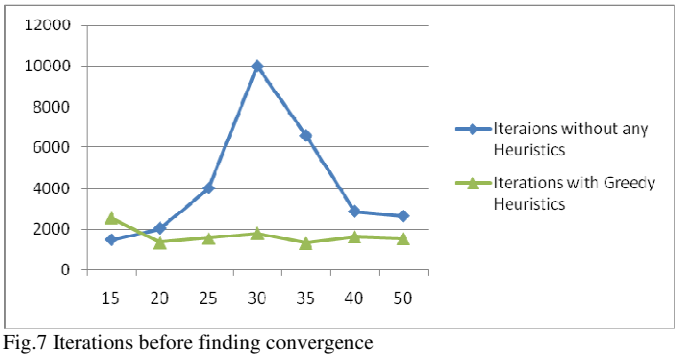

Fig.7 Iterations before finding convergence

V. CONCLUSION AND FUTURE WORK

Generalized Assignment Problem has been extensively studied by many researchers and various heuristics have been proposed. In this paper, the problem of scheduling concurrent users/jobs in the Grid is modelled as a GAP. A fairness index based on deadline and budget is used to do a greedy assignment which provides the initial solution for the Genetic Algorithm. Then the combined optimal schedule for concurrent users is obtained using standard operations of GA. The heuristic initial solution has reduced the number of iterations significantly, which otherwise would take longer time for convergence.

The key aspects this paper addresses are twofold. First, by proposing a Clearing House Broker which addresses the drawbacks arising in negotiations in the case of personalized Grid brokers when trying to optimize independently and second, by suggesting a Heuristic GA based algorithm for Clearing House Broker. The scheduling focuses on achieving better cost payoff for resource providers by maximizing the resource utilization, thereby resulting in more incentives to the Grid resource providers, and also minimizes the overall cost for the combined users by meeting the user level QoS constraints, i.e., deadline and budget. The fairness index, which is used to assign priority for the jobs, assigned higher priority to the users/jobs approaching the deadline. The heuristics have not only reduced the iterations considerably but also resulted in finding a better resource mapping.

HGA algorithm successfully acheived its objectives with encouraging processing speeds. Each iteration on a normal Pentium Centrino laptop for a 25 user test took 20 milliseconds, thus the entire scheduling for 25 concurrent users finished in less than 10 seconds including IO operations, which is very encouraging.

Our future work in this area is to provide multiple initial solutions from Greedy and Linear Programming/Integer Programming solutions, and also to add better heuristics for new offspring breeding. This we believe would not only result in better optimization but also further reduce the iterations. The other direction of work is to add more dynamics into the market model, where the Grid owners from 'time- to-time' announce incentives in terms of reducing price for certain/specific users or brokers. Brokers also act as profit agencies like an 'airline ticketing agents'. The incentives could be a function of time, user and broker. These would result in different architecture, protocols and scheduling algorithms.

ACKNOWLEDGEMENTS

This research is partially supported by the Australian Department of Education, Science and Training awards: Endeavour Research Fellowship and International Science Linkage. The first author is visiting the University of Melbourne on deputation from Dept. of Space, Govt. of India.

REFERENCES

[1] I. Foster, C. Kesselman and S. Tuecke, "The anatomy of the Grid:Enabling scalable virtual organizations," *International Journal of SupercomputerApplications,* vol. 15, no. 3, pp. 200–222, 2001.

[2] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the Grid: An open Grid services architecture for distributed systems integration", *In Open Grid Service Infrastructure WG, Global Grid Forum,* 2002.

[3] Amazon Elastic Compute Cloud (Amazon EC2) http://www.amazon.com/gp/browse.html?node=201590011

[4] R. Buyya, D. Abramson, J. Giddy and H. Stockinger. "Economic models for resource management and scheduling in grid computing. Special Issue on Grid computing Environment", *The Journal of concurrency and Computation:Practice and Experience (CCPE),* Volume 14, Issue13-15, Wiley Press, 2002.

[5] S. Venugopal, R. Buyya and L. Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", *Concurrency and Computation: Practice and Experience,* Volume 18, Issue 6, Pages: 685-699, Wiley Press, New York, USA, May 2006.

[6] D. Abramson, R. Buyya, and J. Giddy. "A computational economy for grid computing and its implementation in the Nimrod-G resource broker", *Future Generation Computer Systems (FGCS) Journal,* Volume 18, Issue 8 Pages: 1061-1074, Elsevier Science, The Netherlands, October, 2002.

[7] R. Ranjan, A. Harwood and R. Buyya, "SLA-Based Coordinated Superscheduling Scheme for Computational Grids", *In Proceedings of the 8th IEEE International Conference on Cluster Computing* (Cluster 2006), Sept. 27-30, 2006, Barcelona, Spain.

[8] C. Dumitrescu and I. Foster, "Usage Policy-based CPU Sharing in Virtual Organizations", *In 5th International Workshop in Grid Computing, 2004,* Pittsburg, PA.

[9] Condor Project, Condor-G, www.cs.wisc.edu/condor/2002.

[10] C. Dumitrescu and I. Foster, "GRUBER: A Grid Resource SLA Broker", *In Euro-Par, Portugal*, September 2005.

[11] C. Dumitrescu, I. Raicu and I. Foster, "DI-GRUBER: A Distributed Approach to Grid Resource Brokering", *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC '05*), Seattle, USA.

[12] P. K. Konugurthi, A. Agarwal and R. Krishnan, "A Fuzzy based Resource Management Framework for High Throughput Clusters", *In Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid,* 555 – 562pp, Chicago, USA.

[13] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 14, Issue 13-15, Pages: 1175-1220, Wiley Press, USA, November - December 2002.

[14] M. Siddiqui, A. Villazón and T. Fahringer, "Grid allocation and reservation---Grid capacity planning with negotiation-based advance reservation for optimized QoS", *In Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, November 11-17, 2006, Tampa, Florida

[15] D. Ouelhadj and J. Garibaldi and J. MacLaren and R. Sakellariou and K. Krishnakumar and A. Meisels, "A Multi-agent Infrastructure and a Service Level agreement Negotiation Protocol for Robust Scheduling in Grid Computing Advances in Grid Computing", *EGC 2005, European Grid Conference,* Amsterdam, The Netherlands, February 14-16, 2005: 651-660

[16] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. A. Huberman. "Tycoon: An Implementation of a Distributed Market-Based Resource Allocation System", Technical Report, HP Labs, USA, Dec. 2004.

[17] P. C. Chu, J. E. Beasley, "A genetic algorithm for the generalised assignment problem", *Computers and Operations Research,* v.24 n.1, p.17-23, Jan. 1997.

[18] S. Martello and P. Toth, "An alogorithm for Generalized Assignment Problem", *Operational Research,* 81:589-603, 1981.

[19] Harald Feltl , Günther R. Raidl, "An improved hybrid genetic algorithm for the generalized assignment problem", *Proceedings of the 2004 ACM symposium on Applied computing,* March 2004, Nicosia, Cyprus.

[20] S. Chapin, J. Karpovich, and A. Grimshaw. "The legion resource management system", *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing,* San Juan, Puerto Rico, 16 April, Springer:Berlin, 1999.

[21] H. Casanova and J. Dongara. Netsolve: A Network server solving computational science problem. *International Journal of Supercomputing Applications and High Perfomance Computing*;11(3); Pages:212-223, 1997.

[22] J. Litzkow, M. Livny, and M.W. Mukta. Condor- A hunter of idle workstations. IEEE, 1988.

[23] B. Bode, D. Halstead, R. Kendall, and D. Jackson, "PBS: The portable batch scheduler and the maui scheduler on linux clusters", *In Proceedings of the 4th Linux Showcase and Conference*, Atlanta, GA, USENIX Press, Berkley,CA, October, 2000.

[24] W. Gentzsh. Sun grid engine: Towards creating a compute power grid. *In Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid 2001), Brisbane, Australia.

[25] I. Foster et al., "The Grid2003 Production Grid: Principles and Practice", *In 13th International Sysmposium on High Performance Distributed Computing*, HPDC 2004: 236-245

[26] D. Goldberg, Genetic Algorithms in Search, *Optimization, and Machine Learning*, Addison-Wesley,1989.