International Conference on Advanced Computer Technology (ICACT) 2011
Proceedings published by International Journal of Computer Applications® (IJCA)

Mapping of Independent Task Classes onto GRIDSIM

G.Malathy
Assistant Professor CSE (Research Scholar),
Velalar College of Engineering and Technology,
Tamilnadu, India.

Dr.A.Saradha
Assiosiate Professor - CSE,
Institute of Road and Transport Technology,
Tamilnadu, India

ABSTRACT:

The motivation of Grid computing is to aggregate the power of widely distributed resources, and provide non-trivial services to users. To achieve this goal, an efficient Grid scheduling System is an essential part of the Grid. Rather than covering the whole Grid scheduling area, this survey provides a review of the subject mainly from the perspective of scheduling algorithms. In this review, the challenges for Grid scheduling are identified. First, the architecture of components involved in scheduling is briefly introduced to provide an intuitive image of the Grid scheduling process. Then various Grid scheduling algorithms are discussed from different points of view, such as static vs. dynamic policies, objective functions, applications models, adaptation, constraints, strategies dealing with dynamic behavior of resources, and so on. Thus, in this paper, the following definition for the term Grid adopted: "A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements". To facilitate the discussion, the following frequently used terms are defined: A task is an atomic unit to be scheduled by the scheduler and assigned to a resource. The properties of a task are parameters like CPU/memory requirement, deadline, priority, etc. A job (or metatask, or application) is a set of atomic tasks that will be carried out on a set of resources. Jobs can have a recursive structure, meaning that jobs are composed of sub-jobs and/or tasks, and sub-jobs can themselves be decomposed further into atomic tasks. In this paper, the term job, application and metatask are interchangeable. A resource is something that is required to carry out an operation, for example: a processor for data processing, a data storage device, or a network link for data transporting. A site (or node) is an autonomous entity composed of one or multiple resources. A task scheduling is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains.

Keywords: OLB, MET, Min_min, GA, Tabu, A*

INTRODUCTION

1. Mapping Independent Tasks

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements. HC environments are well suited to meet the computational demands of large, diverse groups of tasks. The problem of optimally mapping (defined as matching and scheduling)these tasks onto the machines of a distributed HC environment has been shown, in general, to be NP-complete, requiring the development of heuristic techniques. Selecting the best heuristic to use in a given environment, however, remains a difficult problem, because comparisons are often clouded by different underlying assumptions in the original study of each heuristic.Therefore, a collection of 11 heuristics from the literature has been selected, adapted, implemented, and analyzed under one set of common assumptions.It is assumed that the heuristics derive a mapping statically (i.e., off-line). It is also assumed that a metatask (i.e., a set of independent, noncommunicating tasks) is being mapped and that the goal is to minimize the total execution time of the metatask. The 11 heuristics examined are Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min_min, Max_min, Duplex, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A*. This study provides one even basis for comparison and insights into circumstances where one technique will out-perform another. The evaluation procedure is specified, the heuristics are defined, and then comparison results are discussed. It is shown that for the cases studied here, the relatively simple Min_min heuristic performs well in comparison to the other techniques. The matching of tasks to machines and scheduling the execution order of these tasks is referred to as mapping. For these studies, let a metatask be defined as a collection of independent tasks with no intertask data dependencies. The mapping of the metatasks is being performed statically (i.e., off-line, or in a predictive manner). The goal of this mapping is to minimize the total execution time of the metatask. Composed of independent tasks occur in many situations. For example, all of the jobs submitted to a supercomputer center by different users would constitute a metatask. Another example of a metatask would be a group of image processing applications all operating on different images.

Static mapping is utilized in many different types of analyses and environments.The most common use of static mapping is for predictive analyses (e.g., to plan the work for the next day and_or to meet a deadline). For example, assume a NASA center knows it will have a 2-hour communication window with a probe tomorrow. In those 2 hours, NASA center will have to analyze the data the probe sends back and determine if the probe needs to be adjusted before communications blackout. Therefore, the NASA center will want to plan the most efficient way to handle the data a priori and determine if the deadline can be met. Another use of static map-ping is for ``what if " simulation studies. For example, a system administrator may need to justify the benefits of purchasing another machine for an HC suite. Static mapping is also used for post-mortem analyses. For example, a static mapper can be used ex post facto to evaluate how well an on-line (i.e., dynamic) mapper is performing. Future high-powered computational grids [6] will also be able to utilize static mapping techniques to distribute resources and computational power. The wide applicability of static mapping makes it an important area for ongoing research.It is also assumed that each machine executes a single task at a time (i.e., no mul-titasking), in the order in which the tasks are assigned. The size of the metatask (i.e., the number of tasks to execute), the number of machines in the HC suite, are static and known beforehand. This study provides one even basis for comparison and insights into circum-stances where one mapping technique will out-perform another. The evaluation procedure is specified, the heuristics are defined, and then comparison results are shown.

2. Simulation Model

The 11 static mapping heuristics were evaluated using simulated execution times for an HC environment. Because these are static heuristics, it is assumed that an accurate estimate of the expected execution time for each task on each machine is known prior to execution and contained within a ETC (expected time to compute) matrix. The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems [9] Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [2]. One row of the ETC matrix contains the estimated execution times for a given task on each machine. Similarly, one column of the ETC matrix consists of the estimated execution times of a given machine for each task in the metatask. Thus, for an arbitrary task ti and an arbitrary machine mj , ETC(ti , mj) is the estimated execution time of ti on mj .The ETC(ti , mj) entry could be assumed to include the time to move the executables and data associated with task ti from their known source to machine mj . For cases when it is impossible to execute task ti on machine mj (e.g., if specialized hardware is needed), the value of ETC(ti , mj) is set to infinity. For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments. The ETC matrices used were generated using the following method. Initially, a {_1 baseline column vector, B, of floating point values is created. Let ,b be the upper bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, xi b # [1, ,b), and letting B(i)=xi b for 0_i<{. Next, the rows of the ETC matrix are constructed. Each element ETC(ti , mj) in row i of the ETC matrix is created by taking the baseline value, B(i), and multiplying it by a uniform random number, xi, jr , which has an upper bound of ,r . This new random number, xi, jr # [1, ,r), is called a row multiplier. One row requires + different row multipliers, 0_j<+. Each row i of the ETC matrix can then be described as ETC(ti , mj)=B(i)_xi, jr , for 0_j<+. (The baseline column itself does not appear in the final ETC matrix.) This process is repeated for each row until the {_+ ETC matrix is full. Therefore, any given value in the ETC matrix is within the range [1, ,b_,r) [9].To evaluate the heuristics for different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods from [4]. The amount of variance among the execution times of tasks in the metatask for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the upper bound of the random numbers within the baseline column vector.

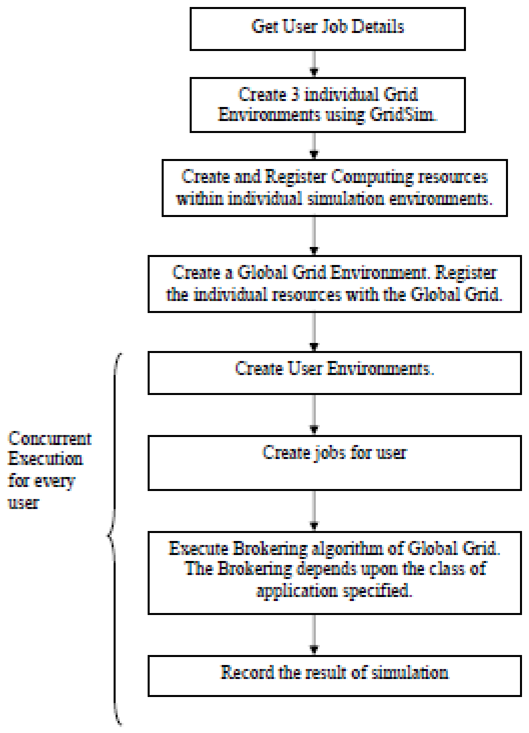

Get User Job Details
Create 3 individual Grid Environments using GridSim.
Create and Register Computing resources within individual simulation environments.
Create a Global Grid Environment. Register the individual resources with the Global Grid.
Create User Environments.
Create jobs for user
Execute Brokering algorithm of Global Grid. The Brokering depends upon the class of application specified.
Record the result of simulation
Concurrent Execution for every user

Fig1. Flow Diagram of Global Grid Simulation

3. Current Model

For a DC based IR-drop analysis, an estimated peak DC current is often used when the circuit design is still incomplete. In [8], this simple DC peak current model is extended to an AC peak current model. Based on the estimated average chip current and an average/peak current factor, the current profile for a single clock cycle is constructed. This current profile is then evenly distributed across all power grid connections by dividing the chip-level current by the number of power grid connections at each time point in the clock cycle. This method results in a low, wide current profile for each gate, and all gate currents are perfectly synchronized. In an actual circuit, however, each gate produces a much narrower and taller current pulse when switched, For a 500MHz design, a typical gate produces a current pulse 50ps wide, only a fraction of the 2ns pulse width that will be generated in the above approach. Also, gate currents are not synchronized, and switch at various times during the clock cycle. Decoupling capacitance is much more effective in supplying the needed charge for many short, asynchronous current spikes than for a set of slow, synchronized current pulses. Thus, this simplified current model can produce a significant error in power grid analysis. An accurate model can be obtained by a fast transistor level simulation of the circuit blocks using chip-level vectors, monitoring the individual gate currents. Each gate can then be modeled by a current source matching its observed profile. However, this procedure is prohibitively expensive for very large processors. More importantly, most power grid design is performed before the circuit design is completed and transistor level simulation can be performed. Once all circuit designs are completed, only limited, small modifications can be made to the power grid. Hence, there is a critical need for a good early current model a model that not only matches the total current profile at the chip-level, but matches also the gate-level current profiles and mimics their random switching behavior.

4. Grid Resistance Models

The parasitic inductances of the power grid in a package must be extracted and modeled to study their effect on the power grid voltage and resonance behavior. The major sources of parasitic inductances in the package are the power planes, ball arrays/bond wires, and package vias. A number of inductance extraction tools and techniques [10] are available to extract the model of the package power network. Defining ports for the package network at each supply and ground input to the package and at connection points to the die, the extracted model can then be reduced to a compact n-port model. The n-port model is a completely dense matrix which represents the self and mutual inductances between port connections. The parasitic resistances of the on-chip power and ground grid are either extracted from the layout with a commercial extraction tool or, in early phases, are determined directly from wire sizes using sheet resistance. The n-port model of the package and the RC elements of the on-chip power model are then combined with the extracted decoupling capacitance models and current source models. Finally, the combined power grid network is simulated using the techniques

5. Heuristic Descriptions

The definitions of the 11 static metatask mapping heuristics are provided below. First, some preliminary terms must be defined. Machine availability time, mat (mj), is the earliest time machine mj can complete the execution of all the tasks that have previously been assigned to it (based on the ETC entries for those tasks). The completion time for a new task ti on machine mj , ct(ti , mj), is the machine availability time for mj plus the execution time of task ti on machine mj , i.e., ct(ti , mj)=mat(mj)+ETC(ti , mj). The performance criterion used to compare the results of the heuristics is the maximum value of ct(ti , mj), for 0_i<{ and 0_j<+. The maxi-mum ct(ti , mj) value, over 0_i<{ and 0_j<+, is the metatask execution time, and is called the makespan [2]. Each heuristic is attempting to minimize the makespan, i.e., finish execution of the metatask as soon as possible. The descriptions below implicitly assume that the machine availability times are updated after each task is mapped. For heuristics where the tasks are considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used. Most of the heuristics discussed here had to be adapted for this problem domain. For many of the heuristics, there are control parameters values and control function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and information in the literature.

Simulated Annealing :

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. 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.

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

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

(1)

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

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

(2)

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

(3)

It is straightforward to prove that an annealing schedule for

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

(4)

Where bi > 0 is a constant parameter and k0 is a sufficiently large constant to satisfy (4), if the generation functions in (1)

is adopted.

OLB: Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the next machine that is expected to be available, regardless of the task's expected execution time on that machine [3]. The intuition behind OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity,but because OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans.

MET: In contrast to OLB, Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability [7]. The motivation behind MET is to give each task to its best machine. This can cause a severe load imbalance across machines. In general, this heuristic is obviously not applicable to HC environments characterized by consistent ETC matrices. MCT: Minimum Completion Time (MCT) assigns each task, in arbitrary order, to the machine with the minimum expected completion time for that task [3].This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The intuition behind MCT is to combine the benefits of OLB and MET, while avoiding the circumstances in which OLB and MET perform poorly.

Min_min: The Min_min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion times, M=[min0_j<+ (ct(ti , mj)), for each ti # U], is found. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine (hence the name Min_min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [13]. Min_min is based on the minimum completion time, as is MCT. However, Min_min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min_min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Let ti be the first task mapped by Min_min onto an empty system. The machine that finishes ti the earliest, say mj, is also the machine that executes ti the fastest. For every task that Min_min maps after ti , the Min_min heuristic changes the availability status of mj by the least possible amount for every assignment. Therefore, the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher for Min_min than for Max_min (defined next). The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest. Max_min: The Max_min heuristic is very similar to Min_min. The Max_min heuristic also begins with the set U of all unmapped tasks. Then, the set of mini-mum completion times, M, is found. Next, the task with the overall maximum completion time from M is selected and assigned to the corresponding machine (hence the name Max_min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [3].Intuitively, Maximum attempts to minimize the penalties incurred from performing tasks with longer execution times. Assume, for example, that the metatask being mapped has many tasks with very short execution times and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a Minmin mapping, where all of the shorter tasks would execute first, and then the longer running task would execute while several machines sit idle. Thus, in cases similar to this example, the Max_min heuristic may give a mapping with a more balanced load across machines and a better makespan.Duplex: The Duplex heuristic is literally a combination of the Min_min and Max_min heuristics. The Duplex heuristic performs both of the Min_min and Max_min heuristics and then uses the better solution [3]. Duplex can be per-formed to exploit the conditions in which either Min_min or Max_min performs well, with negligible overhead.

GA: Genetic Algorithms (GAs) are a technique used for searching large solution spaces [2]. The version of the heuristic used for this study was adapted from [4] for this particular problem domain. Figure 1 shows the steps in a general GA.The GA implemented here operates on a population of 200 chromosomes (possible mappings) for a given metatask. Each chromosome is a {_1 vector, where position i (0_i<{) represents task ti , and the entry in position i is the machine to which the task has been mapped. The initial population is generated using two methods :(a) 200 randomly generated chromosomes from a uniform distribution, or (b) one chromosome that is the Min_min solution (i.e., mapping for the metatask) and 199 random solutions. The latter method is called seeding the population with a Min_min chromosome. The GA actually executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the final solution. Each chromosome has a fitness value, which is the makespan that results from the matching of tasks to machines within that chromosome. After the generation of the initial population, all of the chromosomes in the population are evaluated based on their fitness value, with a smaller fitness value being a better mapping. Then, the main loop in Fig. 1 is entered and a rank-based roulette wheel scheme [11] is used for selection. This scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. Elitism, the property of guaranteeing the best solution remains in the population [3], was also implemented. The population size stays fixed at 200.Next, the crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks. Every chromosome is considered for crossover with a probability of 600. After crossover, the mutation operation is performed. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome, and randomly reassigns it to a new machine. Every chromosome is considered for mutation with a probability of 400. For both crossover and mutation, the random operations select values from a uniform distribution. Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when any one of three conditions are met: (a) 1000 total iterations, (b) no change in the elite chromosome for 150 iterations, or (c) all chromosomes converge to the same mapping. Until the stopping criterium is met, the loop repeats, beginning with the selection step. The stopping criterium that usually occurred in testing was no change in the elite chromosome in 150 iterations. A: Simulated Annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each metatask at a time. This solution uses the same representation as the chromosome for the GA. The initial implementation of SA was evaluated and then modified and refined to give a better final version. Both the initial and final implementations are described below. SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space [10].This probability is based on a system temperature that decreases for each iteration. As the system temperature ``cools,'' it is more difficult for poorer solutions to be accepted. The initial system temperature is the makespan of the initial (random) mapping. The initial SA procedure implemented here is as follows. The first mapping is generated from a uniform random distribution. The mapping is mutated in the same manner as the GA, and the new makespan is evaluated. The decision algorithm for accepting or rejecting the new mapping is based on [10]. If the new makespan is better, the new mapping replaces the old one. If the new makespan worse (larger), a uniform random number z # [0, 1) is selected. Then, z is compared with y, where y=11+e (old makespan-new makespan temperature). If z>y the new (poorer) mapping is accepted; otherwise it is rejected, and the old mapping is kept. GSA: The Genetic Simulated Annealing (GSA) heuristic is a combination of the GA and SA techniques . In general, GSA follows procedures similar tothe GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature and a simplified SA decision process for accepting or rejecting a new chromosome. Specifically, the initial system temperature was set to the average makespan of the initial population and reduced to 900 of its current value for each iteration. Whenever a mutation or crossover occurs, the new chromosome is compared with the corresponding original chromosome. If the new makespan is less than the original makespan plus the system temperature, then the new chromosome is accepted [7]. Otherwise, the original chromosome survives to the next iteration. Therefore, as the system temperature decreases, it is again more difficult for poorer solutions to be accepted. The two stopping criteria used were either (a) no change in the elite chromosome in 150 iterations or (b) 1000 total iterations. The most common stopping criteria was no change in the elite chromosome in 150 iterations.

Tabu: Tabu search is a solution space search that keeps track of the regionsof the solution space which have already been searched so as not to repeat a search near these areas [12]. A solution (mapping) uses the same representation as a chromosome in the GA approach.The implementation of Tabu search used here begins with a random mapping as the initial solution, generated from a uniform distribution. To manipulate the current solution and move through the solution space, a short hop is performed. The intuitive purpose of a short hop is to find the nearest local minimum solution within the solution space. The basic procedure for performing a short hop is to consider, for each possible pair of tasks, each possible pair of machine assignments, while the other {&2 assignments are unchanged. This is done for every possible

pair of tasks. The Pseudocode for the short hop procedure is given in Fig. 2.Let the tasks in the pair under consideration be denoted ti and tj in Fig. 2. (The machine assignments for the other {&2 tasks are held fixed.) The machines to which tasks ti and tj are remapped are mi and mj, respectively. For each possible pair of tasks, each possible pair of machine assignments is considered. Lines 1 through 4 set the boundary values of the different loops. Line 6 or 8 is where each new solution (mapping) is evaluated, and line 9 is where the new solution is considered for acceptance. Each of these new solutions is a short hop. If the new makespan is an improvement, the new solution is saved, replacing the current solution. (This is defined as a successful short hop.) When ti and tj represent the same task (ti=tj), a special case occurs (line 5). In these situations, all machines for that one task are considered.

A*: The final heuristic in the comparison study is the A* heuristic. A* has been applied to many other task allocation problems [9]. The technique used here is similar to [9]. A* is a search technique based on a +-ary tree, beginning at a root node that is a null solution. As the tree grows, nodes represent partial mappings (a subset of tasks is assigned to machines). The partial mapping (solution) of a child node has one more task mapped than the parent node. Call this additional task ta . Each parent node generates + children, one for each possible mapping of ta . After a parent node has done this, the parent node becomes inactive. To keep execution time of the heuristic tractable, there is a pruning process to limit the maximum number of active nodes in the tree at any one time (in this study, to 1024). Each node, n, has a cost function, f (n), associated with it. The cost function is an estimated lower bound on the makespan of the best solution that includes the partial solution represented by node n. Let g(n) represent the makespan of the task machine assignments in the partial solution of node n, i.e., g(n) is the maximum of the machine availability times (max0_ j<+ mat(mj)) based on the set of tasks that have been mapped to machines in node n's partial solution. Let h(n) be a lower-bound estimate on the difference between the makespan of node n's partial solution and the makespan for the best complete solution that includes node n's partial solution. Then, the cost function for node n is computed as f (n) =g(n)+h(n). (2) Therefore, f (n) represents the makespan of the partial solution of node n plus a lower-bound estimate of the time to execute the rest of the (unmapped) tasks in the metatask (the set U).The function h(n) is defined in terms of two functions, h1 (n) and h2 (n), which are two different approaches to deriving a lower-bound estimate. Recall that M = [min0_ j<+ (ct (ti, mj)), for each ti # U]. For node n let mmct (n) be the overall maximum element of M (i.e., the maximum minimum completion time). Intuitively, mmct (n) represents the best possible metatask makespan by making the typically unrealistic assumption that each task in U can be assigned to the machine indicated in M without conflict.

6. Implementation:

A. User Interface:

The user interface was implemented using the Java AWT and Java Swing API. The user can enter the following information for the jobs through the interface:

1) Username – Name of the user submitting the job.

2) Baud Rate – The network communication speed between the user and the resource.

3) Maximum Simulation Time- Total time for which simulation has to be executed.

4) Successive Experiment Delay- Time delay between successive experiments.

5) Scheduling Strategy- No Optimization, Optimize Cost, Optimize Cost and Time, Optimize Cost Plus, Optimize time.

6) Job Type-Loosely Coupled, Tightly Coupled, Workflow, Distributed-Pipelined

7) Gridlet information- Parameters for creating gridlets, gridlet sizes, gridlet length, gridlet file size, gridlet output size.

8) Number of Gridlet- Number of jobs submitted by the user.

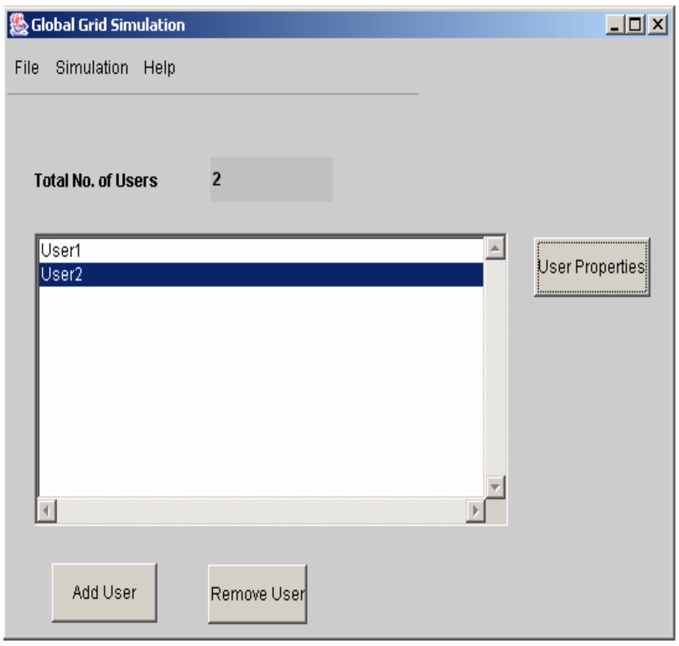
9) Factor-Based or Value-Based- The Budget and Deadline constraints are either value- based or factor-based.

10) Budget and Deadline- Economic constraint values for application scheduling.

The main screen consists of the Add User, Remove User and User Properties buttons. The Add User button is used to add multiple users to the Global Grid. The Remove User button is used to remove users from the Global Grid. The User Properties button is used to view and modify the job information submitted by each user.

7. Grid Environments:

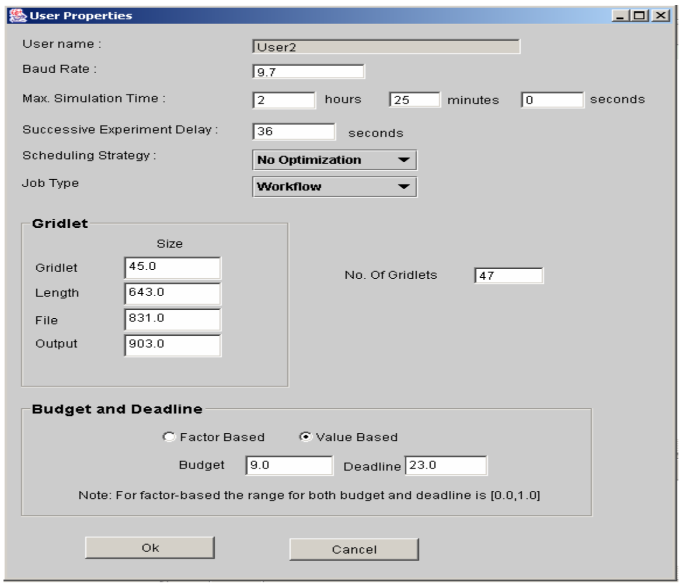
There are three local Grid environments GridEnv1, GridEnv2, GridEnv3 simulated using GridSim. Each GridSim environment has individual resources registered with the Grid Information Service (GIS). One of the most important implementation details of GridSim is that it can have only one GIS even though there are multiple GridSim Environments. All the resource has to be registered with this GIS. In our implementation of the 3 local environments, each one has 3



Total No. of Users 2

User1
User2

Fig2 Main user window



User name : User2
Baud Rate : 9.7
Max. Simulation Time : 2 hours 25 minutes 0 seconds
Successive Experiment Delay : 36 seconds
Scheduling Strategy : No Optimization
Job Type Workflow

Gridlet

	Size
Gridlet	45.0
Length	643.0
File	831.0
Output	903.0

No. Of Gridlets 47

Budget and Deadline

Factor Based [✓ Value Based]

Budget 9.0 Deadline 23.0

Note: For factor-based the range for both budget and deadline is [0.0,1.0]

Fig3 User Properties window

resources. Each resource is a cluster of machines each having 1 or more Processing Elements (PEs) with fixed values of MIPS rating for each machine.

GridSimEnv1: Env1_Resource_0, Env1_Resource_1, Env1_Resource_2.

GridSimEnv2: Env2_Resource_0, Env2_Resource_1, Env2_Resource_2.

GridSimEnv3: Env3_Resource_0, Env3_Resource_1, Env3_Resource_2.

The Global Grid virtually encompasses the three local Grids and has access to all the resources through the common GIS. Each user created in the Global Grid spawns a new execution thread. The job (gridlet) creation for each user is performed concurrently by the thread. The simulation is initiated with the method call to "StartGridSimulation" in the GridSim package. The jobs are then concurrently brokered using the customized broker algorithm. The default broker algorithm of GridSim is not initiated.

To summarize the findings of this section, for consistent ETC matrices, GA gave the best results, Minimum the second best, and MET gave the worst. When the ETC matrices were inconsistent, OLB provided the poorest mappings while the mappings from GA and A* performed the best. For the partially-consistent cases, A still gave the best results, followed closely by Minimum and A*, while MET had the slowest. All results were for metatasks with {=512 tasks executing on +=16 machines, averaged over 100 different trials. For the situations considered in this study, the relative performance of the map-ping heuristics varied based on the characteristics of the HC environments. The GA always gave the best performance. If mapper execution time is also considered,

Minimum gave excellent performance (within 120 of the best) and had a very small execution time. The confidence intervals derived from the mappings for these two heuristics were among the best (smallest) of any of the 11 heuristics. GA was always within \90 of its mean and Minimum was always within \130 of the mean for all 12 cases. This means that, for any future metatask to be mapped, these two heuristics will generate a good make span (within the confidence interval) 950 of the time.

CONCLUSION

Static mapping is useful in predictive analyses, impact studies, and post-mortem analyses. The goal of this study was to provide a basis for comparison and insights into circumstances when one static technique will out-perform another for 11 different heuristics. The characteristics of the ETC matrices used as input for the heuristics and the methods used to generate them were specified. The implementation of a collection of 11 heuristics from the literature was described. The results of the mapping heuristics were discussed, revealing the best heuristics to use in certain scenarios. For the situations, implementations, and parameter values used here, GA consistently gave the best results. The average performance of the relatively simple Minimum heuristic was always within 120 of the GA heuristic.

REFERENCES

[1] Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure Second edition, Morgan-Kaufman, 2004.

[2] M. Baker, R. Buyya, D. Laforenza: Grids and Grid technologies for wide-area distributed computing, Sotware – Practice and Experience, 2002, John Wiley & Sons, Ltd.

[3] M. Baker, R. Buyya and D. Laforenza, Grids and Grid Technologies for Wide-area Distributed Computing, in J. of Software-Practice & Experience, Vol. 32, No.15, pp: 1437-1466, December 2002.

[4] F. Berman, R. Wolski, S. Figueria, J. Schopf and G. Shao, Application-Level Scheduling on Distributed Heterogeneous Networks, in Proc. of the 1996 ACM/IEEE

Conference on Supercomputing, Article No: 39, Pittsburgh, Pennsylvania USA, November 1996.

[5] F. Berman, High-Performance Schedulers, chapter in The Grid: Blueprint for a Future Computing Infrastructure, edited by I. Foster and C. Kesselman, Morgan Kaufmann Publishers, 1998.

[6] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D. Zagorodnov, Adaptive Computing on the Grid Using AppLeS, in IEEE Trans. On Parallel and Distributed Systems (TPDS), Vol.14, No.4, pp.369--382, 2003.

[7] F. Berman, High-Performance Schedulers, chapter in The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1998.

[8] J. Bester, I. Foster, C. Kesselman, J. Tedesco and S. Tuecke, GASS: A Data Movement and Access Service for Wide Area Computing Systems, in Proc. of the 6th Workshop on I/O in Parallel and Distributed Systems, pp.: 78-88, Atlanta, Georgia USA, May 1999.

[9] J Blythe, S Jain, E Deelman, Y Gil, K Vahi and A Mandal,K Kennedy, Task Scheduling Strategies for Workflow-based Applications in Grids, in Proc. Of International Symposium on Cluster Computing and Grid (CCGrid'05), pp.759-767,

Cardiff, UK, May 2005.

[10] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, in J. of Parallel and Distributed Computing, vol.61, No. 6, pp. 810-837, 2001.

[11] R. Buyya, J. Giddy, and D. Abramson, An Evaluation of Economy-based Resource

Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications, in Proc. of the 2nd International Workshop on Active Middleware Services (AMS 2000), pp. 221-230, , Pittsburgh, USA, August 2000.

[12] R. Buyya and D. Abramson and J. Giddy and H. Stockinger, Economic Models for Resource Management and Scheduling in Grid Computing, in J. of Concurrency and Computation: Practice and Experience, Volume 14, Issue.13-15, pp. 1507-1542, Wiley Press, December 2002.

[13] R. Buyya, D. Abramson, and S. Venugopal, The Grid Economy, in Proc. of the IEEE, Vol. 93, No. 3, pp. 698-714, IEEE Press, New York, USA, March 2005.

[14] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd, Grid Flow: Workflow Management for Grid Computing, in Proc. of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid'03), pp.198-205, Tokyo, Japan, May 2003.

Author Profile:

Malathy.G received B.E Degree in Computer Science and Engineering in 1999 from Bharathiyar University, and the M.E degree in Computer Science and Engineering from Anna University, Cheenai in 2007.She is working as an Assistant Professor(SG) in the department of CSE at Velalar College of Engineering and Technology,Erode.Her area of interest includes Database Systems. Computer Networks, Grid Computing and Compiler Design.

Dr.Saradha Arumugam received B.E degree in Electronics and Communication Engineering from Madurai Kamaraj University, Tamilnadu, India in 1988 and M.E degree in Computer Science Engineering from PSG College of Technology, Coimbatore, Tamilnadu, India in 1993 and Ph.D degree in Computer Science Engineering from Government College of Technology, Coimbatore, Tamilnadu, India in 2005.

She is an Associate professor of Computer Science Engineering at Institute of Road and Transport Technology, Tamilnadu, India. Her Current research interests include Pattern recognition, Semantic web technology, Web mining, Information Security, Grid Computing and Bioinformatics.