International Journal of Computer Science and Applications, ©2009 Technomathematics Research Foundation
Vol. 6, No. 1, pp. 53 – 69

AN EFFICIENT APPROACH TO TASK SCHEDULING IN COMPUTATIONAL GRIDS

DR.G.SUDHA SADASIVAM

Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamilnadu 641004, India
sudhasadhasivam@yahoo.co.in

VIJI RAJENDRAN.V

Department of Computer Science and Engineering, NSS College of Engineering, Palghat, Kerala 678008, India
vijirajv@gmail.com

The Resource management in a grid environment is complex. Scheduling strategy plays an important role in the grid environment to schedule the user jobs, and dispatch them to appropriate grid resources. A good task scheduling method is essential to reduce the total time taken for job execution in the grid. In a grid environment, the jobs are processed at the grid resources in a fine-grained form that results in a low computation-communication ratio. So an efficient job grouping-based scheduling system is required to dynamically assemble the individual fine-grained jobs of an application into a group of jobs and then transmit these coarse-grained jobs to the grid resources. The objective of this paper is to develop a scheduling strategy using job groups that optimizes the utilization of processing capabilities of grid resource and reduces the total time taken to process user jobs.
Keywords: Scheduling; grid computing; particle swarm optimization; job grouping.

1. Introduction

A grid coordinates resources that are not subject to centralized control. The creation of large resources through pooling of individual resources from different locations enables the grid to provide powerful computing infra structure. In the grid environment the utility of the combined system is significantly greater than that of the sum of its parts. The term Grid is an analogy to a power grid that provides consistent, pervasive, dependable, transparent access to electricity irrespective of its source [1].

This paper proposes a novel job grouping method using Particle Swarm Optimization (PSO) to reduce the communication overhead and hence enhance the speed of completion of the processes and improve resource utilization. The proposed method improves the computation communication ratio.

The unorganized deployment of applications in the grid environment with a large number of fine-grain jobs increases the communication overhead and results in a low computation-communication ratio. Grid's dynamic nature complicates the planning of the job scheduling activity for minimizing the application processing time. Hence a scheduling strategy is essential to optimize the utilization of grid resource processing capabilities, and reduce total time taken to process user jobs. The grid scheduler should reduce the total transmission time of user jobs to/from the resources, and the overhead processing time of the jobs at the

resources. To achieve this, the jobs should be sent to the resources in a coarse-grained form. In other words, the scheduling strategy should group a number of user jobs (small jobs) together according to a particular grid resource's processing capabilities, and then send the grouped jobs to the resources in parallel.

The granularity time is also essential in the scheduling system to discover the total user jobs that can be processed in a given time. Relationship between the total number of jobs, processing requirements of those jobs, total number of available grid resources, processing capabilities of those resources, and the granularity time should be determined in order to minimize the job execution time, and optimize the utilization of the grid resources.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 describes the design of the scheduling system including the architecture of the scheduler and conventional gridlet grouping algorithm. Section 4 describes the evolutionary approach to job grouping and PSO based gridlet grouping algorithm with load balancing. Section 5 mentions the results and discussion. Finally, Section 6 gives the conclusion.

2. Related Work

In a cluster a set of nodes is placed in one location. A grid is composed of many clusters. In a grid environment, the jobs are processed at the grid resources in a fine-grained form. A study of scheduling heuristics was conducted by James, Hawick and Coddington [2]. Round-robin scheduling, minimal adaptive scheduling, continual adaptive scheduling and first-come-fist-served scheduling algorithms were discussed and compared for various job distributions. In some algorithms, jobs were grouped in equal numbers, while in other algorithms the nodes were made to synchronize after each round of execution [2]. In our proposed approach, the jobs are grouped according the ability of the remote nodes. There are no synchronization steps involved as the grouped jobs are dispatched when the remote resources become available. This eliminates the synchronization overhead.

Gerasoulis and Yang [3] in the context of Directed Acyclic Graph (DAG) scheduling in parallel computing environments, grouped jobs into clusters to reduce communication dependencies among them. The aim of this clustering is to reduce the inter-job communication and thus, decrease the time required for parallel execution. Cluster mapping heuristics [4] aim to maximize the number of jobs that can be executed in parallel on different processors.

As stated by Buyya, Date, et. al. [5], the need for a job grouping method became an imperative research area after the emergence of distributed analysis of brain activity data. The Magneto encephalography (MEG) helmet is used for recording information about brain activities. A 64-sensored MEG instrument produces 0.9 GB of data in an hour and the data is used to generate 7257600 analysis jobs which take about 102 days on a commodity computer. Global grids enable the partnering doctors to share the MEG instrument and allow the analysis jobs to be computed among the distributed computing resources. Large amount of computation power reduces the total time taken for completing the analysis jobs. The main issue is the expense caused from the overhead communication time. This necessitates grouping of jobs.

GridSim based simulations on grouping the grid jobs have been conducted at GRIDS laboratory, University of Melbourne [6] and at the Department of Computer System and Technology, University of Malaya [7]. In these simulations, jobs were merged based on each resource's Million Instructions Per Second (MIPS) and each job's Million Instructions (MI). For example, in order to utilize a resource with 500 MIPS for 3 seconds, jobs were merged into a single job file until the maximum MI of the file was 1500. Job MI, resource MIPS and a fixed network bandwidth are not static values and greatly depend on the surrounding influences.

MIPS or MI are not the preferred benchmarks as the execution times for two programs of similar MI but with different program locality (e.g. program compilation platform) can differ [8]. Moreover, a resource's full MIPS may not be available all the time because of the I/O interrupt signals. N.Muthuvelu [9] presents a grid job scheduling algorithm, based on a parameterized job grouping strategy, which is adaptive to the runtime grid environment. Intervalized average analysis is implemented by the grid broker to determine the current status of the grid before performing the job grouping task.

Task scheduling is a challenging problem in grid environment [10]. Grid scheduling is a NP Complete problem. Heuristic optimization algorithm can be used to solve a variety of NP complete problems. Abraham et al [11] and Braun et al [12] present three basic heuristics implied by Nature for Grid scheduling, namely Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS), and heuristics derived by a combination of these three algorithms.

Particle swarm optimization (PSO) [13] is one of the latest evolutionary optimization techniques inspired by nature. It simulates the process of a swarm of birds in their preying activity. Its ability for global searching has made PSO highly suitable for neural network training, control system analysis and sign, structural optimization and so on. It also has fewer algorithm parameters than either GA or SA [14].

This paper proposes an efficient job scheduling approach that uses job grouping to improve computation communication ratio and utilization of the resources. It also proposes the use of PSO to select the resources to minimize makespan and to bring about efficient load balancing.

3. Design of the System

3.1 *Architecture of gridlet Scheduler*

Figure 1 depicts the architecture of the gridlet scheduler used in the system. The system accepts total number of user jobs, processing requirements or average MI of those jobs, allowed deviation percentage of the MI, processing overhead time of each user job on the grid, granularity size of the job grouping activity and the available grid resources in the grid environment (step 1-2).

After gathering the details of user jobs and the available resources, the system randomly creates jobs with priority assigned to it according to the given average MI and MI deviation percentage (step 3). The scheduler will then select a resource and multiplies the resource

MIPS with the given granularity size (step 4). The jobs are then gathered or grouped according to the resulting total MI of the resource (step 5), and each created group is stored in a list with its associated resource ID (step 6). Eventually, after grouping all jobs, the scheduler submits the job groups to their corresponding resources for job computation (step 7).

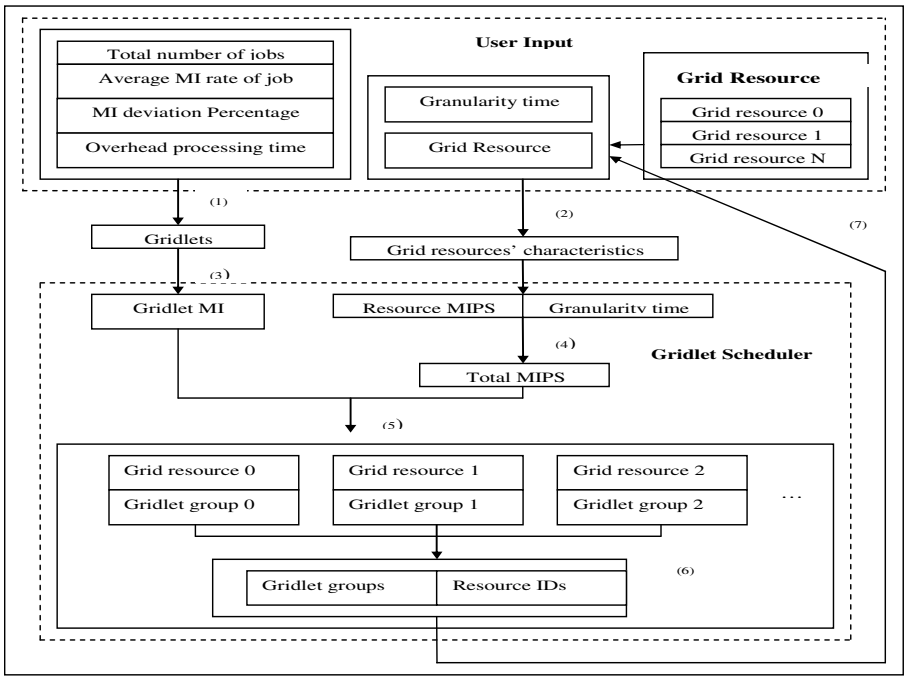

Fig 1 Architecture of the Gridlet Scheduler

3.2 *Operation of Scheduling Framework*

The scheduling framework illustrated in figure 2 depicts the design of the job scheduler and its interactions with other entities. When the user creates a list of jobs in the user machine, these jobs are sent to the job scheduler for scheduling arrangement. The information collector gathers resource information from the Grid information service (GIS). The grid information service (GIS) is a facility that provides information about all the registered resources in a grid. Based on the information, the job scheduling algorithm is used to determine the job grouping and resource selection for grouped jobs. Once all the jobs are put into groups with selected resources, the grouped jobs are dispatched to their corresponding resources for computation.

The grouping and selection service serves as a site where matching of jobs is conducted. The strategy for matching jobs is based on the information gathered from the information collector. There are two steps involved during the matching of jobs. They are job grouping

and job selection. In the job grouping process, jobs submitted by the user to the scheduler are collected and they are grouped together based on the information of resources. The size of a grouped job depends on the processing requirement length expressed in Million Instructions. At the same time, job selection is also being conducted where a grouped job corresponds to the resource in question. The process is performed iteratively until all the jobs are grouped according to their respective resources.

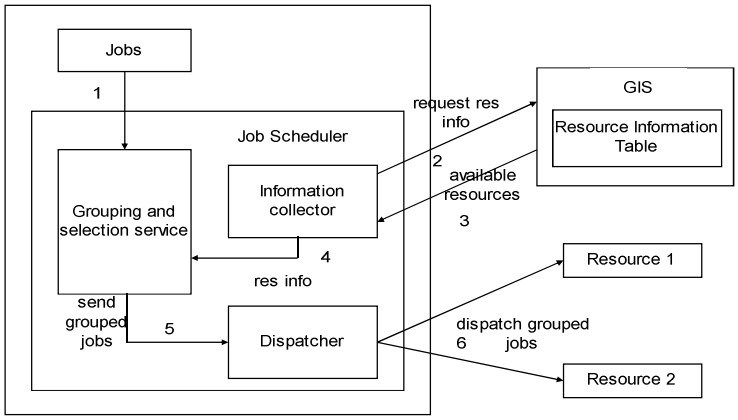

Fig 2 Scheduling Framework for Job Grouping

The dispatcher functions as a sender that transmits the grouped jobs to their respective resources. The dispatcher forwards the grouped jobs based on the schedule made during the matching of jobs with resources. The dispatcher also collects the results of the processed jobs from the resources through input ports.

3.3 *Gridlet Grouping Algorithm*

The following is a listing of steps in gridlet grouping.

1. The scheduler receives the Gridlet_List, J [ ] created in the system.
2. Sort the gridlets according to the priority
3. The scheduler receives the Resources_List, R [ ].
4. Set the Total_Length of gridlet to zero. Set the resource number i to 1.
5. Get MIPS of the resource i.
6. Multiply the Resource_MIPS with Granularity Time specified by the user.
7. Get length (MI) of the first gridlet.
8. If Resource_MIPS is less than the Gridlet_Length
   BEGIN
   - The gridlet cannot be scheduled for the resource.
   - Get the MIPS of next resource and
   - Proceed to step 6.
   END
9. If Resource_MIPS is more than Gridlet_Length

BEGIN

9.1 While Total_Length is less than or equal to Resource_MIPS, and while there are ungrouped gridlets in the Gridlet_List
BEGIN
- Set Total_Length to the summation of previous Total_Length and current Gridlet_Length
- Get the length of the next gridlet
- Repeat step 9.1
END

9.2 If the Total_Length is greater than Resource_ MIPS
BEGIN
- Deduct the length of the last gridlet from the Total_Length
END

END

10. If Total_length is not zero
BEGIN
- Create a new gridlet (Grouped_Gridlet) with the length equals to the Total _Length.
- Set User ID for the Grouped_Gridlet.
- Add the Grouped_Gridlet to a Gridlet_List_2.
- Add the ID of the resource (to which gridlet is to be assigned) into a linked list.
END
11. Set the Total_Length of gridlet to zero.
12. Get the MIPS of next resource.
13. Multiply the Resource_MIPS with Granularity_Time specified by the user.
14. Get the length of next gridlet.
15. Goto step 8.
16. Perform the looping until all the gridlets in the Gridlet_List are grouped into Grouped_Gridlet.
17. If the scheduler receives new Gridlet_List from the user (for dynamic scheduling),
BEGIN
- The scheduler manipulates the unutilized MIPS of each resource.
- Branch to step 6 with the manipulated MIPS taken as the resource capability.
END
18. After all the gridlets are scheduled into groups and each Grouped_Gridlet is assigned with a particular grid resource, send all the Grouped_Gridlets to their corresponding resources.
19. After all the Grouped_Gridlets are processed by the grid resources, and sent back to the I/O queue of the scheduler / system, collect the Grouped_Gridlets from the I/O queue.
20. Set the resource ID, and job execution cost of each Grouped_Gridlet.
21. Get the total simulation time.
22. Display the details of the processed Grouped_Gridlets to the user through GUI.

The overall explanation of algorithm is as follows: once the user jobs are submitted to the broker or scheduler, the scheduler gathers the characteristics of the available grid resources. Then, it selects a particular resource and multiplies the resource MIPS with the granularity size to get the total MI the resource can process within a specified granularity size. The scheduler groups the user jobs by accumulating the MI of each user job while comparing the resulting job total MI with the resource total MI. If the total MI of user jobs is more than the resource MI, the very last MI added to the job total MI will be removed from the job total MI. Eventually, a new job (job group) of accumulated total MI will be created with a unique ID and will be scheduled to execute in the selected resource. This process continues until all the user jobs are grouped into few groups and assigned to the grid resources. The scheduler then sends the job groups to their corresponding resources for further computation. The grid resources process the received job groups and send back the computed job groups to the grid user. The scheduler then gathers the computed job groups from the network through its I/O port or queue.

Even though the above mentioned approach enhances the computation-communication ratio, it does not enhance the utilization of the entire grid. So an evolutionary approach to job grouping in the grid environment is proposed.

4. Evolutionary approach to job grouping

4.1 *PSO approach for Scheduling Problem in Computational Grid*

PSO is used to address independent task assignments problems in parallel distributed computing systems. Conducting searches uses a population of particles. Each particle corresponds to an individual in evolutionary algorithm. To start with, a flock or swarm of particles is randomly generated. Each particle has an updating position vector $\dot{X}$ and updating velocity vector $\dot{V}$ by moving through the problem space. Kennedy and Eberhart [25] proposed the formula of updating position vector $\dot{X}$: $x_{k+1}^i = x_k^i + v_{k+1}^i$ (1) and formula of updating velocity vector $\dot{V}$: $v_{k+1}^i = w_k v_k^i + c_1 r_1 (p_k^i - x_k^i) + c_2 r_2 (p_k^g - x_k^i)$ (2) where c_1 and c_2 are positive constant and r_1 and r_2 are uniformly distributed random number in [0, 1]. The velocity vector V^i is in the range [-V_{max}, V_{max}] [13]. The vector P^i represents the best ever position of each particle and P^g represents the best position obtained so far in the population. Changing velocity this manner enables the particle to search around its individual best position P^i, and also updates its global best position P^g, until the global optimum is reached.

The aim of the scheduling is to improve the efficiency of resource and to minimize the job completion time. Task scheduling method uses the Small Position Value (SPV) rule [15] to convert the continuous position values to its permutation. Each particle denoted as X^i (i=1, 2, 3, …, n) with its position, velocity and fitness value represents a potential solution to resource scheduling. The position of each particle can be represented in 'n' number of dimensions as $x_k^i = [x_1^i, x_2^i, ..., x_n^i]$ where x_k^i is the position value of particle i with respect to the n dimension, and the velocity is represented by $v_k^i = [v_1^i, v_2^i, ..., v_n^i]$ where v_k^i is the velocity of particle i with respect to the n dimension.

Based on SPV rule, the continuous position of the particle is converted to a permutation of sequences S^i. S^i is represented by $s_k^i = [s_1^i, s_2^i, ..., s_n^i]$ where s_k^i is the sequence of task of i[th] particle in the processing order with respect to the n dimension. $r_k^i = [r_1^i, r_2^i, ..., r_n^i]$ is the n dimension task processing on the r_k^i resources. The fitness value evaluated by fitness function represents the particle's quality based on the current sequence R^i. The fitness function is to minimize $\sum_{i=1}^{m} load\_Ri / MIPS\_Ri$. The personal best value denoted as P^i represents the best searching position of the particle so far. For each particle in the population, P^i can be determined and updated at each iteration step.

In grid resource scheduling with the fitness function $f(R^i)$ where R^i is the corresponding sequence of particle X^i, the personal best P^i of the i[th] particle is obtained such that $f(R_k^i) \leq f(R_{k-1}^i)$ where R^i is the corresponding sequence of personal best P^i.For each particle, the personal best is denoted as $p_k^i = [p_1^i, p_2^i, ..., p_n^i]$ where p_k^i is the position value of the i[th] particle best with respect to the n dimension. The best particle in the whole swarm is assigned to the global best denoted as G^i. G^i can be obtained such that $f(R^i) \leq f(R_k^i)$ for k = 1, 2, 3, m, where R^i is the corresponding sequence of particle best P^i. In general, we define the fitness function of the global best as $f(R_{gbest}^i) = f(R_{best}^i)$, and the global best is defined as $g_k^i = [g_1^i, g_2^i, ..., g_n^i]$ where g_k^i is the position value of the global best with respect to the n dimension. Each particle updates its velocity vector based on the experiences of the personal best and the global best in order to update the position of each particle with the velocity currently updated in search space. Each particle keeps track of its own best position and the swarm keeps track of the global best position. When maximum number of iteration is reached, the process will be terminated.

Table 1 Solution Representation

Dimension	x_k^i	s_k^i	r_k^i
0	1.03	3	3
1	3.81	9	1
2	-0.11	1	1
3	-0.39	0	0
4	3.15	7	3
5	3.41	8	0
6	2.64	5	1
7	3.00	6	2
8	0.89	2	2
9	1.52	4	0

A particle represents a possible solution in the population and dimension n corresponds to n tasks. The Smallest Position Value (SPV) rule is used first to find a permutation corresponding to the continuous position X^i. For the n tasks and m resource problem, each particle represents a reasonable scheduling scheme. The position vector $x_k^i = [x_1^i, x_2^i, ..., x_n^i]$ has a continuous set of values. Table 1 illustrates the solution

representation of particle X^i of PSO algorithm for 10 tasks and 4 processors. Based on the SPV rule, the continuous position vector can be transformed to a dispersed value permutation $s_k^i = [s_1^i, s_2^i, ..., s_n^i]$. Then the operation vector $r_k^i = [r_1^i, r_2^i, ..., r_n^i]$ is defined by the following formula: $R_k^i = S_k^i \bmod m$ (3). This is used to find the sequence number of computing processor. The start of sequence number is defined as zero.

4.2 *PSO Algorithm*

The formulae (1) and (2) are used to construct the initial continuous position values and velocity value of each particle. In PSO algorithm, the self-recognition coefficient c1 and social coefficient c2 are 2, and the weight is linear decreased from 0.9 to 0.4.

The complete flow of the PSO algorithm for the task scheduling of grid can be summarized as follows:

Step1: Initialization.

- Set the contents about this algorithm: maximum number of iterations $k_{max,}$, self-recognition coefficient c1 = 2, social coefficient c2 = 2 and inertia factor ω to 0.9 ;
- Initialize position vector $x_k^i = [x_1^i, x_2^i, ..., x_n^i]$ and velocity vector $v_k^i = [v_1^i, v_2^i, ..., v_n^i]$ of each particle randomly;
- Apply the SPV rule to find the permutation $s_k^i = [s_1^i, s_2^i, ..., s_n^i]$;
- Apply the formula (3) to fine the operation vector $r_k^i = [r_1^i, r_2^i, ..., r_n^i]$;
- Evaluate operation vector of each particle; find the best fitness value among the whole swarm f (R^i). Set the global best value G^i = f (R^i).

Step 2: Update iteration variable. iter = iter +1;

Step 3: Update inertia weight. $\omega^{iter} = \omega^{iter-1} * \beta$ where β is a decrement factor

Step 4: Apply the formula (2) to update velocity of each particle;

Step 5: Apply the formula (1) to update position of each particle;

Step 6: Apply the SPV rule to find the permutation $s_k^i = [s_1^i, s_2^i, ..., s_n^i]$;

Step 7: Apply the formula (3) to fine the operation vector $r_k^i = [r_1^i, r_2^i, ..., r_n^i]$

Step 8: Update personal best. Each particle is evaluated by using the operation vector to see if the personal best will improve. If $f(R_{best}^i) \le f(R_k^i)$, then $f(R_{best}^i) = f(R_k^i)$

Step 9: Update global best. If $f(R_{gbest}^i) \le f(R_{best}^i)$ then $f(R_{gbest}^i) = f(R_{best}^i)$

Step 10: Stopping criterion - If the number of iteration exceeds the maximum number of iteration, then stop; otherwise go to step 2.

4.3 *Gridlet Grouping Algorithm with Load Balancing*

1. The scheduler receives the Gridlet_List, J [ ].
2. Sort the gridlets according to the priority
3. The scheduler receives the Resources_List, R [ ].
4. Call the PSO algorithm to assign the gridlets to the resources.
5. Find the Total_length of the gridlets assigned to each resource

6. If Total_length is not zero
   BEGIN
   - Create a new gridlet (Grouped_Gridlet) with the length equals to the Total _Length.
   - Set User ID for the Grouped_Gridlet.
   - Add the Grouped_Gridlet to Gridlet_List_2.
   - Add the ID of the resource into a linked list.

   END
7. If the scheduler receives new Gridlet_List from the user (for dynamic job scheduling)
   BEGIN
   - Find the average length of gridlets to be assigned to each resource.
   - The scheduler manipulates the unutilized MIPS of each resource.
   - Get the length of the first gridlet.

   7.1 While there are resources and ungrouped gridlets in the Gridlet_List
   BEGIN
   While new_Total_Length of gridlet is less than or equal to unutilized_Resource_MIPS
   BEGIN
   - Set new_Total_Length to the summation of previous new_Total_Length and current Gridlet_Length
   - If the new_Total_Length is greater than average task length to be assigned to resources
   BEGIN
   - Deduct the length of the last gridlet from the new_Total_Length
   END
   END
   If the new_Total_Length is greater than unutilized_Resource_MIPS
   BEGIN
   - Deduct the length of the last gridlet from the new_Total_Length
   - Find the total length of the unassigned tasks
   - Find the average length to be assigned to the remaining resources.
   END
   END
   Perform step 6 with this new_Total_Length as the gridlet length on each resource.
8. After all the gridlets are scheduled into groups and each Grouped_Gridlets are assigned with a particular grid resource, send all the Grouped_Gridlets to their corresponding resources.
9. After all the Grouped_Gridlets are processed by the grid resources, and sent back to the I/O queue of the scheduler/system, collect the Grouped_Gridlets from the I/O queue.
10. Set the resource ID, and job execution cost of each Grouped_Gridlets.
11. Get the total simulation time.
12. Display the details of the processed Grouped_Gridlets to the user through GUI.

5. Experimental Results

GridSim is used for simulation guidance. The tests are conducted using nine resources of different MIPS, as shown in Table 2.

Table 2 Grid Resource List

Resource	MIPS	Cost
R1	20	100
R2	24	200
R3	39	300
R4	40	70
R5	50	150
R6	60	400
R7	66	60
R8	72	50
R9	120	210

The MIPS of each resource is computed as follows:
Resource MIPS = Total_PE * PE_MIPS, where
Total_PE = Total number of processing elements (PE) at the resource,
PE_MIPS = MIPS of PE
The total processing cost is computed based on the actual CPU time taken for computing the gridlets at the grid resource and at the cost rate specified at the grid resource, as summarized below:
Process_Cost = T * C, where
T = Total CPU Time for gridlet execution, and
C = Cost per second of the resources.
We have conducted experiments and obtained results for task scheduling with and without job grouping and with and without load balancing.

Experiment 1: Simulation with and without Job Grouping

Simulations were conducted to analyze and compare the differences between two scheduling algorithms, namely first come first serve and job grouping-based algorithm (as explained in section 3.3) in terms of processing time and cost. Figure 3 shows the results of the simulations with and without job grouping method conducted with granularity size of 5 seconds, average gridlet length of 10 MI at a deviation of 10% and an overhead time of 10 seconds. The total processing time and cost are increasing gradually for simulations without job grouping method compared to simulations with job grouping method.
Without grouping, a simulation from 25 to 150 gridlets yields a massive increase of 489% in Total_Process_Time (TPT), whereas simulation with grouping yields only 233.15% rise in terms of in TPT.

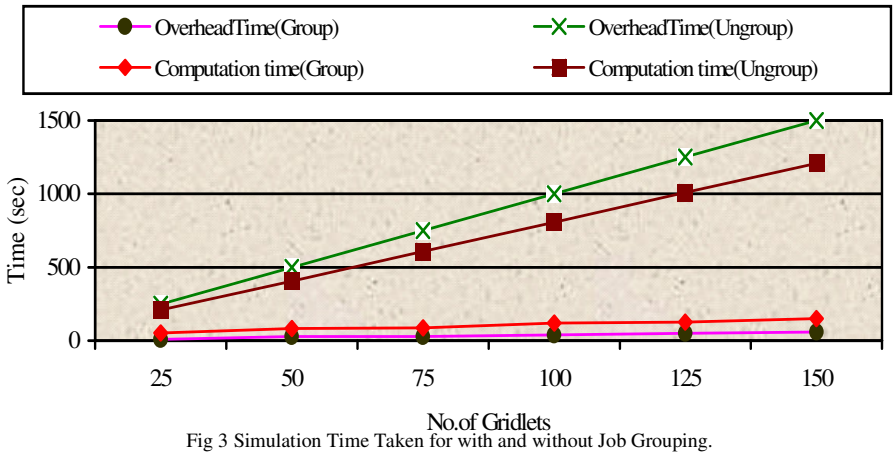

Fig 3 Simulation Time Taken for with and without Job Grouping.

As the number of gridlets grows, the TPT increases linearly for simulation without job grouping since total communication time increased with number of gridlets. In simulation with grouping, the communication time remains constant and major contribution to the total Process_Time comes from gridlet computation time at the resources. With 150 gridlets, six gridlet groups are created. Here, 28.3% of the total Process_Time is spent for communication purpose, whereas in simulation without grouping, 55.4% of total Process_Time is spent for the same communication purpose.

The time each gridlet spends at the grid resource is taken into consideration for computing the total Process_Cost. In simulation with job grouping, only a small number of gridlets (gridlet groups) are sent to each resource and therefore, the amount of total overhead time is reduced. When processing 25 gridlets individually at the grid resource, the total Process_Cost comes up to 5522 units, whereas simulation with job grouping reduces this cost to 1079 units. Figure 4 depicts the total cost incurred when simulation is carried out with and with job grouping on varying number of gridlets having average MI of 10 with a granularity size of 5 seconds.

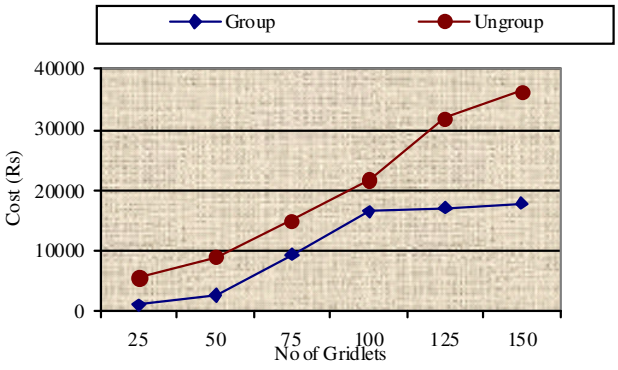

Fig 4 Processing Cost for Simulation with and without Job Grouping

Experiment 2: Simulation with Job grouping for different Granularity time.

Simulations are conducted using different granularity sizes to examine the total time and cost taken to execute gridlets on the grid. Table 3 shows the processing load at each grid resources when simulations are carried out on 50 gridlets having an average length of 50 MI using different granularity sizes.

Table 3 Processing load at grid resources for different granularity time

GT	Resources								
	R1	R2	R3	R4	R5	R6	R7	R8	R9
10	198	202				587		295	1194
15	251	346	57						1308
20	387		754			1181			153
25	451		953						1072
30					512		1964		

When granularity size is 10 seconds, 5 job groups are created (from 50 user jobs) and resource computes job groups of almost balanced MI. Since the gridlet computations at the grid resources are done in parallel and each resource has less processing load (balanced gridlet MI), all the gridlet groups can be computed rapidly, in 170 seconds. In the case of granularity size of 15 seconds, four gridlet groups are created and 66.67% of the total gridlet MI is scheduled to be computed at resource R9 since it can support up to 1800 MI. Average gridlet MI percentage at the other resources is 33.7%. Therefore, R9 spent more time in computing the gridlet group which leads to higher total Process_Time.

In terms of Process_Cost, the resulting cost highly depends on the cost per second located at each resource and total gridlet MI assigned to each resource. In the simulations, cost per second of using resource R3 (300 units) and R6 (400 units) are more than the other resources. Therefore, involving these resources in gridlet computation will increase the total Process_Cost. When the granularity size is 20 seconds, assigning a large number gridlet MI (1181 MI) to R6 results in high total Process_Cost of 47947 units.

From the experiments, it is clear that job grouping method decreases the total processing time and cost. However, assigning a large number of gridlet MI to one particular resource will increase the total processing time and cost. Therefore, during the job grouping activity, a balanced relationship should be determined between total number of groups to be created from job grouping method, resources cost per second, and MI distribution among the selected resources. For this purpose PSO is used.

Experiment 3: Processing gridlets within the Granularity time.

The scheduler is tested using different number of resources and granularity time to find out the total number of grouped gridlets that can be processed successfully within a particular granularity time. Figure 5 shows the number of gridlets completed (out of 150 gridlets)

within the granularity time where each gridlet have an average length of 200 MI. MI deviation percentage is 10% and the gridlet overhead processing time is 2 seconds. All the 150 gridlets are completed at a granularity time of 65.

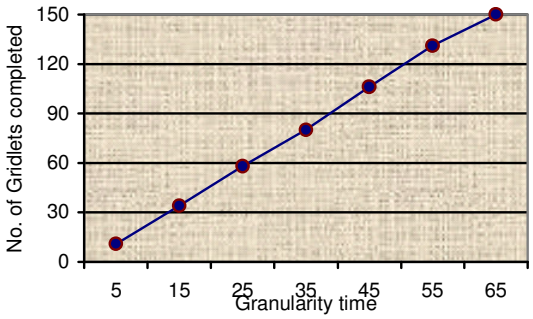

Fig 5. Number of gridlets completed in different granularity time

Experiment 4: Simulation of gridlet grouping using PSO based resource assignment.

Table 4 shows the load at the resources when PSO based resource selection (algorithm given in section 4.3) is used for different number of gridlets with an average length of 15 MI and a deviation of 10%. Granularity size is taken as 10 seconds and overhead time is 1 second. Table 5 shows the load at the resources when resources are selected randomly under the same condition.

From the table it is clear that when PSO based approach is used for resource assignment, load is balanced at all the resources. Simulation results demonstrate that PSO algorithm can get better effect for a large scale optimization problem. Hence task scheduling algorithm based on PSO algorithm can be used in the computational grid environment.

Table 4 Load at resources when PSO based approach is used

No. of gridlets	**Resources**								
	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**
25	45.39	44.05	44.78	27.68	28.66	44.13	43.29	45.31	46.55
50	92.00	86.02	89.63	77.68	75.88	86.23	73.67	70.58	90.90
75	133.27	135.94	117.33	120.77	120.10	117.37	117.42	121.29	134.79
100	183.43	159.78	159.36	165.64	169.94	162.74	170.29	163.13	161.74

Table 5 Load at the resources when resources are selected randomly

No. of gridlets	Resources								
	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**
25	189	181	-	-	-	-	-	-	-
50	194	-	383	-	166	-	-	-	-
75	195	226	208	-	490	-	-	-	-
100	196	-	376	-	495	-	-	-	429

Experiment 5: Comparison of Resource utilization of PSO vs. randomly selected resources

Figure 6 shows the load at the resources when resources are selected randomly (section 3.3) and PSO (section 4.3) is used for 100 gridlets with an average length of 15 MI and a deviation of 10%.

Granularity size is taken as 10 seconds and overhead time is 1 second. When PSO based approach is used, all the resources have almost equal load. Since all the available resources are used, the communication overhead may be increased. Hence parallel transfer of job groups to resources is required. But the time taken for actual processing at each resource is reduced very much when compared with random selection of resources.

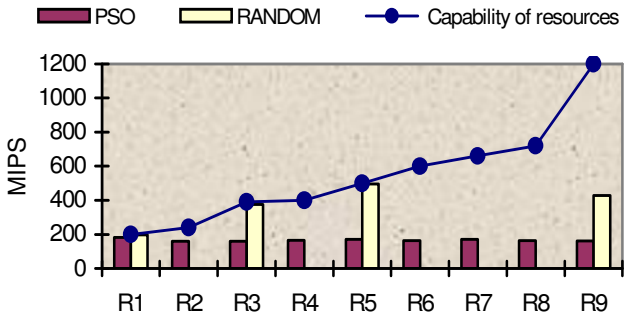

Fig 6 Resource Utilization Graph for 100 Gridlets

Figure 7 shows the percentage of resource utilization when gridlets are given to resources selected randomly vs. PSO approach. Percentage of resource utilization depends on the capability of the resources. Only four resources – R1 (98.2%), R3 (96.4%), R5 (98.98%) and R9 (35.72 %) are used when resources are selected randomly. R1, R3 and R5 are utilized fully. Here the standard deviation of percentage of utilization of resources is 31.09. The standard deviation is a measure of how widely values are dispersed from the average value. When PSO based approach is used, all the resources have equal load. Here the standard deviation is 24.95.

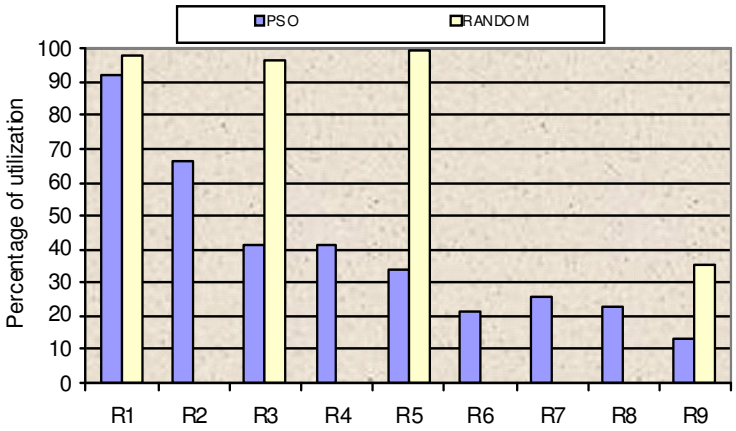

Fig 7 Percentage of Utilization of Resources for PSO vs. Random selection of Resources.

6. Conclusion

Grid system is a hot topic in distributed systems research at this moment. If fine grained jobs are assigned to grid resources which have high computing capability the computation-communication ratio becomes low. So gridlet grouping method where large numbers of gridlets are grouped into a few gridlet groups and sent to the Grid resources is used. Gridlet grouping reduces the transition time of each gridlet to the resource, and the overhead processing time of each gridlet at the resource. The proposed framework gives improved performance and better job scheduling compared to a non-grouping job scheduling framework. Resource selection based on PSO is used to generate an optimal schedule so as to complete the tasks in a minimum time and to utilize the resources efficiently. From the simulated experiment, it was found that the PSO based resource selection enables load balancing. Simulation results demonstrate that PSO algorithm can get better effect for a large scale optimization problem.

References

1. Baker.M, Buyya.R, Laforenza.D, 2002, “Grids and Grid Technologies for Wide-area Distributed Computing”, Software-Practice and Experience, Vol 32, No.15, pp.1437-1466.
2. James.H.A, Hawick. K.A and Coddington. P.D, 1999, “Scheduling Independent Tasks on Metacomputing Systems”, in Proc of Parallel and Distributed Computing Systems (PDCS’99), Aug. 1999, pp. 156--162.
3. Gerasoulis A and Yang T, 1992, “A Comparison of Clustering Heuristics for Scheduling Directed Graphs on Multiprocessors”, Journal of Parallel and Distributed Computing, Vol 16, No. 4, pp. 276-291.

4. Radulescu A. and van Gemund A, 1998, "A Low-Cost Scheduling Algorithm for Distributed-Memory Architectures", Proc. of the Fifth International Conference on High Performance Computing(HiPC 98), Madras, India, pp. 294-301, IEEE Press.
5. Buyya, R., Date, S., Miizuno-Matsumoto, Y., Venogopal, S. and Abramson, D., 2004, "Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A case for eScience on Global Grids", Journal of Concurrency and Computation: Practice and Experience. Vol 17, No. 15, pp.1783-1798.
6. Muthuvelu. N, Liu. J, Lin Soe. N, Venugopal. S, Sulistio. A and Buyya. R, 2005, "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids", in Proc of Australasian Workshop on Grid Computing and e-Research (AusGrid2005), Feb. 2005, pp. 41-48.
7. Ng. W. K, Ang. T. F, Ling. T. C, and Liew. C. S, 2006, "Scheduling Framework for Bandwidth-Aware Job Grouping-based Scheduling in Grid Computing", Malaysian Journal of Computer Science, Vol. 19, pp.117-126.
8. Stokes. J. H, "Behind the benchmarks: SPEC, GFLOPS, MIPS et al.", [Online Document] Jun. 2000, [2007 Sep 14], Available at: http://arstechnica.com/cpu/2q99/benchmarking-2.html
9. Nithiapidary Muthuvelu, Ian Chai and Eswaran. C, 2008, "An Adaptive and Parameterized Job Grouping Algorithm for Scheduling Grid Jobs", in ICACT 2008, Feb. 17-20, pp 975 – 980.
10. Ian Foster and Carl Kesselman eds., 2004, "The Grid: Blueprint for a New Computing Infrastructure", 2nd ed., Morgan Kaufmann Publishers.
11. Abraham. A, Buyya. R and Nath. B, 2000, "Nature's Heuristics for Scheduling Jobs on Computational Grids", The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), Cochin, India, December 2000, pp. 45-52.
12. Braun. R, Siegel. H, Beck. N, Boloni. L, Maheswaran. M, Reuther. A, Robertson. J, Theys. M, Yao. B, Hensgen. D and Freund. R, 2001, "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.
13. Schute. J. F and Groenwold. A. A, 2005, "A Study of Global Optimization using Particle Swarms", Journal of Global Optimization, Kluwer Academic Publisher, Vol 31, pp.93- 108.
14. Lei Zhang, Yuehui Chen and Bo Yang, 2006, "Task Scheduling Based on PSO Algorithm in Computational Grid", Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06).
15. Tasgetiren. M. F, Sevkli. M, Liang. Y. C, Gencyilmaz. G, 2004, "Particle Swarm Optimization Algorithm for Single-Machine Total Weighted Tardiness Problem", Congress on Evolutionary Computation, CEC2004, Portland, Oregan, USA.
16. Rajkumar Buyya, and Manzur Murshed, 2002, "GridSim: A Toolkit for the Modeling, and Simulation of Distributed Resource Management, and Scheduling for Grid Computing", The Journal of Concurrency, and Computation: Practice, and Experience (CCPE), Vol 14, Issue 13-15, Wiley Press, USA, pp.1175-1220.