Task Scheduling for Coarse-Grained Grid Application

Nithiapidary Muthuvelu, Junyang Liu and Nay Lin Soe

Grid Computing and,, Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{nmu, junyangl, nsoe}@students.cs.mu.oz.au

Abstract

Grid Computing is a new, and inevitable technology in the fields of scientific, and engineering, and as well as in commercial, and industrial enterprises. This growing technology facilitates the conduct of virtual organization by bringing together appropriate, and effective human, information, and computing resources for tackling highly complex, and multidisciplinary projects. This project focuses mainly on the scheduling strategies for coarse-grained Grid application. 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 needed to reduce the total time taken for job execution in the Grid. The main purpose of this project is to use the GridSim Toolkit to model coarse-grained Grid application by developing an efficient, and effective task scheduling method.

1.0 INTRODUCTION

GridSim is a Java-based toolkit for modeling, and simulation of Grid resources, and application scheduling [1]. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources, and managing their execution [2]. These features can be used to simulate resource brokers (e.g. Nimrod-G) or Grid schedulers for evaluating performance of scheduling algorithms or heuristics. Other than GridSim, there are few tools available for application scheduling simulation in Grid computing environments, such as Bricks, MicroGrid and SimGrid toolkit.

GridSim entities are namely, user, broker, resource, information service, statistics, shutdown, and report writer. Once GridSim starts, the resource entities register themselves with the Grid Information Service (GIS) entity. The broker entity queries GIS entity for resource discovery, based on the user entity's request. The GIS entity returns a list of registered resources, and their contact details. The broker entity queries the resources for resource configuration, and properties. They respond with dynamic information such as resources cost, capability, availability, load, and other configuration parameters. Broker entity selects the appropriate resources, and sends user jobs (Gridlets) to those resources fro execution. The resources send back the processed Gridlets to the I/O queue of the broker entity. Finally, the user will collect the processed Gridlets from the I/O queue.

This paper is organized as follows: Section 2 presents problem statement of this project, which is followed by project objective, and scope in Section 3, and 4 respectively. Section 5 describes the design of the scheduling system, and the implementation of the design is presented in Section 6. Some experiments or testing are done using the created program, and the results are presented in Section 7. Finally, Section 8 concludes the developed scheduling system (program), and results from Section 7.

2.0 PROBLEM STATEMENT

In a Grid environment, the jobs are processed at the Grid resources in a fine-grained form. Each job is sent one by one (one at a time) to the Grid resource(s), processed individually, and then sent back to the user. Sending, processing, and receiving the jobs one at a time, increases the total amount of time needed to execute all the jobs from a user. The total execution time encompasses the time for transmitting each jobs to the Grid resource, and the overhead processing time for each job at the Grid resource. Moreover, in the case of Grid resource which has a very high processing capabilities (processing effort), sending, and processing a small job that requires very low processing capabilities will lead to poor utilization of that particular resource.

3.0 OBJECTIVE

The purpose of this project is to develop a scheduling strategy that optimizes the utilization of Grid resource processing capabilities, and reduces total time taken to process user jobs. In particular, the project or system should reduce the total time taken in transmitting the user jobs to/from the resources, and also to reduce the overhead processing time of each jobs at the resource. In order to achieve these requirements, 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 send the grouped jobs to that resource. The scheduler also should submit, and receive all the job groups to/from the Grid resources in parallel.

Another purpose of the project is to include the granularity time 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 achieve the minimum job execution time, and optimum or maximum utilization of the Grid resources.

4.0 SCOPE

This project focuses on the scheduling part of the GridSim Toolkit. The GridSim simulation sends, and receives the Gridlets to/from the Grid resources in sequential manner; the user or broker entity gets a Gridlet from the Gridlet list, submits the Gridlet to the Grid resource, and receives the processed Gridlet from the

resource. Then, the broker gets the second Gridlet from the Gridlet list, and performs the subsequent steps. Sending, and receiving one Gridlet at one time increase the Gridlet transmitting time, and cost over the Grid network as well as the overhead processing time, and cost of each Gridlet. Another point is that, if an individual Gridlet with small length is sent to a Grid resource with large MIPS, the Gridlet processing effort at the Grid resource would not utilize the maximum capability of that Grid resource. This will result in wastage of network bandwidth in term of Gridlet transmitting cost and time.

The scheduling strategy of the GridSim simulation in this project is specifically designed for the Gridlets with small size lengths. However, the resulting simulation program still accepts Gridlets with large length (MI). The Grid resources in the system are of user choice, where user has to select the resources from an external file. Finally, only one Grid user is used in the system to be assigned to the Gridlets before sending the Gridlets to the resources.

5.0 DESIGN

This section presents the flowchart of the simulation program, and the architecture of the scheduling strategy. Figure 1.0 depicts the flowchart of the GridSim simulation used in the proposed program or system. The proposed system accepts total number of Gridlets, average Million Instructions (MI) of those Gridlets (Gridlet length), allowed deviation percentage of the MI, granularity time of the simulation, overhead processing time per Gridlet (Gridlet overhead processing time), and the Grid resources from the user. Then, it will initialize the GridSim entities, and parameters. The system then creates the Grid resources as specified by the user for processing the Gridlets. Details of the Grid resources are obtained from a file containing a list of resources with their characteristics. Total PEs (processing elements), and MIPS of each PE are created based on the details found in the file, and therefore, each resource is built with different MIPS. Resource creation followed by the construction of 'Test' class object, where 'Test' is the name of the program. Here, the Grid user, and the Gridlets are created for the simulation. The Gridlets are created based on user specified parameters (total number of Gridlets, average MI of each Gridlet, and the deviation percentage of the MI). Then, the system starts the GridSim simulation. It first gathers the characteristics of the available Grid resources created in the resource creation section in the system. Then, it multiplies the MIPS of each resource with the granularity time. Each resulting value indicates the total MI that a particular resource can process in the given granularity time. Then, the scheduler selects a resource (one by one), and schedules the Gridlets. The scheduler groups the Gridlets according to the calculated MI (processing capabilities) of a selected resource. All the Gridlets created in the program are grouped into a number of groups, and each group is assigned to a particular grid resource. Then, the system will submit all the Gridlet groups to their corresponding Grid resources. The Grid resources process the received Gridlet groups, and send back the processed Gridlet groups to the Grid user.

The system then gathers the processed Gridlets from simulated network through its I/O port or queue. Finally, the system will display the processed Gridlet groups, and some of their characteristics to the user.

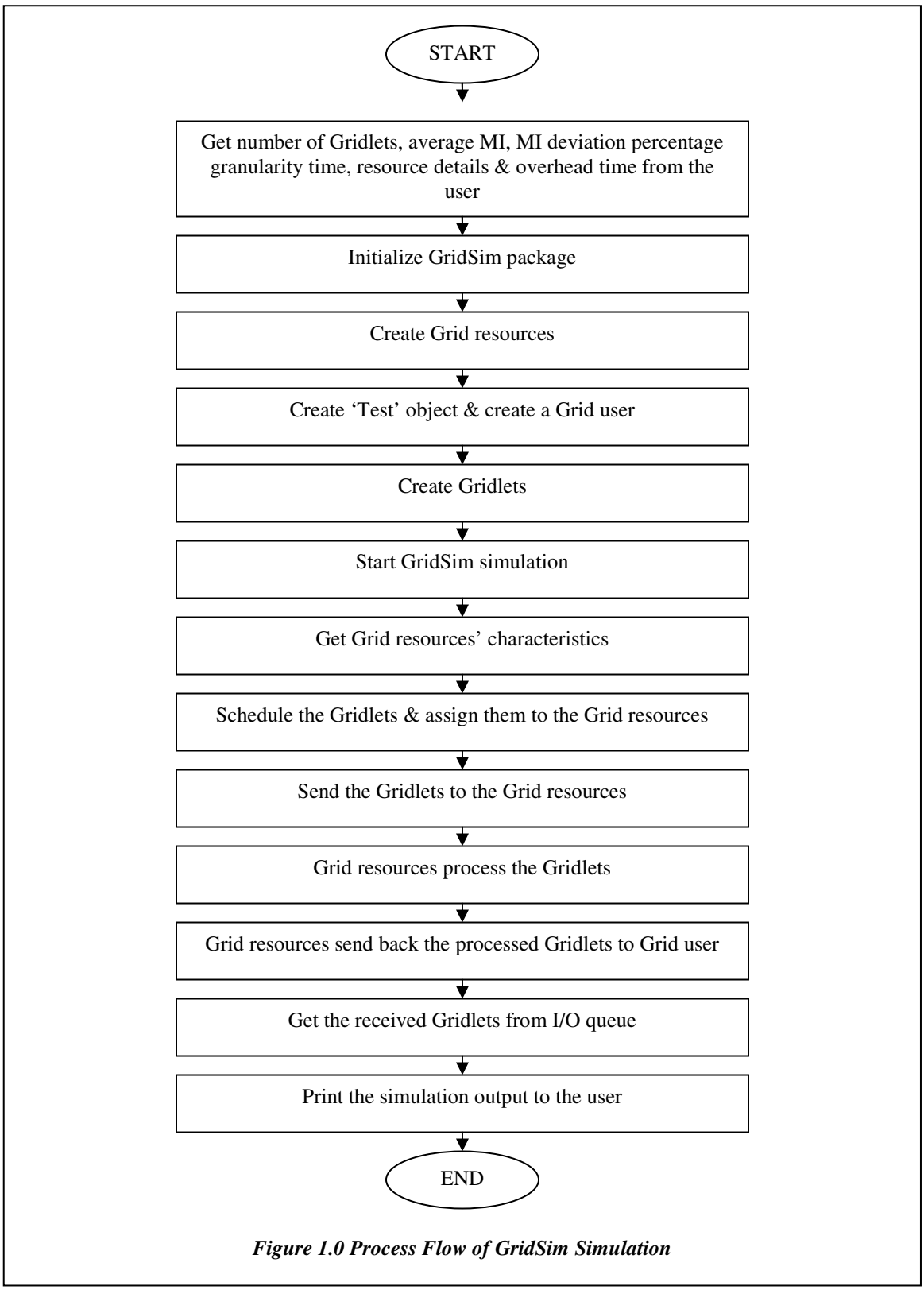


Figure 1.0 Process Flow of GridSim Simulation

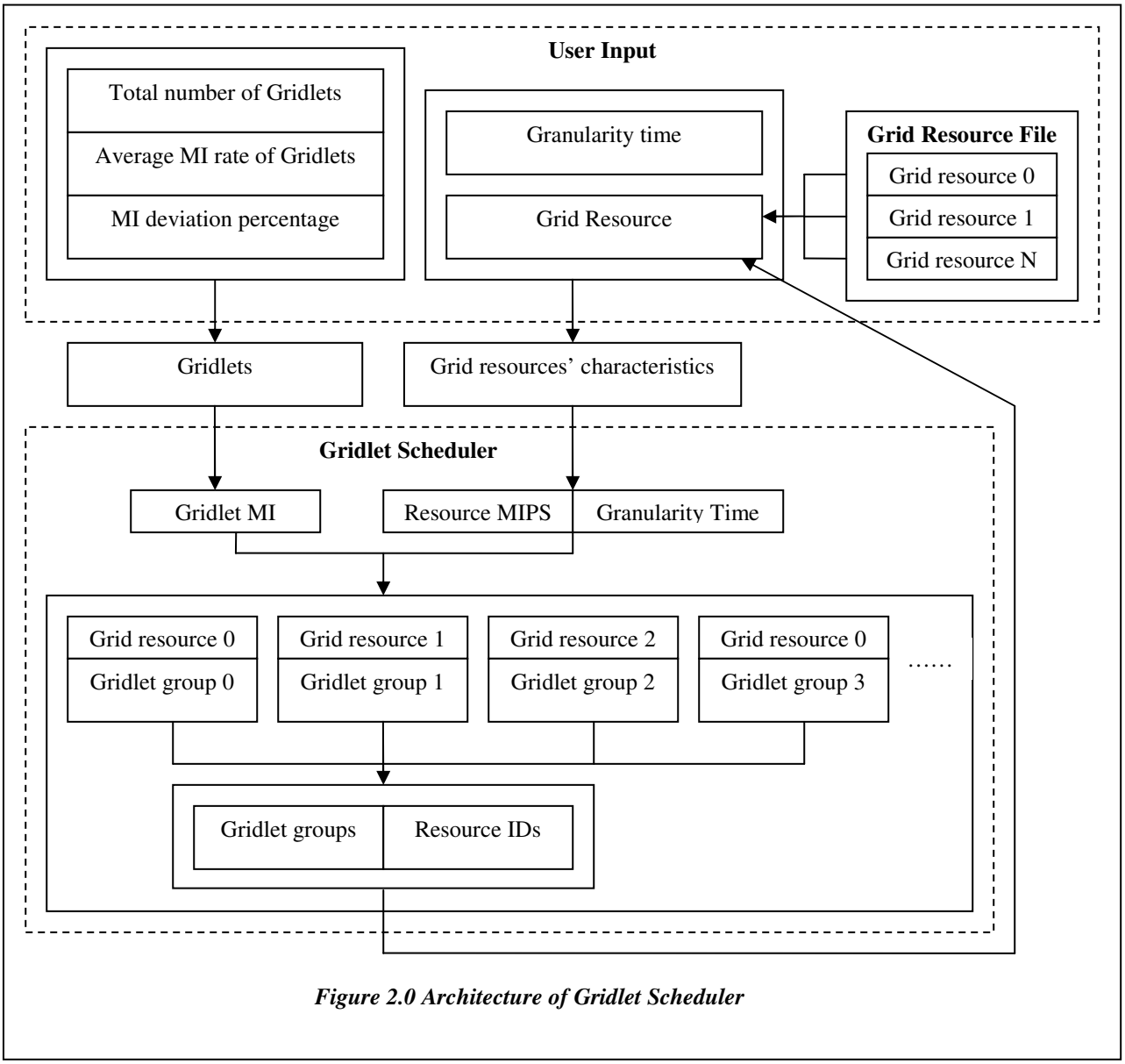


Figure 2.0 Architecture of Gridlet Scheduler

Figure 2.0 depicts the architecture of the Gridlet scheduler that will be used in the proposed system. The scheduler receives the characteristics of Grid resources from the resource file specified by the user. An alternative way is to collect the resource characteristics from the Grid Information Service entity that keeps track of the resources available in the Grid environment. The resource MIPS, granularity time, and Gridlet length are used for the Gridlet grouping task. Grouping task results in a list of Gridlet groups, and a list of resource IDs (IDs of resources assigned to each Gridlet group). The scheduler then sends the Gridlet groups to their assigned resources.

Figure 3.0 shows an example of Gridlet grouping task. In this example 100 Gridlets with small lengths (MI) are grouped into six groups according to the available resources, and the granularity time.

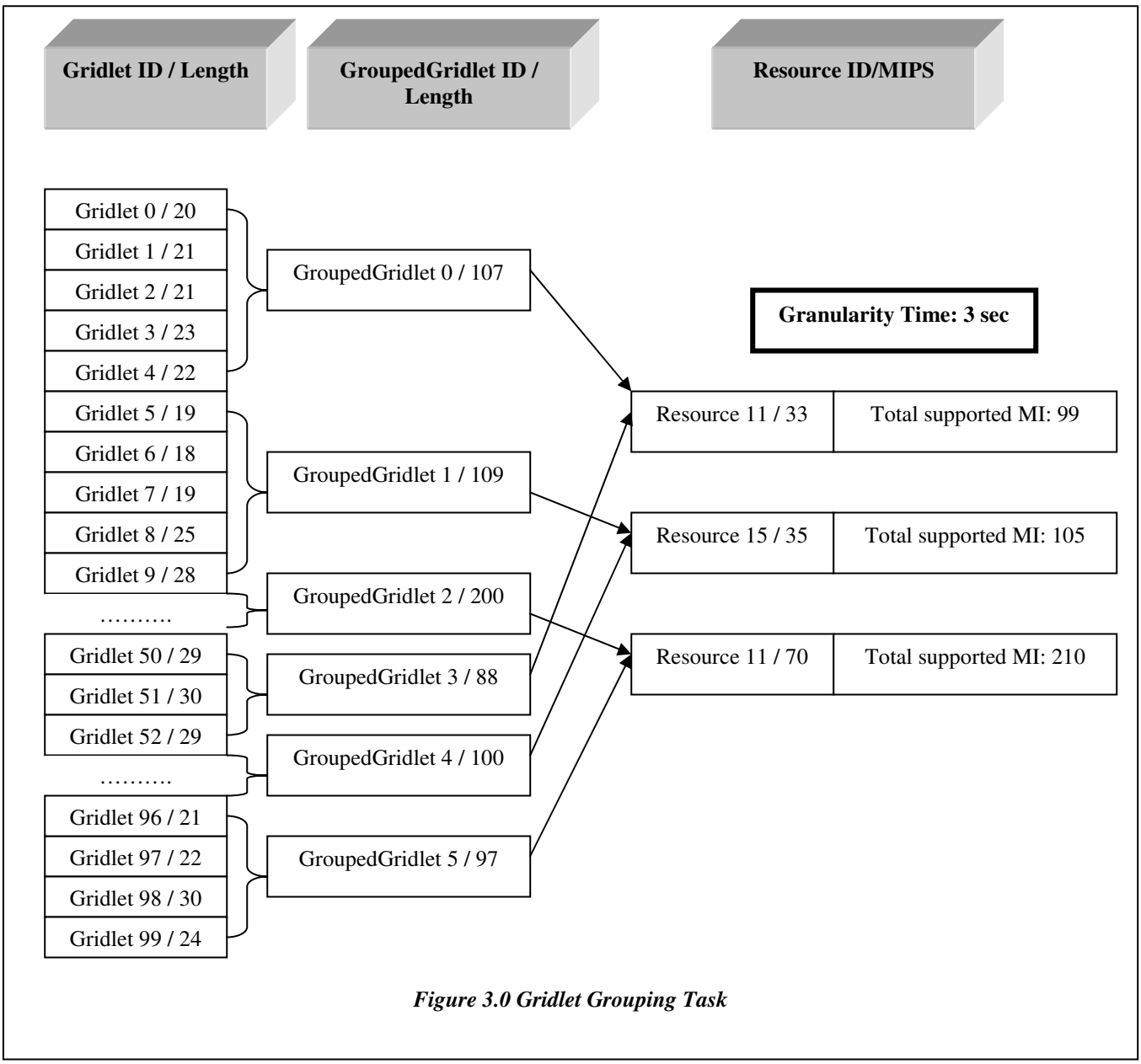


Figure 3.0 Gridlet Grouping Task

6.0 IMPLEMENTATION

6.1 Graphical User Interface

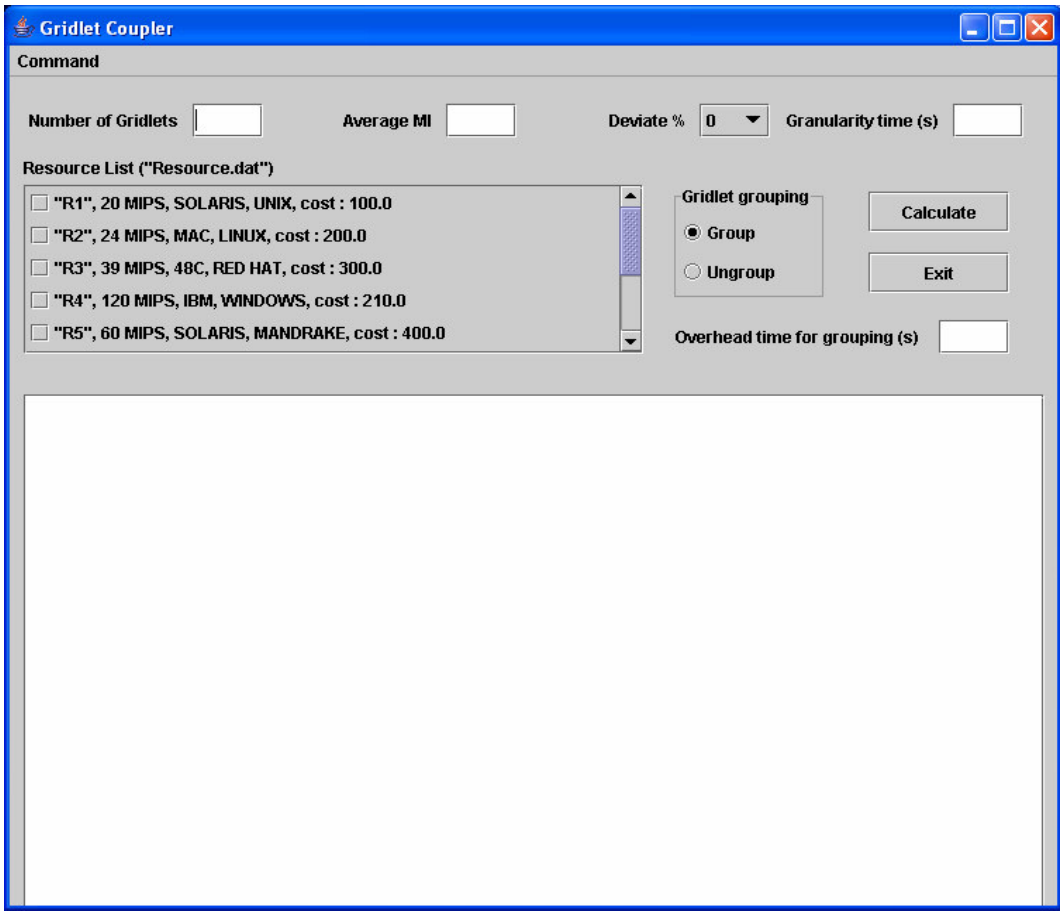


Figure 4.0 GUI for the GridSim Simulation System

Graphical User Interface (GUI) is used for better usability of the system. In order to facilitate software (class) reuse, the GUI is solely implemented in a separate class. Another class is used for reading data (resource information) from user specified resource file. This results in three-class design – first class is for GUI (GUI class or main class), second class is for Gridsim simulation (Gridsim class or support class), and the third class is for resource file. The purposes of the classes are exclusively demarcated as below:

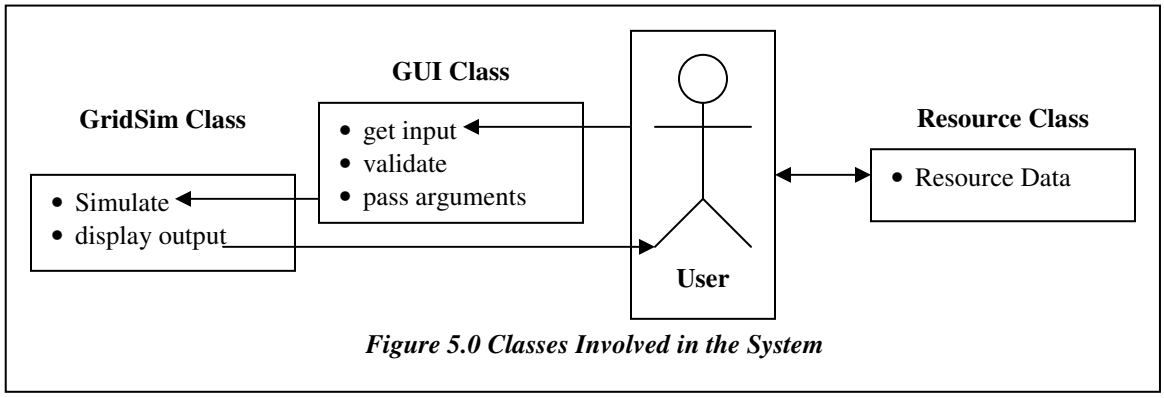


Figure 5.0 Classes Involved in the System

In this section, the aspects of GUI Class are discussed.

GUI Components (For Input and Output)

The components used in GUI class are from the Java swing package. The types of components used are JFrame, JButton, JTextField, JTextArea, JComboBox, JCheckBox, JMenu and JLabel. The inputs for number of Gridlets, their average size, and granularity time are implemented by JTextFields. The percentage of deviation of the Gridlet length can be selected in a JComboBox. JCheckBoxes are used for selecting whether Gridlet grouping should be done or not, and also for selecting the resources. The overall output is displayed in JTextArea. Additionally, two JButtons are incorporated - one for starting Gridsim, and another one for exit. There is also a menu (JMenu) in which the user can set the file name for reading the list of resources.

Listener (Event Handling)

For event handling, it is decided to handle only with an action listener. The action listener listens only to the clicking of the JButtons. Therefore, in order for the user to invoke a function of the program, a button needs to be clicked. Any other GUI actions are going to be ignored.

Validation

The validation of the user inputs are done by using regular expressions. In order for the inputs to be valid, first of all, they all must be non-empty. Secondly, except the input for the average Gridlet length, all others must be a combination of digits only. As for the one for the average Gridlet length any decimal number format is valid. If the inputs conform to these rules, the validation is passed.

Main Method

In command-line single-class design, the initial preparation for Grid simulation (such as initiation) is put in main method. In two-class GUI design, this preparatory part is separated from the Gridsim class, and integrated into actionPerformed method of the GUI class. A few other absolutely essential initialization codes

are also inserted in the GUI class. Otherwise, the purpose of the GUI class is merely for input, validation and output.

Figure 6.0 depicts the input, and output of the system. The user must insert the number of Gridlets, average Gridlet length (MI), MI deviation percentage, granularity time, Gridlet overhead processing time, and the resource file to the system through the GUI. The user can choose to perform the simulation with or without using the Gridlet grouping method. This facilitates the user to compare the output between a conventional simulation, and simulation with Gridlet grouping method.

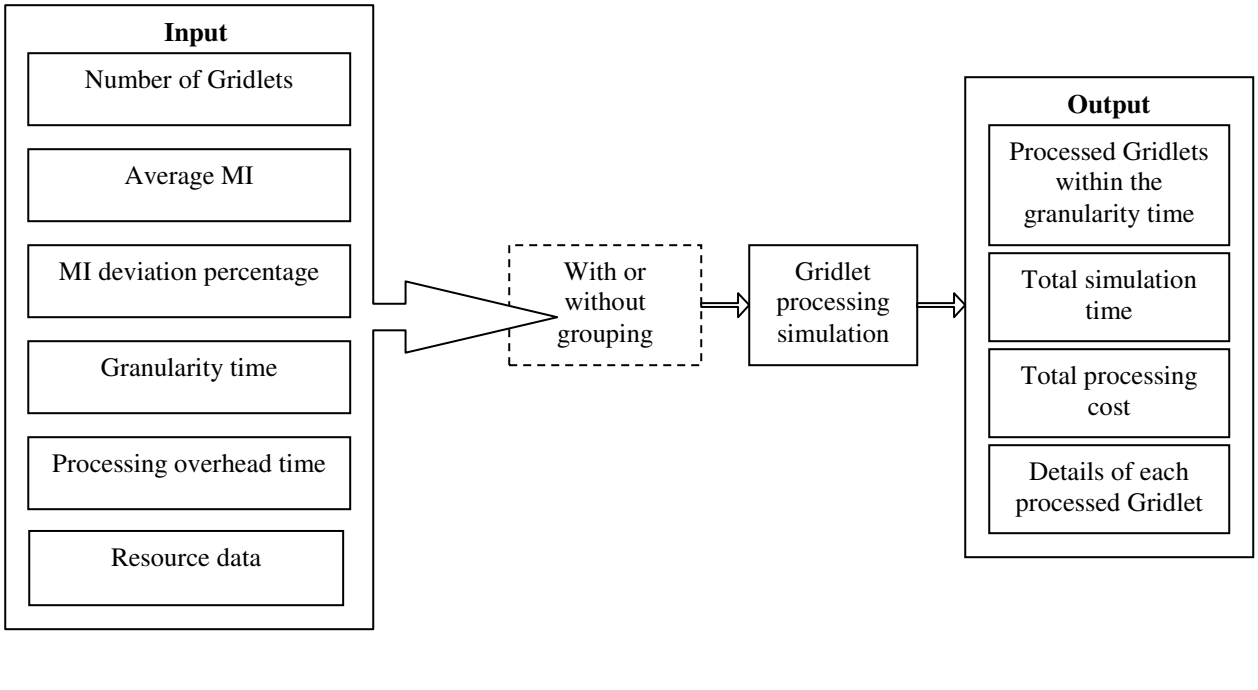


Figure 6.0 Input, output of the system

6.2 Grid Resource Creation

Figure 7.0 shows how the Grid resources are created from a resource file. The resource file contains a list of available resources on the Grid. Each resource has their characteristics specified in the file, namely, resource name, architecture, operating system, number of machines, number of PEs, MIPS of each PE, time zone, processing cost, communication speed, random seed, and resource load during peak hour, off-peak hour, and holiday. The user can choose a resource file, and then select the resources that he wants to use to process his Gridlets. Once the user selects the resources, the system will create those resources to be used for the Gridlet processing task.

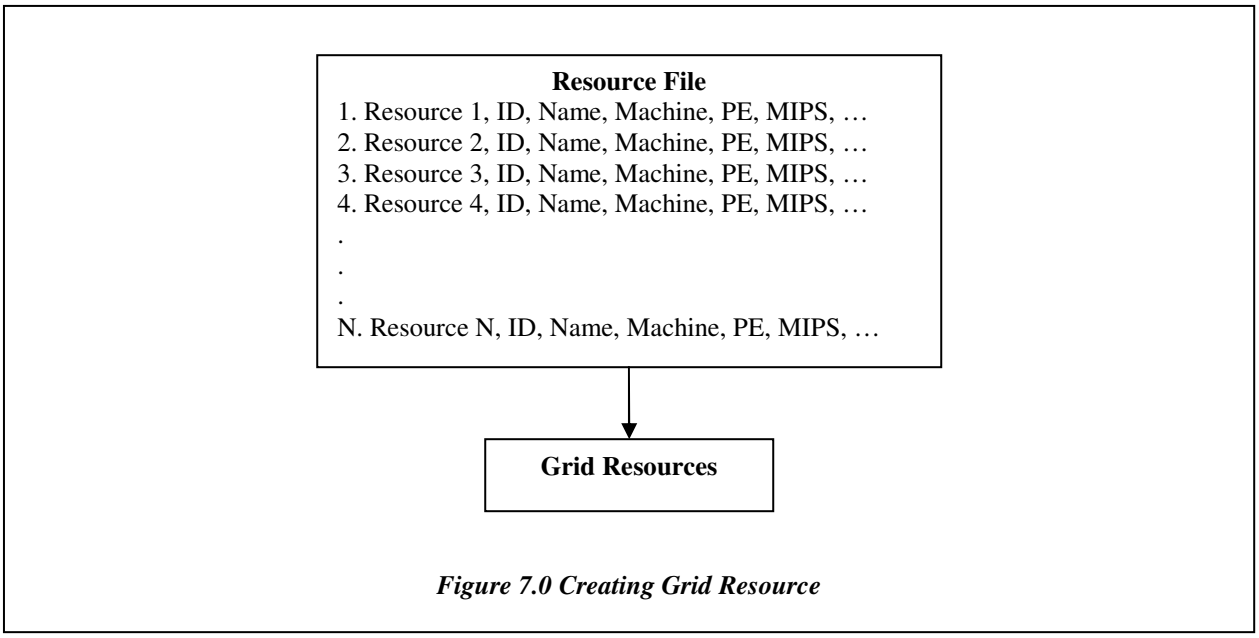


Figure 7.0 Creating Grid Resource

Figure 8.0 displays the code listing for creating the Grid resources.

```
/*****************************************************************
   The createGridResource method that creates Grid resources
 *****************************************************************/
  private GridResource createGridResource(int index)
  {
    // get resource details from resource file
    String      name            = resourceFile.getNameOfResource(index);          // resource name
    String      architecture    = resourceFile.getArchitectureOfResource(index);  // system architecture
    String      OS              = resourceFile.getOSOfResource(index);            // operating system
    int         machines        = resourceFile.getMachinesOfResource(index);      // number of machines in the resource
    int         PE              = resourceFile.getPEOfResource(index);            // number of PEs in the resource
    int         MIPSofPE        = resourceFile.getMIPSofPEOfResource(index);      // MIPS of each PE
    long        seed            = resourceFile.getSeedOfResource(index);
    double      timezone        = resourceFile.getTimezoneOfResource(index);      // time zone this resource located
    double      cost            = resourceFile.getCostOfResource(index);          // the cost of using this resource
    double      baudRate        = resourceFile.getBaudRateOfResource(index);      // communication speed
    double      peakLoad        = resourceFile.getPeakLoadOfResource(index);      // the resource load during peak hour
    double      offPeakLoad     = resourceFile.getOffPeakLoadOfResource(index); // the resource load during off-peak hour
    double      holidayLoad     = resourceFile.getHolidayLoadOfResource(index); // the resource load during holiday

    MachineList mList = new MachineList(); // create machine list to store the PE lists
    PEList peList;
        for(int machineNum = 0; machineNum < machines; machineNum++)
    {
      peList = new PEList(); // create first PE list to store the PEs

      for(int PEnum = 0; PEnum < PE; PEnum++)
      {
        peList.add( new PE(PEnum, MIPSofPE) ); // add create PEs
      }
      mList.add( new Machine(machineNum, peList) ); // add PE list into machine list
    }

   // create a ResourceCharacteristics object to store the properties of a Grid resource: architecture, OS, list of
   // Machines, allocation policy: time- or space-shared, time zone and its price (G$/PE time unit).
    ResourceCharacteristics resConfig = new ResourceCharacteristics(architecture, OS, mList,
                                                                    ResourceCharacteristics.TIME_SHARED,timezone, cost);

   // incorporates weekends so the grid resource is on 7 days a week
   LinkedList Weekends = new LinkedList();
    Weekends.add(new Integer(Calendar.SATURDAY));
    Weekends.add(new Integer(Calendar.SUNDAY));

   // incorporates holidays, but no holidays are set in this Test program
    LinkedList Holidays = new LinkedList();
    GridResource gridRes = null;

    try
    {
       gridRes = new GridResource(name, baudRate, seed, resConfig, peakLoad, offPeakLoad, holidayLoad, Weekends,Holidays)
    }
    catch (Exception e) {
         e.printStackTrace();
       }

    return gridRes;
  }
```

Figure 8.0 Code Listing for Grid Resource Creation

6.3 Gridlet Creation

Figure 9.0 shows how the Gridlets are created in the system. User inserts three parameters that affect the Gridlet creation task, namely, the number of Gridlets, the everage MI of each Gridlet, and the MI deviation percentage. The MI deviation percentage refers to the deviation percentage of average MI of each Gridlet. For example, if the average MI is 20, and deviation percentage is 30%, than the Gridlet length falls between 14 MIPS to 26 MIPS. Other than that, the file size, and output size of each Gridlet varies within the range of 500 with +/- 10%. Each resulting Gridlet will be assigned to a Grid user, and then added into a Gridlet list.

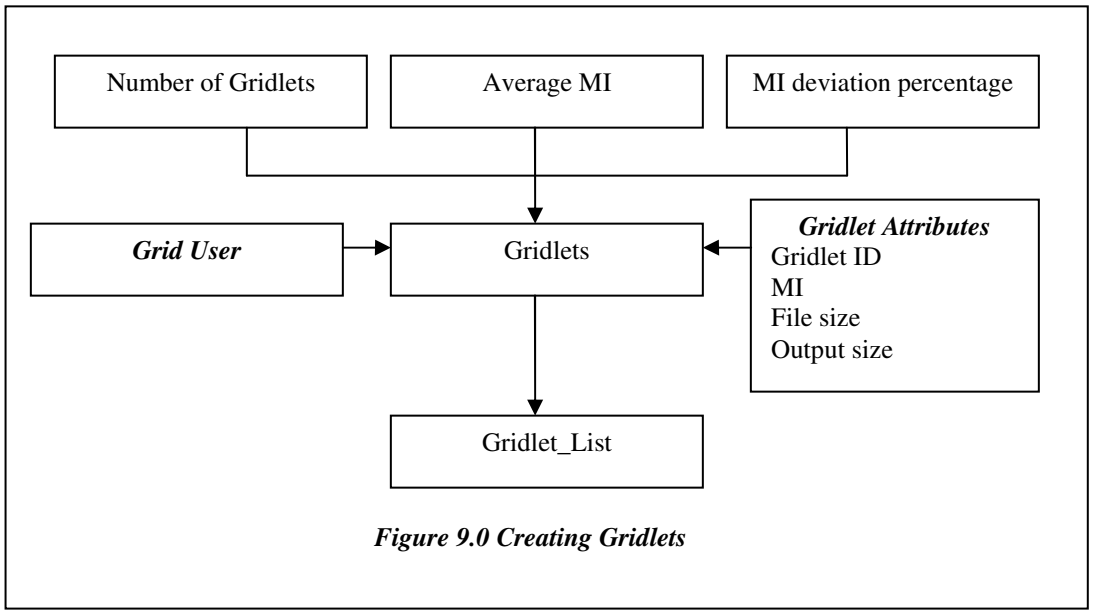


Figure 9.0 Creating Gridlets

Figure 10.0 displays the code listing for creating the Gridlets.

```
/*********************************************************************************
 This createGridlet method creates Gridlets based on user specified MIPS
*********************************************************************************/
private GridletList createGridlet(int userID)
{
    // creates a list to store Gridlets
    GridletList list = new GridletList();

    // variable declarations
     int__id = 0;
     double _length;
     long __file_size;
     long __output_size;
     double _rateDP;

    // create Gridlets by using the seed for the random function
     long__seed__= 11L*13*17*19*23+1;
     Random _random _= new Random(seed);

    // convert the deviate percentage from % form to point form
     rateDP = (double)deviatePercentage/100.0;

     for (int i = 0; i < numOfGridlets; i++)
     {
      // Gridlet length that varies within the range of user specified MIPS rating +/- deviation %
       length                    = (double) GridSimRandom.real(averageMI,rateDP,rateDP,random.nextDouble());
       // Gridlet file size that varies within the range 500 +/- 10%
       file_size                 = (long) GridSimRandom.real(500,0.1,0.1,random.nextDouble());
       // Gridlet output size that varies within the range 500 +/- 10%
       output_size               = (long) GridSimRandom.real(500,0.1,0.1,random.nextDouble());
       // creates a new Gridlet object
       Gridlet gridlet           = new Gridlet(id + i, length, file_size, output_size);
       // set user ID for the Gridlets
       gridlet.setUserID(userID);
       // add the Gridlet into a list
       list.add(gridlet);
     }
     return list;
 }
```

Figure 10.0 Code Listing for Gridlet Creation

6.4 Gridlet Grouping and Scheduling

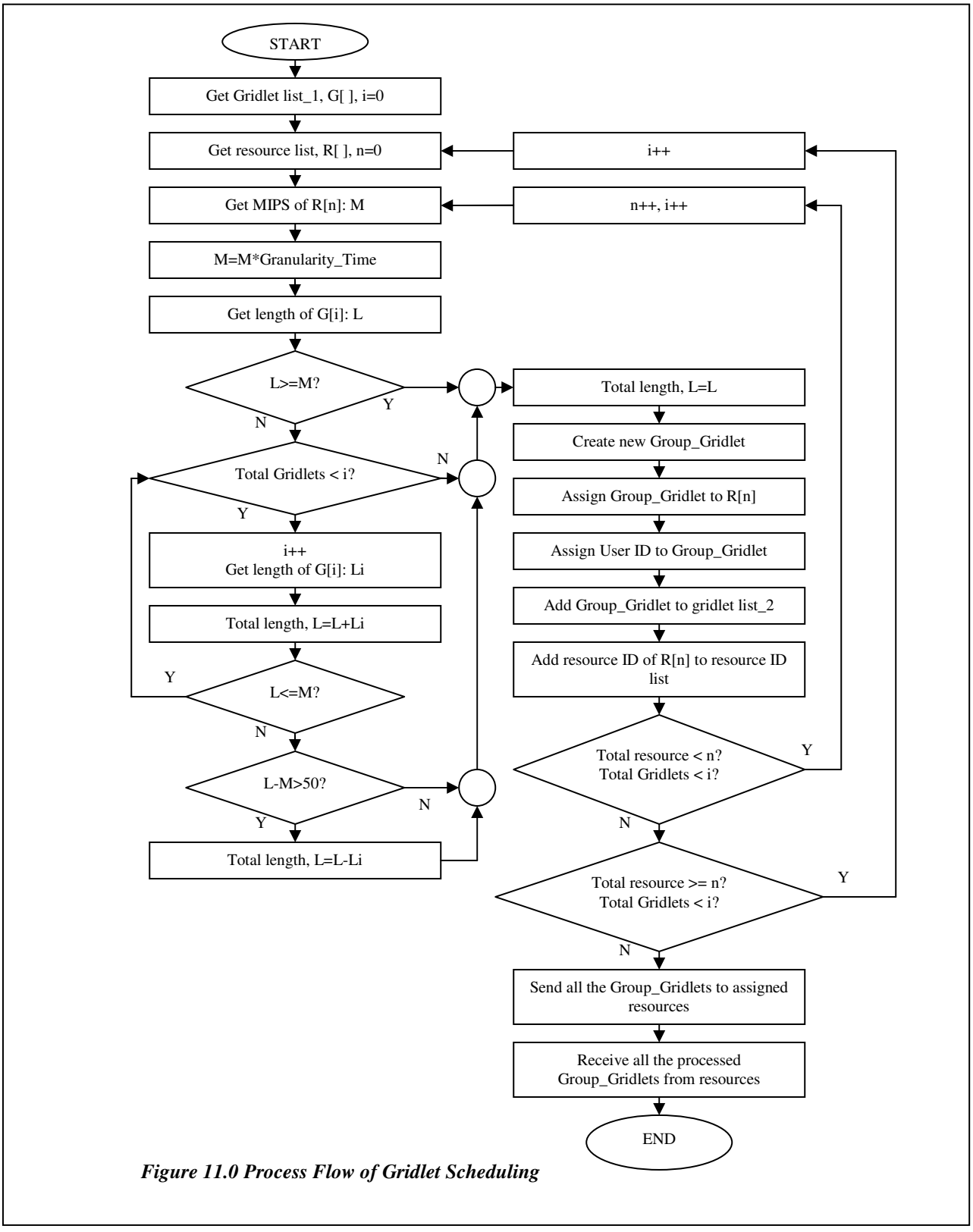


Figure 11.0 Process Flow of Gridlet Scheduling

```
/**************************************************************************
    The core method that handles communications among GridSim entities
**************************************************************************/
public void body()
{
  int       i;
  int       resourceID[ ]       = new int[this.totalResource_];
  double    resourceCost[ ]     = new double[this.totalResource_];
  String    resourceName[ ]     = new String[this.totalResource_];
  double    resourceMIPS[ ]     = new double[this.totalResource_];

  LinkedList resList;
  ResourceCharacteristics resChar;

  // Waiting to get list of Grid resources
  while (true)
  {
          // pause for a while to wait for the Grid resources to finish registering to GIS
          super.gridSimHold(1.0); // hold by 1 second

          esList                  = getGridResourceList();
           if (resList.size()     == this.totalResource_)
                  break;
           else
                  output          += "\nWaiting to get list of resources ..... \n";
   }

   output += "\n================ Receiving Characteristics of the Selected Grid Resources ================\n\n";
  // a loop to get all the available Grid resources
  for (i=0; i < this.totalResource_; i++)
  {
          // resource list contains list of resource IDs
          resourceID[i]           = ( (Integer)resList.get(i) ).intValue();
          // requests to resource entity to send its characteristics
          send(resourceID[i], GridSimTags.SCHEDULE_NOW,GridSimTags.RESOURCE_CHARACTERISTICS, this.ID_);
           // waiting to get a resource characteristics
          resChar                 = (ResourceCharacteristics) receiveEventObject();
          resourceName[i]         = resChar.getResourceName();
          resourceCost[i]         = resChar.getCostPerSec();
          resourceMIPS[i]         = resChar.getMIPSRating();
          output                  += "Resource " + resourceName[i] + ", with id = " + resourceID[i]
                                  + ", cost = " +  resourceCost[i] + ", MIPS = " +  resourceMIPS[i] + "\n";
          // record this event into "stat.txt" file
          recordStatistics("\"Received ResourceCharacteristics from " + resourceName[i] + ", with id = " + resourceID[i]
                                  + ", cost = " +  resourceCost[i] + ", MIPS = " +  resourceMIPS[i] +"\"", "" );
  }

  output += "\n==============================START SIMULATION================================\n";

  // variable declarations
   String           info;
   int              m=0, n=0, p=0, q=0;
   int              id=0;
  int               checkPoint=0, processedGridlet=0;
   double           Time1, Time2, Total_Time;
   double           total_length, MIPS_Rate_Machine;
   Gridlet          gridlet, groupedGridlet;
   GridletList      list2                   = new GridletList();
   LinkedList       TargetResource          = new LinkedList();
   LinkedList       FGridlet                = new LinkedList();
   LinkedList       LGridlet                = new LinkedList();

   // start timer
   Time1 = clock();
```

```
// Gridlets are grouped together according to resource MIPS rating and sent to that resource
if(grouping == true)
{
	// a loop to get a Grid resouce's MIPS rating and group the Gridlets according to that rating
	for(i = 0; i <  this.list_.size();) // loop to get the Gridlet
	{
		// check point to detect whether all the gridlets are processed within the given granularity time
		checkPoint++;
		if(checkPoint == 2)
			processedGridlet = i-1;

		for(n=0; n < (this.totalResource_) && (i < this.list_.size()); n++, i++) // loop to get the resource
		{
			p				= i;
			total_length			= 0.0;
			MIPS_Rate_Machine		= 0.0;
			id				= n;

			// get MIPS of a resource and multiply it with the granularity time
			MIPS_Rate_Machine = resourceMIPS[id]*granTime;

			output	+= "\nThe supported MI for granularity time " + granTime + " second(s), by Resource "
				+ resourceName[id] + " (ID:" + resourceID[id] + ")" + " is " + MIPS_Rate_Machine;

			// get one Gridlet
			gridlet = (Gridlet) this.list_.get(i);

			// if the resouce MIPS is less than Gridlet's MI, then assign the Gridlet to that resource
			if( MIPS_Rate_Machine <= gridlet.getGridletLength())
			{
				// assign the indidual Gridlet to the resource without any grouping process
				total_length = gridlet.getGridletLength();
			}

			// else, group the Gridlets based on resource MIPS
			else
			{
				while((total_length <= MIPS_Rate_Machine) && (i< this.list_.size()))
				{
					gridlet		= (Gridlet) this.list_.get(i);
					total_length	= total_length + gridlet.getGridletLength();
					i++;
				}

				i=i-1;

				// if the difference between total MI (total length) and resource MIPS is more than 50MI,
				// then deduct the last Gridlet from the group
				if((total_length - MIPS_Rate_Machine) > 50)
				{
					gridlet		= (Gridlet) this.list_.get(i);
					output		+= "Deducting the last Gridlet's length from the total length \n";
					total_length	= total_length - gridlet.getGridletLength();
					i		= i-1;
				}
			}

			// create the GroupGridlets
			groupedGridlet = new Gridlet(m,total_length,1000,1000);
			// set Gridlet user for the GroupGridlets
			groupedGridlet.setUserID(this.ID_.intValue());
```

```
                    // add the GroupGridlets into a list
                    list2.add(groupedGridlet);
                    // set the resource ID for each GroupGridlets
                    TargetResource.add(new Integer(id));
                    q           =i-p+1;
                    output      +="\nGroupGridlet " + m + "; composed of " + q + " Gridlets, (Gridlet " + p + " to Gridlet " + i
                                + "); Total Length: " + total_length + "; Resource ID: " + resourceID[id] + "\n";

                    FGridlet.add(new Integer(p));
                    LGridlet.add(new Integer(i));
                    // record this event into "stat.txt" file
                    recordStatistics("\"GroupGridlet " + m + ", Total Gridlet (" + p + " to " + i + "): " +  q + ",
                                        Total Length: " + total_length + ", Resource ID: " + resourceID[id] + "\"", "");
                    m++;
                    }
            }

            // send GroupGridlets to the Grid resources
            output += "\n=================== Sending Gridlets to Grid Resources ======================\n";
            i=0;
            while(i < list2.size())
            {
                    // get the assigned resource ID
                    id          = ( (Integer)TargetResource.get(i) ).intValue();
                    // get the GroupGridlet
                    gridlet     = (Gridlet) list2.get(i);
                    info        = "groupedGridlet " + gridlet.getGridletID();
                    output      +="Sending " + info + " to Resource " +  resourceName[id] + ", ID: " + resourceID[id] + "\n";
                    // send one GroupGridlet to the assigned Grid resource
                    gridletSubmit(gridlet, resourceID[id], 0.0, false);
                    // record this event into "stat.txt" file
                    recordStatistics("\"Submit " + info + " to " + resourceName[id] + "\"", "");
                    i++;
            }

            // receive the GroupGridlets from the I/O queue
            output += "\n======================== Receiving Processed Gridlets ========================\n";
            i=0;
            while(i < list2.size())
            {
                    // waiting to receive a Gridlet back from resource entity
                    gridlet     = this.gridletReceive();
                    // when a Grid resource finished processing the Gridlet, set the resource ID and its cost to perform the job
                    gridlet.setResourceParameter( gridlet.getResourceID() , gridlet.getCostPerSec());
                    info        = "GroupedGridlet " + gridlet.getGridletID();
                    output      +="Receiving " + info + " from " + gridlet.getResourceName(gridlet.getResourceID())
                                + ", ID: " + gridlet.getResourceID() + "\n";
                    // recod this event into "stat.txt" file for statistical purposes
                    recordStatistics("\"Received " + info +  " from " + gridlet.getResourceName(gridlet.getResourceID())
                                + "\"", gridlet.getProcessingCost());
                    // store the received Gridlet into a new GridletList object
                    this.receiveList_.add(gridlet);
                    i++;
            }
            output += "\n============================================================================\n";

            // total overhead time
            overhead            = overhead*list2.size();
    } // End of If
```

Figure 12.0 Code Listing for Gridlet Scheduling

Figure 11.0 shows the process flow involved in scheduling (grouping) the Gridlets, and sending the Gridlets to the appropriate resources. Figure 12.0 displays the code listing for Gridlet scheduling task. The following scheduling algorithm describes in detail the steps involved in the scheduling process as depicted by Figure 9.0, and 10.0.

Gridlet Scheduling Algorithm

1. The scheduler receives the Gridlet_List created in the system.
2. The scheduler receives the Resources_List (resources selected by the user from resource file).
3. Set the Total_Length of Gridlet to zero.
4. Get MIPS of first resource.
5. Multiply the Resource_MIPS with Granularity_Time specified by the user.
6. Get length (MI) of the first Gridlet.
7. If Resource_MIPS is less than or equal to Gridlet_Length

   BEGIN
   - Set Total_Length to the current Gridlet_Length
   - Proceed with step 9

   END
8. If Resource_MIPS is more than Gridlet_Length

   BEGIN

   8.1 While Total_Length is less than or equal to Resource_MIPS, and while there are ungrouped Gridlets in the Gridlet_List

   BEGIN
   - Get the length of the next Gridlet
   - Set Total_Length to the summation of previous Total_Length and current Gridlet_Length
   - Repeat step 8.1

   END

   8.2 If the difference between Resource_MIPS and Total_Length is more than 50

   BEGIN
   - Deduct the length of the last Gridlet from the Total_Length

   END

   END
9. Create a new Gridlet (Grouped_Gridlet) with the length equals to the Total _Length.
10. Set User ID for the Grouped_Gridlet.
11. Add the Grouped_Gridlet to a Gridlet_List_2 (a new Gridlet list).
12. Add the ID of the resource (whose MIPS is being used in the current comparison) into a linked list.

13. Set the Total_Length of Gridlet to zero.
14. Get the MIPS of next resource. If all the resources from the resource list are assigned with Grouped_Gridlets, continue the scheduling process by getting the MIPS of the first resource in the list.
15. Multiply the Resource_MIPS with Granularity_Time specified by the user.
16. Get the length of next Gridlet.
17. Repeat step 7.
18. Perform the looping until all the Gridlets in the Gridlet_List are grouped into Grouped_Gridlet.
19. After all the Gridlets are scheduled into groups, and each Gouped_Gridlets are assigned with a particular Grid resource, send all the Gouped_Gridlets to their corresponding resources.
20. After all the Gouped_Gridlets are processed by the Gris resources, and sent back to the I/O queue of the scheduler/system, collect the Gouped_Gridlets from the I/O queue.
21. Set the resource ID, and job execution cost of each Gouped_Gridlets.
22. Store the collected Gouped_Gridlets into a new Gridlet_List_3.
23. Get the total simulation time.
24. Display the details of the processed Gouped_Gridlets to the user through GUI.

6.5 Simulation Time

The total simulation time is calculated in seconds based on the overhead time for processing Gridlets at the Grid resources, and the time taken to perform the scheduling (grouping) task, sending Gridlets to the resources, and receiving back the processed Gridlets. The overhead processing time is taken from the user. This is shown in Figure 11.0.

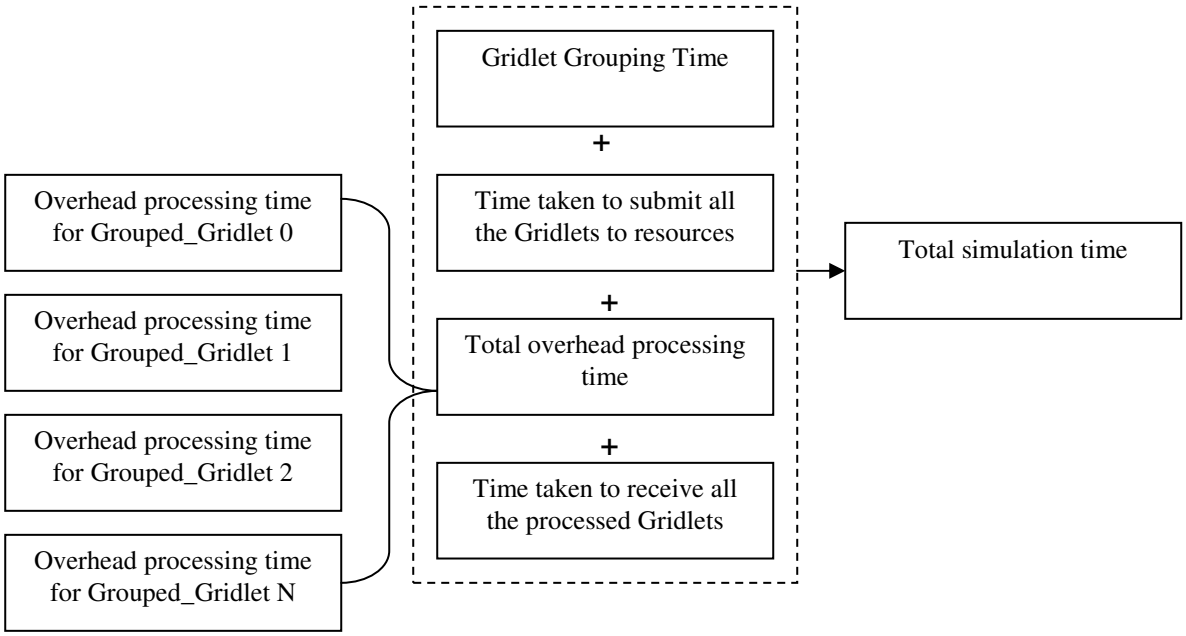


Figure 13.0 Calculating Total Simulation Time

Figure 14.0 displays the code listing for calculating total simulation time.

```
// start timer
Time1 = clock();

…….. SIMULATION……..

// total overhead time
overhead            = overhead*list2.size();

……

Time2               = clock(); // get the finish time
Total_Time          = Time2-Time1; // total simulation time
output              += "\nTotal overhead time: " + overhead + " seconds";
output              += "\nTotal Simulation Time: " + Total_Time + " seconds";
Total_Time          = Total_Time + (double)overhead;
output              += "\nTotal Time: " + Total_Time + " seconds\n";
```

Figure 14.0 Code Listing for Calculating Simulation Time

7.0 EXPERIMENTS AND DISCUSSIONS

The system or the scheduling program is tested using different inputs (total number of Gridlet, average MI of Gridlets, MI deviation percentage, granularity time, resource MIPS, and Gridlet overhead processing time), and the results are given in the following sections. The tests are done using nine resources of different MIPS, namely, R1: 20 MIPS, R2: 24 MIPS, R3: 39 MIPS, R4: 120 MIPS, R5: 60 MIPS, R6: 72 MIPS, R7: 66 MIPS, R8: 40 MIPS, and R9: 50 MIPS.

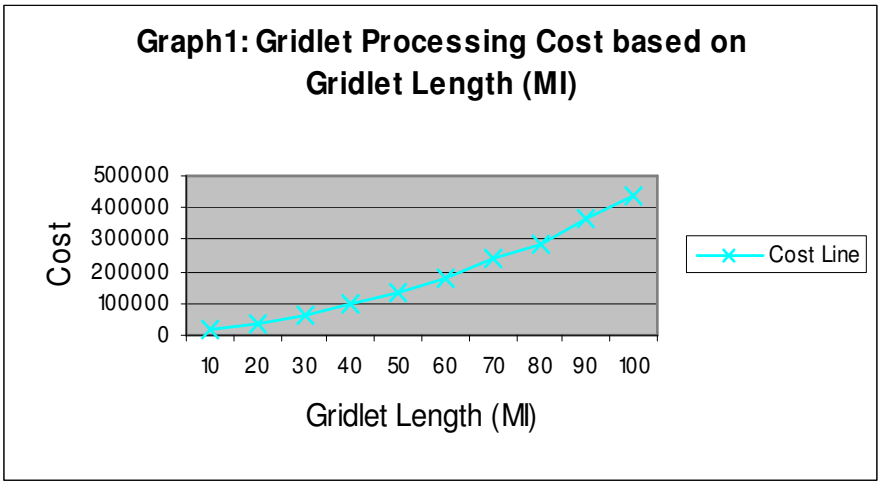
A_MI	***: Average MI rating of Gridlet or Gridlet length in MI***
G_Time	***: Granularity Time in seconds***
R_MIPS	***: Resource processing capabilities in MIPS***
D_%	***: MI Deviation Percentage***
OH_Time	***: Overhead Processing Time of Gridlet in seconds***
SimTime	***: Simulation Time in seconds***
Group	***: Number of Gridlet groups, resulted from Gridlet grouping process***
Cost	***: Processing cost of the Gridlets***
	Processing Cost = actual CPU Time * cost per sec
G_Time	***: Granularity time in seconds***

7.1 Experiment 1.0: Gridlets with Various Lengths (MI)

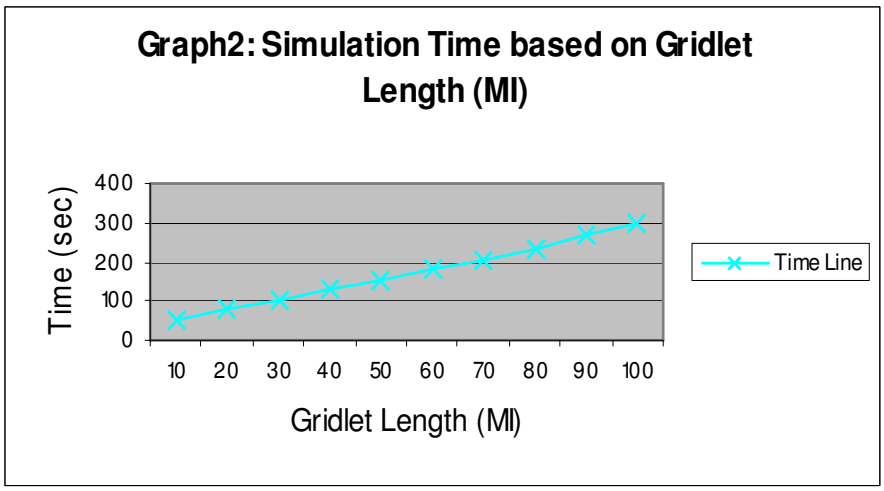
In this section, the scheduler is tested using 100 Gridlets of different average MI. The resulting Gridlet groups, total simulation time, and processing cost are given in the table.

A_MI	Group	SimTime	Cost	
10	4	54.25	16935.6	
20	8	77.57	36183.7	***Gridlets:100***
30	11	100.06	59739.2	***D_%:10***
40	15	128.26	94418.5	***G_Time:10***
50	18	152.04	133620.0	***Resource: R1,R2,R3***
60	21	179.00	176889.9	***R_MIPS:20,24,39***
70	24	202.01	238973.7	***OH_Time:4***
80	28	233.11	285047.6	
90	33	267.38	364229.7	
100	37	298.32	433678.0	

Table 1.0 100 Gridlets with Different Gridlet Lengths



(i)



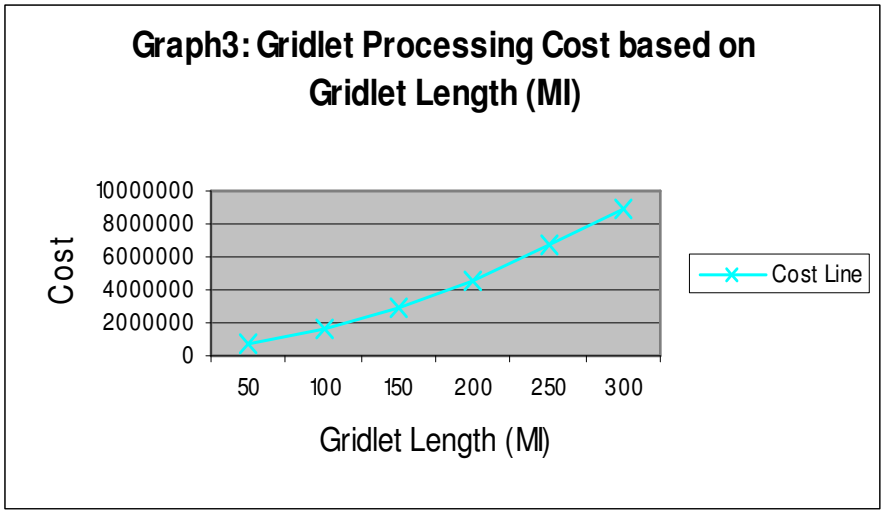
(ii)

Figure 15.0 Gridlet Processing Cost (i), and Simulation Time (ii) based on Different Gridlet Lengths for 100 Gridlets

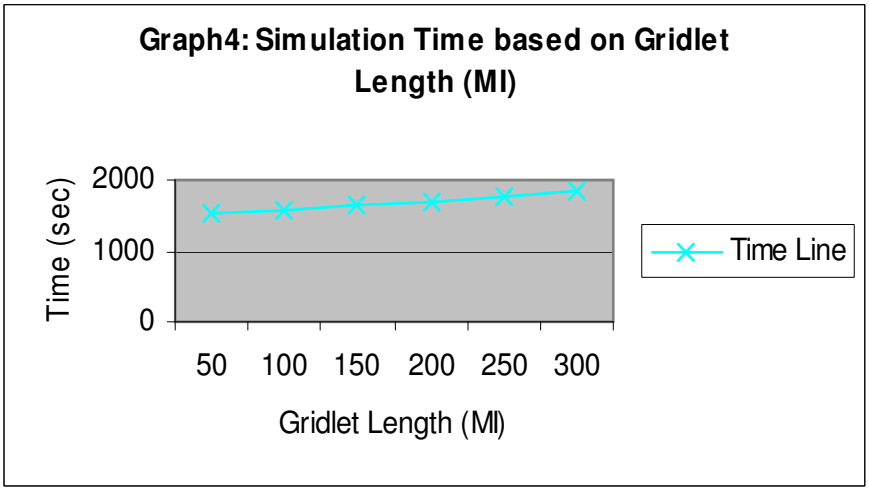
The scheduler is tested using 1000 Gridlets of different average MI. The resulting Gridlet groups, total simulation time, and processing cost are given in the table.

A_MI	Group	SimTime	Cost	
50	6	1510.70	813802.6	***Gridlets:1000***
100	12	1576.36	1711522.0	***D_%:30***
150	17	1634.27	2863629.0	***G_Time:180***
200	23	1702.48	4464123.0	***Resource: R1,R2,R3,R4,R5***
250	28	1761.43	6763464.0	***R_MIPS:20,24,39,120,60***
300	34	1839.17	8999351.0	***OH_Time:10***

Table 2.0 1000 Gridlets with Different Gridlet Lengths



(i)



(ii)

Figure 16.0 Gridlet Processing Cost (i), and Simulation Time (ii) based on Different Gridlet Lengths for 1000 Gridlets

The total simulation time, and cost increase as the average MI of Gridlets increases. This is because, once the Gridlet length increases, the processing load of the resources will increase as well. The total number of Gridlet groups rises as the Gridlet length increases since Gridlet grouping depends on the supported MIPS of each resource.

7.2 Experiment 2.0: Simulation With and Without Gridlet Grouping

In this section, the scheduler is tested using different number of Gridlets, and granularity time with, and without going through the Gridlet grouping task. The average Gridlet length is 20 MI; MI deviation percentage is 10%; Gridlet overhead processing time is 5 seconds; and the Grid resources are R1, R2, R3, and R4 with a total 203 MIPS.

Granularity time: 5 seconds

	With Grouping			Without Grouping	
Gridlets	**Group**	**SimTime**	**Cost**	**SimTime**	**Cost**
100	8	93.17	31927.0	603.33	36115.7
200	16	147.23	63179.4	1203.54	70860.3
300	24	202.16	95296.8	1803.38	106955.3
400	32	260.00	127630.1	2403.28	142567.6
500	40	315.08	159297.2	3002.96	177143.2
600	47	366.56	190829.1	3603.75	213919.0
700	55	421.92	222346.8	4204.20	247909.6
800	63	478.50	253651.6	4803.90	282601.2
900	70	530.01	284828.9	5404.30	316905.7
1000	77	579.90	316730.2	6004.53	352676.6
A_MI:20	***D_%:10%***	***G_Time:5***	***R_MIPS: 20,24,39,120***		***OH_Time:5***

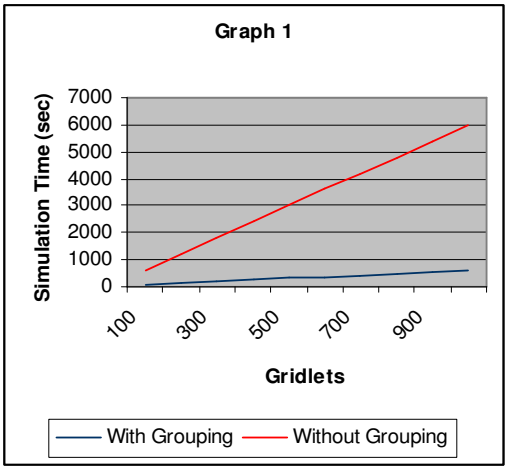


Table 3.0 Simulation With and Without Grouping for Granularity Time of 5 Seconds

Granularity time: 10 seconds

	With Grouping			Without Grouping	
Gridlets	**Group**	**SimTime**	**Cost**	**SimTime**	**Cost**
100	4	105.02	31315.3	604.28	36630.0
200	8	130.28	62842.6	1205.12	73302.2
300	12	158.30	95792.8	1804.34	109680.0
400	16	186.14	132129.3	2405.10	145763.1
500	20	214.30	169957.4	3004.16	181779.3
600	24	243.15	209876.2	3603.35	217804.4
700	28	270.18	254819.2	4203.81	252717.9
800	32	299.34	301143.4	4803.50	288519.3
900	36	334.90	351894.4	5405.06	324481.7
1000	40	366.32	408494.5	6005.29	359661.9
A_MI:20	***D_%:10%***	***G_Time:10***	***R_MIPS: 20,24,39,120***		***OH_Time:5***

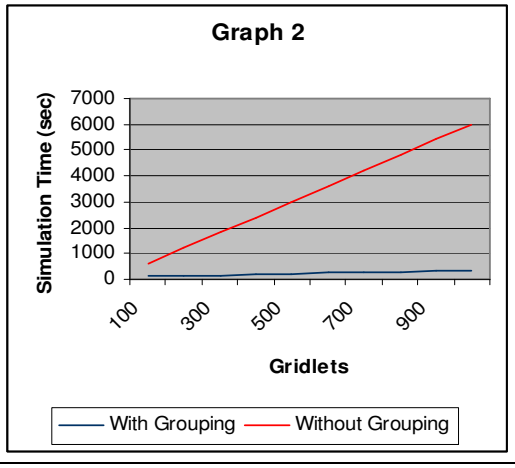


Table 4.0 Simulation With and Without Grouping for Granularity Time of 10 Seconds

Granularity time: 15 seconds

	With Grouping			Without Grouping	
Gridlets	**Group**	**SimTime**	**Cost**	**SimTime**	**Cost**
100	4	78.35	33569.3	604.64	39354.2
200	7	165.62	62323.5	1205.71	73848.4
300	8	170.62	94509.7	1804.63	109551.8
400	12	198.40	131966.1	2404.43	146207.8
500	15	222.22	172245.4	3004.10	178082.2
600	16	227.22	209989.5	3603.29	215397.0
700	20	254.84	267517.7	4203.74	251952.5
800	22	272.42	310808.1	4804.38	282676.3
900	24	282.42	360748.8	5404.78	321291.9
1000	28	310.56	440387.4	6005.01	360374.7
A_MI:20	***D_%:10%***	***G_Time:15***	***R_MIPS: 20,24,39,120***		***OH_Time:5***

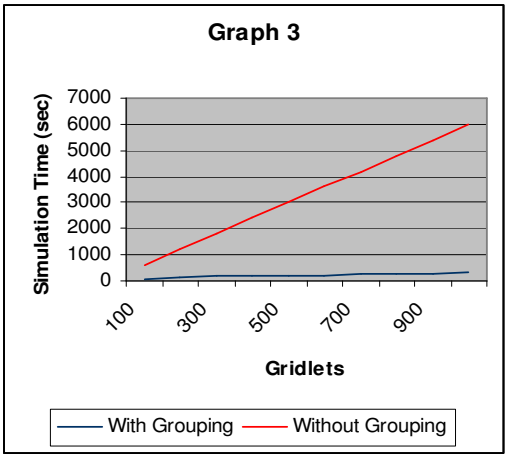


Table 5.0 Simulation With and Without Grouping for Granularity Time of 15 Seconds

Granularity time: 20 seconds

	With Grouping			Without Grouping	
Gridlets	**Group**	**SimTime**	**Cost**	**SimTime**	**Cost**
100	4	88.50	34822.8	604.65	41474.5
200	4	182.19	62583.7	1204.85	71740.8
300	8	210.33	97770.5	1804.34	112817.7
400	8	210.33	125571.8	2404.14	143705.1
500	12	239.40	168498.3	3003.20	182374.9
600	12	239.40	196577.0	3602.42	213197.9
700	15	261.75	261479.4	4204.01	251471.5
800	16	266.75	289732.3	4803.71	283471.1
900	19	289.68	360326.9	5404.11	321058.1
1000	20	294.68	388698.3	6004.34	353549.8
A_MI:20	***D_%:10%***	***G_Time:20***	***R_MIPS: 20,24,39,120***		***OH_Time:5***

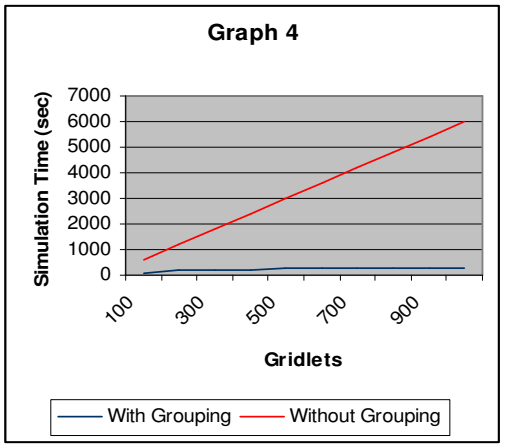


Table 6.0 Simulation With and Without Grouping for Granularity Time of 20 Seconds

Granularity time: 25 seconds

	With Grouping			Without Grouping	
Gridlets	**Group**	**SimTime**	**Cost**	**SimTime**	**Cost**
100	3	89.76	35373.9	604.66	44155.4
200	4	155.30	64419.7	1204.86	77652.9
300	6	241.16	91347.1	1807.11	113430.7
400	8	251.16	129165.2	2406.91	154999.8
500	8	251.16	156884.3	3005.97	185216.5
600	11	274.17	201115.3	3605.16	222575.0
700	12	279.17	231948.5	4205.61	258218.9
800	14	297.18	272476.9	4805.31	288741.9
900	16	307.18	340266.8	5405.71	333139.4
1000	16	307.18	368136.9	6005.94	363618.8
A_MI:20	***D_%:10%***	***G_Time:25***	***R_MIPS: 20,24,39,120***		***OH_Time:5***

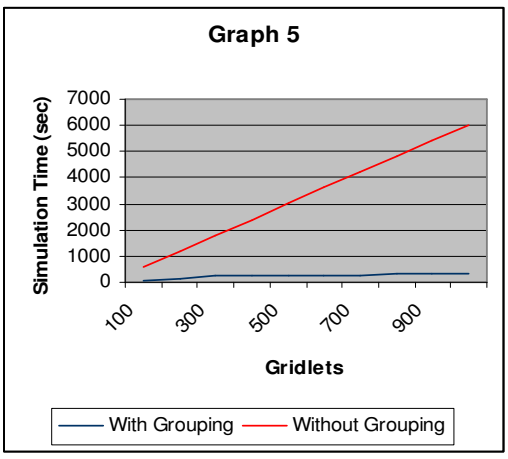


Table 7.0 Simulation With and Without Grouping for Granularity Time of 25 Seconds

The total simulation time, and cost increase gradually as the number of Gridlet increases if the simulation is done without Gridlet grouping method. Simulation with Gridlet grouping method results in reduced total simulation time. The total processing costs for the above experiments do not differ much between the two types of simulation.

In simulation with Gridlet grouping method, the total transmitting time of Gridlets is decreased since a large number of Gridlets are grouped into a few Gridlets, and sent to the Grid resources. In addition, the overhead processing time of each Gridlet is cut down since only a small number Gridlets (grouped Gridlets) are processed at the Grid resources. On the other hand, sending one Gridlet at one time (simulation without Gridlet grouping method) takes into account the transmitting effort, and overhead processing time of each, small Gridlets. The total processing cost depends on the MI, and total CPU time (per second) used for processing the Gridlets.

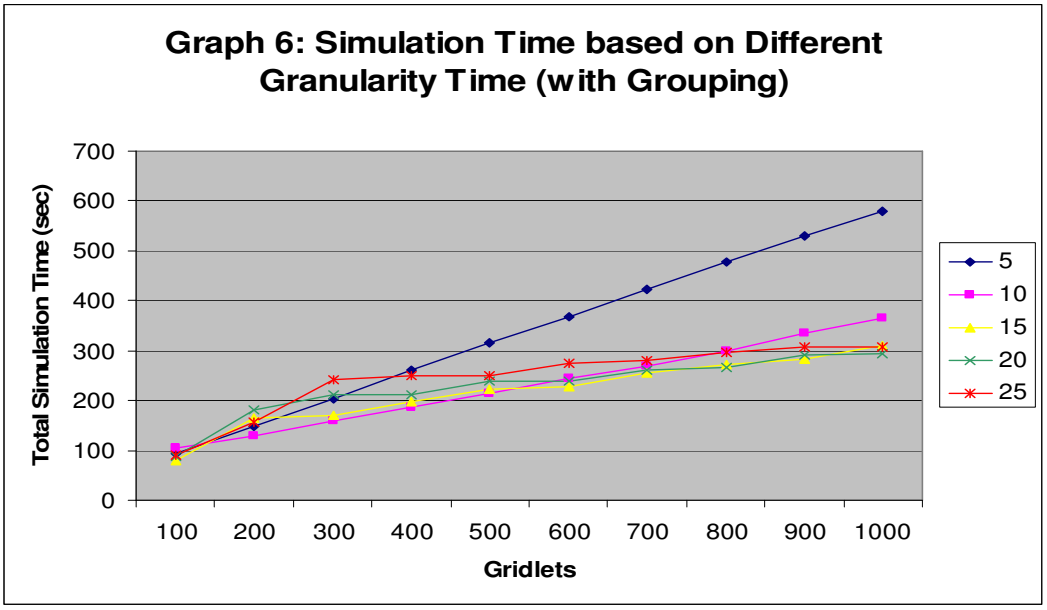


Figure 17.0 Simulation Time based on Different Granularity Time for Simulation With Grouping

Graph 6 shows the simulation time for different granularity time in simulation with Gridlet grouping. Higher the granularity time, lesser the total simulation time. This is because, the given granularity time is multiplied with the resource MIPS before the simulation starts. The resulting MIPS from multiplication depicts the total MI that the resource can process within the given granularity time. Therefore, if the granularity time is high, each resource can support more MI within that particular granularity time.

7.3 Experiment 3.0: Processing Gridlets within the Granularity Time

In this section, the scheduler is tested using different number of Gridlets (with Gridlet grouping method), resources, and granularity time to find out the total number of grouped Gridlets that can be processed successfully within a particular granularity time. The average Gridlet length is 40 MI; MI deviation percentage is 20%; and the Gridlet overhead processing time is 10 seconds.

Total Number of Gridlets is 100

		Granularity Time (sec)					
Resource	**R_MIPS**	**10**	**20**	**30**	**40**	**50**	**60**
R1	20	4	10	15	20	25	30
R1-R2	20,24	10	23	34	45	56	67
R1-R3	20,24,39	21	43	64	84	**100**	**100**
R1-R4	20,24,39,120	52	**100**	**100**	**100**		
R1-R5	20,24,39,120, 60	68					
R1-R6	20,24,39,120,60, 72	86					
R1-R7	20,24,39,120,60, 72,66	**100**					
Gridlet:100 OH_Time:10 A_MI:40 D_%:20%							

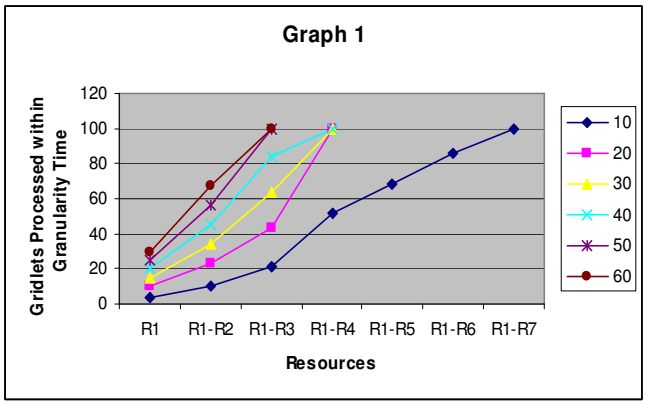


Table 8.0 Total number of Processed Gridlets (out of 100 Gridlets) for Different Granularity Time, and Resources

Total Number of Gridlets is 200

		Granularity Time (sec)					
Resource	**R_MIPS**	**10**	**20**	**30**	**40**	**50**	**60**
R1	20	4	10	15	20	25	30
R1-R2	20,24	10	23	34	45	56	67
R1-R3	20,24,39	21	43	64	84	105	126
R1-R4	20,24,39,120	52	103	153	**200**	**200**	**200**
R1-R5	20,24,39,120, 60	68	133	**200**			
R1-R6	20,24,39,120,60, 72	86	170				
R1-R7	20,24,39,120,60, 72,66	103	**200**				
R1-R8	20,24,39,120,60, 72,66,40	114					
R1-R9	20,24,39,120,60, 72,66,40,50	127					
Gridlet:200 OH_Time:10 A_MI:40 D_%:20%							

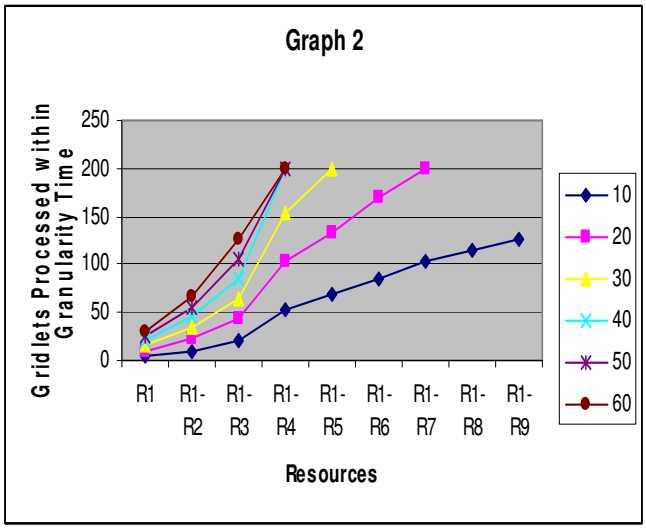


Table 9.0 Total number of Processed Gridlets (out of 200 Gridlets) for Different Granularity Time, and Resources

Total Number of Gridlets is 300

		Granularity Time (sec)					
Resource	**R_MIPS**	**10**	**20**	**30**	**40**	**50**	**60**
R1	20	4	10	15	20	25	30
R1-R2	20,24	10	23	34	45	56	67
R1-R3	20,24,39	21	43	64	84	105	126
R1-R4	20,24,39,120	52	103	153	206	256	**300**
R1-R5	20,24,39,120, 60	68	133	200	265	**300**	
R1-R6	20,24,39,120,60, 72	86	170	253	**300**		
R1-R7	20,24,39,120,60, 72,66	103	205	**300**			
R1-R8	20,24,39,120,60, 72,66,40	114	225				
R1-R9	20,24,39,120,60, 72,66,40,50	127	250				
Gridlet:300 OH_Time:10 A_MI:40 D_%:20%							

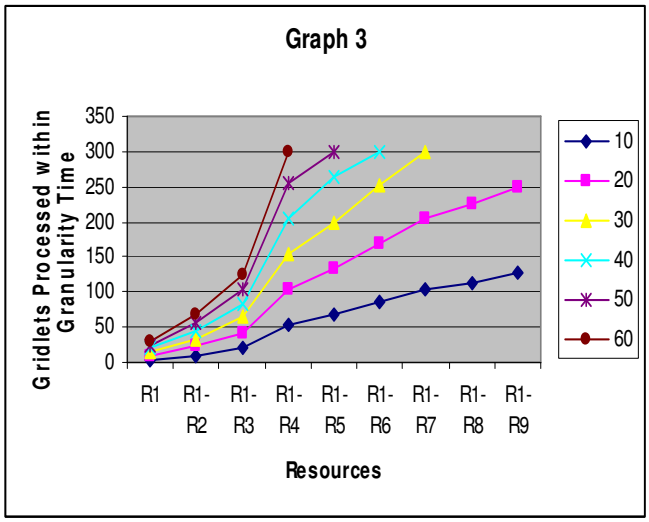


Table 10.0 Total number of Processed Gridlets (out of 300 Gridlets) for Different Granularity Time, and Resources

From the above tables, it is clear that when the granularity time is less, more resources are needed to process the given Gridlets within that granularity time. If the granularity time is high, the total number of resources used for processing the Gridlets can be reduced. As mentioned earlier, the given granularity time is multiplied with the resource MIPS before the simulation starts, and therefore, the supported MI of each resource will increase if the granularity time is high.

	Granularity Time (sec)					
Resource	**10**	**20**	**30**	**40**	**50**	**60**
R1:20	200	400	600	800	1000	1200
R2:24	240	480	720	960	1200	1440
R3:39	390	780	1170	1560	1950	2340
R4:120	1200	2400	3600	4800	6000	7200
R5:60	600	1200	1800	2400	3000	3600
R6:72	720	1440	2160	2880	3600	4320
R7:66	660	1320	1980	2640	3300	3960
R8:40	400	800	1200	1600	2000	2400
R9:50	500	1000	1500	2000	2500	3000

Table 11.0 Total Gridlet Lengths (MI) Supported by Each Resource

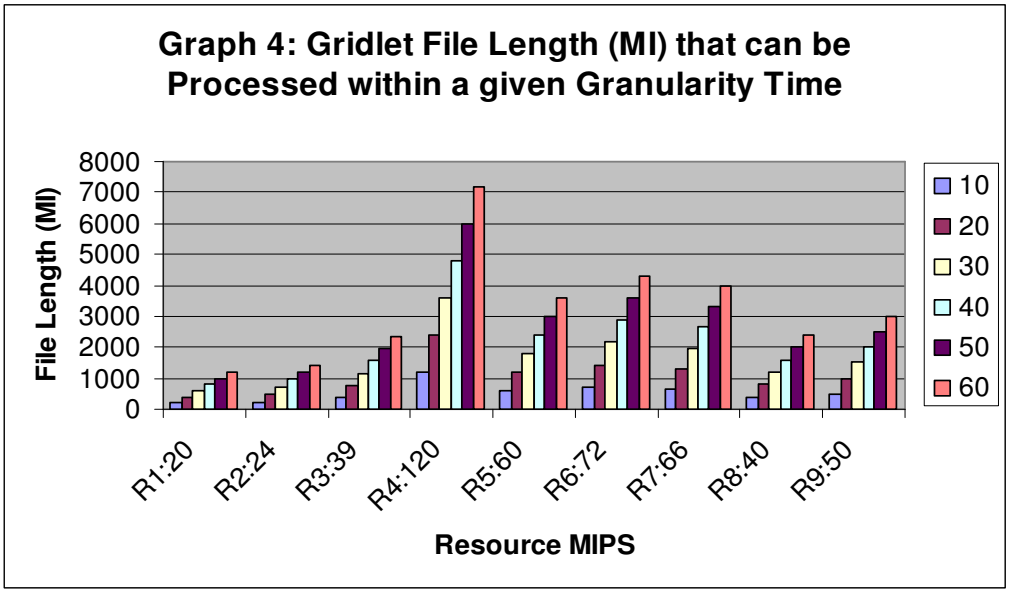


Figure 18.0 Gridlet Length Supported by Each Resource at a Given Granularity Time

Table 11.0, and graph 4 show the total Gridlet Lengths (MI) that can be processed by each resource within different granularity time. Higher the granularity time, higher the capability of each resource for processing the Gridlets.

8.0 CONCLUSION

From the experiments or tests in section 7.0, it is clear that the total simulation time highly depends number of Gridlets, Gridlet length (MI), resource MIPS, and Gridlet overhead processing time. Higher the number of Gridlets (or higher the Gridlet length), longer the simulation time. The simulation time can be decreased by performing Gridlet grouping method where large number of Gridlets are grouped into a few Gridlet groups, and sent to the Grid resources. Gridlet grouping reduces the transition time of each Gridlet to the resource, and the overhead processing time of each Gridlet at the resource. In addition, Gridlet grouping method will fully utilize the processing capabilities of the resources since the Gridlet grouping is done based on the processing capability of each resource. Another way to reduce the simulation time is to increase the number of resources used to execute the Gridlets. However, this will increase the cost of involving more processing resources in the simulation.

Granularity time is used in the experiments to test the simulation effort in a slightly different way. The purpose is to test how many Gridlets can be processed by a group of resources within a particular time. Higher the granularity time, higher the number of Gridlets that can be completed within that granularity time. However, the simulation still takes into account the Gridlet length, and resource MIPS.

In short, simulation time of a number of Gridlets can be reduced with higher granularity time, higher resource MIPS, and Gridlet grouping method. In other words, coarse-grained Grid application will reduce the total time taken to execute all the user jobs in a Grid environment compared to fine-grained Grid application.

REFERENCE

[1] Mark Baker, Rajkumar Buyya, and Domenico Laforenza, ***Grids, and Grid Technologies for Wide-Area Distributed Computing,*** International Journal of Software: Practice, and Experience (SPE), Volume 32, Issue 15, Pages: 1437-1466, Wiley Press, USA, December 2002.

[2] Rajkumar Buyya, and Manzur Murshed, ***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), Volume 14, Issue 13-15, Pages: 1175-1220, Wiley Press, USA, November - December 2002.